Une boîte à outils pour la vérification de programmes LOTOS

Jean-Claude FERNANDEZ* Hubert GARAVEL† Laurent MOUNIER*
Anne RASSE* Carlos RODRIGUEZ† Joseph SIFAKIS*

*LGI-IMAG
IMAG Campus BP 53X
38041 GRENOBLE cedex
FRANCE

†VERILOG Rhône-Alpes
Forum du Pré Milliet
38330 MONTBONNOT SAINT-MARTIN
FRANCE

e-mail: sifakis@imag.imag.fr

Cet article présente les logiciels Aldébaran, Cæsar, Cæsar.adt *et* Cléopâtre *qui constituent une* boîte à outils *pour la compilation et la vérification de programmes* Lotos. *Les principes de fonctionnement des différents outils sont exposés, ainsi que leurs performances et leurs limitations respectives. L'utilisation pratique de la boîte à outils est illustrée par un exemple de vérification formelle du protocole de diffusion atomique rel/REL.*

Mots-clés : compilation, vérification formelle, diagnostic, Lotos, *graphes d'états, bisimulations, logiques temporelles.*

Introduction

Le besoin de méthodes rigoureuses d'aide à la conception des systèmes parallèles a motivé de nombreux travaux de recherche sur les formalismes de spécification et les méthodes et outils de vérification associés.

Par vérification, on entend la comparaison d'un système — que l'on suppose donné par un *programme* exprimé dans un langage possédant une

sémantique opérationnelle bien définie — avec ses *spécifications*, c'est-à-dire la description des services qu'il doit rendre à l'ensemble de ses utilisateurs. Selon le formalisme utilisé pour les spécifications, on distingue deux approches :

spécifications comportementales : elles décrivent le *comportement* du système observé à un certain niveau d'abstraction. De telles spécifications sont généralement exprimées par un *système de transitions*, c'est-à-dire un ensemble d'états et une relation de transition entre ces états. Les algèbres de processus, les automates, ou même le langage utilisé pour décrire le système à vérifier, peuvent être employés à cet effet.

Compte-tenu que le programme à vérifier et ses spécifications comportementales peuvent tous deux être représentés par des systèmes de transitions, la vérification consiste à les comparer au moyen de *relations d'équivalence ou de préordre*. Toute procédure de décision pour ces relations constitue une méthode de vérification.

spécifications logiques : elles caractérisent des *propriétés globales* du système, telles que l'absence de blocage, l'exclusion mutuelle ou l'équité. Parmi les formalismes utilisés, les *logiques temporelles* s'avèrent être bien adaptées, car elles permettent de décrire globalement l'évolution du système dans le temps.

Dans ce cas, une *relation de satisfaction* liant les programmes aux formules logiques est définie ; une propriété est alors une assertion exprimant le fait que le programme satisfait une formule. Toute procédure de décision pour la relation de satisfaction définit une méthode de vérification.

L'étude des systèmes parallèles et réactifs (tels que les protocoles de communication, les systèmes de commande temps-réel et la description de matériel) a montré que des aspects essentiels de leur fonctionnement peuvent être modélisés par des systèmes de transitions finis.

Les méthodes de vérification les plus étudiées pour les systèmes finis sont les méthodes *basées sur les modèles* (*model-based methods*, qui incluent le cas particulier du *model-checking*). Elles consistent à engendrer un modèle fini (généralement un système de transitions) à partir du programme, puis à comparer ce modèle aux spécifications grâce à une procédure de décision. Bien que ces techniques soient efficaces et complètement automatisables, leur principale limitation est leur com-

plexité, en général exponentielle par rapport à la taille du programme.

Il existe de nombreux formalismes pour la description des systèmes parallèles, parmi lesquels le langage LOTOS [ISO88] occupe une place privilégiée, du fait notamment de son statut de norme internationale. Ce langage a fait l'objet de multiples travaux qui ont abouti à la réalisation d'outils : simulateurs [GHHL88] [vE89], compilateurs [MdM88], etc. Cependant, le problème de la vérification a été assez peu abordé, bien que LOTOS possède une sémantique formelle adaptée.

Cet article présente une *boîte à outils* pour la vérification formelle de programmes LOTOS. Elle intègre un ensemble cohérent d'outils permettant de traiter des spécifications comportementales et logiques, en utilisant les méthodes basées sur les modèles. Sont successivement décrits l'architecture générale de la boîte à outils, ses différents constituants, et un exemple montrant son utilisation pour la vérification d'un protocole de diffusion atomique.

1 Présentation de la boîte à outils

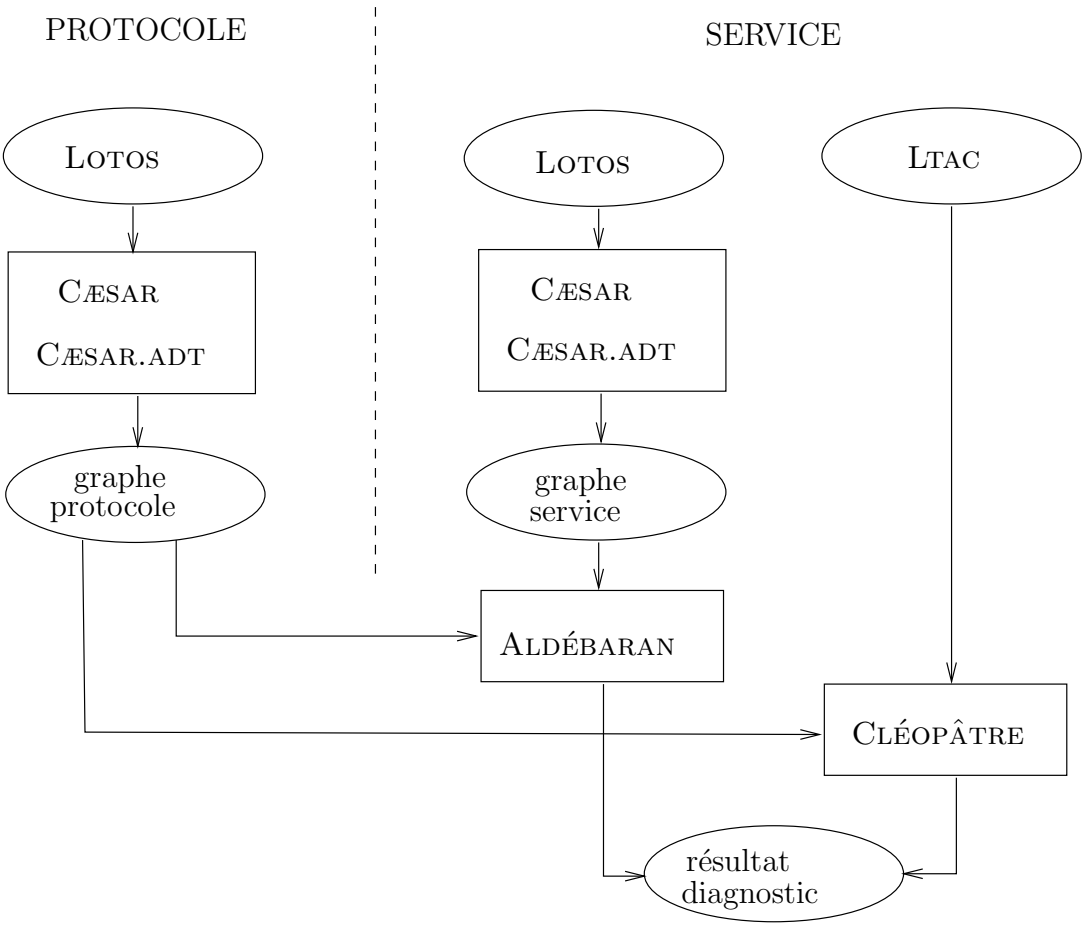
Les composants regroupés dans cette boîte à outils peuvent être classés en deux catégories :

les compilateurs : utilisés en amont, leur rôle est de traduire le programme à vérifier vers un modèle, appelé ici *graphe* (ou *graphe d'états*, ou *système de transitions étiquetées*, ou *automate d'états finis*). La boîte à outils contient deux compilateurs, CÆSAR.ADT et CÆSAR, qui traitent respectivement les données et le contrôle des programmes LOTOS.

les vérificateurs : utilisés en aval, ils effectuent des vérifications sur les graphes engendrés par les compilateurs. La boîte à outils comprend un vérificateur de spécifications comportementales, ALDÉBARAN, qui compare deux graphes modulo diverses relations, ainsi qu'un vérificateur de spécifications logiques, CLÉOPÂTRE, qui évalue des formules de la logique temporelle arborescente LTAC [QS83] sur un graphe. Ces outils sont capables, lorsque le programme à vérifier est incorrect, de fournir à l'utilisateur des *diagnostics* en termes de séquences dans le graphe.

La figure ci-dessous met en évidence l'architecture de la boîte à outils. Le protocole à vérifier est décrit en LOTOS, puis traduit vers un graphe.

Le service attendu est spécifié, soit par un programme LOTOS lui-même traduit en graphe, soit par des formules logiques LTAC.



2 Présentation de l'outil CÆSAR.ADT

CÆSAR.ADT [Gar89b] est un compilateur qui traduit les définitions de types abstraits figurant dans un programme LOTOS en un programme C. CÆSAR.ADT permet donc, à partir d'une spécification formelle, d'obtenir automatiquement une implémentation prototype correspondante.

2.1 Fonctionnalités de CÆSAR.ADT

En entrée, CÆSAR.ADT prend un programme LOTOS dont il ne traite que les définitions de types abstraits (les définitions de processus étant ignorées).

En sortie, CÆSAR.ADT produit une bibliothèque en langage C contenant, pour chaque sorte (*resp.* opération) définie dans le programme LOTOS un type (*resp.* une fonction) C qui l'implémente. Cette bibliothèque est destinée à être incluse dans un programme principal.

Le programme LOTOS doit comporter certains *commentaires spéciaux* qui permettent d'établir une correspondance entre le nom des objets LOTOS et le nom des objets C qui les implémentent.

CÆSAR.ADT impose certaines restrictions sur le sous-ensemble de LOTOS traité. L'utilisateur doit respecter la discipline de *programmation par constructeurs*, c'est-à-dire :

- Diviser l'ensemble des opérations LOTOS en *constructeurs* (opérations primitives) et *non-constructeurs* (opérations dérivées, exprimées en fonction des constructeurs). Les opérateurs constructeurs doivent être indiqués à l'aide de commentaires spéciaux.
- Orienter les équations pour qu'elles se comportent comme des règles de réécriture, le membre droit spécifiant comment le membre gauche doit être réécrit. La stratégie de réécriture choisie est l'appel par valeur, complété par une règle de priorité décroissante entre équations (si plusieurs équations peuvent simultanément s'appliquer, alors la première d'entre elles a précédence sur les suivantes).
- Veiller à ce que toute partie gauche d'équation ait la forme $f(v_1, \ldots\ v_n)$, avec $n \geq 0$, où f est un non-constructeur et $v_1, \ldots\ v_n$ des sous-termes formés uniquement de constructeurs et de variables quantifiées universellement (les non-constructeurs étant proscrits).

Ces restrictions constituent un bon compromis :

- Elles préservent la puissance d'expression théorique : toute définition de types abstraits LOTOS dont les constructeurs sont connus peut être automatiquement mise sous la forme demandée. En outre, les équations conditionnelles sont acceptées.
- Elles induisent une méthodologie pour définir des types abstraits exécutables de manière simple et intuitive.
- Elles permettent à CÆSAR.ADT d'effectuer un certain nombre de vérifications statiques sur la complétude et la consistance des équations.
- Elles autorisent CÆSAR.ADT à engendrer rapidement un code efficace.

La version actuelle de CÆSAR.ADT souffre de limitations conjoncturelles, qui ne sont pas dûes à une impossibilité théorique (par exemple les types paramétrés ne sont pas traités).

En revanche, elle permet à l'utilisateur d'importer, au sein du programme LOTOS, des types et des fonctions C définis de manière externe. Cette possibilité de combiner des types abstraits algébriques avec des types concrets définis séparément s'avère très utile en pratique.

2.2 Principes de fonctionnement de CÆSAR.ADT

CÆSAR.ADT effectue donc la traduction d'un formalisme déclaratif (les types abstraits de LOTOS) vers un langage impératif (le langage C). Après une phase d'analyse de la syntaxe, basée sur le système SYNTAX[1], suivie d'une phase d'analyse de la sémantique statique, cette traduction comprend trois étapes successives :

- La première étape effectue des transformations sur les équations (linéarisation, ...) afin de leur donner une forme exploitable par les phases suivantes.
- La seconde étape détermine la représentation en C des sortes et des constructeurs. La manière dont une sorte S est implémentée dépend exclusivement de l'ensemble des constructeurs dont le résultat est de sorte S. CÆSAR.ADT met en œuvre un algorithme général capable de compiler une sorte quelconque, complété par une collection d'algorithmes spécialisés pour implémenter de manière optimale certains cas particuliers (entiers, énumérations, tuples, ...).
- La troisième étape implémente les non-constructeurs, en utilisant un algorithme de compilation du *pattern-matching* [Sch88]. La fonction C engendrée pour un non-constructeur F est construite à partir des équations qui définissent F en partie gauche.

2.3 Evaluation et perspectives de CÆSAR.ADT

Les algorithmes mis en œuvre dans CÆSAR.ADT ont un coût suffisamment faible pour qu'aucun phénomène d'*explosion combinatoire* ne soit à redouter, ni en temps, ni en mémoire, ni en taille du code C engendré. A titre d'exemple, un "vrai" programme LOTOS comportant 2000 lignes de types abstraits (20 définitions de sortes et 290 définitions d'opérations) se compile en 20 secondes sur une station SUN 4 pour produire 8400 lignes de code C.

[1] SYNTAX est une marque déposée de l'INRIA

A l'heure actuelle, CÆSAR.ADT est principalement utilisé en conjonction avec CÆSAR pour effectuer la vérification formelle de protocoles et de systèmes répartis décrits en LOTOS. Mais on peut également l'employer dans d'autres domaines : par exemple, les types abstraits semblent bien adaptés pour spécifier les structures de données et les algorithmes utilisés dans les compilateurs. CÆSAR.ADT a ainsi servi à produire un compilateur pour la logique temporelle étendue XTL et un autre pour l'algèbre de processus temporisée ATP.

En ce qui concerne les évolutions futures, plusieurs sujets sont à l'étude, notamment l'implémentation optimale pour une plus large classe de types abstraits, ainsi que l'introduction d'une stratégie de gestion mémoire spécifiquement adaptée à l'utilisation de CÆSAR.

3 Présentation de l'outil CÆSAR

CÆSAR [Gar89a] [GS90] est un outil de compilation et de vérification pour les programmes LOTOS[2]. L'objectif initial de CÆSAR est d'appliquer à LOTOS les méthodes basées sur les modèles, en traduisant le programme source à vérifier en un graphe qui modélise toutes les évolutions possibles (états et séquences d'exécution) du programme.

3.1 Fonctionnalités de CÆSAR

En entrée, CÆSAR prend le programme LOTOS à vérifier, ainsi qu'une implémentation en C pour les types abstraits qu'il contient (soit écrite à la main, soit engendrée automatiquement par CÆSAR.ADT).

En sortie, CÆSAR produit, entre autres, un réseau de Petri étendu et un graphe. Les informations contenues dans ce graphe peuvent être exploitées par divers outils (réducteurs d'automates, évaluateurs de logiques temporelles ou de μ-calculs, outils de diagnostics). CÆSAR est capable d'engendrer ce graphe sous des formats multiples afin d'interfacer de nombreux outils existants : en premier lieu ALDÉBARAN et CLÉOPÂTRE (LGI-IMAG), ainsi que AUTO et AUTOGRAPH (INRIA), MEC (Université de Bordeaux), PIPN (LAAS-VERILOG), XESAR (LGI-IMAG), ...

CÆSAR impose la restriction suivante : ne sont acceptés que les programmes LOTOS qui ne comportent aucun appel récursif de processus à gauche ou à droite d'un opérateur de composition parallèle "|[...]|", ni

[2]Cet outil a été réalisé avec le soutien de l'INRIA

à gauche d'un opérateur de composition séquentielle "`>>`", ni à gauche d'un opérateur d'interruption "`[>`".

Ces contraintes visent à interdire la création/destruction dynamique d'instances de processus [GS90]. En pratique, elles réalisent un bon compromis entre le pouvoir d'expression offert à l'utilisateur et l'efficacité des algorithmes de compilation et de vérification.

3.2 Principes de fonctionnement de CÆSAR

Après les phases d'analyse de la syntaxe et de la sémantique statique (identiques à celles effectuées par CÆSAR.ADT), la traduction du programme LOTOS en graphe se fait en quatre étapes successives, qui sont définies formellement dans [Gar89a] et résumées dans [GS90] :

- la phase d'*expansion* traduit le programme LOTOS en un programme SUBLOTOS équivalent, SUBLOTOS étant une algèbre de processus que l'on peut voir comme un sous-ensemble simplifié de LOTOS.
- la phase de *génération* traduit le programme SUBLOTOS en une forme intermédiaire, appelée *réseau*, qui comprend :
  - une *partie contrôle*, basée sur les réseaux de Petri, qui consiste en un ensemble de *places* et de *transitions* sur lesquelles une structuration en *unités* permet de conserver la décomposition en processus communicants, qui existe dans le programme source LOTOS ;
  - une *partie données*, formée d'un ensemble de *variables* globales typées, dont les valeurs peuvent être consultées ou modifiées par des *actions* attachées aux transitions du réseau.
- la phase d'*optimisation* a pour objectif de réduire la complexité du réseau (c'est-à-dire de diminuer le nombre de places, de transitions, d'unités et de variables) en appliquant une liste de transformations qui préservent la sémantique pour la relation de bisimulation forte (*cf.*section 4.2). Ces optimisations portent aussi bien sur la partie contrôle du réseau (grâce à des transformations propres aux réseaux de Petri) que sur la partie données (grâce à des techniques d'analyse du flux des données analogues à celles utilisées dans les compilateurs).
- la phase de *simulation* construit le graphe correspondant au réseau ainsi optimisé. Elle effectue une exploration exhaustive du graphe ;

tous les états rencontrés sont conservés en mémoire principale, tandis que les arcs du graphe sont écrits au fur et à mesure dans un fichier. Dans son principe, cette phase s'apparente à la construction du graphe des marquages d'un réseau de Petri, mais il faut également prendre en compte la partie données du réseau (on doit pour cela engendrer, compiler puis exécuter un programme C qui fait appel à l'implémentation en C des types abstraits du programme LOTOS).

3.3 Evaluation et perspectives de CÆSAR

La version actuelle permet de construire, sur des stations de travail SUN, des graphes de grande taille (de l'ordre du million d'états) dans des temps raisonnables. La vitesse de génération peut atteindre 500 états par seconde, mais elle varie fortement selon le programme traité et la quantité de mémoire vive disponible.

Parmi les diverses applications de l'outil CÆSAR, on peut notamment citer la vérification d'un protocole de diffusion atomique [BM91] (*cf.* § 6), celle d'un sous-ensemble du protocole FIP [ADV90] et celle d'un protocole assurant la sécurité des dépassements entre véhicules automobiles [EFJ90].

A l'heure actuelle, les méthodes basées sur les modèles constituent une solution réaliste pour la vérification de petits et moyens programmes. Cependant elles ne sont pas directement applicables pour des problèmes plus importants, à cause de l'*explosion* du nombre d'états.

On peut en effet leur reprocher de commencer par engendrer des graphes de grande taille (pendant la phase de simulation de CÆSAR) pour les réduire ensuite (par exemple en utilisant ALDÉBARAN). Une solution attrayante consiste à ne pas vérifier *après* la simulation, mais *pendant* [JJ89], ou même *avant* [GS90] : en appliquant, pendant la phase d'optimisation, diverses transformations sur les réseaux de Petri engendrés par CÆSAR, on obtient des réductions au niveau du réseau qui, mêmes minimes, produisent des réductions considérables au niveau du graphe, ceci tout en préservant l'équivalence de bisimulation forte ou des équivalences plus faibles, comme l'équivalence de sûreté [Rod88].

Enfin, la méthode de traduction mise en œuvre par CÆSAR semble applicable à d'autres domaines que la vérification : simulation interactive ou intensive (au sens de l'outil VEDA [JGM88]), génération de code séquentiel ou distribué, génération de tests (dans ce sens, [ADV90] propose une utilisation conjointe de CÆSAR et d'un système de génération

de testeurs canoniques).

4 Présentation de l'outil ALDÉBARAN

ALDÉBARAN [Fer88] [Fer90] est un outil permettant la réduction et la comparaison de graphes par rapport à différentes relations d'équivalence et de préordre.

Parmi les relations utilisées pour la vérification des systèmes parallèles, la bisimulation forte [Par81] joue un rôle central d'un point de vue théorique. Cependant, cette relation est trop forte sur le plan pratique car elle identifie des graphes ayant *exactement* le même comportement. Aussi, afin de pouvoir prendre en compte lors de la comparaison certains critères d'abstraction, comme par exemple l'ensemble des actions non observables, on a recours à des relations d'équivalence ou de préordre plus faibles que la bisimulation forte (au sens où une relation est plus faible qu'une autre lorsqu'elle confond plus d'états).

Outre la bisimulation forte, ALDÉBARAN implémente notamment l'équivalence observationnelle [Mil80], l'équivalence par modèle d'acceptation [GS86], ainsi que le préordre et l'équivalence de sûreté [Rod88] qui préservent les *propriétés de sûreté* des systèmes [BFG+91].

4.1 Fonctionnalités d'ALDÉBARAN

ALDÉBARAN offre deux fonctionnalités correspondant à deux besoins pratiques distincts :

- On peut *comparer* le graphe d'un programme avec celui de ses spécifications comportementales modulo une relation d'équivalence ou de préordre. Dans ce cas, les deux graphes et la relation doivent être fournis en entrée, et le résultat de la comparaison (*vrai* ou *faux*) est obtenu en sortie. En outre, lorsque les deux graphes ne sont pas équivalents, ALDÉBARAN fournit un *diagnostic* constitué d'un ensemble de séquences d'exécution menant, à partir des états initiaux des deux graphes et en suivant les mêmes actions, à deux états où la non-équivalence apparaît clairement (il existe une action ne pouvant être effectuée que dans un seul des deux états).

- On peut *réduire* un graphe, c'est-à-dire calculer son *quotient* par rapport à une relation d'équivalence donnée (c'est-à-dire le plus petit

graphe équivalent pour cette relation au graphe original). Dans ce cas, le graphe et la relation d'équivalence sont fournis en entrée, et le graphe quotient est obtenu en sortie.

4.2 Principes de fonctionnement d'ALDÉBARAN

Deux approches principales existent pour décider si deux graphes se bisimulent fortement, qui peuvent toutes deux être étendues au cas des bisimulations faibles.

- La première approche, basée sur un calcul de point fixe, procède par raffinements successifs d'une partition initiale des états du graphe. Après stabilisation, la partition obtenue fournit les classes d'équivalence du graphe pour la bisimulation forte (c'est-à-dire les états du graphe quotient). Par suite, comparer deux graphes pour la bisimulation forte revient à comparer leurs quotients. Un algorithme efficace de raffinement de partitions, proposé par Paige & Tarjan [PT87], a été implémenté dans ALDÉBARAN.

  Cette méthode peut également être étendue aux bisimulations faibles en modifiant le graphe initial avant d'appliquer l'algorithme de Paige & Tarjan. Cette modification consiste à appliquer des opérations de fermeture transitive qui prennent en compte les critères d'abstraction (phase de *saturation*).

  La principale limitation de cette méthode est son coût en mémoire : elle nécessite en effet de mémoriser les états et les transitions des deux graphes. Ceci est particulièrement critique après la saturation qui accroît considérablement le nombre des transitions.

  En outre, les diagnostics obtenus lors de la comparaison sont difficiles à interpréter, car il s'agit de séquences d'exécution dans les graphes quotients (et non dans les graphes originaux), ce qui ne permet pas à l'utilisateur de faire un lien avec le programme source.

- La seconde approche [FM90] permet la comparaison "à la volée" de deux graphes en les parcourant en profondeur simultanément, ce qui s'assimile au calcul d'un produit synchrone d'automates. Selon la définition du produit synchrone utilisée, il est possible de traiter aussi bien la bisimulation forte que diverses bisimulations faibles ou relations de préordre. Cependant, à la différence de celui de Paige & Tarjan, cet algorithme ne permet pas d'effectuer des réductions de graphe.

En théorie cette méthode peut s'avérer plus coûteuse en temps que l'algorithme de Paige & Tarjan, mais sa complexité diminue lorsque l'un des deux systèmes est déterministe (ce qui est généralement le cas en pratique). En outre, dans le cas des bisimulations faibles, la phase de saturation n'est plus nécessaire. Il a été montré enfin que l'espace mémoire utilisé pour la comparaison "à la volée" pouvait être borné sans augmenter de façon significative le temps de calcul [JJ91].

En pratique, son introduction dans ALDÉBARAN a permis de traiter des exemples de plus grande taille et d'améliorer les temps d'exécution dans le cas des bisimulations faibles. Il présente également l'avantage de fournir des diagnostics directement exprimés en terme de séquences d'exécution sur les graphes originaux.

4.3 Evaluation et perspectives d'ALDÉBARAN

ALDÉBARAN a été réalisé en langage C, sous UNIX. Les performances obtenues sur stations de travail SUN 3 ou SUN 4 sont satisfaisantes pour des exemples de taille moyenne : en quelques minutes, il est possible de traiter des graphes de plusieurs milliers d'états modulo la bisimulation forte ou l'équivalence observationnelle, ou de comparer des graphes d'une centaine de milliers d'états modulo l'équivalence de sûreté.

Le plus souvent, ALDÉBARAN est utilisé avec CÆSAR de manière interactive, par exemple pour comparer le graphe d'un protocole au graphe du service attendu. ALDÉBARAN peut aussi être utilisé de façon "interne" par d'autres logiciels appelés à minimiser des graphes. A l'heure actuelle, ce type d'interface existe avec CLÉOPÂTRE et OCMIN, un optimiseur de code OC.

A l'avenir, il est prévu de rajouter de nouvelles équivalences à ALDÉBARAN. L'équivalence de trace, l'équivalence de branchement [vGW89] et l'équivalence de délai [NMV90] sont actuellement en cours d'implémentation.

Le problème de l'*explosion* du nombre d'états limite l'utilisation actuelle d'ALDÉBARAN à la comparaison de graphes de taille moyenne. Une solution prometteuse consisterait à combiner la phase de comparaison avec la phase de génération du graphe. En effet, l'algorithme "à la volée", qui ne nécessite pas de mémoriser de façon globale l'ensemble des deux graphes à comparer, permet d'envisager ce type d'approche. La comparaison ne se ferait donc plus à partir des graphes entièrement construits, mais au

fur et à mesure de leur construction.

5 Présentation de l'outil CLÉOPÂTRE

CLÉOPÂTRE est un outil de validation pour des spécifications exprimées dans la logique temporelle arborescente LTAC [QS83]. Il comprend :

- un module de vérification [Rod88], qui a pour but de décider, sur le graphe du programme, de la validité des formules,
- un module de diagnostic [Ras90], qui, lorsqu'une formule est fausse, fournit un diagnostic en termes de chemins dans le graphe.

5.1 Fonctionnalités de CLÉOPÂTRE

CLÉOPÂTRE prend en entrée le graphe engendré par CÆSAR à partir du programme LOTOS à vérifier et un ensemble de formules de la logique LTAC. Le sous-ensemble des formules LTAC utilisées pour la vérification des programmes LOTOS est décrit par la grammaire suivante :

$$T \mid init \mid enable(a) \mid after(a) \mid sink \mid f \ \wedge \ g \mid \neg f \mid inev[f]g \mid pot[f]g$$

où f et g sont des formules et a est un label étiquetant une transition du graphe.

Les modèles pour les formules de cette logique sont des *arbres d'exécution*, c'est-à-dire des arbres infinis obtenus en développant le graphe à partir d'un état. Une formule caractérise ainsi un ensemble d'états du graphe, l'ensemble des racines de ses modèles. On dit que ces états *satisfont* la formule et on note $s \models f$ le fait qu'un état s satisfait une formule f. La relation $\models$ est définie comme suit :

- Tous les états satisfont T.
- L'état initial du programme satisfait *init*.
- Un état s satisfait $enable(a)$ si l'action a peut être exécutée à partir de s.
- Un état satisfait $after(a)$ s'il est atteint juste après l'exécution de l'action a.
- Un état satisfait *sink* s'il n'a pas de successeur dans le graphe.

- Un état satisfait $f \wedge g$ s'il satisfait f et g.
- Un état satisfait $\neg f$ s'il ne satisfait pas f.
- Un état s satisfait $inev[f]g$ si, quelle que soit l'évolution possible à partir de s, f est satisfaite jusqu'à ce que g devienne vraie.
- Un état s satisfait $pot[f]g$ s'il existe une évolution à partir de s telle que f est satisfaite jusqu'à ce que g devienne vraie.

Dans la suite, les abréviations suivantes sont utilisées :

$$\begin{aligned} al[f]g &= \neg pot[f]\neg g \\ some[f]g &= \neg inev[f]\neg g \\ f \vee g &= \neg(\neg f \wedge \neg g) \\ f \Rightarrow g &= \neg f \vee g \end{aligned}$$

On appelle *opérateurs temporels* les opérateurs *pot*, *some*, *al* et *inev*. Ils permettent l'expression des propriétés usuelles des protocoles, parmi lesquelles on distingue généralement [AL88] :

- les propriétés de *sûreté*, exprimant le fait qu'il n'est pas possible que quelque chose de mauvais se produise. Une telle propriété sera décrite par une formule du type "$\neg pot\ mauvais$" ou "$al\ \neg mauvais$",
- les propriétés de *vivacité*, exprimant le fait qu'il est inévitable que quelque chose de bon se produise. Une telle propriété pourra être décrite par une formule du type "$inev\ bon$".

En sortie, CLÉOPÂTRE produit le résultat de l'évaluation de chaque formule, c'est-à-dire l'un des messages suivants : *formule valide*, *formule toujours fausse* ou *formule fausse dans k états sur n*. Dans les deux derniers cas, CLÉOPÂTRE fournit à l'utilisateur un diagnostic expliquant la cause de l'erreur. Une propriété de sûreté ou de vivacité portant sur *toutes* les exécutions du programme, il suffit, si elle n'est pas valide, de mettre en évidence une séquence d'exécution *contre-exemple*.

5.2 Principes de fonctionnement de CLÉOPÂTRE

L'évaluation des formules logiques met en œuvre des algorithmes optimisés de parcours de graphe [Rod88]. Chaque opérateur de la logique LTAC est ainsi évalué par un algorithme linéaire en temps et en espace par rapport à la taille du graphe.

Lorsqu'une formule f' n'est pas valide, CLÉOPÂTRE cherche alors à expliquer pourquoi il existe un état s qui satisfait f, où $f = \neg f'$. Un diagnostic pour "*f' n'est pas valide*" est donc une *explication* de $s \models f$. Pour expliquer pourquoi $s \models f$, on écrit f sous la forme disjonctive $\vee_i \wedge_j f_{i,j}$ où les $f_{i,j}$ sont :

- soit des prédicats T, $init$, $sink$, $enable(a)$, $after(a)$ ou leur négation,
- soit de la forme $op[f]g$, où $op = al$, $inev$, pot ou $some$, et où f et g sont sous forme disjonctive.

On procède alors par induction sur la structure de f en considérant l'opérateur le plus externe : on obtient l'explication de $s \models f$ en montrant que certaines sous-formules de f sont satisfaites par des états accessibles à partir de s, puis en répétant ceci récursivement sur les sous-formules de f :

- Pour T, $init$, $sink$ et leurs négations, l'explication est obtenue simplement en considérant l'état s ;
- Pour $enable(a)$, $after(a)$ et leurs négations, l'explication est obtenue en considérant les transitions qui arrivent à s ou celles qui en partent ;
- Pour les formules du type $f \wedge g$ (*resp.* $f \vee g$), on doit expliquer pourquoi $s \models f$ et $s \models g$ (*resp.* $s \models f$ ou $s \models g$) ;
- Pour les opérateurs al et $inev$, engendrer une explication n'aurait pas de sens, car ces opérateurs expriment des propriétés de toutes les exécutions du programme ; une explication devrait donc contenir l'ensemble des exécutions à partir d'un état, information de taille trop importante pour être exploitable. En pratique, ce cas ne se présente pas lorsque l'on traite uniquement des propriétés de vivacité et de sûreté.
- Pour les opérateurs pot et $some$, engendrer une explication consiste à rechercher un chemin partant de s dont les états doivent satisfaire certaines conditions. En outre, pour l'opérateur $some$, il faut parfois mettre en évidence un chemin infini, ce qui nécessite la détermination des composantes fortement connexes du graphe. Parmi tous les chemins répondant au problème, on sélectionne ceux de longueur minimale. En outre, CLÉOPÂTRE peut présenter l'ensemble des chemins sélectionnés de manière concise, sous la forme d'une expression ω-régulière sur le vocabulaire des actions du programme.

Comme l'algorithme d'évaluation des formules, l'algorithme de génération des diagnostics est linéaire en temps et en espace.

5.3 Evaluation et perspectives de CLÉOPÂTRE

CLÉOPÂTRE a été initialement réalisé pour XESAR [Rod88], outil de vérification pour une variante du langage ESTELLE. CLÉOPÂTRE a été ensuite adapté à LOTOS, afin de répondre au besoin d'un outil de diagnostic pour ce langage.

La version actuelle de CLÉOPÂTRE est capable d'évaluer en quelques secondes une formule de complexité moyenne (deux opérateurs temporels imbriqués) sur des graphes ayant plusieurs centaines de milliers d'états.

Concernant les évolutions futures de cet outil, il est prévu de l'adapter à d'autres formalismes de spécification, tels que les automates observateurs [JGM88] ou les expressions régulières. En outre, le problème du diagnostic étant très voisin de celui de la génération de séquences de test guidées par les propriétés, il est envisageable d'utiliser CLÉOPÂTRE à cet effet.

6 Vérification du protocole rel/REL

Cette section présente un exemple d'utilisation de la boîte à outils pour la vérification [BM91] d'un protocole de diffusion atomique. Un autre exemple de vérification formelle d'un protocole de même nature peut être trouvé dans [BGR+90].

6.1 Le protocole *rel/REL*

Le protocole *rel/REL* [SE90] permet d'effectuer des communications atomiques entre une station émettrice et un groupe de stations réceptrices, tout en tolérant les pannes potentielles de chacune des stations engagées dans la communication. Plus précisément, si une station E envoie un message m à un groupe de stations G, alors la propriété d'atomicité suivante est assurée : soit m est reçu par tous les éléments "en fonctionnement" de G, soit aucun d'entre eux ne le reçoit, et ce, malgré un nombre arbitraire de pannes dans le groupe $\{E\} \cup G$.

De telles facilités de communication sont requises, entre autres, pour la gestion des objets répliqués dans les systèmes tolérant les fautes, où des contraintes d'intégrité entre les différentes copies des objets doivent êtres

préservées.

Le protocole repose sur un protocole de niveau *transport*, qui assure une transmission *fiable* de messages entre deux stations quelconques du réseau. De plus, on considère que lorsqu'une station tombe en panne elle cesse définitivement d'émettre et recevoir des messages (*fail-silent behaviour*).

Le principe du protocole *rel/REL* est celui du verrouillage transactionnel à deux phases : l'émetteur envoie séquentiellement deux copies du message à transmettre à chacun de ses destinataires. Chaque message est identifié de manière unique par un numéro et une information indiquant s'il s'agit d'une première copie ou d'une deuxième copie.

Chaque station S ayant reçu la première copie est en attente de la seconde. Si celle-ci n'est pas reçue avant l'expiration d'un certain délai, alors S peut supposer que l'émetteur a subi une panne et que, par suite, certains des destinataires n'ont reçu aucune copie du message. S relaie alors l'émetteur en diffusant, à son tour, les deux copies du même message vers l'ensemble des destinataires (et donc en utilisant récursivement le protocole *rel/REL*). Toutefois, afin de limiter le nombre total de copies émises, cette rediffusion est elle-même interrompue si la seconde copie du message est reçue, venant soit de l'émetteur initial, soit d'un autre destinataire déjà en train de relayer l'émetteur.

On s'intéresse ici à l'une des deux versions du protocole décrites dans [SE90], celle qui préserve l'ordre des messages envoyés par un émetteur donné (rel/REL_{fifo}).

6.2 Description formelle du protocole

La description en LOTOS du protocole et des structures de données associées (numéros de messages, adresses des stations, tables et files d'attente pour mémoriser les messages reçus) n'a pas présenté de problème majeur. La seule difficulté a concerné la modélisation du fait que chaque station puisse tomber en panne à tout instant. La solution retenue s'inspire du style de programmation par contraintes (*constraint-oriented*) naturel en LOTOS : le comportement d'une station est représenté par la composition parallèle d'un processus décrivant son comportement normal et d'un processus simulant la panne éventuelle.

6.3 Description formelle du service et vérification

Les propriétés constituant le service attendu de rel/REL_{fifo} sont au nombre de deux ; elles peuvent être formalisées, soit à l'aide de la logique LTAC, soit par la donnée d'un système de transitions représentant le comportement attendu du protocole en fonction d'une relation d'équivalence et d'un ensemble d'actions visibles.

La première propriété concerne l'*atomicité* des transactions : "*un message doit être reçu, soit par l'ensemble de ses destinataires qui ne sont pas en panne, soit par aucun d'entre eux*". En notant get_i une réception de la première copie du message par la station i, et $crash_i$ l'action modélisant une panne de la station i, la propriété se formule ainsi : "*Pour toutes stations (i,j), il n'est pas possible qu'il existe une séquence d'exécution du protocole contenant l'action* get_i *et ne contenant pas l'action* $crash_i$ (la station i a reçu le message et elle n'a pas subi de panne) *et ne contenant ni l'action* get_j *et ni l'action* $crash_j$ (la station j est toujours en attente du message)."

Pour cela, on note $attente_i$ le fait que la station i soit en attente du message (elle ne l'a pas reçu et elle n'est pas en panne) :

$$attente_i = \neg after(get_i) \wedge \neg after(crash_i)$$

De plus, les hypothèses faites sur le comportement des stations en panne (*fail-silent behaviour*) impliquent que pour une station i, il n'est pas possible d'avoir une réception de message après une panne :

$$al[T](crash_i \Rightarrow al[T](\neg get_i))$$

La propriété d'atomicité s'exprime alors par la formule suivante :

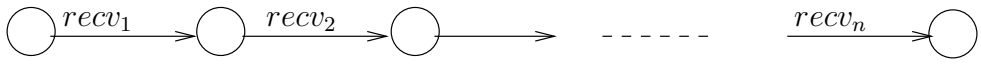
$$\neg \bigvee_{i \neq j} (pot[attente_j](after(get_i) \wedge some[T](attente_j) \wedge \neg after(crash_i)))$$

Cette propriété a été vérifiée grâce à CLÉOPÂTRE sur le graphe du protocole généré par CÆSAR (50 000 états et 150 000 transitions) pour une station émettrice, deux stations réceptrices et un message. En effet, l'étude du protocole a montré qu'il suffisait de faire la vérification avec un seul message [BM91]. Les diagnostics de CLÉOPÂTRE se sont révélés utiles pour la mise au point.

La seconde propriété à vérifier concerne la *préservation de l'ordre* des messages : "*la réception des messages envoyés par un émetteur donné doit préserver l'ordre dans lequel ils ont été émis*". Il s'agit d'une propriété de

sûreté (*cf.* section 5.1) qui exprime qu'un message reçu respecte certaines conditions, mais qui ne garantit pas que tout message émis soit reçu.

Cette propriété étant indépendante de l'émetteur et du récepteur considérés, il suffit de la vérifier pour un couple (émetteur, récepteur) arbitrairement fixé. En numérotant de manière croissante chaque message émis et en notant $recv_i$ la réception du message i par le récepteur considéré, le comportement attendu de ce récepteur se représente alors (modulo l'équivalence de sûreté) par le graphe de la figure suivante :



L'équivalence de sûreté prend ici tout son intérêt car elle autorise une expression très simple du service (on est en droit de l'utiliser car la propriété à prouver est bien une propriété de sûreté). Le graphe ci-dessus aurait été beaucoup plus complexe si l'on avait utilisé une autre équivalence, comme par exemple l'équivalence observationnelle.

La vérification s'est faite avec ALDÉBARAN en comparant ce graphe, modulo l'équivalence de sûreté, avec celui produit par CÆSAR (650000 états et 2000000 transitions) pour une station émettrice, deux stations réceptrices et trois messages. En raison de la taille du graphe du protocole, la comparaison a été faite en utilisant l'algorithme "à la volée".

La vérification du protocole rel/REL_{fifo} a permis d'apporter plusieurs précisions par rapport à la spécification informelle qui en était fournie. En outre, elle a apporté des informations intéressant l'implémentation, notamment l'existence de bornes sur la taille des structures de données utilisées (*threads*).

Conclusion

Cet article a présenté un ensemble d'outils destinés à la vérification formelle de systèmes décrits en LOTOS. Ces outils offrent une grande flexibilité d'utilisation, tant du fait de leur modularité que de leur intégration poussée. Leurs performances ont permis de les utiliser pour la vérification de protocoles relativement complexes.

Ces outils peuvent évoluer dans plusieurs directions. Des améliorations spécifiques à chaque outil ont déjà été suggérées. D'autres améliorations concernent l'architecture même de la boîte à outils, notamment l'ajout de nouveaux outils de vérification et de diagnostic pour des formalismes autres que les logiques temporelles classiques, ainsi que le développement

d'un format de graphe indépendant du langage source, qui permettrait de renforcer la coopération entre outils et d'élargir leur champ d'application.

Références

[**ADV90**] Pierre Azema, Khalil Drira, and François Vernadat. A Bus Instrumentation Protocol Specified in LOTOS. In Juan Quemada, José Manas, and Enrique Vázquez, editors, *Proceedings of the 3rd International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*. North-Holland, novembre 1990.

[**AL88**] M. Abadi and L. Lamport. The existence of refinement mappings. SRC 29, Digital Equipment Corporation, août 1988.

[**BFG+91**] Ahmed Bouajjani, Jean-Claude Fernandez, Susanne Graf, Carlos Rodríguez, and Joseph Sifakis. Safety for Branching Time Semantics. In *Proceedings of 18th ICALP*. Springer Verlag, juillet 1991.

[**BGR+90**] M. Baptista, S. Graf, J.-L. Richier, L. Rodrigues, C. Rodríguez, P. Verissimo, and J. Voiron. Formal Specification and Verification of a Network Independent Atomic Multicast Protocol. In Juan Quemada, José Manas, and Enrique Vázquez, editors, *Proceedings of the 3rd International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*. North-Holland, novembre 1990.

[**BM91**] Simon Bainbridge and Laurent Mounier. Specification and Verification of a Reliable Multicast Protocol. Technical Report HPL-91-163, Hewlett-Packard Laboratories, Bristol, U.K., octobre 1991.

[**EFJ90**] Patrik Ernberg, Lars-åke Fredlund, and Bengt Jonsson. Specification and Validation of a Simple Overtaking Protocol using LOTOS. T 90006, Swedish Institute of Computer Science, Kista, Sweden, octobre 1990.

[**Fer88**] Jean-Claude Fernandez. *ALDEBARAN : un système de vérification par réduction de processus communicants*. Thèse de Doctorat, Université Joseph Fourier (Grenoble), mai 1988.

[**Fer90**] Jean-Claude Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13(2–3):219–236, mai 1990.

[**FM90**] Jean-Claude Fernandez and Laurent Mounier. Verifying Bisimulations "On the Fly". In Juan Quemada, José Manas, and Enrique Vázquez, editors, *Proceedings of the 3rd International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*. North-Holland, novembre 1990.

[**Gar89a**] Hubert Garavel. *Compilation et vérification de programmes LOTOS*. Thèse de Doctorat, Université Joseph Fourier (Grenoble), novembre 1989.

[**Gar89b**] Hubert Garavel. Compilation of LOTOS Abstract Data Types. In Son T. Vuong, editor, *Proceedings of the 2nd International Conference on Formal Description Techniques FORTE'89 (Vancouver B.C., Canada)*, pages 147–162. North-Holland, décembre 1989.

[**GHHL88**] R. Guillemot, R. Haj-Hussein, and L. Logrippo. Executing Large LOTOS Specifications. In S. Aggarwal and K. Sabnani, editors, *Proceedings of the 8th International Workshop on Protocol Specification, Testing and Verification (Atlantic City, NJ, USA)*, pages 399–410. IFIP, North-Holland, 1988.

[**GS86**] Susanne Graf and Joseph Sifakis. Readiness Semantics for Processes with Silent Actions. Rapport SPECTRE C3, Laboratoire de Génie Informatique — Institut IMAG, Grenoble, novembre 1986.

[**GS90**] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, pages 379–394. IFIP, North-Holland, juin 1990.

[**ISO88**] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, septembre 1988.

[**JGM88**] Claude Jard, Roland Groz, and Jean-François Monin. Development of VEDA: A Prototyping Tool for Distributed Systems. *IEEE Transactions on Software Engineering*, 14(3), mars 1988.

[**JJ89**] Claude Jard and Thierry Jeron. On-Line Model-Checking for Finite Linear Temporal Logic Specifications. In Joseph Sifakis, editor, *Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems (Grenoble, France)*, volume 407 of *Lecture Notes in Computer Science*, pages 189–196. Springer Verlag, juin 1989.

[**JJ91**] Claude Jard and Thierry Jéron. Bounded-Memory Algorithms for Verification On-the-Fly. In K. G. Larsen and A. Skou, editors, *Proceedings of the 3rd Workshop on Computer-Aided Verification (Aalborg, Denmark)*, volume 575 of *Lecture Notes in Computer Science*, Berlin, juillet 1991. Springer Verlag.

[**MdM88**] J. A. Manas and T. de Miguel. From LOTOS to C. In Kenneth J. Turner, editor, *Proceedings of the 1st International Conference on Formal Description Techniques FORTE'88 (Stirling, Scotland)*, pages 79–84. North-Holland, septembre 1988.

[**Mil80**] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[**NMV90**] Rocco De Nicola, Ugo Montanari, and Frits Vaandrager. Back and Forth Bisimulations. CS R9021, Centrum voor Wiskunde en Informatica, Amsterdam, mai 1990.

[**Par81**] David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, mars 1981.

[**PT87**] Robert Paige and Robert E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, décembre 1987.

[**QS83**] Jean-Pierre Queille and Joseph Sifakis. Fairness and Related Properties in Transition Systems — A Temporal Logic to Deal with Fairness. *Acta Informatica*, 19:195–220, 1983.

[**Ras90**] Anne Rasse. *CLEO : diagnostic des erreurs en XESAR*. Thèse de Doctorat, Institut National Polytechnique de Grenoble, juin 1990.

[**Rod88**] Carlos Rodríguez. *Spécification et validation de systèmes en XESAR*. Thèse de Doctorat, Institut National Polytechnique de Grenoble, mai 1988.

[**Sch88**] Philippe Schnoebelen. Refined Compilation of Pattern-Matching for Functional Languages. *Science of Computer Programming*, 11:133–159, 1988.

[**SE90**] Santosh K. Shrivastava and Paul. D. Ezhilchelvan. rel/REL: A Family of Re-

liable Multicast Protocol for High-Speed Networks. Technical Report (in preparation), University of Newcastle, Dept. of Computer Science, U.K, 1990.

[**vE89**] Peter van Eijk. *The Design of a Simulator Tool*. In Peter van Eijk et al., editors, *The Formal Description Technique LOTOS*. North-Holland, 1989.

[**vGW89**] R. J. van Glabbeek and W. P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics (extended abstract). CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989. Also in proc. IFIP 11th World Computer Congress, San Francisco, 1989.

Curriculum vitae

Jean-Claude Fernandez, docteur en informatique, a soutenu une thèse traitant des méthodes efficaces pour le calcul des bisimulations, qui a conduit à la réalisation de l'outil ALDÉBARAN. Il est actuellement maître de conférences à l'université Joseph Fourier de Grenoble.

Hubert Garavel, ingénieur ENSIMAG, docteur en informatique, a consacré sa thèse à la compilation et à la vérification des programmes LOTOS, travail pour lequel il a obtenu le prix IBM 1990 du Jeune Chercheur en Informatique. Depuis 1989 il fait partie du groupe d'Etudes Avancées de VERILOG.

Laurent Mounier, actuellement en thèse sous la direction de Jean-Claude Fernandez, étudie les méthodes de vérification basées sur les bisimulations, notamment les nouveaux algorithmes mis en œuvre dans l'outil ALDÉBARAN.

Anne Rasse, ingénieur ENSIMAG, docteur en informatique, a consacré sa thèse aux techniques de diagnostic utilisées dans l'outil CLÉOPÂTRE. Elle enseigne actuellement à l'ENSIMAG.

Carlos Rodriguez, ingénieur, docteur en informatique, a soutenu une thèse sur le système XESAR, portant notamment sur les algorithmes d'évaluation des formules LTAC. Depuis 1990 il fait partie du groupe d'Etudes Avancées de VERILOG.

Joseph Sifakis, directeur de recherches au CNRS, anime le projet SPECTRE de l'IMAG. Ses travaux portent sur les méthodes de spécification et de vérification des systèmes parallèles.