INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet VASY

Validation de Systèmes

Rhône-Alpes

THÈME 1C

Table des matières

1 Composition de l'équipe

Responsable scientifique
Hubert Garavel [DR2 INRIA]

Assistantes de projet
Valérie Gardès
Anne-Marie Saez [à partir du 21 octobre 2002]

Personnel Inria
Radu Mateescu [CR1 INRIA]
Frédéric Lang [CR2 INRIA]

Personnel Bull
Solofo Ramangalahy [responsable d'action BULL]

Personnel Université Pierre Mendès France
Aurore Collomb [ATER, à partir du 1er septembre 2002]

Ingénieurs experts
Damien Bergamini [à partir du 1er septembre 2002]
David Champelovier [à partir du 1er septembre 2002]
Nicolas Descoubes [à partir du 1er septembre 2002]
Bruno Ondet
Frédéric Tronel

Chercheur post-doctorant
Gordon Pace [boursier ERCIM, jusqu'au 31 août 2002]

Chercheur doctorant
Christophe Joubert [à partir du 1er octobre 2002]

Stagiaires
Gilles Badoil [stagiaire de probatoire CNAM, du 24 janvier au 26 mars 2002]
Christophe Joubert [stagiaire DEA Université Joseph Fourier, jusqu'au 30 juin 2002]
Ghislain Kergadallan [stagiaire de probatoire CNAM, du 30 avril au 2 juillet 2002]
Gilles Stragier [stagiaire Université Libre de Bruxelles, jusqu'au 15 janvier 2002]

2 Présentation et objectifs généraux

2.1 Introduction

Créé le 1er janvier 2000 et faisant suite à l'action VASY "Recherche et Applications" (1er janvier 1997 – 31 décembre 1999), le projet VASY s'inscrit dans la problématique de la conception de systèmes sûrs par l'utilisation de méthodes formelles.

Plus précisément, nous nous intéressons à tout système (matériel, logiciel, de télécommunication) faisant intervenir du parallélisme *asynchrone*, c'est-à-dire tout système dont on peut modéliser le comportement parallèle par une sémantique d'entrelacement des

événements (*interleaving semantics*).

Pour la conception de systèmes sûrs, nous préconisons l'utilisation de techniques de description formelle, complétées par des outils informatiques adaptés, offrant des fonctionnalités de simulation, prototypage rapide, vérification et génération de tests.

Parmi les différentes approches existant pour la vérification, nous concentrons nos efforts sur l'approche dite *model checking*, qui recouvre un grand nombre de techniques spécialisées (vérification énumérative, à la volée, symbolique, etc.). Ces techniques, bien que moins générales que les approches par preuves (*theorem proving*), possèdent pourtant l'avantage de permettre une détection automatique, rapide et économique d'erreurs de conception dans des systèmes complexes.

Nos travaux se situent au confluent de deux grandes approches en méthodes formelles : l'approche basée sur des *modèles* (très répandue en Amérique du Nord) et l'approche basée sur des *langages* (plus développée en Europe) :

- Sous le terme de *modèles*, on désigne diverses représentations de programmes parallèles (automates, réseaux d'automates communicants, réseaux de Petri, diagrammes de décision binaire, etc.) ainsi que les algorithmes de vérification qui s'y appliquent. D'un point de vue théorique, il importe de rechercher des résultats généraux, donc indépendants de tout langage de description particulier, ce qui incite à la recherche de modèles mathématiques simples et expressifs.
- En pratique, ces modèles sont souvent trop rudimentaires pour servir à la description directe d'un système complexe (une telle approche est fastidieuse et comporte un fort risque d'erreur). C'est pourquoi il est indispensable de s'appuyer sur des formalismes de plus haut niveau (c'est-à-dire des *langages*) permettant de décrire des problèmes réels et complexes sous forme de programmes. Ces programmes sont ensuite analysés et traduits automatiquement vers des modèles sur lesquels opèrent les algorithmes de vérification.

Pour mener à bien la vérification de systèmes complexes (de taille "industrielle"), il nous semble nécessaire de maîtriser simultanément la technologie des modèles et celle des langages.

2.2 Technologie des modèles – vérification

Par vérification, on entend la comparaison — à un certain niveau d'abstraction — d'un système avec les *propriétés* qui décrivent le fonctionnement attendu du système (c'est-à-dire les services que celui-ci doit fournir).

Les techniques de vérification que nous mettons en œuvre reposent en grande partie sur le modèle des *systèmes de transitions étiquetées* (ou, plus simplement, *automates*, ou encore *graphes*) composés d'un ensemble d'états, d'un état initial et d'une relation de transition entre ces états. Ces techniques consistent à engendrer automatiquement, à partir de la description du système à vérifier, un graphe fini qui en modélise le comportement, puis à vérifier les propriétés sur le graphe grâce à une procédure de décision.

Selon le formalisme utilisé pour exprimer les propriétés, on distingue deux approches :

- *Propriétés comportementales :* elles décrivent le fonctionnement du système sous forme

d'automates (ou bien en utilisant des descriptions de plus haut niveau que l'on traduit ensuite en automates). Etant donné que le système à vérifier et ses propriétés comportementales peuvent tous deux être représentés par des automates, la vérification consiste à les comparer au moyen de *relations d'équivalence ou de préordre*.
Concernant la vérification de propriétés comportementales, nous développons un outil de minimisation pour la bisimulation forte et la bisimulation de branchement, qui permet aussi la minimisation de systèmes stochastiques et probabilistes. En outre, nous collaborons avec d'autres équipes qui développent des outils basés sur les relations d'équivalence et de préordre.

– *Propriétés logiques :* elles caractérisent des propriétés essentielles du système, telles que l'absence de blocage, l'exclusion mutuelle ou l'équité. Parmi les formalismes utilisés, les *logiques temporelles* et le *μ-calcul modal* s'avèrent bien adaptés pour décrire l'évolution du système dans le temps. Dans ce cas, la vérification consiste à s'assurer que l'automate modélisant le système à vérifier satisfait un ensemble de propriétés logiques.
Concernant la vérification de propriétés logiques, nos travaux dans ce domaine portent sur l'évaluation efficace du μ-calcul modal et sur son extension par des variables typées, afin de prendre en compte les données contenues dans les états et les transitions du graphe. Cette extension (dont nous avons mis en évidence l'utilité sur de nombreux exemples, notamment industriels) permet d'exprimer des propriétés qu'il n'est pas possible d'écrire en μ-calcul standard comme, par exemple, le fait qu'une variable donnée soit toujours croissante sur un chemin d'exécution. Nous travaillons aussi à la conception et à l'implémentation d'algorithmes d'évaluation efficaces pour cette extension du μ-calcul.

Bien que ces techniques soient performantes et automatisables, leur principale limitation réside dans le problème de l'*explosion d'états*, qui survient lorsque le nombre d'états du système à vérifier dépasse les capacités en mémoire de la machine. C'est pourquoi nous fournissons des technologies logicielles (voir § 4.1) permettant de manipuler ces graphes de deux manières :

– soit sous forme *explicite*, en gardant en mémoire l'ensemble des états et des transitions (vérification *énumérative*) ;
– soit sous forme *implicite*, en explorant dynamiquement les parties du graphe en fonction des besoins (vérification *à la volée*).

2.3 Technologie des langages – compilation

En ce qui concerne les langages, il nous semble essentiel de s'appuyer sur des langages possédant simultanément un *caractère exécutable* et une *sémantique formelle*, ceci pour plusieurs raisons :

– Les techniques de *model checking* nécessitent de pouvoir exécuter efficacement les programmes à vérifier.
– La modélisation de systèmes critiques ne saurait reposer sur des langages dont la sémantique ne serait pas rigoureusement définie, car cela conduit bien souvent à des ambiguïtés et des divergences d'interprétation (notamment entre concepteurs et

implémenteurs).

– En général, les techniques de *model checking* ne peuvent garantir la correction d'un système infini, puisqu'elles ne peuvent vérifier que des abstractions finies de ce système. C'est pourquoi on doit utiliser aussi des techniques de preuve, lesquelles ne s'appliquent qu'aux langages ayant une sémantique formelle.

Dans ce contexte, nos travaux actuels portent sur plusieurs langages :

– Nous nous intéressons depuis longtemps au langage LOTOS, le seul langage de description de protocoles ayant le statut de norme internationale [ISO88] et possédant les propriétés ci-dessus. Il s'agit d'un langage basé sur les concepts des algèbres de processus (notamment CCS [Mil89] et CSP [Hoa85]) pour la description du contrôle et les types abstraits algébriques [EM85] pour la description des données. LOTOS autorise à la fois la description du parallélisme asynchrone (aspects liés à la répartition, la synchronisation et la communication entre tâches) et celle des structures de données complexes manipulées dans les protocoles et les systèmes distribués.
  Nous utilisons LOTOS pour diverses études de cas industrielles et nous développons des outils logiciels pour ce langage dans le cadre de la boîte à outils CADP (voir § 4.1).
– Les algèbres de processus sont des formalismes particulièrement bien adaptés à la spécification des protocoles de télécommunication et des systèmes répartis. Cependant, en dépit d'une base mathématique rigoureuse, d'efforts de normalisation (notamment ceux concernant le langage LOTOS) et d'un nombre croissant d'études de cas traitées avec succès, les algèbres de processus ne sont pas encore pleinement acceptées en milieu industriel. Elles se voient parfois supplantées par des langages dont l'apparence plus conviviale (syntaxe graphique ou proche des langages algorithmiques classiques) masque une absence de sémantique formelle assez préoccupante lorsqu'il s'agit de modéliser et de valider des systèmes critiques.
  Les besoins en méthodes formelles et vérification allant en croissant, il est nécessaire de réfléchir à de nouveaux langages qui combineraient les fondements théoriques rigoureux et l'expressivité des algèbres de processus avec une simplicité d'utilisation permettant d'assurer une meilleure diffusion industrielle.
  Cette réflexion est également guidée par l'apparition de protocoles à contraintes temporelles fortes — protocoles utilisés dans les réseaux à haut débit — pour lesquels les aspects temporels doivent être pris en compte de manière quantitative, et non plus seulement qualitative.
  Nous travaillons sur ces questions, notamment dans le cadre de la révision de la norme LOTOS entreprise à l'ISO depuis 1992 ; cette révision a donné naissance en 2001 à la norme

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

[Mil89] R. MILNER, *Communication and Concurrency*, Prentice-Hall, 1989.

[Hoa85] C. A. R. HOARE, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[EM85] H. EHRIG, B. MAHR, *Fundamentals of Algebraic Specification 1 — Equations and Initial Semantics*, *EATCS Monographs on Theoretical Computer Science*, *6*, Springer Verlag, 1985.

internationale E-LOTOS (*Enhanced*-LOTOS) [ISO01] qui vise à conjuguer une expressivité sémantique accrue (par exemple, avec l'introduction du temps quantitatif) et une facilité d'apprentissage pour des non-experts. L'historique de nos contributions à l'ISO est disponible sur notre serveur Web, à l'adresse `http://www.inrialpes.fr/vasy/elotos`.
En parallèle, nous étudions une variante simplifiée du langage E-LOTOS, appelée LOTOS NT (LOTOS *Nouvelle Technologie*) [3], dans laquelle nous avons introduit certains concepts qui nous semblent pertinents (ce qui n'est pas toujours chose aisée dans une norme internationale).
Comme E-LOTOS, LOTOS NT se compose de trois parties : une *partie données*, qui permet une description naturelle des types de données et des fonctions tout en étant facilement analysable et implémentable, une *partie contrôle*, qui étend l'algèbre de processus de LOTOS par des constructions plus expressives et la prise en compte du temps quantitatif, et des *modules*, qui autorisent la structuration et la réutilisation des descriptions LOTOS NT.
La différence essentielle entre les deux langages réside dans le fait que LOTOS NT est un langage impératif alors que E-LOTOS s'inscrit dans un cadre fonctionnel. De plus, LOTOS NT se distingue d'E-LOTOS sur certains aspects (typage statique, surcharge d'opérateurs, tableaux) qui en font un langage plus facile à utiliser et plus simple à implémenter.
Nous travaillons sur l'implémentation de LOTOS NT pour lequel nous développons le compilateur TRAIAN (voir § 4.2).

2.4 Implémentation et expérimentation

Dans la mesure du possible, nous essayons de valider nos propositions par le développement d'outils et l'application de ces outils à des études de cas complexes, souvent industrielles. Cette confrontation systématique avec les problèmes d'implémentation et d'expérimentation est un aspect essentiel de notre approche.

3 Domaines d'applications

Les modèles théoriques que nous utilisons (automates, algèbres de processus, bisimulations, logiques temporelles) et les logiciels que nous développons sont suffisamment généraux pour ne pas dépendre trop étroitement d'un seul secteur applicatif.

Nos méthodes peuvent s'appliquer à tout système ou protocole composé d'agents distribués communiquant par messages. Ce cadre conceptuel trouve de nombreuses incarnations dans le domaine du logiciel, du matériel et des télécommunications. Les études de cas conduites ces dernières années avec la boîte à outils CADP (voir notamment § 5.3.3) illustrent bien cette diversité applicative :

[ISO01] ISO/IEC, « Enhancements to LOTOS (E-LOTOS) », *International Standard n° 15437:2001*, International Organization for Standardization — Information Technology, Genève, septembre 2001.

- *architectures matérielles :* circuits asynchrones, arbitrage de bus, cohérence de caches, conception conjointe matériel-logiciel ;
- *bases de données :* protocoles transactionnels, bases de connaissances distribuées, gestion de stocks ;
- *électronique grand public :* télécommandes audiovisuelles, vidéo à la demande, bus FIREWIRE, réseaux locaux domestiques ;
- *protocoles de sécurité :* authentification, commerce électronique, distribution de clés cryptographiques ;
- *systèmes embarqués :* applications sur cartes à puce, contrôle de trafic aérien ;
- *systèmes répartis :* mémoire virtuelle, systèmes de fichiers répartis, algorithmes d'élection, de reconfiguration dynamique et de tolérance aux pannes ;
- *télécommunications :* réseaux à haut débit, administration de réseaux, téléphonie mobile, interactions de services téléphoniques ;
- *interactions homme-machine :* interfaces graphiques, visualisation de données biomédicales, etc.

4 Logiciels

4.1 La boîte à outils CADP

Participants : Damien Bergamini, Nicolas Descoubes, Hubert Garavel [correspondant], Christophe Joubert, Frédéric Lang, Radu Mateescu, Bruno Ondet, Gordon Pace, Gilles Stragier.

Mots clés : application critique, application répartie, compilation, concurrence, génération de code, génie logiciel, logique temporelle, méthodes formelles, modélisation, mu-calcul, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

En collaboration avec le laboratoire VERIMAG, nous développons la boîte à outils CADP (CÆSAR/ALDÉBARAN *Development Package*) pour l'ingénierie des protocoles et des systèmes distribués [27] (voir `http://www.inrialpes.fr/vasy/cadp`). Au sein de cette boîte à outils, nous avons en charge les logiciels suivants :

- CÆSAR [6, 2] est un compilateur qui produit, à partir d'un programme LOTOS, du code exécutable ou des modèles sur lesquels différentes méthodes de vérification peuvent être appliquées. Le programme source LOTOS est traduit successivement en une algèbre de processus simplifiée, un réseau de Petri étendu avec des variables et des transitions atomiques, et, finalement, un système de transitions étiquetées obtenu par simulation exhaustive du réseau de Petri.
- CÆSAR.ADT [7] est un compilateur qui traduit les définitions de types abstraits LOTOS vers des bibliothèques de types et de fonctions en langage C. La traduction met en œuvre un algorithme de compilation par filtrage et des techniques pour la reconnaissance

des classes de types usuels (nombres entiers, énumérations, tuples, listes, etc.) qui sont identifiées automatiquement et implémentées de manière optimale.

- OPEN/CÆSAR [8] est un environnement logiciel extensible permettant de construire des outils de simulation, de vérification et de génération de tests sur des graphes représentés sous forme implicite. Ces outils peuvent être développés de manière simple, modulaire et indépendante du langage utilisé pour décrire les systèmes à valider. De ce point de vue, l'environnement OPEN/CÆSAR est l'un des constituants essentiels de la boîte à outils CADP, puisqu'il effectue la jonction entre les outils dédiés aux langages et les outils opérant sur les modèles. L'environnement OPEN/CÆSAR comprend un ensemble de bibliothèques avec leurs interfaces de programmation, ainsi que divers outils parmi lesquels :
  - EVALUATOR [10, 12], qui évalue à la volée des formules de μ-calcul régulier d'alternance 1,
  - EXECUTOR, qui permet l'exécution aléatoire,
  - EXHIBITOR, qui recherche à la volée des séquences d'exécution caractérisées par une expression régulière,
  - GENERATOR et REDUCTOR, qui construisent le graphe des états accessibles,
  - SIMULATOR, XSIMULATOR et OCIS, qui permettent la simulation interactive, et
  - TERMINATOR, qui recherche les états de blocage.
- BCG (*Binary Coded Graphs*) est un format qui utilise des techniques efficaces de compression permettant de stocker des graphes (représentés sous forme explicite) sur disque de manière très compacte. Ce format joue un rôle central dans la boîte à outils CADP. Il est indépendant du langage source et des outils de vérification. En outre, il contient suffisamment d'informations pour que les outils qui l'exploitent puissent fournir à l'utilisateur des diagnostics précis dans les termes du programme source. Pour exploiter le format BCG, nous développons un environnement logiciel qui se compose de bibliothèques avec leurs interfaces de programmation et de plusieurs outils, notamment :
  - BCG_DRAW, qui construit et affiche une représentation 2D en POSTSCRIPT d'un graphe,
  - BCG_EDIT, qui permet de modifier interactivement la représentation graphique produite par BCG_DRAW,
  - BCG_INFO, qui affiche diverses informations statistiques concernant un graphe,
  - BCG_IO, qui effectue des conversions entre BCG et divers autres formats de graphes,
  - BCG_LABELS, qui permet de masquer et/ou de renommer par des expressions régulières les étiquettes d'un graphe,
  - BCG_MIN, qui permet de minimiser un graphe selon la bisimulation forte ou la bisimulation de branchement (éventuellement étendue au cas des systèmes probabilistes ou stochastiques), et
  - BCG_OPEN, qui permet d'appliquer à tout graphe BCG les outils disponibles dans l'environnement OPEN/CÆSAR.
- XTL (*eXecutable Temporal Language*) [11, 9] est un langage adapté à l'expression des algorithmes d'évaluation et de diagnostic pour les formules de logiques temporelles telles

que HML [HM85], CTL [CES86], ACTL [NV90], etc. D'inspiration fonctionnelle, ce langage offre des primitives d'accès à toutes les informations contenues dans les graphes BCG : états, étiquettes des transitions, fonctions *successeurs* et *prédécesseurs*, ainsi qu'aux types et fonctions du programme source. Il permet la définition de fonctions récursives qui servent à calculer des prédicats de base et des modalités temporelles caractérisant des ensembles d'états et de transitions.
- SVL (*Script Verification Language*) est un langage permettant d'exprimer de manière simple des scénarios de vérification élaborés, lesquels seront exécutés en appelant, dans un ordre approprié, les différents outils de CADP avec les paramètres adéquats.

A ces outils s'ajoutent ceux développés par le laboratoire VERIMAG et l'ex-projet PAMPA (Rennes) :
- ALDÉBARAN effectue la comparaison et la minimisation de graphes selon diverses relations d'équivalence ou de préordre,
- EXP.OPEN et PROJECTOR calculent des produits et des abstractions d'automates communicants,
- TGV (*Test Generation based on Verification*) engendre automatiquement des tests de conformité en fonction d'objectifs de tests définis par l'utilisateur.

Tous ces outils — ainsi que d'autres développés par l'ex-projet MEIJE (Sophia-Antipolis) et les Universités de Liège et d'Ottawa — sont intégrés au sein de l'interface graphique EUCALYPTUS (développée en TCL/TK) qui offre un accès facile et uniforme aux différents outils, en masquant à l'utilisateur les conventions d'appel et les formats spécifiques à chaque outil.

Dans la compétition actuelle entre les différents langages, méthodologies et outils proposés pour la spécification et la vérification formelle, la boîte à outils CADP possède plusieurs avantages : elle s'appuie sur un langage normalisé, comporte des outils robustes (bien que perfectibles) et regroupe une communauté importante d'utilisateurs.

4.2 Le compilateur TRAIAN

Participants : David Champelovier, Hubert Garavel [correspondant], Frédéric Lang.

Mots clés : application critique, application répartie, compilation, concurrence, génération de code, génie logiciel, méthodes formelles, modélisation, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Pour le langage LOTOS NT, nous développons un compilateur, appelé TRAIAN, dont le but

[HM85] M. HENNESSY, R. MILNER, « Algebraic Laws for Nondeterminism and Concurrency », *Journal of the ACM 32*, 1985, p. 137–161.

[CES86] E. M. CLARKE, E. A. EMERSON, A. P. SISTLA, « Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications », *ACM Transactions on Programming Languages and Systems 8*, 2, avril 1986, p. 244–263.

[NV90] R. D. NICOLA, F. W. VAANDRAGER, *Action versus State Based Logics for Transition Systems*, *Lecture Notes in Computer Science*, *469*, Springer Verlag, 1990, p. 407–419.

est de traduire automatiquement une description LOTOS NT vers un programme C pouvant ensuite être utilisé à des fins de simulation, de prototypage rapide, de vérification et de test.

La version actuelle de TRAIAN effectue l'analyse lexicale et syntaxique, la construction des arbres de syntaxe abstraite, les vérifications de sémantique statique et la génération de code C pour les définitions de types et de fonctions contenues dans les descriptions LOTOS NT.

Bien que cette version de TRAIAN soit encore incomplète (elle ne traite pas les définitions de processus LOTOS NT), elle a d'ores et déjà trouvé un champ d'application original et utile : la construction de compilateurs [20]. L'approche suivie consiste à utiliser conjointement l'outil SYNTAX développé à l'INRIA Rocquencourt (pour les aspects lexicaux et syntaxiques) et le langage LOTOS NT (pour les aspects sémantiques, notamment la définition, la construction et le parcours des arbres abstraits). Il est également possible de programmer directement en langage C certaines parties du compilateur, ce qui offre une souplesse appréciable pour certains traitements spécifiques.

C'est ainsi que tous les compilateurs récents de l'équipe VASY — notamment EVALUATOR 4.0 (voir § 5.1.9), EXP.OPEN (voir § 5.1.2), NTIF (voir § 5.2.2) et SVL (voir § 5.1.10) — comportent une forte proportion de code LOTOS NT qui est ensuite traduit en code C par le compilateur TRAIAN. L'utilisation combinée des technologies SYNTAX, LOTOS NT et TRAIAN, donne entière satisfaction, tant pour la qualité des compilateurs produits que pour la facilité et la rapidité du développement lui-même.

Le compilateur TRAIAN est diffusé sur Internet : il existe une page Web qui lui est consacrée (voir `http://www.inrialpes.fr/vasy/traian`) à partir de laquelle on peut télécharger librement le compilateur.

5 Résultats nouveaux

5.1 Technologie des modèles – vérification

Mots clés : automate, bisimulation, compilation, concurrence, génération de code, génie logiciel, logique temporelle, méthodes formelles, modélisation, mu-calcul, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Résumé : *En 2002, nous avons étendu et amélioré divers outils de* CADP *et nous avons entrepris le développement de nouveaux outils visant à faciliter l'utilisation des techniques formelles en milieu industriel. Nos travaux ont porté sur les techniques de vérification compositionnelle (outils* BCG_MIN, EXP.OPEN, PROJECTOR *et* SVL*), la vérification de traces (outil* SEQ.OPEN*), l'évaluation de performances (outils* BCG_MIN, BCG_STEADY, BCG_TRANSIENT *et* DETERMINATOR*), la vérification massivement parallèle et distribuée (outils* DISTRIBUTOR *et* BCG_MERGE*) et la vérification à la volée (outils* CÆSAR_SOLVE, BISIMULATOR *et* EVALUATOR *4.0).*

5.1.1 Développement de l'outil BCG_MIN

Participants : Damien Bergamini, Hubert Garavel.

Développé en collaboration avec Holger Hermanns (Université de Twente, Pays-Bas), BCG_MIN est un outil de minimisation de graphes, qui traite trois sortes de graphes (tous encodés dans le format BCG) :

- des systèmes de transitions "ordinaires", tels que ceux produits à partir de descriptions LOTOS ;
- des systèmes de transitions "probabilistes" dont chaque transition peut être étiquetée, soit par une action a "ordinaire", soit par une probabilité $p \in [0,1]$, soit par un couple (a,p) ;
- des systèmes de transitions "stochastiques" dont chaque transition peut être étiquetée, soit par une action a "ordinaire", soit par un paramètre réel λ qui détermine une loi de distribution exponentielle donnée par $prob(x > t) = e^{-\lambda t}$, soit par un couple (a,λ).

De par l'existence de ces différents types de transitions, les modèles acceptés par BCG_MIN sont suffisamment généraux pour couvrir de nombreux modèles probabilistes[1] et stochastiques[2].

Pour minimiser les systèmes de transitions ordinaires, BCG_MIN implémente (une variante de) l'algorithme de Kanellakis et Smolka [KS90] pour la bisimulation forte et l'algorithme de Groote et Vaandrager [GV90] pour la bisimulation de branchement. Pour les systèmes de transitions probabilistes et stochastiques, BCG_MIN implémente l'algorithme de Hermanns et Siegle [HS99].

BCG_MIN permet de minimiser efficacement des graphes de grande taille que d'autres outils plus anciens (ALDÉBARAN et FC2MIN notamment) ne parviennent pas à traiter par manque de mémoire. Ainsi, BCG_MIN est cité dans [GvdP00] comme "*the best implementation of the*

[1] *Discrete Time Markov Chains, Discrete Time Markov Reward Models, Alternating Probabilistic LTS, Discrete Time Markov Decision Processes, Generative Probabilistic LTS, Reactive Probabilistic LTS, Stratified probabilistic LTS.*

[2] *Continuous Time Markov Chains, Continuous Time Markov Reward Models, Continuous Time Markov Decision Processes, Interactive Markov Chains, Timed Processes for Performance (TIPP) Models, Performance Evaluation Process Algebra (PEPA) Models, Extended Markovian Process Algebra (EMPA) Models.*

[KS90] P. C. KANELLAKIS, S. A. SMOLKA, « CCS expressions, finite state processes, and three problems of equivalence », *Information and Computation 86*, 1, mai 1990, p. 43–68.

[GV90] J. GROOTE, F. VAANDRAGER, « An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence », *in : Proceedings of the 17th ICALP (Warwick)*, M. S. Patterson (éditeur), *Lecture Notes in Computer Science*, *443*, Springer Verlag, p. 626–638, 1990.

[HS99] H. HERMANNS, M. SIEGLE, « Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation », *in : Proceedings of the 5th International AMAST Workshop ARTS'99 (Bamberg, Germany)*, J.-P. Katoen (éditeur), *Lecture Notes in Computer Science*, *1601*, Springer Verlag, p. 244–265, mai 1999.

[GvdP00] J. GROOTE, J. VAN DE POL, « State space reduction using partial τ-confluence », *in : Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science MFCS'2000 (Bratislava, Slovakia)*, M. Nielsen, B. Rovan (éditeurs), *Lecture Notes in Computer*

standard algorithm for the branching bisimulation".

Un article d'ensemble présentant l'utilisation de BCG_MIN pour la vérification fonctionnelle et l'évaluation de performances (avec application au protocole d'arbitrage du bus SCSI-2) a été publié [19].

En 2002, les travaux relatifs à BCG_MIN ont porté sur deux points :

- Les algorithmes pour la bisimulation forte ont été spécialisés pour augmenter leur rapidité. Sur divers exemples, un gain de 15% a été constaté.
- Poursuivant les efforts entrepris en 2001, nous avons continué à optimiser les structures de données utilisées dans BCG_MIN, afin de les rendre plus compactes et de pouvoir traiter de plus gros graphes.
  Ainsi, la taille mémoire allouée pour un état est elle passée de 20 à 16 octets dans le cas des systèmes de transitions ordinaires, et de 28 à 24 octets dans le cas des systèmes de transitions probabilistes ou stochastiques.
  De même, la taille mémoire allouée pour un bloc (c'est-à-dire, un état du graphe minimisé) est passée de 32 à 24 octets dans le cas des systèmes de transitions ordinaires, et de 40 à 28 octets dans le cas des systèmes de transitions probabilistes ou stochastiques.
  Enfin, nous avons supprimé certaines allocations dynamiques de mémoire, ce qui a permis de réduire l'espace mémoire maximum requis par BCG_MIN.

5.1.2 Développement de l'outil EXP.OPEN 2.0

Participants : Hubert Garavel, Frédéric Lang, Gordon Pace.

Outre BCG_MIN (voir § 5.1.1), un second axe de nos travaux sur la vérification compositionnelle a porté sur la refonte de l'outil EXP.OPEN de CADP qui construit à la volée le système de transitions étiquetées correspondant à un système d'automates communicants interconnectés par les opérateurs de composition parallèle et de masquage du langage LOTOS.

En 2002, nous avons entrepris le développement d'une nouvelle version 2.0 de l'outil EXP.OPEN qui corrige diverses limitations de l'ancienne version 1.0 et en étend considérablement les fonctionnalités. Développée à l'aide du générateur de compilateurs SYNTAX et du compilateur TRAIAN (voir § 4.2), cette nouvelle version (1 700 lignes de code SYNTAX, 5 800 lignes de code LOTOS NT et 1 500 lignes de code C) apporte les améliorations suivantes :

- Alors que la version précédente d'EXP.OPEN était centrée sur la vérification compositionnelle de programmes LOTOS, la nouvelle version d'EXP.OPEN rend possible la vérification compositionnelle pour d'autres formalismes, dont les algèbres de processus CCS, CSP et μCRL, les langages E-LOTOS et LOTOS, et les automates communicants tels qu'ils existent dans les outils MEC et FC2TOOLS. Dans ce but, le langage d'entrée pour la description de systèmes d'automates communicants a été étendu, tout en préservant

Science, *1893*, Springer Verlag, p. 383–393, Berlin, août 2000. Paru également comme rapport technique CWI SEN-R0008, Amsterdam, mars 2000.

strictement la compatibilité ascendante avec la version antérieure d'Exp.Open. Outre les opérateurs Lotos de composition parallèle et de masquage, ce langage comprend désormais :
  - les opérateurs de composition parallèle des algèbres de processus Ccs, Csp et μCrl ;
  - les opérateurs de composition parallèle "graphique" des langages E-Lotos et Lotos NT [4] ;
  - la composition parallèle basée sur les "vecteurs de synchronisation" des outils Mec et Fc2Tools ;
  - des opérateurs étendus de masquage et de renommage d'actions, avec utilisation d'expressions régulières ;
  - des options pour adapter le nom des actions masquées et des actions de terminaison selon les différentes conventions de nommage utilisées en Ccs, Csp, μCrl, etc.
- Exp.Open 2.0 donne à l'utilisateur une meilleure compréhension du système de transitions étiquetées construit en indiquant, pour chaque action de celui-ci, les opérations successives (synchronisations, masquages, renommages) qui ont engendré cette action.
- Exp.Open 2.0 permet de traduire le système d'automates communicants donné en entrée vers un réseau de Petri équivalent. Nous avons ainsi réalisé une interconnexion d'Exp.Open avec l'outil Pep de vérification de réseaux de Petri développé à l'Université d'Oldenburg (Allemagne).

Les premières expérimentations d'Exp.Open 2.0 font apparaître, sur les exemples testés, jusqu'à 50% de gain en temps et en mémoire par rapport à l'ancienne version.

Par ailleurs, nous avons étudié l'introduction dans Exp.Open d'ordres partiels ("τ-confluence") afin de limiter l'explosion d'états tout en préservant l'équivalence modulo la bisimulation de branchement. Cette approche consiste à donner priorité à certaines transitions masquées sans considérer les entrelacements potentiels de ces transitions avec d'autres transitions.

Nos premières expériences montrent que, si les automates du système ne sont pas préalablement minimisés pour la bisimulation de branchement, l'utilisation d'ordres partiels permet un gain important (jusqu'à 54% d'états et 72% de transitions en moins) par rapport à la génération brute du système. Même lorsque tous les automates du système ont été minimisés pour la bisimulation de branchement, des gains (plus faibles) ont été également observés, ce qui démontre la complémentarité entre vérification compositionnelle et ordres partiels.

5.1.3 Développement de l'outil PROJECTOR 2.0

Participants : Nicolas Descoubes, Hubert Garavel, Frédéric Lang, Radu Mateescu, Bruno Ondet, Gordon Pace, Frédéric Tronel.

Outre Bcg_Min (voir § 5.1.1) et Exp.Open (voir § 5.1.2), un troisième axe de nos travaux sur la vérification compositionnelle a porté sur l'amélioration des performances de l'outil Projector de Cadp qui implémente l'opération d'abstraction sur des comporte-

ments [GLS96,KM97].

Pour remédier à des problèmes de lenteur et de consommation mémoire excessive pouvant empêcher l'utilisation effective de la vérification compositionnelle avec abstractions, nous avions entrepris, en 2001, le développement d'une nouvelle version 2.0 de l'outil PROJECTOR en partant uniquement des définitions théoriques de l'opérateur d'abstraction [KM97], sans nous baser sur l'ancienne version 1.2 de PROJECTOR.

En 2002, le développement de PROJECTOR 2.0 (1 500 lignes de code C) a été quasiment achevé.

Nous avons effectué une campagne de test rigoureuse permettant de comparer les résultats produits par l'ancienne et la nouvelle version de PROJECTOR, ce qui a permis de déceler et de corriger une dizaine d'erreurs. A cette occasion, une base de tests de non-régression automatisés a été développée.

5.1.4 Développement de l'outil SEQ.OPEN

Participants : Hubert Garavel, Radu Mateescu, Bruno Ondet.

Nous nous intéressons au problème de la vérification automatisée des traces d'exécution produites par un système fonctionnant en "boîte noire". En pratique, cette approche est la seule possible lorsque les seules informations que l'on peut observer sont les interactions entre ce système et son environnement, l'état interne du système n'étant pas connu, soit parce que le code source n'est pas disponible, soit parce que le système est trop complexe pour être analysé exhaustivement.

En 2001, un outil prototype appelé SEQ.OPEN avait été développé pour la vérification de traces. Cet outil permet d'appliquer à un ensemble de traces d'exécution encodées dans le format SEQ (format texte utilisé dans CADP pour représenter les traces) les différents outils disponibles dans l'environnement OPEN/CÆSAR (et notamment l'outil de vérification EVALUATOR 3.0). Un avantage important de SEQ.OPEN réside dans le fait qu'il fonctionne sans charger en mémoire la trace d'exécution et peut, par conséquent, s'appliquer à des traces de très grande taille.

En 2002, dans le cadre des contrats FORMALFAME (voir § 6.2) et ARCHWARE (voir § 6.1), nous avons entièrement réécrit l'outil SEQ.OPEN. La nouvelle version (1 200 lignes de code C) prend en compte la totalité du format SEQ. Les performances ont été soigneusement améliorées (voir § 5.3.1 pour des données expérimentales relatives au protocole de cohérence de caches "FAME"), d'une part, en pré-compilant le code de SEQ.OPEN afin de réduire le nombre d'appels

[GLS96] S. GRAF, G. LÜTTGEN, B. STEFFEN, « Compositional Minimisation of Finite State Systems using Interface Specifications », *Formal Aspects of Computation 8*, septembre 1996.

[KM97] J.-P. KRIMM, L. MOUNIER, « Compositional State Space Generation from LOTOS Programs », *in : Proceedings of TACAS'97 Tools and Algorithms for the Construction and Analysis of Systems (University of Twente, Enschede, The Netherlands)*, E. Brinksma (éditeur), *Lecture Notes in Computer Science*, *1217*, Springer Verlag, Berlin, avril 1997. Version étendue avec preuves parue comme rapport de recherche VERIMAG RR97-01.

au compilateur C et, d'autre part, en introduisant un cache logiciel pour optimiser les accès aux transitions et étiquettes fréquemment visitées.

Un article sur SEQ.OPEN a été écrit et soumis à une conférence internationale.

5.1.5 Développement des outils DISTRIBUTOR 2.1 et BCG_MERGE 3.0

Participants : Nicolas Descoubes, Hubert Garavel, Christophe Joubert, Radu Mateescu, Gilles Stragier.

Nous étudions depuis 1999 l'utilisation de machines parallèles pour améliorer les performances des algorithmes de vérification énumérative.

En effet, ces algorithmes — qui nécessitent l'exploration et le stockage de graphes de dimensions importantes (plusieurs millions d'états) — sont souvent limités par la puissance de calcul et l'espace mémoire des machines séquentielles actuelles. C'est pourquoi nous souhaitons repousser ces limites en exploitant au mieux les possibilités des machines parallèles de type MIMD à mémoire distribuée (grappes de PC, réseaux de stations de travail) disponibles dans les laboratoires de recherche.

Nos efforts se sont concentrés sur la parallélisation de l'algorithme de construction du graphe [1], qui constitue un goulot d'étranglement pour la vérification puisqu'il requiert un espace mémoire considérable pour stocker tous les états accessibles. La parallélisation de cet algorithme devrait permettre, en remplaçant la mémoire d'une seule machine par celle de dizaines ou de centaines de machines, de gagner plusieurs ordres de grandeur dans la complexité des graphes traités.

Nos travaux avaient abouti en 2001 à deux outils prototypes :

- DISTRIBUTOR 2.0 (6 200 lignes de code C), qui répartit la construction d'un graphe sur N machines communiquant deux à deux au moyen de *sockets*. Chaque machine est chargée de construire une partie du graphe sous forme d'un fichier au format BCG, les états étant répartis entre les mémoires locales des machines au moyen d'une fonction de hachage déterminée statiquement.
- BCG_MERGE 2.0 (450 lignes de code C), qui assemble les N parties de graphe construites sur chaque machine par DISTRIBUTOR afin d'obtenir — après une renumérotation appropriée des états — un fichier BCG unique représentant le graphe complet.

Ces outils ont été expérimentés sur diverses architectures : stations de travail sous SOLARIS reliées par un réseau ETHERNET 100 Mbits et grappe de 225 PC du projet APACHE. Les résultats obtenus sur diverses spécifications LOTOS (protocole du réseau HAVi, protocole d'arbitrage du bus SCSI-2, protocole d'élection sur des anneaux à jeton, etc.), traitées sur des configurations allant jusqu'à 70 machines, ont montré des accélérations importantes et un bon équilibrage de charge entre les machines.

En 2002, nos travaux dans ce domaine ont été poursuivis. Nous avons effectué une étude bibliographique détaillée [29] de l'état de l'art et donné un cours intitulé "*Parallel and Distributed Model Checking*" à l'école d'été SFM'02-MC (voir § 8.4).

Après avoir constaté que les outils DISTRIBUTOR et BCG_MERGE comportaient des fragments de code similaires susceptibles d'être factorisés et réutilisés pour de futurs outils, nous avons développé une bibliothèque de code, appelée CÆSAR_NETWORK (2 800 lignes de code C), qui regroupe toutes les fonctionnalités communes : gestion du fichier de configuration des machines contenant les paramètres du calcul distribué, protocole de déploiement de processus sur un ensemble de machines distantes, envoi et réception de messages utilisant des *sockets* bloquantes ou non-bloquantes, gestion des tampons de communication, etc.

Nous avons ensuite entrepris d'adapter DISTRIBUTOR pour tirer partie de cette bibliothèque. Ce travail a conduit à une nouvelle version DISTRIBUTOR 2.1 (3 300 lignes de code C), actuellement en cours d'expérimentation. Nous avons également réfléchi au moyen de traiter les types de données dynamiques (listes, arbres, ensembles, etc.) présents dans les systèmes à vérifier, ces types n'étant pas acceptés, pour l'instant, par DISTRIBUTOR.

Par ailleurs, nous avons entièrement réécrit l'outil BCG_MERGE afin d'utiliser la bibliothèque CÆSAR_NETWORK. La nouvelle version BCG_MERGE 3.0 (700 lignes de code C) apporte en outre deux améliorations :

- Elle permet de traiter des graphes plus grands sur un nombre N de machines plus élevé. Alors que la version précédente de BCG_MERGE consistait en un seul processus séquentiel qui commençait par rapatrier sur la machine frontale les N parties de graphe produites par DISTRIBUTOR avant de les fusionner en un seul graphe BCG, la nouvelle version comporte un processus frontal et N processus, répartis sur les machines distantes, qui envoient au processus frontal les transitions contenues dans les N parties de graphe.
- La nouvelle version met en œuvre un algorithme de tri distribué qui ordonne les transitions du graphe résultat par numéros d'états source croissants, ce qui améliore la compacité du fichier BCG produit et rend sa génération plus rapide.

Enfin, nous avons étudié le moyen d'appliquer les outils de CADP aux très grands graphes produits par DISTRIBUTOR et BCG_MERGE. A cette fin, nous avons conçu deux extensions de l'outil BCG_OPEN. Ces extensions [31] permettent de manipuler des graphes trop grands pour être contenus dans la mémoire vive d'un ordinateur standard ; l'une utilise une représentation sur disque dur, l'autre exploite les mémoires vives d'un ensemble d'ordinateurs en réseau.

5.1.6 Développement des outils BCG_STEADY, BCG_TRANSIENT et DETERMINATOR

Participants : Hubert Garavel, Christophe Joubert, Bruno Ondet, Radu Mateescu.

Depuis plusieurs années, en collaboration avec Holger Hermanns (Université de Twente, Pays-Bas), nous développons des outils pour la génération et la minimisation compositionnelles de systèmes probabilistes et stochastiques. Ces travaux (voir § 5.1.1) ont fait l'objet d'une publication [19].

En 2002, les travaux dans cette direction ont été poursuivis avec le développement de trois nouveaux outils (DETERMINATOR, BCG_STEADY et BCG_TRANSIENT) qui ont fait l'objet d'une publication [22].

L'outil DETERMINATOR prend en entrée un graphe G_1 (encodé dans le format BCG) comportant :

- des transitions "ordinaires", c'est-à-dire étiquetées par une action a (produite, par exemple, à partir d'une description LOTOS) ;
- des transitions "probabilistes", étiquetées, soit par une probabilité $p \in [0,1]$, soit par un couple (a,p) ;
- des transitions "stochastiques", étiquetées, soit par un paramètre réel λ qui détermine une loi de distribution exponentielle donnée par $prob(x > t) = e^{-\lambda t}$, soit par un couple (a,λ) ;

et tente de traduire ce graphe en une *chaîne de Markov à temps continu* (CMTC), c'est-à-dire un graphe G_2 (encodé au format BCG) ne comportant plus que des transitions stochastiques. Il s'agit donc d'éliminer les transitions ordinaires et probabilistes tout en conservant les informations stochastiques présentes dans G_1. Une difficulté réside dans le fait que G_1 peut comporter du non-déterminisme, ce qui rend la traduction impossible dans le cas général. C'est pourquoi DETERMINATOR ne traite que le cas des graphes G_1 vérifiant une condition suffisante (*well-specified check*) [DS99] qui garantit que, quelle que soit la manière dont on résout les choix non-déterministes, la CMTC résultante est unique. Ceci permet d'éliminer le non-déterminisme sans altérer le comportement stochastique du système. L'algorithme de traduction implémenté dans DISTRIBUTOR est une variante de celui présenté dans [DS99] ; il fonctionne à la volée en s'appuyant sur les fonctionnalités de l'environnement OPEN/CÆSAR.

Bien que l'outil de minimisation BCG_MIN permette aussi, dans une certaine mesure, d'éliminer le non-déterminisme (en effectuant des réductions plus générales basées sur le concept de *lumpability*), il diffère de DETERMINATOR : BCG_MIN ne traite pas le cas des graphes comportant à la fois des transitions probabilistes et stochastiques ; il n'élimine pas les transitions ordinaires et son algorithme est plus coûteux que celui de DETERMINATOR.

Les outils BCG_STEADY et BCG_TRANSIENT prennent en entrée une CMTC — pouvant avoir été produite par DETERMINATOR — à partir de laquelle ils construisent une représentation interne (matrice creuse) rassemblant les informations stochastiques contenues dans cette CMTC. Sur cette matrice creuse, ils appliquent ensuite des algorithmes numériques pour l'évaluation de performances :

- BCG_STEADY calcule, pour chaque état de la CMTC, la probabilité de se trouver, à long terme (c'est-à-dire, à l'équilibre), dans cet état. Ce calcul s'effectue de manière itérative en utilisant un algorithme de type Gauss-Seidel [Ste94].
- BCG_TRANSIENT calcule, pour chaque état de la CMTC et pour chaque instant d'un ensemble discret fourni par l'utilisateur, la probabilité de se trouver à cet instant dans cet état. Ce calcul s'effectue selon la méthode d'uniformisation [Ste94] en utilisant l'algorithme de Fox-Glynn [FG87].

[DS99] D. D. DEAVOURS, W. H. SANDERS, « An Efficient Well-Specified Check », *in : Proceedings of the 8th International Workshop on Petri Nets and Performance Models PNPM'99 (Zaragoza, Spain)*, IEEE Press, p. 124–133, 1999.

[Ste94] W. J. STEWART, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.

[FG87] B. FOX, P. GLYNN, « Computing Poisson Probabilities », *Communications of the ACM 31*, 4,

A partir des probabilités ainsi calculées, BCG_STEADY et BCG_TRANSIENT peuvent aussi fournir des mesures de débit (*throughput*) pour chacune des étiquettes de transitions présentes dans la CMTC.

Pour faciliter l'interprétation des données ainsi calculées, BCG_STEADY et BCG_TRANSIENT délivrent leurs résultats dans le format universel CSV (*Comma Separated Values*) qui est directement utilisable par des tableurs (notamment EXCEL) et des outils de visualisation (notamment GNUPLOT).

5.1.7 Développement de la bibliothèque CÆSAR_SOLVE

Participant : Radu Mateescu.

La vérification *à la volée* (voir § 2.2) est un moyen de combattre l'explosion d'états en construisant incrémentalement des graphes au fur et à mesure des besoins de la vérification. Elle permet de détecter des erreurs dans des graphes dont la taille est trop importante pour qu'ils puissent être entièrement explorés. Sous-jacent à différents problèmes de vérification à la volée, notamment l'évaluation de formules de logique temporelle (*model checking*) et le calcul de relations d'équivalence ou de préordre (*equivalence checking*), il existe un formalisme intermédiaire utile, les systèmes d'équations booléennes, sur lequel portent nos efforts.

En 2002, nous avons poursuivi le développement, entrepris en 2001, d'une bibliothèque logicielle, appelée CÆSAR_SOLVE (5 000 lignes de code C), permettant la résolution à la volée des systèmes d'équations booléennes d'alternance 1 (c'est-à-dire, sans récursion mutuelle entre les équations de plus petit et de plus grand point fixe). Cette bibliothèque est également capable de générer des diagnostics (exemples et contre-exemples).

La méthode de résolution employée repose sur les graphes booléens, qui offrent une représentation simple des dépendances entre variables booléennes. La bibliothèque CÆSAR_SOLVE comporte quatre algorithmes de résolution à la volée :

- L'algorithme A1 utilise un parcours en profondeur du graphe booléen. C'est un algorithme général, au sens où il ne fait aucune hypothèse sur la forme des équations booléennes à résoudre. Une première version de cet algorithme avait été implémentée dans l'outil EVALUATOR 3.0 pour vérifier à la volée des formules du μ-calcul régulier d'alternance 1. Etant inclus dans la bibliothèque CÆSAR_SOLVE, cet algorithme pourra désormais être utilisé pour de nouvelles applications.
- L'algorithme A2 utilise un parcours en largeur du graphe booléen. Comme A1, il s'agit d'un algorithme général. En outre, il permet de produire des diagnostics de profondeur réduite, voire minimale lorsque ces diagnostics sont des séquences de transitions (notamment dans le cas des contre-exemples produits pour les propriétés de sûreté exprimées en logique temporelle).
- L'algorithme A3 utilise un parcours en profondeur du graphe booléen. Il est optimisé pour réduire la consommation mémoire dans le cas des graphes booléens sans circuit. De

1987, p. 440–445.

tels graphes sont obtenus lorsque l'on vérifie des propriétés en logique temporelle sur des graphes sans circuit ou lorsque l'on vérifie l'inclusion de séquences de transitions dans des graphes quelconques.
- L'algorithme A4 utilise un parcours en profondeur du graphe booléen. Il est optimisé pour réduire la consommation mémoire dans le cas des graphes booléens disjonctifs ou conjonctifs (c'est-à-dire, dont les circuits comprennent uniquement des variables disjonctives ou uniquement des variables conjonctives). De tels graphes booléens sont obtenus lorsque l'on vérifie des propriétés temporelles exprimées en logique ACTL ou PDL et lorsque l'on vérifie l'équivalence de deux graphes dont l'un est déterministe.

La bibliothèque CÆSAR_SOLVE a fait l'objet d'une publication [25]. Elle constitue le moteur de vérification pour les outils BISIMULATOR (voir § 5.1.8) et EVALUATOR 4.0 (voir § 5.1.9).

5.1.8 Développement de l'outil BISIMULATOR

Participant : Radu Mateescu.

L'utilisation de relations d'équivalence pour la vérification comporte deux aspects complémentaires : la *minimisation* d'un graphe — telle qu'effectuée notamment par l'outil BCG_MIN (voir § 5.1.1) — et la *comparaison* de deux graphes selon une certaine relation d'équivalence.

C'est ce second aspect qu'aborde l'outil BISIMULATOR. Cet outil prend en entrée deux graphes à comparer (l'un représenté implicitement au moyen de l'environnement OPEN/CÆSAR [8], l'autre représenté explicitement sous forme de fichier BCG) et détermine s'ils sont tous deux équivalents ou si l'un d'eux est inclus dans l'autre (au sens de diverses relations d'équivalence ou de préordre).

BISIMULATOR fonctionne *à la volée*, c'est-à-dire qu'il explore dynamiquement, au fur et à mesure des besoins, les parties utiles du graphe représenté implicitement, sans nécessiter de construire préalablement la totalité de ce graphe, ce qui pourrait s'avérer prohibitif.

De plus, l'utilisation des environnements génériques BCG et OPEN/CÆSAR rend BISIMULATOR totalement indépendant du langage source utilisé pour décrire les deux graphes à comparer. Ceci constitue un progrès significatif par rapport à des outils plus anciens — tels qu'ALDÉBARAN et FC2IMPLICIT — qui implémentent également certains algorithmes de comparaison à la volée, mais de manière limitée, puisqu'ils imposent au graphe implicite d'être fourni sous la forme d'un produit d'automates communicants, alors que BISIMULATOR peut s'appliquer directement à un programme écrit dans un langage de haut niveau (par exemple, LOTOS).

L'approche utilisée dans BISIMULATOR consiste à formuler le problème de la comparaison de deux graphes en termes d'un système d'équations booléennes. Comparer les graphes à la volée revient alors à effectuer la résolution locale du système booléen, c'est-à-dire à calculer la valeur de vérité de la variable représentant l'équivalence des états initiaux des deux graphes.

Une fonctionnalité importante de BISIMULATOR consiste en la production de diagnostics

expliquant pourquoi deux graphes sont équivalents ou inclus l'un dans l'autre (*exemples*) ou, au contraire, pourquoi ils ne le sont pas (*contre-exemples*). En pratique, ces diagnostics ont plusieurs utilités :

- Les exemples produits par BISIMULATOR pour l'inclusion au sens du préordre de la bisimulation forte pourraient améliorer les fonctionnalités du simulateur graphique OCIS en lui permettant de relire interactivement les parties de graphes (traces d'exécution, scénarios, diagnostics, etc.) produites par d'autres outils de CADP tels que EXECUTOR, EXHIBITOR, EVALUATOR 3.0, etc.
- Les contre-exemples produits par BISIMULATOR permettent à l'utilisateur de comprendre pourquoi deux graphes ne sont pas équivalents ou inclus : ces contre-exemples sont des graphes acycliques rassemblant toutes les séquences de transitions qui, exécutées simultanément dans les deux graphes, mènent à des états non équivalents. De ce fait, les contre-exemples de BISIMULATOR sont (beaucoup) plus compacts que ceux d'ALDÉBARAN, qui se limitent à un ensemble de séquences d'exécution construites séparément.

En 2002, nous avons poursuivi le développement, entrepris en 2001, de l'outil BISIMULATOR (7 500 lignes de code C). Nos travaux ont porté dans deux directions :

- La version précédente de BISIMULATOR prenait en compte uniquement la relation d'équivalence forte, qui ne permet pas d'abstraire les transitions internes présentes dans les graphes. Nous avons étendu BISIMULATOR pour traiter trois relations d'équivalence faibles : l'équivalence observationnelle, l'équivalence $\tau^*.a$ et l'équivalence de sûreté. Pour chacune de ces relations, nous avons développé une traduction du problème de la vérification en termes de systèmes d'équations booléennes, permettant ainsi la connexion à la bibliothèque générique CÆSAR_SOLVE (voir § 5.1.7).
- Nous avons également développé la traduction des diagnostics fournis par la bibliothèque CÆSAR_SOLVE en termes de sous-graphes (portions communes aux deux graphes que l'on compare) représentés au format BCG.

Les comparaisons expérimentales avec l'outil ALDÉBARAN ont montré les résultats suivants :

- Pour la bisimulation forte, la vitesse d'exécution de BISIMULATOR est environ 25 fois supérieure à celle des algorithmes de comparaison à la volée implémentés dans ALDÉBARAN (mesures effectuées sur des graphes allant jusqu'à 10 000 états, au-delà les temps d'exécution des algorithmes à la volée devenant prohibitifs en présence de non-déterminisme).
- Pour l'équivalence $\tau^*.a$ et l'équivalence de sûreté, la version actuelle de BISIMULATOR présente des performances comparables à celles d'ALDÉBARAN en temps d'exécution, mais avec une consommation mémoire jusqu'à quatre fois inférieure (il convient de noter qu'ALDÉBARAN n'implémente pas la comparaison à la volée pour l'équivalence observationnelle).

5.1.9 Développement de l'outil EVALUATOR 4.0

Participant : Radu Mateescu.

La boîte à outils CADP comporte actuellement deux évaluateurs de logique temporelle : XTL [9], qui permet de vérifier de façon énumérative des formules temporelles comportant des données et EVALUATOR 3.0 [10, 12], qui permet de vérifier à la volée des formules du μ-calcul régulier d'alternance 1 sans données.

Bien que ces deux logiciels aient été utilisés avec succès pour valider des dizaines d'applications critiques, il serait souhaitable, dans un souci d'ergonomie, de proposer aux utilisateurs de CADP un seul outil d'évaluation de logique temporelle qui réunirait l'ensemble des fonctionnalités actuellement fournies par XTL et EVALUATOR 3.0 de manière séparée. C'est pourquoi, nous avons entrepris, en 2000, un travail d'unification d'XTL et d'EVALUATOR 3.0 qui devrait, à terme, aboutir à un outil de vérification unique, appelé EVALUATOR 4.0. A notre connaissance, il n'existe à l'heure actuelle aucun outil de vérification réunissant toutes ces fonctionnalités.

Cet outil permettra de vérifier à la volée des propriétés temporelles comportant des données. Le langage d'entrée d'EVALUATOR 4.0 sera basé sur des formules d'un μ-calcul régulier d'alternance 1 étendu avec des variables typées ; comme XTL, il offrira des primitives de manipulation des états et des transitions dans les formules logiques, permettant de définir des propriétés temporelles non-standard (comme par exemple le fait qu'un état possède une transition vers lui-même, ou encore le fait qu'une séquence de transitions finie comporte le même nombre d'occurrences de deux actions A et B).

En 2002, nous avons poursuivi le développement de l'outil EVALUATOR 4.0. Nos travaux ont concerné les aspects suivants :

- Nous avons étendu les expressions régulières présentes dans le langage d'entrée d'EVALUATOR 4.0, qui jusqu'alors ne comportaient que des opérateurs réguliers classiques, avec des constructions inspirées des langages de programmation séquentielle (instructions conditionnelles "**if-then-else**", boucles "**while**", "**repeat**", etc.). Des propriétés très complexes portant sur les données peuvent être maintenant exprimées uniquement au moyen de modalités contenant des expressions régulières étendues. Ceci permet de réduire drastiquement l'emploi des opérateurs de point fixe, qui est difficile et source d'erreurs même pour les utilisateurs expérimentés.
  Nous avons effectué la compilation de ces nouvelles constructions (analyse lexicale et syntaxique, analyse sémantique, traduction vers des opérateurs de point fixe paramétrés par des données), ainsi que leur traduction vers des systèmes d'équations booléennes, ce qui a permis de réaliser la connexion avec la bibliothèque générique CÆSAR_SOLVE (voir § 5.1.7), qui sert de moteur à la vérification.
- Nous avons également développé la traduction des diagnostics (exemples et contre-exemples) fournis par la bibliothèque CÆSAR_SOLVE en termes de sous-graphes (portions du graphe sur lequel les formules temporelles sont évaluées) représentés en format BCG.

L'outil EVALUATOR 4.0 (30 000 lignes de code SYNTAX, LOTOS NT et C) est actuellement

opérationnel et entièrement compatible avec la version précédente EVALUATOR 3.0 (le langage d'entrée d'EVALUATOR 4.0 étant une extension conservative de celui d'EVALUATOR 3.0).

Pour les propriétés sans données, EVALUATOR 4.0 exhibe des performances nettement supérieures par rapport à EVALUATOR 3.0 (jusqu'à dix fois plus rapide et jusqu'à huit fois plus économique en mémoire).

5.1.10 Autres développements d'outils

Participants : Damien Bergamini, Hubert Garavel, Frédéric Lang, Bruno Ondet, Frédéric Tronel.

Nous avons également poursuivi l'amélioration des outils et des composants logiciels qui constituent le cœur de la boîte à outils CADP :

- La bibliothèque CÆSAR_HASH a été enrichie par six fonctions de hachage supplémentaires.
- L'outil BCG_INFO a été enrichi de nouvelles fonctionnalités (affichage des états inaccessibles, des états non-déterministes, des plus courts chemins menant à un état donné).
- L'algorithme de reconnaissance des étiquettes de transitions probabilistes et stochastiques utilisé par BCG_MIN a été généralisé au cas des transitions temporisées et inséré dans une bibliothèque commune aux outils BCG_MIN, BCG_STEADY, BCG_TRANSIENT et DETERMINATOR.
- L'outil BCG_IO a été étendu pour effectuer la traduction des graphes BCG vers le format d'entrée de l'outil ETMCC (*Erlangen-Twente Markov Chain Checker*), ce qui permet désormais d'évaluer des formules de la logique temporelle stochastique CSL sur des chaînes de Markov produites par CADP.
- Nous avons entrepris le développement d'un nouvel outil, appelé BCG_GRAPH (460 lignes de code C), qui permet de produire le graphe BCG (minimal pour la bisimulation forte) correspondant à un canal de communication asynchrone. Ce graphe est paramétré par le nombre de places maximal de la file FIFO associée au canal et par le nombre de messages différents que ce canal peut recevoir. La génération est efficace : pour un canal comportant 10 places et 10 transitions, BCG_GRAPH produit un graphe d'environ 185 000 états et 1 850 000 transitions en une minute sur une machine standard.
- Pour permettre la factorisation des structures de données allouées dynamiquement (par exemple, dans les programmes LOTOS), une bibliothèque spécialisée CÆSAR_TABLE_2 avait été développée (voir rapport d'activité de VASY en 2000). En 2002, nous avons repris cette activité dans le cadre de la génération distribuée d'espace d'états (voir § 5.1.5). Après analyse des performances et des limitations de la bibliothèque CÆSAR_TABLE_2, nous avons choisi de fusionner les fonctionnalités de cette bibliothèque avec une autre bibliothèque CÆSAR_TABLE_1 destinée à stocker de très grands espaces d'états.
  Nous avons alors entrepris d'améliorer la bibliothèque CÆSAR_TABLE_1 sur divers points. D'une part, nous avons augmenté sa capacité de stockage (2^{32} éléments maximum au lieu de 2^{24} précédemment). D'autre part, nous l'avons optimisée afin de permettre le stockage efficace d'un petit nombre (quelques centaines) d'éléments.

– Enfin, nous avons travaillé à l'amélioration du langage SVL (*Script Verification Language*), un langage de haut niveau pour l'écriture de scénarios de vérification, ainsi que du compilateur associé à ce langage.
La version SVL 2.0 produite en 2001 a fait l'objet d'une publication [23]. Elle a été utilisée notamment par l'Université de Twente (Pays-Bas) pour la vérification compositionnelle de systèmes stochastiques [19], ainsi que par la société ERICSSON (Suède), avec des retours d'utilisation très positifs.
En 2002, nous avons produit une nouvelle version 2.1 du langage et du compilateur SVL qui, outre diverses corrections d'anomalies, apporte plusieurs améliorations. En particulier, les stratégies prédéfinies de vérification compositionnelle ont été optimisées, afin d'éliminer les appels inutiles aux outils de minimisation après renommage de portes sur un automate déjà minimisé. Par ailleurs, des variables *shell* peuvent désormais être utilisées dans la définition des noms de portes et de processus, étendant les possibilités de paramétrage offertes par le langage.

5.2 Technologie des langages – compilation

Mots clés : algèbre de processus, automate, compilation, concurrence, génération de code, génie logiciel, méthodes formelles, modélisation, parallélisme asynchrone, spécification formelle, synchronisation, système distribué, temps réel.

Résumé : *Les travaux consacrés depuis 1992 à la définition du langage* E-LOTOS *— travaux auxquels* VASY *a activement participé — ont trouvé leur consécration en 2001 avec l'adoption officielle de* E-LOTOS *comme norme internationale* ISO/IEC [ISO01]. *Forte de ce succès, l'équipe* VASY *a poursuivi ses efforts pour la compilation des méthodes formelles de nouvelle génération telles que* E-LOTOS *et* LOTOS NT *(voir § 2.3), ce qui nécessite de traiter à la fois la partie données et la partie contrôle des langages* E-LOTOS *et* LOTOS NT.

5.2.1 Compilation de la partie données du langage E-LOTOS

Participants : David Champelovier, Hubert Garavel, Frédéric Lang.

Concernant la partie données du langage E-LOTOS, nous avons continué à diffuser le compilateur TRAIAN (voir § 4.2) et à l'utiliser intensivement, au sein de l'équipe VASY, comme un méta-outil pour l'écriture de compilateurs. En 2002, cette approche a fait l'objet d'une publication [20] et d'une démonstration lors de la conférence *Compiler Construction*.

Nous avons continué à faire évoluer le compilateur TRAIAN (plus de 48 000 lignes de code). Nous avons corrigé diverses erreurs, principalement dans l'analyse sémantique (test d'exhausti-

[ISO01] ISO/IEC, « Enhancements to LOTOS (E-LOTOS) », *International Standard n° 15437:2001*, International Organization for Standardization — Information Technology, Genève, septembre 2001.

vité du filtrage, validité du mode de passage des paramètres des fonctions) et dans la génération du code pour les exceptions. Suite à ces corrections, une nouvelle version 2.2 de TRAIAN a été diffusée (voir § 8.1).

Nous avons ensuite commencé à intégrer dans TRAIAN 2.2 plusieurs optimisations étudiées par Claude Chaudet en 1999 (voir le rapport d'activité 1999 de VASY, § 5.2.3), notamment : la représentation des types énumérés en utilisant le nombre minimal de bits nécessaires, la représentation des types isomorphes aux entiers naturels, la suppression des pointeurs pour la représentation des types non récursifs, la réduction du nombre de pointeurs pour la représentation des types mutuellement récursifs, la factorisation des sous-expressions communes et leur mise sous forme canonique en utilisant des tables de hachage pour réduire la quantité de mémoire allouée.

En parallèle, nous avons entrepris la conception d'une version 3.0 de TRAIAN susceptible de compiler aussi bien la partie données de LOTOS que celle de LOTOS NT, c'est-à-dire capable à terme de remplacer aussi bien CÆSAR.ADT que TRAIAN 2.2. Dans ce but, nous avons entamé le recensement des fonctionnalités attendues de ce futur compilateur, que nous avons classées dans une base de données comprenant actuellement 140 entrées.

5.2.2 Compilation de la partie contrôle du langage E-LOTOS

Participants : Aurore Collomb, Hubert Garavel, Frédéric Lang.

La compilation de la partie contrôle de E-LOTOS et LOTOS NT est un problème difficile, compte-tenu de la richesse de ces langages, qui combinent parallélisme asynchrone, temps quantitatif et exceptions. Nous avons choisi d'aborder ces différents problèmes de manière progressive en nous concentrant, dans un premier temps, sur les processus séquentiels présents dans les langages E-LOTOS et LOTOS NT. Pour les modéliser, nous avons conçu un formalisme appelé NTIF (*New Technology Intermediate Form*) destiné à servir de langage intermédiaire dans la compilation et la vérification énumérative des processus E-LOTOS et LOTOS NT.

Dans son principe, NTIF permet de spécifier des automates définis par un ensemble d'états et de transitions et paramétrés par des variables d'état typées. A chaque transition sont attachés une action (permettant la communication avec l'environnement selon la sémantique du rendez-vous propre aux algèbres de processus) ainsi qu'un fragment de code séquentiel permettant de consulter et/ou de modifier la valeur des variables d'états ; ce fragment de code est exprimé avec des structures de contrôle évoluées (instructions "`case`", "`if-then-else`", "`while`", etc.).

De ce fait, NTIF pallie les principales limitations des modèles "conditions/actions" (ou automates "à commandes gardées") couramment utilisés en vérification : l'absence, dans ces modèles, de structures de contrôle évoluées oblige l'utilisateur à introduire de nombreux états et transitions intermédiaires, ainsi qu'à dupliquer de nombreuses conditions booléennes, ce qui constitue une source importante d'erreurs. De plus, NTIF possède une sémantique intuitive, facile à apprendre, car proche de celle des langages algorithmiques et fonctionnels.

Nous avons conçu deux logiciels pour NTIF : NT2DOT, qui permet de visualiser sous

forme structurée les descriptions NTIF en les traduisant vers le format DOT utilisé par l'outil GRAPHVIZ d'AT&T, et NT2IF, qui traduit les descriptions NTIF vers le formalisme "conditions/actions" IF [BFG+99], la traduction s'effectuant par "dépliage symbolique" des transitions NTIF en transitions IF plus élémentaires.

Le formalisme NTIF et les outils associés ont été validés sur deux études de cas significatives : la spécification d'un porte-monnaie électronique multi-devises (EMV *Common Electronic Purse Standard* version 2.2 5-2000) et les commandes administratives d'un système d'exploitation destiné aux cartes à puces pour la téléphonie mobile 3GPP (*3rd Generation Partnership Project*).

En 2002, nous avons amélioré NTIF en lui ajoutant de nouvelles instructions et en renforçant les contrôles effectués dans la sémantique statique. Les outils NT2IF et NT2DOT ont été adaptés en conséquence et quelques corrections d'anomalies leur ont été apportées. Un article décrivant le formalisme NTIF a été publié [21].

Nous avons ensuite orienté nos travaux vers la prise en compte des aspects de temps quantitatif. Après une étude bibliographique des travaux sur les algèbres de processus temporisées, nos recherches ont porté sur l'introduction du temps (notions de délai, d'urgence, etc.) dans NTIF, ainsi que sur une traduction éventuelle de NTIF vers les modèles utilisés en entrée par des outils de vérification temporisés tels que TREX et UPPAAL.

5.3 Etudes de cas et applications pratiques

Mots clés : activité de conception, algorithme distribué, application critique, application répartie, architecture multiprocesseur, architecture parallèle, atomicité, automate, cohérence de caches, concurrence, génération de code, génération de test, génie logiciel, ingénierie des protocoles, logique temporelle, mémoire répartie, modélisation, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, temps réel, test, travail coopératif, validation, vérification de programme.

Résumé : *Nous accordons une grande importance au traitement d'exemples réalistes, notamment issus de besoins industriels, qui nous permettent de vérifier l'adéquation de nos méthodes et outils, et d'identifier de nouvelles orientations de recherche pour résoudre les problèmes rencontrés.*

5.3.1 Protocole de cohérence de caches CC-NUMA "Fame"

Participants : Hubert Garavel, Radu Mateescu, Bruno Ondet, Solofo Ramangalahy.

Depuis octobre 1998, nous collaborons avec BULL dans le cadre du contrat FORMALFAME

[BFG+99] M. BOZGA, J.-C. FERNANDEZ, L. GHIRVU, S. GRAF, J.-P. KRIMM, L. MOUNIER, « IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems », *in : Proceedings of World Congress on Formal Methods in the Development of Computing Systems FM'99 (Toulouse, France)*, J. Wing, J. Woodcock (éditeurs), Springer Verlag, septembre 1999.

(voir § 6.2) qui cherche à promouvoir l'utilisation de méthodes formelles pour la vérification et le test d'architectures multiprocesseurs. Nos travaux portent sur FAME, une architecture CC-NUMA développée au centre BULL des Clayes-sous-Bois (France) et basée sur des processeurs 64 bits INTEL ITANIUM. Nous nous concentrons sur la validation et le test d'un circuit complexe (appelé B-SPS) ayant une double fonctionnalité : il effectue le routage entre les processeurs et les nœuds d'entrées/sorties et il implémente le protocole de cohérence de caches de l'architecture FAME. La complexité de ce circuit provient à la fois du parallélisme interne inhérent au routage et des accès multiples aux données partagées dont il convient de préserver l'intégrité.

Le fonctionnement du circuit B-SPS est décrit par un document de référence (100 pages) combinant des spécifications informelles en langue naturelle avec des tables états/transitions. Le circuit lui-même est réalisé par un modèle en langage VERILOG. Un modèle logiciel en langage C++ du circuit B-SPS permet, soit d'émuler le comportement attendu du circuit, soit de valider le comportement du modèle VERILOG. Dans le cadre de FORMALFAME, une description formelle en LOTOS du circuit B-SPS a été développée en 2000.

Les travaux conduits dans FORMALFAME introduisent des aspects formels dans la méthodologie de validation et de test utilisée par BULL. En 2002, les travaux ont porté sur trois points :

- *Description formelle du protocole de cohérence de caches* : la description formelle en LOTOS du circuit B-SPS est désormais maintenue au centre BULL des Clayes-sous-Bois qui l'améliore et la fait évoluer en fonction des révisions du protocole de cohérence de caches. En 2002, Jack Abily et Romain Henry ont développé de nouvelles descriptions LOTOS détaillées pour deux parties du circuit B-SPS (le bloc PRR et l'unité ILU) afin d'expérimenter la méthodologie FORMALFAME au niveau des blocs et des unités (c'est-à-dire à un niveau plus bas que le niveau système correspondant à la totalité du circuit B-SPS).
- *Génération automatique de tests* : l'un des objectifs de FORMALFAME consiste à dériver automatiquement des cas de test à partir de la description LOTOS, ce qui garantit la correction des tests par rapport à la modélisation formelle. Ainsi, l'effort investi dans la modélisation trouve son utilité non seulement pour la vérification, mais aussi pour la génération automatique de tests qui s'ajoutent aux tests écrits manuellement.
  En 2002, l'approche dont nous avions montré la faisabilité en 2001 a été approfondie au centre BULL des Clayes-sous-Bois. Dans cette approche, basée sur l'utilisation du compilateur CÆSAR en mode "génération de code" (EXEC/CÆSAR), les tests sont obtenus par des exécutions aléatoires de la description LOTOS selon différentes distributions de probabilité. Le code C produit par EXEC/CÆSAR a été connecté à la plate-forme de test de BULL (outil TESTBUILDER [26] de la société CADENCE) afin que celle-ci soit directement pilotée par la description LOTOS. L'expérimentation de cette approche pour le bloc PRR et l'unité ILU a été entreprise.
- *Vérification automatique de traces d'exécution* : nous avons poursuivi nos travaux sur la validation automatique des traces d'exécution produites soit par le modèle VERILOG, soit par l'émulateur C++ du circuit B-SPS. Ces traces sont issues de scénarios de test (déterministes ou aléatoires) et peuvent comporter plusieurs dizaines de milliers de mes-

sages, ce qui rend leur analyse manuelle par relecture prohibitive, surtout dans le cas de transactions imbriquées.
En 2002, nous avons finalisé la méthode consistant à spécifier en logique temporelle des propriétés de correction sur les traces et à vérifier ces propriétés grâce aux outils EVALUATOR 3.0 et SEQ.OPEN (voir § 5.1.4) de CADP. Les propriétés sont produites automatiquement à partir des tables états/transitions figurant dans la spécification de référence du circuit B-SPS. Les résultats de la vérification permettent de mesurer précisément la couverture *fonctionnelle* (déterminée en fonction de la spécification de référence) par un ensemble de traces considéré. Cette couverture, différente de la couverture *structurelle* (évaluée par rapport au modèle VERILOG), n'était pas mesurable objectivement avec les méthodes de test utilisées antérieurement par BULL.
Cette méthode de test (qui avait permis de détecter des erreurs et de rajouter des tests manquants) a été transférée à BULL. Elle est désormais appliquée systématiquement lors de chaque révision du circuit B-SPS. Les améliorations de l'outil SEQ.OPEN effectuées en 2002 ont permis des gains en temps appréciables (d'environ 30%) par rapport à 2001. Ainsi, le temps nécessaire pour effectuer, sur une machine standard (processeur Pentium III 700 MHz avec 1 giga-octet de mémoire vive), 7,4 millions de vérifications unitaires (c'est-à-dire, vérification d'une propriété sur une trace) a été ramené de 31 heures à 23 heures.

Suite à ces résultats, le contrat FORMALFAME devrait être prolongé pour l'année 2003.

5.3.2 Protocole de déploiement de composants Java

Participants : Hubert Garavel, Radu Mateescu, Frédéric Tronel.

Dans le cadre de PARFUMS (voir § 6.3), projet pré-compétitif du Réseau National des Technologiques Logicielles (RNTL), nous travaillons depuis octobre 2001 sur un protocole de déploiement conçu et implémenté par la société SCALAGENT, issue de l'ex-projet SIRAC de l'INRIA. Parmi ses diverses applications potentielles, ce protocole doit servir à installer et à configurer un ensemble d'agents mobiles (composants JAVA) sur des onduleurs fabriqués par la société MGE-UPS.

En 2002, nous avons poursuivi nos travaux sur la modélisation formelle et la vérification de ce protocole, avec les résultats suivants :

- Une étude attentive de la spécification en langue naturelle du protocole a révélé diverses imprécisions et ambiguïtés qui ont été résolues par SCALAGENT.
- Il existe, dans le cadre de PARFUMS, un logiciel permettant de décrire graphiquement une configuration d'agents à déployer ; ce logiciel stocke la configuration voulue dans un fichier au format XML (DTD XOLAN pour la description d'architectures logicielles) qui est lu par le protocole de déploiement SCALAGENT.
  Pour la vérification, nous avons poursuivi le développement d'un traducteur automatique capable, à partir de la description XOLAN d'une configuration d'agents à déployer, de générer une description LOTOS modélisant le fonctionnement dynamique global ob-

servable lors du déploiement de ces agents. Ce traducteur (10 000 lignes de code OCAML utilisant le préprocesseur CAMLP4) lit le format XOLAN et construit un arbre abstrait. La traduction prend en compte :

- D'une part, les aspects comportementaux, en associant, à chaque agent JAVA du protocole de déploiement, un processus LOTOS (qui décrit en fait un automate communicant). Ceci nécessite un traitement particulier, compte-tenu du fait que la définition des agents JAVA fait un usage intensif de la notion d'héritage, notion absente du langage LOTOS. Plus précisément, le protocole de déploiement est défini par une hiérarchie de classes dont chacune décrit partiellement le comportement des agents (et des sous-classes éventuelles) qui la composent. Le traducteur restitue l'héritage en composant le comportement propre de chaque agent avec les comportements de toutes les classes parentes de cet agent.
- D'autre part, les aspects architecturaux, en générant une partie de code LOTOS décrivant la mise en parallèle et la synchronisation des différents processus LOTOS ainsi obtenus. De par la conception même du protocole SCALAGENT, la composition des processus se fait selon un schéma arborescent.

  Dans un premier temps, nous avons expérimenté une approche simple (analogue à celle utilisée dans [CGMdP01]), dans laquelle tous ces processus communiquent par l'intermédiaire d'un processus central unique modélisant un tampon FIFO. Toutefois, nous avons constaté que cette approche provoquait une explosion d'états en vérification énumérative en même temps qu'elle rendait difficile la vérification compositionnelle.

  C'est pourquoi, dans un second temps, nous avons mis en place une approche plus sophistiquée dans laquelle le processus tampon central est remplacé par autant de processus tampons décentralisés qu'il existe de canaux de communication entre agents susceptibles de communiquer. Cette approche favorise la vérification compositionnelle en augmentant le degré de synchronisation entre chacun des processus séquentiels et en introduisant des masquages de transitions à tous les niveaux de l'arborescence du protocole.

  Dans ce but, le traducteur a été étendu pour générer un script de vérification SVL (voir § 5.1.10) correspondant à la partie architecturale. Ce script décrit un scénario de vérification compositionnelle utilisant la minimisation par la bisimulation de branchement. Les processus tampons sont engendrés automatiquement à l'aide de l'outil prototype BCG_GRAPH (voir § 5.1.10).
- Nous avons appliqué ce traducteur à l'étude d'une configuration simple (90 lignes de description XOLAN) qui a produit 3 000 lignes de code LOTOS (19 processus en parallèle) et 230 lignes de SVL. L'emploi de la vérification compositionnelle et des processus tampons multiples a permis de surmonter l'explosion d'états qui se produisait lors de précédentes tentatives : la taille des systèmes de transitions produits durant la vérification composi-

[CGMdP01] M. A. CORNEJO, H. GARAVEL, R. MATEESCU, N. DE PALMA, « Specification and Verification of a Dynamic Reconfiguration Protocol for Agent-Based Applications », *in : Proceedings of the 3rd IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems DAIS'2001 (Krakow, Poland)*, A. Laurentowski, J. Kosinski, Z. Mossurska, R. Ruchala (éditeurs), IFIP, Kluwer Academic Publishers, p. 229–242, septembre 2001. Version étendue parue comme rapport de recherche INRIA RR-4222.

tionnelle n'excède alors jamais 6 600 états.

5.3.3 Autres études de cas

D'autres équipes ont également utilisé la boîte à outils CADP pour diverses études de cas. Pour ne citer que les travaux publiés en 2002, on peut mentionner :

- la vérification d'une unité d'acquisition de données embarquée dans les hélicoptères LYNX [FIK+02] ;
- la vérification de l'architecture JAVASPACES de SUN MICROSYSTEMS [vdPE02a,vdPE02b] ;
- la vérification de l'architecture à espace de données distribué SPLICE conçue par THALÈS Pays-Bas [HvdP01,HvdP02] ;
- la vérification d'un système de verrouillage distribué présent dans un commutateur ATM d'ERICSSON [AED02] ;
- la vérification du protocole d'authentification à clé publique Needham-Schroeder [Pan02] ;
- l'analyse d'architectures logicielles [Ker02].

Par ailleurs, d'autres équipes de recherche ont adopté les composants logiciels offerts par CADP (et notamment par les environnements BCG et OPEN/CÆSAR) pour développer leurs propres outils. Nous pouvons citer les réalisations suivantes :

- l'utilisation de CADP au sein d'un environnement pour la vérification d'applications

[FIK+02] W. J. FOKKINK, N. IOUSTINOVA, E. KESSELER, J. VAN DE POL, Y. S. USENKO, Y. A. YUSHTEIN, « Refinement and Verification Applied to an In-flight Data Acquisition Unit », *in : Proceedings of the 13th International Conference on Concurrency Theory CONCUR'02 (Brno, Czech Republic)*, L. Brim, P. Jancar, M. Kretinski, A. Kucera (éditeurs), *Lecture Notes in Computer Science*, *2421*, Springer Verlag, p. 1–23, août 2002.

[vdPE02a] J. VAN DE POL, M. V. ESPADA, « Formal Specification of JavaSpaces Architecture using mu-CRL », *in : Proceedings of the 5th International Conference on Coordination Models and Languages COORDINATION'02 (York, UK)*, F. Arbab, C. Talcott (éditeurs), *Lecture Notes in Computer Science*, *2315*, Springer Verlag, p. 274–290, avril 2002.

[vdPE02b] J. VAN DE POL, M. V. ESPADA, « muCRL Specification of Event Notification in Javaspaces », *in : Proceedings of X Jornadas de Concurrencia (Jaca, Spain)*, University de Zaragoza, p. 191–204, juin 2002.

[HvdP01] J. HOOMAN, J. VAN DE POL, « Verifying Replication on a Distributed Shared Data Space with Time Stamps », *in : Proceedings of 2nd Progress Workshop on Embedded Systems (Veldhoven, The Netherlands)*, F. Karelse (éditeur), Technology Foundation (STW), octobre 2001.

[HvdP02] J. HOOMAN, J. VAN DE POL, « Formal Verification of Replication on a Distributed Data Space Architecture », *in : Proceedings of the ACM Symposium on Applied Computing SAC'02 (Madrid, Spain)*, H. Haddad, G. Papadopoulos (éditeurs), Universidad Carlos III, mars 2002.

[AED02] T. ARTS, C. B. EARLE, J. DERRICK, « Verifying Erlang Code: A Resource Locker Case-Study », *in : Proceedings of the 11th International Symposium of Formal Methods Europe FME'2002 (Copenhagen, Denmark)*, L.-H. Eriksson, P. A. Lindsay (éditeurs), *Lecture Notes in Computer Science*, *2391*, Springer Verlag, p. 184–203, juillet 2002.

[Pan02] J. PANG, « Analysis of a Security Protocol in muCRL », *Technical Report n°SEN-R0201*, CWI, Amsterdam, The Netherlands, janvier 2002.

[Ker02] A. KERSCHBAUMER, « Non-Refinement Transformation of Software Architectural Patterns », *in : Proceedings of the International Workshop on Refinement of Critical Systems: Methods, Tools and Experience RCS'02 (Grenoble, France)*, M. Butler, T. Muntean (éditeurs), janvier 2002.

multimédia [AP02] ;
- le développement d'outils de minimisation (basés sur les outils BCG) pour les graphes d'accessibilité d'automates temporisés [Loh02] ;
- l'outil MCRL.OPEN développé au CWI (Pays-Bas) qui effectue la connexion à OPEN/CÆSAR du format intermédiaire TBF utilisé dans la compilation du langage μCRL.

6 Contrats industriels (nationaux, européens et internationaux)

6.1 Contrat européen IST ArchWare

Participants : Aurore Collomb, Damien Bergamini, David Champelovier, Nicolas Descoubes, Christophe Joubert, Hubert Garavel, Frédéric Lang, Radu Mateescu.

Mots clés : activité de conception, architecture logicielle, concurrence, génie logiciel, modélisation, parallélisme asynchrone, simulation, spécification formelle, synchronisation, système distribué, validation, vérification de programme.

ARCHWARE (*Architecting Evolvable Software*) est un projet du programme européen *Information Society Technologies* (IST-2001-32360). Démarré au 1er janvier 2002, ARCHWARE regroupe le Consortium de Recherche de Pise (CPR), la société Engineering (Italie), l'Université de Savoie (laboratoire LLP/CESALP et Association Interaction Université-Economie — INTERUNEC), la société THÉSAME (France), les Universités de Manchester et de St. Andrews (Royaume-Uni), ainsi que l'équipe VASY de l'INRIA.

L'objectif d'ARCHWARE est de créer un environnement intégré pour l'architecture et l'ingénierie des systèmes logiciels évolutifs soumis à des exigences fonctionnelles et des critères de performance. Centré autour d'un langage de description pour les architectures logicielles, cet environnement offrira des fonctionnalités pour la définition de styles architecturaux propres à divers domaines d'activités, ainsi que des outils d'ingénierie pour l'analyse des descriptions architecturales. La contribution de VASY dans ARCHWARE porte sur la description et la vérification de propriétés fonctionnelles.

[AP02] T. ARTS, J. J. S. PENAS, « Global Scheduler Properties Derived from Local Restrictions », *in : Proceedings of the ACM Sigplan Erlang Workshop (Pittsburgh, USA)*, J. Hughes (éditeur), ACM, octobre 2002.

[Loh02] C. LOHR, *Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-Lotos*, thèse de doctorat, Institut National Polytechnique de Toulouse, décembre 2002.

6.2 Contrat FormalFame (Bull)

Participants : Hubert Garavel, Radu Mateescu, Bruno Ondet, Solofo Ramangalahy.

Mots clés : activité de conception, algorithme distribué, application répartie, architecture multiprocesseur, architecture parallèle, automate, cohérence de caches, compilation, génération de code, génération de test, mémoire répartie, modélisation, parallélisme asynchrone, programmation parallèle, protocole de communication, spécification formelle, synchronisation, système distribué, vérification de programme.

Depuis 1995, nous entretenons une collaboration de longue durée avec BULL, collaboration à laquelle l'ex-projet PAMPA a participé jusqu'en décembre 2000. Il s'agit de démontrer que les méthodes formelles et les outils développés à l'INRIA pour la validation et le test des protocoles de télécommunication peuvent aussi être appliqués avec succès aux architectures multiprocesseurs développées par BULL. L'objectif à long terme est d'offrir une chaîne complète et intégrée d'outils pour la spécification formelle, la simulation, le prototypage rapide, la vérification, la génération de tests et leur exécution.

Une première étape de cette collaboration a eu lieu entre 1995 et 1998 dans le cadre de l'action VASY du GIE BULL-INRIA DYADE. Elle a été consacrée à deux études de cas successives : le protocole d'arbitrage de bus de l'architecture POWERSCALE [CGM+96] et le protocole de cohérence de caches de l'architecture multiprocesseur POLYKID [5]. Le résultat a été jugé positif : la faisabilité de l'approche proposée a été démontrée et BULL a manifesté son intérêt à poursuivre la collaboration avec de nouvelles architectures.

Depuis octobre 1998, nos travaux portent sur FAME, une architecture multiprocesseur CC-NUMA développée par BULL et basée sur des processeurs 64 bits INTEL ITANIUM. D'abord informelle, cette collaboration a été officialisée en 1999 sous la forme d'une action DYADE, intitulée FORMALFAME, qui a duré jusqu'à la fin du GIE DYADE en mars 2001. Depuis cette date, la collaboration se poursuit sous la forme d'un contrat BULL-INRIA, pour lequel nous avons conservé le nom de FORMALFAME. Ce contrat réunit le projet VASY et l'équipe de Sylvie Lesmanne (qui a succédé à Anne Kaszynski en mars 2002) au centre BULL des Clayes-sous-Bois (France), la coordination étant assurée par un ingénieur BULL (M. Zendri, puis N. Zuanon, puis S. Ramangalahy) installé dans les locaux de l'Unité de Recherche INRIA Rhône-Alpes.

FORMALFAME cible ses efforts sur les composants critiques de l'architecture FAME, successivement : le circuit CCS qui gère les communications pour un groupe de quatre processeurs, le circuit NCS qui gère les communications réseau et, depuis décembre 1999, le circuit B-SPS qui implémente le protocole de cohérence de caches. Pour chacun de ces composants, une

[CGM+96] G. CHEHAIBAR, H. GARAVEL, L. MOUNIER, N. TAWBI, F. ZULIAN, « Specification and Verification of the PowerScale Bus Arbitration Protocol: An Industrial Experiment with LOTOS », *in : Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'96 (Kaiserslautern, Germany)*, R. Gotzhein, J. Bredereke (éditeurs), IFIP, Chapman & Hall, p. 435–450, octobre 1996. Version intégrale parue comme rapport de recherche INRIA RR-2958.

description LOTOS a été élaborée et sert de modèle de référence pour la vérification et le test.

En 2002, les résultats obtenus par FORMALFAME concernent, d'une part, la validation et le test du circuit B-SPS (voir § 5.3.1) et, d'autre part, l'amélioration des différents outils de CADP.

6.3 Contrat RNTL Parfums (MGE-UPS, Scalagent, Silicomp)

Participants : Hubert Garavel, Radu Mateescu, Frédéric Tronel.

Mots clés : activité de conception, algorithme distribué, application répartie, concurrence, génie logiciel, ingénierie des protocoles, modélisation, langage à objets, langage d'interface, parallélisme asynchrone, protocole de communication, spécification formelle, synchronisation, système distribué, validation, vérification de programme.

PARFUMS (*Pervasive Agents for Reliable and Flexible UPS Management Systems*) est un projet pré-compétitif du Réseau National des Technologies Logicielles (RNTL). D'une durée de 2 ans, ce projet a débuté en mai 2001 et regroupe les sociétés MGE-UPS, SCALAGENT (issue de l'ex-équipe SIRAC de l'INRIA), SILICOMP et l'équipe VASY de l'INRIA.

L'objectif de PARFUMS est la mise en œuvre d'une architecture flexible et fiable à base de composants JAVA pour permettre l'administration d'onduleurs à distance depuis divers équipements (ordinateurs, téléphones portables, assistants personnels, etc.). Cette architecture s'appuie sur l'infrastructure logicielle développée dans le projet SIRAC : bus logiciel tolérant les pannes, modèle de programmation par agents (éventuellement mobiles) communiquant par messages, et outils de développement associés. Il s'agit de faciliter le déploiement des logiciels d'administration, leur surveillance, leur reconfiguration et leur mise à jour, tout en réduisant le coût de ces fonctions.

En 2002, la contribution de VASY au projet PARFUMS a porté essentiellement sur la modélisation et la vérification compositionnelle du protocole de déploiement (voir § 5.3.2).

7 Actions régionales, nationales et internationales

7.1 Actions nationales

En 2002, nous avons collaboré avec plusieurs projets INRIA :

- APACHE (Rhône-Alpes) : utilisation de la plate-forme "grappe de PCs" développée par le projet APACHE pour l'expérimentation d'algorithmes de vérification massivement parallèles (voir § 5.1.5) ;
- SIRAC/SARDES (Rhône-Alpes) : collaboration dans le cadre du contrat RNTL PARFUMS (voir § 6.3) ;
- LANDES (Rennes), LEMME (Sophia-Antipolis), OASIS (Sophia-Antipolis) et VERTECS (Rennes) : collaboration dans le cadre de l'ARC MODOCOP.

Nous entretenons également des relations scientifiques avec d'autres équipes françaises :

- Laboratoire ISIMA/LIMOS (Clermont-Ferrand) : méthodes de conception conjointe matériel-logiciel combinant LOTOS, LOTOS NT et VHDL (Pierre Wodey) ;
- Laboratoire VERIMAG (Grenoble) : collaboration dans le cadre de l'ARC MODOCOP.

7.2 Actions internationales

7.2.1 Groupes de travail internationaux

- Le projet VASY est membre de FMICS (*Formal Methods for Industrial Critical Systems*), l'un des onze groupes de travail d'ERCIM (voir `http://www.inrialpes.fr/vasy/fmics`). De juillet 1999 à juillet 2001, H. Garavel a été responsable de ce groupe de travail. Depuis juillet 2002, il est membre du comité de direction (FMICS *Board*), en charge des actions de dissémination.
- H. Garavel est membre du comité technique (*ETItorial Board*) de la plate-forme d'intégration logicielle ETI (*Electronic Tool Integration*) accessible en ligne par Internet (voir `http://www.eti-service.org`).

7.2.2 Relations bilatérales internationales

Nous entretenons des relations scientifiques avec plusieurs universités et centres de recherche internationaux. En 2002, outre les partenaires avec lesquels nous collaborons contractuellement, nous avons eu des échanges suivis avec :

- l'Université de Göthenburg en Suède (Thomas Arts),
- l'Université de Twente (Axel Belinfante et Holger Hermanns),
- l'Université de Kent à Canterbury (Clara Benac Earle),
- l'Université Libre de Bruxelles (Thierry Massart),
- le CWI à Amsterdam (Stefan Blom et Jaco van de Pol).

Dans le cadre de sa 3^{e} année de magistère d'informatique, Ch. Joubert a séjourné trois mois à l'Université de Twente (du 27 juin au 27 septembre 2002).

7.3 Accueil de chercheurs français et étrangers

- Radu Iosif, de l'Université du Kansas (Etats-Unis), nous a rendu visite le 4 janvier 2002 puis du 20 mai au 5 juin 2002. Il a donné un séminaire intitulé "*Exploiting Symmetry in Explicit State Software Model Checking*".
- John Derrick, de l'Université de Kent à Canterbury, nous a rendu visite le 28 janvier 2002.
- Clara Benac Earle, de l'Université de Kent à Canterbury, nous a rendu visite du 4 au 18 mars 2002. Elle a donné un séminaire intitulé "*Development of Formally Verified Erlang Programs*".

- Aurore Collomb, du laboratoire VÉRIMAG, Grenoble, nous a rendu visite le 7 mars 2002. Elle a donné un séminaire intitulé "Vérification d'automates étendus : algorithmes d'analyse symbolique et mise en œuvre".
- Gregor v. Bochmann, de l'Université d'Ottawa (Canada), nous a rendu visite le 14 mars 2002. Il a donné un séminaire intitulé "*Submodule Construction and Supervisory Control : A Generalization*".
- Lars-Ake Fredlund, du SICS (*Swedish Institute of Computer Science*) à Kista (Suède), nous a rendu visite le 18 mars 2002.
- Emmanuel Godard, du laboratoire LABRI à Bordeaux, nous a rendu visite le 11 avril 2002.
- Holger Hermanns, de l'Université de Twente (Pays-Bas), nous a rendu visite du 24 au 26 avril 2002, puis du 8 au 10 octobre 2002.
- Gilles Barthe, du projet LEMME à l'INRIA Sophia-Antipolis, a participé au séminaire annuel de VASY organisé à Aix-les-Bains du 28 au 31 octobre 2002. Il a donné un exposé intitulé "*Tool-Assisted Specification and Verification of the JavaCard*".
- Au cours du même séminaire, Jean-Bernard Stefani, responsable du projet SARDES de l'INRIA Rhône-Alpes, a donné un exposé intitulé "*The M-calculus : A Higher-order Distributed Process Calculus*".

8 Diffusion de résultats

8.1 Diffusion de logiciels

Le projet VASY diffuse principalement deux logiciels : la boîte à outils CADP (voir § 4.1) et le compilateur TRAIAN (voir § 4.2). En 2002, les faits marquants concernant la diffusion de CADP sont les suivants :

- Nous avons préparé et diffusé des beta-versions successives (2002-a, ... 2002-f) des outils CADP, apportant de nombreuses améliorations aux outils déjà existants.
- Nous avons effectué des démonstrations publiques de CADP pendant l'école d'été SFM'02-MC (*School on Formal Methods — Model Checking*, Bertinoro, Italie, 9–14 septembre 2002) et à l'occasion du colloque *Tools Day* associé à la conférence CONCUR'2002 (Brno, République Tchèque, 24 août 2002).
- Le nombre de contrats de licence signés pour CADP est passé de 246 à 274 et nous avons octroyé en 2002 des licences CADP pour 769 machines différentes dans le monde.

La distribution du compilateur TRAIAN s'est poursuivie : 71 sites différents ont téléchargé le compilateur en 2002. Nous avons préparé une nouvelle version 2.2 du compilateur TRAIAN mise en ligne le 5 novembre 2002.

8.2 Animation de la communauté scientifique

- H. Garavel a été membre du comité de programme de TACAS'2002 (*8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*,

Grenoble, France, 6–14 avril 2002) et responsable (*tool chair*) de la sélection des logiciels présentés.
- H. Garavel a été, conjointement avec Rance Cleaveland, président du comité de programme de FMICS'2002 (*7th International Workshop on Formal Methods for Industrial Critical Systems*, Málaga, Espagne, 12-13 juillet 2002).
- H. Garavel a été membre du comité de programme de PDMC'2002 (*Workshop on Parallel and Distributed Model Checking*, Brno, République Tchèque, 19 août 2002).
- H. Garavel a été membre du comité de programme de FORTE'2002 (*IFIP TC6 WG 6.1 Joint International Conference on Formal Techniques for Networked and Distributed Systems*, Rice University, Houston, Texas, 11–14 novembre 2002).
- F. Lang a organisé la troisième réunion de l'ARC MODOCOP à l'INRIA Rhône-Alpes le 25 novembre 2002.
- Depuis septembre 2002, H. Garavel est membre du comité de pilotage d'ETAPS (*European Joint Conferences on Theory and Practice of Software*, regroupant les conférences CC, ESOP, FASE, FOSSACS et TACAS).
- En collaboration avec Stefania Gnesi (IEI-CNR, Pise) et Ina Schieferdecker (GMD-FOKUS, Berlin), H. Garavel est responsable d'un numéro spécial de la revue SCP (*Science of Computer Programming*), à paraître en mars 2003, qui rassemble les meilleurs articles de FMICS'2000.
- H. Garavel est, conjointement avec John Hatcliff (Kansas State University), président du comité de programme de TACAS'2003 (*9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Varsovie, Pologne, 5–13 avril 2003).
- R. Mateescu est membre du comité de programme de VVEIS'2003 (*First International Workshop on Verification and Validation of Enterprise Information Systems*, Angers, France, 22 avril 2003).
- R. Mateescu est membre du comité de programme de FMICS'2003 (*8th International Workshop on Formal Methods for Industrial Critical Systems*, Trondheim, Norvège, 5–7 juin 2003).
- H. Garavel est membre du comité de programme de PDMC'2003 (*2nd International Workshop on Parallel and Distributed Model Checking*, Boulder, Colorado, USA, 14 juillet 2003).
- R. Mateescu est membre du comité de programme de SOFTMC'2003 (*2nd Workshop on Software Model Checking*, Boulder, Colorado, USA, 14 juillet 2003).

8.3 Enseignement universitaire

Le projet VASY est équipe d'accueil pour :

- le DEA "Informatique : Systèmes et Communication" commun à l'Institut National Polytechnique de Grenoble et à l'Université Joseph-Fourier, et
- le DEA "Informatique : communication et coopération dans les systèmes à agents" commun à l'Université de Savoie et à l'Ecole des Mines de Saint-Etienne.

En 2002 :

- R. Mateescu a dispensé le cours "Temps Réel" destiné aux étudiants en 3^{e} année de l'Ensimag (21 heures annuelles).
- F. Lang a assuré, conjointement avec Flavio Oquendo, le cours "Méthodes formelles pour l'ingénierie des logiciels : spécification et vérification de protocoles" destiné aux étudiants du Dea d'informatique de l'Université de Savoie (24 heures annuelles). Ce cours a été transmis simultanément en visioconférence aux étudiants du Dea de l'Ecole des Mines de Saint-Etienne.
- A. Collomb a dispensé les cours "Travailler avec un ordinateur", "Produits documentaires électroniques" et "Documents électroniques" à l'Iut Information et Communication de l'Université Pierre Mendès-France (192 heures annuelles).
- R. Mateescu et S. Ramangalahy ont encadré le stage de probatoire Cnam de G. Badoil intitulé "Vérification de circuits : problèmes et solutions", qui a été soutenu à Grenoble le 26 mars 2002.
- H. Garavel et R. Mateescu ont encadré le stage de Dea de Ch. Joubert intitulé "Techniques et outils pour la construction massivement parallèle de systèmes de transitions", qui a été soutenu à Grenoble le 19 juin 2002.
- F. Lang a participé au jury de Dea de D. Bergamini à Grenoble le 20 juin 2002.
- H. Garavel et F. Lang ont encadré le stage de probatoire Cnam de G. Kergadallan intitulé "Construction de compilateurs avec la technologie Syntax + Traian", qui a été soutenu à Grenoble le 1er juillet 2002.
- H. Garavel et R. Mateescu ont co-encadré avec Thierry Massart (Université Libre de Bruxelles) le stage de maîtrise d'informatique de G. Stragier intitulé "Prise en charge de systèmes de transitions étiquetées de grande taille (utilisation d'un cache et d'un réseau d'ordinateurs)", qui a été soutenu à Bruxelles le 6 septembre 2002.
- H. Garavel a participé au jury de magistère d'informatique de Ch. Joubert à Grenoble le 4 octobre 2002.
- R. Mateescu est membre de la commission de spécialistes (Cse) de l'Université de Savoie (section 27).
- H. Garavel a participé au jury de thèse de Christophe Lohr intitulée "Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-Lotos", qui a été soutenue à Toulouse le 19 décembre 2002.

8.4 Participation à des colloques, séminaires, invitations

Nous avons présenté des communications dans plusieurs conférences et colloques internationaux (voir à ce sujet la liste de nos publications). En outre :

- H. Garavel a donné un exposé sur les activités scientifiques du projet Vasy lors de la première réunion de l'Arc Modocop à l'Inria Sophia-Antipolis le 13 mars 2002.
- H. Garavel a donné un exposé invité intitulé "Panorama des outils Cadp" lors des dixièmes journées Fac (Formalisation des Activités Concurrentes) au laboratoire Laas-Cnrs de Toulouse le 27 mars 2002.
- R. Mateescu a donné un séminaire d'Ur intitulé "*Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus*" à l'Inria Sophia-Antipolis le 27 mars 2002.

- F. Lang a donné un exposé intitulé "Compilation et techniques de description formelle de nouvelle génération" à l'occasion d'un séminaire Qsl (Qualité et Sûreté du Logiciel) à l'Inria Lorraine le 28 mars 2002.
- H. Garavel, Ch. Joubert, F. Lang, R. Mateescu et G. Pace ont participé à la conférence Etaps'2002 (*European Joint Conferences on Theory and Practice of Software*) qui a eu lieu à Grenoble du 6 au 14 avril 2002.
- F. Lang a donné un exposé sur le modèle Ntif lors de la seconde réunion de l'Arc Modocop à l'Inria Rennes le 2 juillet 2002.
- H. Garavel a présidé le colloque Fmics'2002 (Ercim *Working Group on Formal Methods for Industrial Critical Systems*), Málaga (Espagne) du 12 au 13 juillet 2002.
- H. Garavel a donné un exposé invité intitulé "*An Overview of CADP*" à l'Université de Málaga (Espagne) le 16 juillet 2002.
- F. Lang a donné un exposé intitulé "*An Overview of CADP 2001*" lors du *Tools Day* de la conférence Concur'2002, Brno (République Tchèque) le 24 août 2002.
- H. Garavel et F. Lang ont participé à l'école d'été Sfm'02-MC (*School on Formal Methods — Model Checking*), Bertinoro, Italie, du 9 au 14 septembre 2002. H. Garavel y a donné un cours intitulé "*Parallel and Distributed Model Checking*". F. Lang y a effectué une démonstration en public de la boîte à outils Cadp.
- Tous les membres du projet Vasy ont participé à un séminaire d'équipe à Aix-les-Bains du 29 au 31 octobre 2002.
- F. Tronel a donné un exposé intitulé "Vérification d'un protocole distribué basé sur des agents Java" lors de la troisième réunion de l'Arc Modocop à l'Inria Rhône-Alpes le 25 novembre 2002.
- H. Garavel et F. Lang ont effectué une visite au Cea de Saclay le 16 décembre 2002. A cette occasion, H. Garavel a présenté la boîte à outils Cadp et F. Lang a donné un exposé sur Ntif.

8.5 Prix scientifique

Lors de la 10^{e} édition des carrefours de la Fondation Rhône-Alpes Futur, qui s'est tenue à Lyon le 21 novembre 2002, R. Mateescu a remporté le 1er Prix "Technologies de l'information" décerné par France Télécom, pour la présentation intitulée "Evaluator 3.0 : un outil pour la vérification des systèmes critiques".

9 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] H. Garavel, R. Mateescu, I. Smarandache, « Parallel State Space Construction for Model-Checking », *in : Proceedings of the 8th International SPIN Workshop on Model Checking of Software SPIN'2001 (Toronto, Canada)*, M. B. Dwyer (éditeur), *Lecture Notes in Computer Science, 2057*, Springer Verlag, p. 217–234, Berlin, mai 2001. Version révisée parue comme rapport de recherche INRIA RR-4341 (décembre 2001).

[2] H. Garavel, J. Sifakis, « Compilation and Verification of LOTOS Specifications », *in : Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, L. Logrippo, R. L. Probert, H. Ural (éditeurs), IFIP, North-Holland, p. 379–394, juin 1990.

[3] H. Garavel, M. Sighireanu, « Towards a Second Generation of Formal Description Techniques – Rationale for the Design of E-LOTOS », *in : Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems FMICS'98 (Amsterdam, The Netherlands)*, J.-F. Groote, B. Luttik, J. Wamel (éditeurs), CWI, p. 187–230, Amsterdam, mai 1998. Présentation invitée.

[4] H. Garavel, M. Sighireanu, « A Graphical Parallel Composition Operator for Process Algebras », *in : Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'99 (Beijing, China)*, J. Wu, Q. Gao, S. T. Chanson (éditeurs), IFIP, Kluwer Academic Publishers, p. 185–202, octobre 1999.

[5] H. Garavel, C. Viho, M. Zendri, « System Design of a CC-NUMA Multiprocessor Architecture using Formal Specification, Model-Checking, Co-Simulation, and Test Generation », *Springer International Journal on Software Tools for Technology Transfer (STTT) 3*, 3, juillet 2001, p. 314–331, paru également comme rapport de recherche INRIA RR-4041.

[6] H. Garavel, *Compilation et vérification de programmes LOTOS*, Thèse de doctorat, Université Joseph Fourier (Grenoble), novembre 1989.

[7] H. Garavel, « Compilation of LOTOS Abstract Data Types », *in : Proceedings of the 2nd International Conference on Formal Description Techniques FORTE'89 (Vancouver B.C., Canada)*, S. T. Vuong (éditeur), North-Holland, p. 147–162, décembre 1989.

[8] H. Garavel, « OPEN/CÆSAR : An Open Software Architecture for Verification, Simulation, and Testing », *in : Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, B. Steffen (éditeur), *Lecture Notes in Computer Science*, *1384*, Springer Verlag, p. 68–84, Berlin, mars 1998. Version étendue parue comme rapport de recherche INRIA RR-3352.

[9] R. Mateescu, H. Garavel, « XTL : A Meta-Language and Tool for Temporal Logic Model-Checking », *in : Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, T. Margaria (éditeur), BRICS, p. 33–42, juillet 1998.

[10] R. Mateescu, M. Sighireanu, « Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus », *in : Proceedings of the 5th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany)*, S. Gnesi, I. Schieferdecker, A. Rennoch (éditeurs), *GMD Report 91*, p. 65–86, Berlin, avril 2000. Paru également comme rapport de recherche INRIA RR-3899.

[11] R. Mateescu, *Vérification des propriétés temporelles des programmes parallèles*, Thèse de doctorat, Institut National Polytechnique de Grenoble, avril 1998.

[12] R. Mateescu, « Efficient Diagnostic Generation for Boolean Equation Systems », *in : Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000 (Berlin, Germany)*, S. Graf, M. Schwartzbach (éditeurs), *Lecture Notes in Computer Science*, *1785*, Springer Verlag, p. 251–265, mars 2000. Version étendue parue comme rapport de recherche INRIA RR-3861.

Livres et monographies

[13] R. Cleaveland, H. Garavel (éditeurs), *Proceedings of the 7th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2002 (Málaga, Espagne)*, *Electronic Notes in Theoretical Computer Science*, *66.2*, Elsevier, juillet 2002.

[14] H. Garavel, J. Hatcliff (éditeurs), *Proceedings of the 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, *Lecture Notes in Computer Science*, *2619*, Springer Verlag, avril 2003, à paraître.

Articles et chapitres de livre

[15] H. Garavel, S. Gnesi, I. Schieferdecker, « Special issue on the 5th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany) », *Science of Computer Programming 46*, 3, mars 2003, à paraître.

[16] R. Mateescu, M. Sighireanu, « Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus », *Science of Computer Programming 46*, 3, mars 2003, p. 255–281, à paraître.

Communications à des congrès, colloques, etc.

[17] E. Asarin, G. Pace, G. Schneider, S. Yovine, « SPeeDI – A Verification Tool for Polygonal Hybrid Systems », *in : Proceedings of the 14th International Conference on Computer Aided Verification CAV'2002 (Copenhagen, Denmark)*, E. Brinksma, K. G. Larsen (éditeurs), *Lecture Notes in Computer Science*, *2404*, Springer Verlag, p. 354–358, juillet 2002.

[18] K. Claessen, G. Pace, « An Embedded Language Framework for Hardware Compilation », *in : Proceedings of the 4th International Workshop on Designing Correct Circuits DCC'02 (Grenoble, France)*, avril 2002.

[19] H. Garavel, H. Hermanns, « On Combining Functional Verification and Performance Evaluation using CADP », *in : Proceedings of the 11th International Symposium of Formal Methods Europe FME'2002 (Copenhagen, Denmark)*, L.-H. Eriksson, P. A. Lindsay (éditeurs), *Lecture Notes in Computer Science*, *2391*, Springer Verlag, p. 410–429, juillet 2002.

[20] H. Garavel, F. Lang, R. Mateescu, « Compiler Construction using LOTOS NT », *in : Proceedings of the International Conference on Compiler Construction CC 2002 (Grenoble, France)*, N. Horspool (éditeur), *Lecture Notes in Computer Science*, *2304*, Springer Verlag, p. 9–13, avril 2002.

[21] H. Garavel, F. Lang, « NTIF : A General Symbolic Model for Communicating Sequential Processes with Data », *in : Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2002 (Houston, Texas)*, D. A. Peled, M. Y. Vardi (éditeurs), *Lecture Notes in Computer Science*, *2529*, Springer Verlag, p. 276–291, novembre 2002.

[22] H. Hermanns, C. Joubert, « A Set of Performance and Dependability Analysis Components for CADP », *in : Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, H. Garavel, J. Hatcliff (éditeurs), *Lecture Notes in Computer Science*, *2619*, Springer Verlag, avril 2003. à paraître.

[23] F. Lang, « Compositional Verification using SVL Scripts », *in : Proceedings of the 8th International Conference on Tools and Algorithms for Construction and Analysis of Systems TACAS'2002*

(Grenoble, France), J.-P. Katoen, P. Stevens (éditeurs), *Lecture Notes in Computer Science*, *2280*, Springer Verlag, p. 465–469, avril 2002.

[24] R. Mateescu, « Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems », *in : Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2002 (Grenoble, France)*, J.-P. Katoen, P. Stevens (éditeurs), *Lecture Notes in Computer Science*, *2280*, Springer Verlag, p. 281–295, avril 2002.

[25] R. Mateescu, « A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems », *in : Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, H. Garavel, J. Hatcliff (éditeurs), *Lecture Notes in Computer Science*, *2619*, Springer Verlag, avril 2003. à paraître.

Divers

[26] G. Badoil, *Vérification de circuits : problèmes et solutions — Exemple de TestBuilder*, mémoire de probatoire en informatique, Conservatoire National des Arts et Métiers, Grenoble, mars 2002.

[27] H. Garavel, F. Lang, R. Mateescu, « An overview of CADP 2001 », *in : European Association for Software Science and Technology (EASST) Newsletter*, *4*, août 2002, publié également comme rapport technique Inria RT-0254, `http://www.easst.org/newsletter/index.html`.

[28] C. Joubert, *Software Tools for Combining Functional Verification and Performance Evaluation Using CADP*, mémoire de magistère d'informatique, Université Joseph Fourier, Grenoble, octobre 2002.

[29] C. Joubert, *Techniques et outils pour la construction massivement parallèle de systèmes de transitions*, mémoire de DEA en informatique, Institut National Polytechnique de Grenoble et Université Joseph Fourier, Grenoble, juin 2002.

[30] G. Kergadallan, *La technologie Syntax/Traian pour la construction de compilateurs*, mémoire de probatoire en informatique, Conservatoire National des Arts et Métiers, Grenoble, juillet 2002.

[31] G. Stragier, *Prise en charge de systèmes de transitions étiquetées de grande taille – Utilisation d'un cache et d'un réseau d'ordinateurs*, mémoire de maîtrise en informatique, Université Libre de Bruxelles, septembre 2002.