Lionel Morel

Adresse professionelle :
Laboratoire Verimag
Centre Equation
2 Av. de Vignate
38610 Gières
FRANCE
e-mail : Lionel.Morel@imag.fr
web : http://www-verimag.imag.fr/~lmorel

Age : 26 ans
Date de Naissance : 15 Mai 1978
Lieu de Naissance : La Mure (38-France)
Nationalité : Française

Adresse personnelle : 12 rue Linné
38000 Grenoble
FRANCE

Situation actuelle

Attaché Temporaire à l'Enseignement et à la Recherche (poste de 1/2 ATER) - Institut National Polytechnique de Grenoble et laboratoire Verimag.

Centre d'intérêt de recherche Méthodes formelles, systèmes réactifs embarqués, langages synchrones, verification, model-checking, theorem-proving, spécification par contrats, langages de programmation, compilation.
Activités d'enseignement : Architecture des ordinateurs - ENSIMAG et Département Telecom (1ère année)

Formation

2001 - 2005 : **Thèse de Doctorat en Informatique** à Verimag - INPG
Titre: Exploitation des structures régulières et des spécifications locales
pour le développement correct de systèmes réactifs de grande taille.
Jury : Président : Mr Jacques Chassin de Kergommeaux (ENSIMAG - LSR, Grenoble
Rapporteurs : Mr Marc Pouzet (LIP6-Univ. Pierre et Marie Curie, Paris-6)
Mr Patrice Quinton (Antenne Bretagne de l'ENS-Cachan)
Examinateur : Mr Jean-Louis Colaço (Société Esterel Technologies)
Directrice : Mme Florence Maraninchi

Juillet à Septembre 2001 : **Stage de Magistère**
Département d'Informatique et de Mathématiques - Université de Stirling (Écosse, GB).
Sujet: Extension d'une logique *à-la*-HML avec des opérateurs '*'
Directrice: Dr Carron Shankland

2000 - 2001 : **DEA en Informatique** à Verimag - Université Joseph Fourier - Grenoble
Titre: Compilation efficace des itérateurs de tableaux LUSTRE
Directrice: Pr F. Maraninchi
Mention : Bien

1999 - 2000 : **Maitrise d'Informatique** Université Joseph Fourier - Grenoble
Mention : Assez Bien

1999 - 2000 : **Licence d'Informatique** Université Joseph Fourier - Grenoble
Mention : Passable

1999 - 2000 : **DEUG MIAS** Université Joseph Fourier - Grenoble
Mention : Passable

Synthèse des Activités de Recherche

Doctorat

Dates : 2001 - 2004
Lieu : Équipe *Langages synchrones et Systèmes Réactifs* - Laboratoire Verimag
Titre : Extraction d'obligations de preuve d'après la structure des programmes. Application au langage LUSTRE
Directrice : Pr Florence Maraninchi
Résumé : Le travail entreprit dans cette thèse s'inscrit dans le contexte du développement et de la validation des systèmes réactifs synchrones. Il vise à tirer parti de certaines formes de structuration des programmes durant le processus de développement et de validation. Nous étudions premièrement l'utilisation d'opérateurs réguliers de types "itérateurs" qui permettent d'exprimer assez facilement des programmes réguliers manipulant des tableaux. Nous montrons aussi comment, au moment de la validation, on peut tirer partie de ces structures régulières pour rendre la preuve d'une propriété plus simple. Nous nous intéressons ensuite à la spécification dite "par contrat" où un couple *(assume, guarantee)* est associé à chaque composant pour spécifier les hypothèses sur l'environnement et les propriétés satisfaites par le composant sous ces hypothèses. Nous montrons l'intérêt de tels contrats à la fois en terme de spécification et de vérification pour le cas particulier des systèmes synchrones. Nous proposons une série d'algorithmes de transformations de programmes (aussi bien autour de l'utilisation des itérateurs que des contrats) utilisable comme pre-processeur d'objectifs de preuve pour les outils de validation. Nos propositions, notamment sur l'aspect langage des itérateurs, ont répondu à des besoins rencontrés dans les applications industrielles, particulièrement autour du langage Lustre, auquel nous appliquons nos résultat. Ces propositions seront bientôt incluses dans la version industrielle du langage.
Mots-clés : programmation synchrone, Lustre, contrats "assume-guarantee", itérateurs de tableaux, validation

Magistère

Dates : Juillet à Septembre 2001
Lieu : Department of Computing Science and Mathematics at the University of Stirling, Scotland.
Titre : Extension de la logique FULL avec des opérations itératives
Directrice : Dr Carron Shankland
Résumé : LOTOS (Language Of Temporal Ordering Specification) est une technique de description formelle des systèmes distribués et concurrents. Depuis quelques années, le département d'informatique de l'université de Stirling travaille sur une sémantique symbolique de LOTOS centrée sur les Systèmes de Transitions Symboliques (STSs) et sur des techniques de validation associées. En particulier une logique a été développée (FULL) permettant d'exprimer des propriétés sur les STSs. Une des restrictions majeures de cette logique réside dans l'impossibilité d'exprimer des propriétés portant sur des comportements arbitrairement longs. Parmi ces propriétés, on retiendra les propriétés de sûreté (quelque chose de mauvais n'arrivera jamais) et de vivacité (quelque chose de bon arrivera sûrement un jour) qui jouent un rôle central dans la validation des systèmes. Notre travail dans ce contexte a été de proposer une extension à la logique (nommée FULL*) permettant d'exprimer de telles propriétés. Nous avons aussi étudié l'expressivité de la logique (notamment par rapport aux propriétés de sûreté et de vivacité). Ce travail fait l'objet d'une publication à la conférence AMAST'04 et sera présenté en juillet 2004.
Mots-clefs : LOTOS, Vérification Formelle, Logique Temporelle, Systèmes d'États Infinis, Représentation Symbolique.

Diplôme d'Études Approfondies

Dates : 2000 - 2001
Lieu : Équipe *Langages synchrones et Systèmes Réactifs* - Laboratoire Verimag
Titre : Compilation Efficace d'Itérateurs de Tableaux LUSTRE

Résumé : Les tableaux ont été introduits dans le langage LUSTRE à partir de la version V4 (destinée principalement à la compilation vers du matériel). Les opérateurs de manipulation de tableaux alors proposés étaient particulièrement adapté à une cible matériel (notamment sur des cartes programmables Xilinx). Pour une utilisation dans un cadre purement logiciel, plusieurs inconvénients ont été associés aux tableaux de Luste-V4, notamment concernant l'efficacité du code produit (les tableaux sont expansés en variables indépendantes). Dans ce travail, nous présentons de nouveaux opérateurs (adaptés des itérateurs utilisés dans les langages fonctionnels) pour lesquels nous proposons un schéma de compilation vers du code impératif efficace. Un prototype de compilateur a également été développé qui implémente la solution présentée.
Directrice : Pr Florence Maraninchi
Mots-clefs : Systèmes embarqués, compilation, itérateurs, code efficace, boucles, programmation synchrone.

Publications

Conférences Internationales avec Actes et Comité de Lecture

- **Logical-Time Contracts for Reactive Embedded Components**. F. Maraninchi and L. Morel. *30th EUROMICRO Conference on Component-Based Software Engineering Track ECBSE'04, Rennes, France, September 2004*. Auteur principal: **L. Morel**; Présenté par: **L. Morel**;
- **Arrays and Contracts for the Specification and Analysis of Regular Systems**. F. Maraninchi and L. Morel. *4th International Conference on Application of Concurrency to System Design ACSD'04, Hamilton, Ontario Canada, June 2004*. Auteur principal: **L. Morel**; Présenté par: **L. Morel**;
- **Expressing Iterative Properties Logically in a Symbolic Setting**. C. Shankland, J. Bryans, and L. Morel. *10th International Conference on Algebraic Methodology And Software Technology AMAST'04, Stirling, Scotland, July 2004*. Auteur principal: **C. Shankland**; Présenté pour: **C. Shankland**;
- **Efficient Compilation of Array Iterators for** LUSTRE. L. Morel. *1st Workshop on Synchronous Languages, Applications, and Programs, SLAP02, Grenoble, April 2002*. Présenté par: **L. Morel**.

Rapports

- **Exploitation des structures régulières et des spécifications locales pour le développement correct de systèmes réactifs de grande taille**. L. Morel. *Thèse de Doctorat, Mars 2005*.
- **Une logique modale temporelle basée sur une sémantique symbolique de Full-LOTOS**. L. Morel. *Rapport de Magistère, Septembre 2001*
- **Compilation Efficace d'It rateurs de Tableaux Lustre**. L. Morel. *Mémoire de DEA, June 2001*

Présentations dans des Séminaires

- **Contrats pour** LUSTRE. Séminaire de l'équipe "Langages Synchrones" de Verimag, Février 2004;
- **First Steps In Extracting Proof Obligations from Data Structures of** LUSTRE **Programs**. *Synchron'02 - La-Londe-Les-Maures 2002*;
- **Efficient Compilation of Array Iterators for** LUSTRE. *Synchron'01 - 8th International Workshop on Synchronous Programming*, Dagshtul 2001;
- **Efficient compilation of array iterators for the synchronous data-flow language lustre**. Séminaire du département d'informatique et de mathématiques de l'Université de Stirling, Écosse. Septembre 2001; `http://www.cs.stir.ac.uk/seminars/aut01.html`.

Synthèse des Activités d'Enseignement

Le tableau ci-dessous synthétise mes activités d'enseignement par ordre chronologique. Lorsqu'un enseignement est indiqué pour deux années scolaires, le volume horaire correspond à une seule année. A ce jour, le cumul des heures effectuées atteint approximativement 200h équivalent-TD. Chaque enseignement est détaillé plus loin dans le chapitre 2.

Année(s)	Statut	Intitulé	Public concerné / Établissement	Volume Horaire
2000-01	Vacataire	Soutien Thèmes : Découverte d'UNIX, programmation avec C.	DEUG MIAS 1ère Année UJF (BAC+1)	TD : 12h
2001-02 et 2002-03	Vacataire	Langages et Grammaires, Expression Relationnelle (SQL)	DEUG MIAS 2ème Année UJF (BAC+2)	TD : 15h TP : 18h
2001-02	Vacataire	Projet de Compilation	Année Spéciale (BAC+5) ENSIMAG-INPG	Cours : 3h TP : 24h
2002-03	Vacataire	Architectures Logicielle et Matérielle	Licence d'Informatique (BAC+3) UFRIMA - UJF	TD : 18h
2002-03	Vacataire	Projet C	Première Année (BAC+3) ENSIMAG-INPG	TD : 18h
2002-03 et 2003-04	Vacataire	Projet de Programmation	Licence d'informatique (BAC+3) UFRIMA-UJF	TD : 18h
2003-04	Vacataire	Architectures Logicielle et Matérielle	Première Année RICM (BAC+3) EPUG	TD : 18h
2003-04	1/2 ATER	Architectures Logicielle et Matérielle	Première Année (BAC+3) ENSIMAG et Département Telecom	TD : 96h

Abréviations utilisées dans le reste du document pour désigner des établissements ou des filières d'enseignements:

UJF :	Université Joseph Fourier (Grenoble I)
UFRIMA :	UFR Informatique, Mathématiques Appliquées
MIAS :	Filière Mathématiques, Informatique et Applications aux Sciences
ENSIMAG :	École Nationale Supérieure d'Informatique et Mathématiques Appliquées de Grenoble
INPG :	École Nationale Polytechnique de Grenoble
RICM :	Filière Réseaux Informatiques et Communication Multimédia
EPUG :	École Polytechnique de l'Université Joseph Fourier de Grenoble (ex ISTG)

Activités de Recherche

Le contexte général de mes travaux de recherche est celui de la spécification et de la validation formelles des systèmes informatiques.
En 2000-2001, j'ai effectué mon DEA au sein du laboratoire Verimag sur le thème de la compilation efficace des itérateurs de tableaux LUSTRE. D'Octobre 2001 à Mars 2005, j'ai effectué une thèse à Verimag pendant laquelle j'ai étudié les possibilités de prise en compte de la structure des programmes (notamment pour le langage LUSTRE) dans la validation des programmes. La section 1.1 décrit ces 2 travaux. Jusqu'en Aout 2005, je suis ATER à l'ENSIMAG et Verimag, où je poursuis mes travaux de recherches dans la continuité de ma thèse.
Pendant l'été 2001 (juillet à septembre compris) j'ai effectué mon stage de magistère au sein du Département d'Informatique et Mathématiques de l'Université de Stirling (en Écosse). La section 1.2 décrit ce travail et en donne quelques perspectives.

1.1. Spécification et Validation des Systèmes Réactifs

Verimag est une unité mixte de recherche affiliée au CNRS, à l'UJF et à l'INPG dirigée par Mr Joseph Sifakis. Depuis sa création en 1993, le laboratoire mènent des travaux visant à produire des outils théoriques et techniques pour permettre le développement de systèmes embarqués de qualité maîtrisée. Pendant les 10 dernières années, Verimag a contribué activement au développement de l'état de l'art dans les domaines des langages synchrones, de la vérification par model-checking, du test, et de la modélisation des systèmes. Les résultats du laboratoire trouvent de nombreuses applications industrielles, notamment dans des outils pour le développement des logiciels et systèmes embarqués.
En 2000-2001 j'ai effectué mon DEA au sein du laboratoire, plus précisément dans l'équipe "Langages Synchrones et Systèmes Réactifs" (dirigée par Mr Nicolas Halbwachs) sous la direction de Mme Florence Maraninchi. Durant cette année, j'ai étudié la génération de code efficace à partir des itérateurs de tableaux LUSTRE (voir section 1.1.2).
D'Octobre 2001 à Mars 2005 j'ai effectué une thèse toujours sous la direction de Mme Maraninchi (thèse pour laquelle j'ai reçu une Bourse de Docteur Ingénieur BDI du CNRS). Présentant une certaine continuité avec mon travail de DEA, mon sujet de thèse consistait à étudier en quelles mesures certaines formes de structurations des programmes réactifs synchrones (particulièrement pour le langage LUSTRE) peuvent être peuvent être exploitées pour la validation des systèmes réactifs. La section 1.1.3 présentent ce travail, son contexte, les réalisations pratiques qui en ont découlé ainsi que des perspectives que je suis à l'heure actuelle en train d'explorer.

1.1.1 Contexte Général

Le cadre général de ce travail est celui du développement des *systèmes réactifs*. Les systèmes réactifs ont une interaction constante avec leur environnement : leur temps de réponse doit correspondre aux nécessités de ce dernier. On les différencie notamment des systèmes transformationnels qui disposent de toutes leurs entrées à leur initialisation et délivrent leurs sorties à leur terminaison (comme un compilateur). Beaucoup de systèmes réactifs sont aussi *critiques*, leur dysfonctionnement ayant un impact direct néfaste sur leur environnement. En particulier, toute erreur dynamique est à proscrire. Ces systèmes sont utilisés dans le domaine du transport (avions, trains) où de l'industrie (centrales nucléaires).

Approche synchrone La famille des langages et formalismes synchrones a représenté un contribution importante dans le domaine de la programmation des systèmes réactifs. Cette approche est à la base des langages Esterel [BG92], Signal[GAPT85] et LUSTRE [HCRP91]. Elle repose sur l'hypothèse de synchronisme, qui

établit que le temps de réaction d'un système est nul. D'un point de vue *externe*, cela veut dire que les sorties sont produites de façon simultanée avec la réception des entrées. Ceci est clairement impossible à implémenter. Néanmoins, un système synchrone fonctionne parfaitement du moment qu'il réagit assez rapidement, par rapport à la vitesse "imposée" par son environnement. Par exemple, si les modifications significatives dans les entrées du système arrivent au plus toutes les secondes, alors le système doit pouvoir répondre en au plus une seconde.
Lorsqu'on spécifie un système réactif à l'aide d'un langage synchrone, on doit donc vérifier que le code produit par compilation exécute bien une réaction suffisamment rapidement. On calcule donc un temps d'exécution "au pire". Ce calcul est habituellement rendu difficile par les constructions récursives ou de boucles présentes dans les langages de programmation. Mais ces constructions n'apparaissent pas dans les langages synchrones. La structure du code produit est telle qu'on peut *toujours* fournir une sur-approximation du temps d'exécution.
D'un point de vue *interne*, l'hypothèse de synchronisme établit que le délai de communication entre les composants d'un programme est nul. La sémantique de composition dans un langage synchrone est donc simple : on peut composer une multitude de programmes sans changer le temps de réponse global. On ne cherche pas à produire, à partir de programmes synchrones, des programmes parallèles, auquel cas cet aspect de l'hypothèse de synchronisme serait en parfaite contradiction avec la réalité de l'exécution du programme. Les langages synchrones sont au contraire destinés à être compilés vers du code séquentiel centralisé ; la composition parallèle et le mécanisme de communication entre composants ne sont que des mécanismes de description, au niveau du langage. Les programmes sont compilés vers du code séquentiel et n'impliquent ni parallélisme explicite ni communication au moment de l'exécution.
Enfin, les langages synchrones ne sont pas seulement des langages de spécification mais permettent réellement la *programmation* des systèmes. Leurs environnements de programmation fournissent des compilateurs efficaces pour différentes cibles logicielles ou matérielles. Basés sur des sémantiques bien définies formellement, il est facile de les connecter à des outils de validation tels que débogueurs, générateurs de cas de test, prouveurs. Le code typique pour un programme synchrone est constitué d'une boucle infinie. A chaque tour de cette boucle, le programme reçoit ses entrées, calcule ses sorties et mémorise les valeurs nécessaires à la suite de son exécution (voir figure 1.1).

```
<Initialiser la mémoire>
Pour toujours faire
   <lire les entrées>
   <calculer les sorties>
   <mettre à jour la mémoire>
```

FIG. 1.1 – code séquentiel pour un programme synchrone

Lustre LUSTRE est un langage flot de données synchrone *dédié* à la programmation des systèmes réactifs. Il est développé depuis le milieu des années 80 à partir d'une idée de Paul Caspi et Nicolas Halbwachs. Le but est de fournir un cadre formel de développement de logiciels sûrs et efficaces. A ce titre, certaines caractéristiques de langages plus généraux ne sont pas inclus dans LUSTRE : manipulation explicite de la mémoire, allocation dynamique de données, etc. On cherche ainsi à éviter tout erreur dynamique. Par contre, le langage contient des mécanismes de description spécifiques à la programmation synchrone flot de données, notamment la manipulation explicite des flots et des horloges. Le langage LUSTRE permet de décrire des programmes réactifs à partir de *noeuds*. Les noeuds sont des composants *déterministes et réactifs* que l'on définit en Lustre par des ensembles de variables d'entrées, de sorties et des variables locales ainsi que des équations définissant les comportement des variables locales et de sortie. Un système est nécessairement défini à l'aide d'au moins un noeud qui représente son comportement global.
La société Esterel Technologies commercialise un environnement de programmation, appelé SCADE basé sur LUSTRE. De nombreuses applications industrielles ont lieu grâce aux liens qu'entretient l'équipe avec des

sociétés telles que Schneider Electrics, Aérospatiale, la RATP où encore la SNCF.

1.1.2 Compilation efficace des itérateurs de tableaux Lustre

Cette partie présente le travail effectué durant mon DEA. L'article [Mor02] publié en 2001 au Premier Workshop International sur les Langages, Applications et Programmes Synchrones synthétise ces travaux.

Structures de données LUSTRE permet la manipulation de structures de données au travers de langages hôtes, comme C. La définition dans LUSTRE de types complexes ne semble pas toujours nécessaire puisque ces types ne servent fréquemment qu'à *véhiculer* des informations structurées. L'inconvénient est que la manipulation de telles structures n'est pas maîtrisée par les outils LUSTRE. En particulier, on ne peut pas espérer prouver des propriétés liées à la structuration des données si celle-ci n'est pas décrite en LUSTRE.
L'équipe synchrone a donc proposé d'introduire dans LUSTRE les structures de données les plus fréquemment utilisées telles que les types à champs multiples (appelés usuellement "*records*") et les tableaux.

Cas des tableaux Afin d'éviter toute erreur dynamique, l'introduction des tableaux dans le langage lui-même doit répondre aux critères de sûreté suivant :

- les tailles doivent être connues statiquement
- on ne doit pas pouvoir accéder à des éléments d'un tableau par des indices dynamiques

Les tableaux ont été introduits dans LUSTRE-V4 pour la description de systèmes matériels. Les techniques de compilation mises en place (expansion des tableaux en variables indépendantes) sont particulièrement adaptées à la programmation de circuits.
Puis, LUSTRE a été utilisé pour le développement de systèmes logiciels. Il s'est alors avéré qu'à la fois l'expansion et les primitives de manipulation de tableaux (concaténation, extraction de tranches, récursivité statique) ne sont pas pratiques pour le développement logiciel. L'équipe synchrone de VERIMAG a alors proposé l'introduction dans le langage de 4 nouveaux opérateurs appelés "*itérateurs de tableaux*". Le but était de fournir des opérateurs plus sûrs, plus faciles à manipuler et pour lesquels le code généré serait plus efficace (code avec tableaux et boucles de parcours sur ces tableaux).

Apports et réalisations Le travail effectué durant ce DEA a consisté à étudier la compilation vers du code efficace de programmes LUSTRE manipulant ces opérateurs tableaux. A partir des itérateurs, inspirés d'opérateurs itératifs qu'on trouve dans les langages fonctionnels (tels que *fold* ou *map*[Bir88, OM92]), on cherche à générer du code impératif avec une boucle de type "for" pour chacune des itérations utilisées dans le programme LUSTRE.
Lorsque, par exemple, un map (qui consiste à appliquer le même noeud à tous les éléments d'un tableau) est appliqué à un tableau T pour calculer un tableau T', on obtient dans le code impératif généré une boucle de type "for" dont le corps est constitué d'une application du noeud itéré sur un élément du tableau T pour calculer l'élément correspondant du tableau T'. Un algorithme de génération de code a été proposé et implémenté dans un prototype de compilateur.
Parallèlement, nous nous sommes aussi intéressés à l'optimisation des enchaînements des itérations. Considérons l'exemple de la figure 1.2 qui montre graphiquement un enchaînement de 2 opérateurs map. Le premier, $map(f)$, itère le noeud f sur T (c'est-à-dire applique f à tous les éléments de T) pour calculer T'. Le second, $map(g)$, itère g sur T' pour calculer T''. Intuitivement, si T' n'est pas utilisé ailleurs que comme intermédiaire dans cet enchaînement d'itérations, on pourrait l'éliminer et remplacer ces deux itérations par une seule itération d'une composition de f et g (figure 1.3). Ces transformations sont inspirées des notions de *listlessness* et de *déforestation* introduite par Mr P. Wadler[Wad84, Wad90].
Nous avons donné une axiomatisation des transformations d'enchaînements d'itérations. Un algorithme d'optimisation a été proposé qui implémente ces transformations. Une implémentation a été réalisée comme un

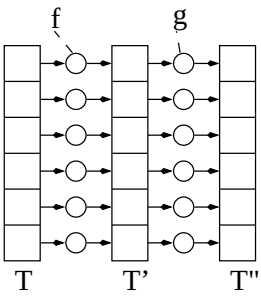

FIG. 1.2 – L'enchaînement $map(f)$ suivi de $map(g)$

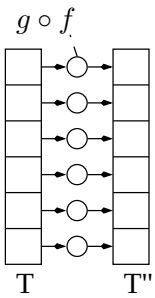

FIG. 1.3 – L'enchaînement d'itérations après optimisation

module externe au prototype de compilateur évoqué plus haut. Pour cela, j'ai utilisé un outil général de transformations de programmes (*Stratego*) dans lequel j'ai pu exprimer les axiomes de transformations comme des règles de réécritures basées sur le langage de terme de Stratego. Cette approche nous a permis de valider rapidement nos propositions. Notre prototype n'ayant pas pour but de servir de base pour un compilateur académique, il était plus rapide de décrire par des règles de réécriture (telles que celle utilisées dans Stratego) des versions simplifiées de l'algorithme d'optimisation que d'implémenter l'algorithme lui-même.
Durant tout ce travail, j'ai étudié un nombre important d'exemples (pour certains tirés d'une étude de cas sur laquelle l'équipe travaillait pour Airbus). L'intégralité de ce travail est décrit dans mon rapport de DEA, disponible en ligne à l'adresse: `http://www-verimag.imag.fr/~lmorel/`.

Transfert Directement après ce DEA, une implémentation complète des algorithmes de transformations et de génération de code a été développé par Jean-Louis Colaço, de la société Esterel Technologie qui développe l'outil Scade, version industriel du langage LUSTRE. Depuis début 2003, une nouvelle version du langage et du compilateur académique est développé au laboratoire Verimag par Mr N. Halbwachs, P. Raymond et Y. Bouzouzou. Ce compilateur prend en compte les itérateurs de tableaux et traite les enchaînements d'itérateurs de la manière décrite plus haut.

1.1.3 Utilisation de la structure des programmes pour la validation

Problématique originale Dans la continuité de mon travail de DEA, principalement centré sur des considérations de *compilation*, la question suivante s'est posé: "dans quelle mesure est-il possible de tirer partie des structures régulières de programmes impliquées par l'utilisation des itérateurs de tableaux pour la validation ?". Dans le processus de vérification (que ce soit pour du test ou pour de la vérification par model-checking) les tableaux étaient jusque-là expansé en variables indépendante (comme pour la compilation en LUSTRE-V4). Nous pouvons identifier deux principaux inconvénients à cette approche : 1- la taille du programme "explose" lors de l'expansion des tableaux. Le model-checking de tels programmes est alors plus difficile voir impossible à réaliser; 2- la structuration du programme est perdue: mais d'une part cette information est peut-être importante pour la validation, d'autre part, en cas de réponse négative du vérificateur pour une propriété donnée, le retour d'information sur les origines potentielles (typiquement extraction de contre-exemple) de l'erreur détectée est impossible à assurer.

Extraction de sous-objectifs de validation Schématiquement, on a donc une structuration particulière des programmes de haut niveau que nous perdons, à bas niveau, lors du processus de validation. L'idée principale de ce travail a consisté à proposer une série de *pré-traitement* sur des programmes avec itérations qui permettent, à partir d'un objectif de validation portant sur des tableaux, d'extraire des *sous-objectifs de validation* portant sur des éléments de tableaux que l'on peut ensuite déchargé aux outils standard de validation LUSTRE.

Considérons par exemple le cas d'un programme calculant la valeur d'un tableau T à l'aide d'un $map(f)$ (où f est un noeud) et sur lequel on souhaite prouver que tous les éléments de T sont positifs. Pour cela, il est suffisant de prouver que l'application de la noeud f produit une valeur toujours positive. Évidemment ce cas est trivial: la propriété est un invariant de la boucle $map(f)$. Dans le cas général, on procède en fait à un renforcement de la propriété à prouver. On va systématiquement considérer que cette propriété est un invariant. Suivant les méthodes classiques de la logique de Hoare[Hoa69], il est alors suffisant de prouver la propriété est vrai avant le passage dans la boucle et qu'elle est préservée par un passage dans celle-ci. Une réponse positive indique toujours que la propriété est satisfaite. Par contre une réponse négative ne peut être interprétée: -soit la propriété est un invariant et n'est pas vérifiée -soit la propriété n'est pas un invariant.
Nous nous intéressons ensuite à l'extension de cette méthode aux programmes manipulant des enchaînements d'itérations comme décrit plus haut. Nous montrons que la transformation décrite ci-dessus se propage facilement à ces enchaînements d'itérations.

Contrats pour des systèmes réactifs En essayant d'élargir la réflexion engagée au début de la thèse, nous avons considéré la forme la plus simple de structuration des programmes en LUSTRE, celle de *noeud*. Pour cela, nous avons commencé à étudier la notion de *contrats* dans le cadre des systèmes réactifs. Un contrat spécifie comment un composant interagit avec son environnement. Il est constitué d'une clause *assume* (ou "require") qui correspond aux hypothèses que fait le composant sur son environnement, et d'une clause *guarantee* qui spécifie quels comportements le composant assure lorsque placé dans un environnement satisfaisant la clause *assume*.
La notion de contrat est étudiée pour différentes formes de composants. Elle a été proposé par Mr Bertrand Meyer [Mey92] (puis par exemple [FF00]) dans le cadre de la programmation orientée objet (particulièrement le langage Eiffel). Des approches similaires à la notre ont été proposée pour le développement des systèmes purement matériel ([WB93, WB94]). Parallèlement, les méthodes compositionnelles ont proposé d'étudier les raisonnements circulaires liés à la composition des composants: Si l'on considère 2 composants P et Q qui communiquent entre eux et qu'on veut prouver que P satisfait une propriété ϕ sous une hypothèse α et que Q satisfait une propriété ψ sous l'hypothèse β, nous devons prouver que ψ (la propriété vérifiée par Q) implique α (l'hypothèse de P) et inversement que ϕ implique β. Dans ce domaine, on peut noter principalement les travaux de Mr Abadi et Mr L. Lamport[AL93, AL95]et de Mr K. McMillan[McM97]. Dans le cadre des systèmes réactifs, on peut noter le travail de Mr De Alfaro et Mr Henzinger[dAH01] qui s'intéressent plus à la synthèse de contrôleurs depuis des spécifications par contrat, ainsi qu'au travail de Mr Holenderski[Hol00] qui s'intéresse à la vérification compositionelle de réseaux de composants synchrones.

Programmation défensive L'étude des contrats pour LUSTRE a débuté avec le stage de magistère de Mr Marc Vareille[Var02] dans lequel il étudiait l'interprétation des contrats dans le cadre du débogage. La thèse de Mr Fabien Gaucher[Gau03] portait sur l'étude du débogage algorithmique de programmes synchrones et a vu notamment la naissance de l'outil Ludic. Le travail de M. Vareille consistait à augmenter l'interpréteur du débogueur LUSTRE afin de pouvoir vérifier qu'un contrat est bien satisfait pendant la simulation d'un programme.

Manipulations de contrats : aide à la conception Dans cette thèse, nous avons repris les définitions des contrats présentée dans [Var02], et nous nous sommes intéressés tout d'abord à leur sémantique. Puis nous avons proposé une série de propriétés standards associées à la notion de contrat: est-ce qu'un composant satisfait le contrat qu'on lui a associé ? Est-ce que, dans un contexte donné, le contrat d'un composant est satisfait ? On a montré aussi comment construire le contrat d'une composition de composants, étant donné les contrats des composants eux-mêmes.
Faisant le lien avec la problématique que nous avons exposé plus tôt sur l'utilisation des tableaux, nous avons aussi étudié les questions suivantes: 1- Étant donné le contrat d'un composant, on peut construire de manière automatique un contrat pour une itération de ce composant; 2- On peut aussi déterminer si le contrat est compatible avec l'utilisation qu'on en fait dans une itération (notion de *"branchabilité"* d'un composant avec lui-même).

A partir d'un noeud à contrat, on peut générer des obligations de preuve pour chacune de ces questions. La résolution de ces obligations de preuves peut ensuite être déléguée aux outils de validation standard.

Un outils de pré-traitement de programmes LUSTRE **pour la validation** D'une manière générale, nous proposons *un ensemble d'algorithmes de transformations de programmes atour des structures d'itérations et par contrats des programmes* LUSTRE. L'ensemble de ces algorithmes a été implémenté dans un outils de pré-traitement des objectifs de preuve LUSTRE. L'interface graphique permet de visualiser le programme en cours de traitement et d'appliquer sur les noeuds de ce programme les manipulations que nous avons indiqué plus haut : 1- concernant les itérations et 2- concernant les contrats de noeuds. Ces transformations génère des objectifs de preuve sous la forme de noeuds LUSTRE, qui peuvent ensuite être transmis aux outils de model-checking ou de test pour LUSTRE.
Le travail réalisé pendant cette thèse a déjà fait l'objet de 2 publications. La première [MM04a] porte sur l'utilité des itérations et de la spécification par contrats dans le développement des systèmes réguliers. Nous présentons les avantages de la méthodologie développée en l'illustrant avec une des études de cas étudiée dans la thèse. La seconde [MM04b] présente la notion de contrat pour les systèmes réactif et compare notre approche à celles utilisées aussi bien dans le monde de la programmation orientée objet que dans celui du développement de systèmes matériel.

1.1.4 Perspectives

Les paragraphes suivants abordent quelques thématiques de recherche qui découlent directement du travail effectué durant cette thèse. Nous en étudions actuellement un partie.

Vers une notion de composants embarqués La notion de contrat que nous avons introduit dans ce travail est répandue depuis de nombreuses années dans le domaine du génie logiciel orienté objet. Par contre, elle est assez récente dans le cadre des systèmes réactifs. Elle semble prometteuse, notamment au regard des études de cas que nous présentons dans les chapitres 6 et 7. Le projet Alidecs (Langages et Atelier Intégré pour le Développement de Composants Embarqués Sûrs) démarre en septembre 2004 pour une durée de 2 ans, dans le cadre de l'ACI2 sécurité). Plusieurs laboratoires (équipe Sémantique, Preuve et Implémentation du LIP-6, équipe Synchrone de Verimag, projets INRIA Pop-Art et Mimosa, équipe CMOS du LaMI) tentent de mettre en commun leurs efforts afin d'« étudier les systèmes embarqués critiques de grande taille, pour lesquels la réutilisation devient un problème crucial ». Le projet privilégie une approche langage des aspects suivants : · support au cycle de vie, depuis les phases initiales de spécification jusqu'au code embarqué efficace ; · mécanismes de composition modulaires utilisables aussi bien dans les programmes, dans leurs spécifications, dans la description des environnements ; · validation précoce des assemblages de composants ; · validation précoce du comportement dynamique des systèmes par simulation des programmes et des spécifications, tout au long des phases de développement ; la possibilité d'exécuter des programmes incomplets est un des objectifs prioritaires3 . Les contrats tels que nous les introduisons dans cette thèse devraient aider à répondre à plusieurs de ces critères. Les contrats seront aussi étudiés dans le cadre du projet européen Assert (Automated proof-based System and Software Engineering for Real-Time4) qui démarre lui aussi en septembre 2004 pour une durée de 3 ans. Ce projet regroupe plusieurs industriels de l'avionique et du transport (l'Agence Spatiale Européenne, Alcatel Space, EADS, Dassault, Esterel Technologies, Prover,...) ainsi que plu- sieurs laboratoires universitaires européens (ETH Zurich, INRIA, CNRS-LAAS, Verimag,...), et a comme objectif global de mutualiser les efforts des participants en terme de spécification correcte des

Une sémantique commune pour les différents formalismes autour de LUSTRE En étudiant les contrats pour LUSTRE, nous nous sommes trouvé devant la nécessité de définir une forme de description commune pour la sémantique des noeuds *et* des contrats. Par la suite, nous avons trouvé intéressant d'essayer d'exprimer les sémantiques d'autres formes de spécifications utilisées dans l'équipe. A l'heure actuelle, nous sommes

toujours en train d'explorer cette voie qui nous permet aussi de comparer les pouvoirs d'expressions de ces différentes formes de spécification.

Exécution de contrats Une fois définie dans un cadre commun les sémantiques de différentes formes de spécification (noeuds lustre déterministes, comportements non-déterministes), nous pouvons imaginer avoir à simuler des réseaux de composants dont certains sont décrits sous la forme de noeuds lustre, d'autres ne possèdent qu'un contrat (sans noeud déterministe associé), d'autres encore prennent d'autres formes. Il est alors nécessaire de déterminer dans quel ordre effectuer les calculs permettant la simulation des différents composants et comment effectuer ces calculs lorsque le comportement est décrit par un contrat. C'est le but d'un travail que nous avons débuté récemment avec F. Maraninchi, N. Halbwachs et P. Raymond.

Contrats pour les systèmes GALS Les GALS[Cas01] sont des systèmes *Globalement Asynchrones, Localement Synchrones* ils sont constitués de composants synchrones (écrit par exemple en LUSTRE) qui communiquent par des moyens asynchrones (files d'attente, rendez-vous...). Dans [HB02], N. Halbwachs et S. Baghadi ont étudié comment les communications asynchrones pouvaient modélisé à l'aide du systèmes d'horloge du langage LUSTRE. Dans un avenir plus lointain, nous aimerions étudier la sémantique et l'utilité des contrats de composants dans le cadre des systèmes GALS.

1.2. Stage de magistère : Extension temporelle de la logique modale FULL

A la suite de mon DEA, et avant d'entreprendre ma thèse à Verimag, j'ai effectué mon stage de troisième année de magistère d'informatique au sein du Département d'Informatique et de Mathématiques, sous la direction de Mme Carron Shankland. Le cadre général reste la validation des systèmes. Nous ne nous intéressons plus à des systèmes réactifs mais à des systèmes distribués ou concurrents.
Le sujet de ce travail n'avait pas été décidé à l'avance: je me suis intégré à l' équipe du projet DIET (comprenant Mme C. Shankland, Mme Savi Maharaj ainsi que Mr Jeremy Bryans) en proposant moi-même un avancement dans une direction que les membres de l'équipe n'avait pas encore pu explorer.
Le projet *DIET* (pour Developing Implementation and Extending Theory) a pour but le développement théorique et l'implémentation d'une approche symbolique pour le raisonnement autour du langage LOTOS. LOTOS est un langage formel (inspiré de CSP[Hoa89])utilisé pour la description des systèmes distribués et concurrents. Dans [CS01], C. Shankland et Mme Muffy Calder ont proposé une sémantique symbolique du langage sous la forme de Systèmes de Transitions Symboliques (STSs). Cette sémantique étend la sémantique standard en donnant un sens aux comportement paramétrés par des données (jusque là oublié par les approches classiques) et fournit une représentation finie de ces comportements. Une bisimulation symbolique a aussi été définie.
Une partie de ce projet a consisté au développement d'une logique modale pour l'expression de propriétés sur les STSs. Cette logique, nommée FULL, permet l'expression de propriétés manipulant les données, grâce à des opérateurs classiques de quantifications sur ces donnée ("*pour tout*", et "*il existe*"). [CMS02] décrit cette logique et montre qu'elle est adéquate vis-à-vis de la bisimulation sur les STSs.
Une des limitations de la logique FULL est qu'elle ne peut exprimer que des propriétés sur des chemins des STSs d'une longueur fixe et connue. Pourtant, les propriétés les plus intéressantes en validation (sûreté et vivacité) manipulent nécessairement des chemins de longueurs arbitrairement grande voir infinie.
Les propriétés de sûreté permettent d'exprimer le fait que quelque chose (de mauvais en général) n'arrivera jamais quelque soit le cheminement suivi dans le parcours du STS. Les propriétés de vivacité, elles, expriment le fait que quelque chose (de bon) arrivera nécessairement un jour.
Mon travail dans ce contexte a consisté à étudier les possibilités de définitions d'opérateurs permettant l'écriture de telles propriétés. J'ai proposé une extension à FULL (nommée sobrement FULL*) pour laquelle j'ai défini des opérateurs similaires à ceux des expressions régulières (concaténation de comportements et répétitions éventuellement vides) partiellement inspiré de ceux qu'on trouve dans la logique XTL[Mat98] de l'outil CADP. Les choix de ces opérateurs a été le fruit d'une longue réflexion pour trouver un compromis entre le pouvoir d'expression de la logique et la facilité d'écriture. De nombreuses logiques (parmi lesquelles la plus connue

est certainement ACTL) ont ainsi été étudiées. Une implémentation partielle de cette extension a été faite en utilisant le langage XTL. De nombreux exemples de propriétés ont été développés tout au long de ce travail.
Ce travail a fait l'objet d'une publication[SBM04] dans la conférence AMAST'04 (International Conference on Algebraic Methods And Software Technology). Le papier sera présenté par Mme C. Shankland au mois de Juillet 2004.

Note personnelle: *Cette expérience a été très enrichissante tant d'un point de vue scientifique que personnel: outre le fait de travailler avec une nouvelle équipe, sur un sujet nouveau pour moi (et d'avoir la possibilité d'explorer une partie de l'informatique que je n'avais encore pas étudié), j'ai du apprendre à* travailler *dans une autre langue (que je maîtrise par ailleurs) et à communiquer mon travail (lors de mon séjour j'ai fait un exposé sur le travail réalisé là-bas et un exposé sur mes travaux de DEA).*

Activités d'enseignement

Cette section présente en détails les enseignements auxquels j'ai participé depuis mon DEA, durant ma thèse et, dernièrement dans le cadre de mes fonctions d'ATER. Pendant ma thèse, j'ai effectué des enseignements aussi bien au sein de l'Université scientifique Joseph Fourier de Grenoble et de l'Institut National Polytechnique de Grenoble. Mon ATER s'est déroulé au sein de l'ENSIMAG. Jusqu'à présent, mes enseignements se sont orientés selon deux thèmes principaux:

- l'étude des architectures des ordinateurs;
- l'étude des langages de programmation et des formalismes associés.

Évidemment, un tel choix n'a pas été fait dans l'idée d'éviter d'autres thèmes, mais dans celle d'approfondir deux thèmes pour lesquels j'ai une sensibilité particulière et pour lesquels je me sentais près, sur une période courte, à assumer des enseignements. J'ai participé à des enseignements pour ces deux thèmes à des niveaux différents (principalement premier et second cycles, et une fois en troisième cycle) et pour des publics différents (étudiants universitaires, élèves-ingénieurs, voir même ingénieurs non-informaticiens en formation 'double-compétence').
Ces enseignements se font sous la forme de séances de travaux dirigés (parfois augmentées de séances de travaux pratiques, surtout au niveau DEUG) étalées sur des périodes de 3 mois. De manière orthogonale, j'ai souhaité participer à l'encadrement de projets dans lesquels les étudiants s'investissent à temps plein pendant des périodes allant d'une semaine à 1 mois. Je considère ces deux facettes comme indispensables à tout enseignement de l'informatique, les projets permettant aux étudiants de mettre en pratique dans des situations (plus) réelles les formalismes et les méthodologies qu'ils apprennent durant les phases plus théoriques de leur cursus. Dans les pages qui suivent, j'ai détaillé les contenus, niveau d'études, volume horaire de ces enseignements répartis en 3 catégories:

- "Architectures Logicielles et Matérielles";
- "Langages, Compilation";
- "Pédagogie par Projets".

Pour chaque enseignement, j'ai aussi indiqué quelques remarques sur la façon dont je le perçois, tant d'un point de vu personnel que pédagogique.

2.1. Architectures Logicielles et Matérielles

2.1.1 ALM - UFRIMA et filière RICM-Polytech'Grenoble

En 2002-2003, j'ai donné des séances de Travaux Dirigés "logiciel" du module ALM en licence d'informatique, sous la responsabilité de Mr Paul Amblard (responsable du module) et de Mme Fabienne Lagnier (responsable des TD "logiciel"). En 2003-2004, j'ai donné des séances de TD similaires pour leur contenu mais cette fois ci à des étudiants en 1ère année de filière RICM[1] de Polytech' Grenoble[2], sous la responsabilité de Mr Pascal Sicard. Chaque année, j'avais la charge d'un groupe de TD d'une trentaine d'étudiants pour 11 séances de 1h30 chacune.

Contenu L'enseignement d'architecture logicielle et matérielle, que ce soit en licence d'informatique ou en 1ère année de la filière RICM, vise à fournir aux étudiants les bases conceptuelles pour la compréhension du

[1] Réseaux Informatiques et Communication Multimédia
[2] École Polytechnique de l'Université Grenoble I

fonctionnement d'un ordinateur. On s'intéresse principalement à 2 thèmes : 1- Le fonctionnement du matériel présent dans un ordinateur, principalement le processeur et les différentes formes de mémoires ainsi que les communications existant entre ces matériels; 2- la traduction d'un programme de *haut-niveau* (écrit par l'utilisateur) en des instructions de *bas-niveau* directement interprétables par le matériel.
Pour ces 2 années, j'ai enseigné des Travaux Dirigés portant sur le second thème. On cherche à montrer aux étudiants comment les instructions qu'ils utilisent dans leur programmes (principalement structure de contrôles et appels de procédures) sont traduites par un compilateur en des instructions dans un langage d'assemblage et comment le processeur interprète ces instructions. On distingue pour cela 2 niveaux de traduction: 1- La traduction des instructions d'un programme impératif en langage d'assemblage; 2- Le codage des instructions du langage d'assemblage en mots binaires directement interprétables par les circuits logiques qui constituent le processeur.

Commentaires En licence d'informatique, les séances de TD contiennent toutes une partie importante de "cours" puisque le cours magistral d'ALM ne présente pas en détail les notions spécifiques au TD "logiciel". Cela nécessite une préparation plus importante et une coordination particulière entre les enseignants des différents groupes. Il est par contre plus facile d'organiser les séances et d'adapter les points à traiter en fonction des difficultés rencontrées par les étudiants. En RICM, par contre, les séances de TD complète un cours magistral. J'ai tout de même rencontré, comme le reste de l'équipe de TD, la nécessité de rappels assez conséquents sur les notions vues en cours.
D'enseigner la même matière dans deux cadres différents m'a beaucoup appris sur les étudiants et les différences qui existent dans les façons qu'ils ont d'aborder une matière. J'ai noté un intérêt plus systématique chez les étudiants de RICM qui hésitent moins à s'investir dans le travail qu'on leur demande. On doit alors répondre à de véritables avalanches de questions, mais qui reste "au niveau" du cours. En licence, l'intérêt apporté à la matière enseignée est plus disparate dans un groupe de TD. Par contre, lorsque cet intérêt existe il est plus profond: on décèle des étudiants plus autonomes et plus près à approfondir les questions soulevées.
Pendant le semestre, les étudiants étaient évalués sur des travaux pratiques dans lesquelles ils devaient coder en langage assembleur quelques algorithmes classiques. J'ai ainsi eu à corriger des comptes-rendus de TP durant ces 2 années. 1 ou 2 séances sur machines encadrées ont été mise en place dans lesquelles nous montrons aux étudiants l'utilisation de base des logiciels tels que compilateur et débogueur.

2.1.2 2004-05 : 1/2 ATER ENSIMAG

Depuis Septembre 2004 (et jusqu'à fin aout 2005), je suis ATER à l'ENSIMAG (INPG-Grenoble). Mes activités d'enseignement portent essentiellement sur l'architecture des ordinateurs. Je donne des TDs et encadre des séances de TPs dans plusieurs modules détaillés ci-dessous, à la fois pour des étudiants en première année à l'ENSIMAG et pour des étudiants en première au département TELECOM (joint entre l'ENSIMAG et l'ENSERG, l'École Nationale Supérieur d'Électronique et Radioélectricité de Grenoble).

2.1.2.1 Digital Circuits

Objectifs de l'enseignement Cet enseignement a pour but de donner les éléments nécessaires à la compréhension du fonctionnement du matériel informatique à travers l'étude de la conception des circuits digitaux.

Contenu

- Algèbre de Boole : calcul booléen, fonctions booléennes, minimisation
- Electronique : semi-conducteurs, transistor bipolaire, transistors MOS, bascules et portes
- Circuits combinatoires : portes, multiplexeurs, logique programmable (ROM, PLA,FPGA) et méthodes de synthèse
- Arithmétique binaire, circuits itératifs et cellulaires
- Circuits séquentiels : bascules, registres, automates et méthodes de synthèse

- Synthèse de circuits complexes, comportant un contrôleur et une partie opérative.

Un ensemble de six séances de travaux pratiques illustre et complète cet enseignement ; l'objectif est de réaliser différents types de circuits à partir de boîtiers de faible complexité et/ou d'observer leur comportement.

J'ai participé à cet enseignement:

- pour l'ecandrement des séances de TPs.

2.1.2.2 Computer Design

Objectifs de l'enseignement Cet enseignement présente les notions de base de l'architecture des ordinateurs : la structure interne d'un processeur, son jeu d'instructions et son exécution, ainsi que les dispositifs d'entrée-sortie, et illustre les principes de la conception moderne de circuits complexes par un projet qui consitse à réaliser un processeur.

Contenu

- Le processeur : spécification (notion de programme machine, jeu d'instructions), étude de cas (conception d'un processeur simple, choix de la partie opérative, spécification et conception de la partie contrôle), les différents types de processeurs (CISC, RISC, ...)
- Les entrées-sorties : modes d'entrée-sortie (mode programmé, mode par interruptions, accès direct mémoire, instructions d'entrée- sortie), traitement des interruptions par le processeur (priorité, masquage, commutation de contexte)

Projet A partir de la spécification d'un processeur simple par son jeu d'instructions, les étudiants doivent concevoir un prototype de ce processeur sur un circuit programmable. Le projet est accompagné d'un enseignement "flot de conception", qui donne une vue d'ensemble sur les méthodes et outils utilisés.

J'ai participé à cet enseignement:

- en donnant des séances de TD à la fois pour des étudiants de première année de la filière TELECOM et pour des étudiants de première année de l'ENSIMAG;
- pour l'encadrement des séances de TPs proposées dans le cadre du projet en première année de la filière TELECOM.

2.2. Langages, Compilation

2.2.1 Informatique - Langages, Expression relationnelle

Pendant 2 années universitaires consécutives 2001-2002 et 2002-2003, j'ai pris en charge un groupe de TD, dans le cadre de l'enseignement d'informatique de second semestre de deuxième année de DEUG MIAS[3], à l'UJF[4]. Ce enseignement se déroulait sous la responsabilité de Mme Catherine Parent-Vigouroux et Mme Marie-Christine Fauvet et comportait, pour chaque groupe, une séance de TD (1h30) et une séance de TP (1h45) par semaine pendant 11 semaines.

Contenu Les objectifs de cet enseignement sont les suivants. On cherche à expliciter la perception des aspects syntaxiques des langages (outils de description adéquats, écriture d'analyseurs). On tente de sensibiliser les étudiants à un mode d'expression déclarative fondé sur le modèle relationnel. On établit aussi le lien entre divers styles d'expression (fonctionnel, actionnel, relationnel). On aide les étudiants à renforcer leur savoir-faire de

[3]Mathématiques, Informatique et Application aux Sciences
[4]Université Joseph Fourier

programmation en ce qui concerne le codage méthodique des algorithmes, la structuration et la documentation des programmes, la modification des programmes et l'utilisation de tests. Enfin, on leur permet de concrétiser, par l'expérimentation, la compréhension des principes et techniques de paramétrisation et de modularité.
L'enseignement est découpé en 2 parties distinctes : Automates et grammaires et Expression relationnelle. Chaque partie comporte une part d'expérimentation importante (réalisée en TP encadrés) qui permet aux étudiants de concrétiser les notions formelles aborder en cours et en TD. La partie "Automates et grammaires" comprend: 1- Pour les séances TD: étude des expressions régulières, analyse lexicale d'un langage, automates reconnaisseurs ; grammaires, analyse syntaxique d'un langage, analyseurs récursifs descendants; 2- Pour la partie TP: Expérimentation autour d'un analyseur d'expressions. La partie "Expression relationnelle" comprend: 1- Pour la partie TD: ensembles et relations ; algèbre relationnelle ; introduction au langage SQL; 2-Pour la partie TP: Expérimentation autour de l'interrogation de bases de données.

Commentaires Un des buts de cet enseignement (en complément avec les enseignements sur les langages fonctionnels) est de montrer aux étudiants les différentes formes de programmation et de faire ressortir leurs utilités respectives. Le module permet de mettre en parallèle dans un temps assez court 2 de ces formes. Grâce aux TP, on arrive à faire sentir aux étudiants que l'on ne pourrait par exemple pas gérer facilement une base de donnée avec un langage impératif.
L'organisation TD/TP est bénéfique, car elle permet d'accompagner l'étudiant depuis les concepts formels jusqu'à la réalisation et l'exécution d'un programme.
Une des difficultés liées à cet enseignement est le fait qu'une partie des étudiants ne se destinent pas à des études en informatiques et qu'ils ont du mal à s'intéresser et à s'investir dans l'apprentissage de la matière. Il faut alors adapter le discours à chacun: ne pas perdre les étudiants les moins motivés en approfondissant trop certaines notions avec les étudiants plus motivés. Cet enseignement m'a aussi donné l'occasion d'apprendre à corriger des travaux écrits. Pendant le semestre, les étudiants ont du: 1- rendre deux comptes-rendus de TP rédigé (un pour chaque thème abordé); 2- rendre un "examen blanc", donné à mis parcours comme devoir à la maison, reprenant généralement le sujet de l'année précédente. J'ai ainsi du corriger des comptes-rendus de TP ainsi qu'un examen complet (même si la note ne comptait pas dans l'évaluation finale des étudiants). J'ai par ailleurs toujours sollicité les étudiants pour qu'ils me rendent des exercices de leur propre chef.

2.2.2 Automates et Applications

En 2002-2003, j'ai donné des séances de Travaux Dirigés dans le cadre du cours "Automates et Applications" pour des étudiants de première année du département "Télécom" de l'ENSIMAG. Cet enseignement se déroulait sous la direction de Mme Florence Maraninchi et comprenait 9 cours magistraux donnés par Mme Maraninchi et 9 séances de TD de 1h30 par groupe. J'avais la charge d'un groupe d'à peu près 30 étudiants.

Contenu Le but du module consiste à fournir à des étudiants visant plutôt un métier dans le domaine des télécommunications, une formation de base sur les langages et les moyens de description des langages, ainsi que sur les automates qui permettent de décrire le comportement d'un système informatique. La continuité est assurée en 2ème année dans le cours de compilation du module "Génie Logiciel".
L'enseignement est découpé en 2 parties. Dans la *partie Langages*, on commence par montrer comment décrire des langages à l'aide du formalisme des expressions régulières (ER). On insiste ensuite sur les limitations des ER dont l'expressivité est trop faible dans la pratique. On introduit ensuite les grammaires hors-contexte sur lesquelles on présente la notion d'ambiguïté. On présente aussi différentes notations possibles pour décrire les grammaires: BNF, EBNF, diagrammes syntaxiques.
Dans la partie Automates, on commence par étudier les automates reconnaisseurs et leur relation avec les ER. On décrit les transformations classiques sur ces automates (minimisation, élimination des ϵ-transitions, déterminisation, etc...). Ensuite, on introduit les automates transducteurs suivant les modèles de Moore et de Mealy. On explique enfin les notions de déterminisme et de réactivité sur ces modèles.

Commentaires La place de ce module dans la formation globale des étudiants en filière TELECOM est particulière. Ces étudiants ne se destinent pas à un travail d'informaticien et il n'est pas évident de leur montrer l'utilité des formalismes présentés dans leur contexte. Pourtant ces notions font partie des bases de l'informatique "pratique" avec laquelle ils devront certainement travailler (compilateur, langage de programmation, notion de "machine").
J'ai tenté d'appuyer mon discours sur cette idée: il me paraît important pour une personne utilisant un compilateur de savoir "un minimum" de chose sur le fonctionnement du compilateur lui-même. Ce module est d'ailleurs conçu comme une introduction à certaines notions formelles qui sont approfondis en seconde année, dans le module "compilation".
Dans cet enseignement, j'ai eu la charge d'un groupe de TD. J'ai ainsi assuré 9 séances de 1h30. J'ai aussi corrigé une partie des copies d'examen.

2.2.3 Programmation et Algorithmique de base

En 2000-2001, j'ai assuré 12 heures de soutien pour des étudiants de DEUG MIAS 1ère année (DSU, Université Joseph Fourier). Mon travail consistait à assurer une permanences de 1h30 à 2h par semaine (pendant 8 semaines) et de répondre aux questions posées par les étudiants. Mon intérêt pour cet enseignement a principalement résidé dans le fait que les étudiants ne me considéraient pas vraiment comme un enseignant (je n'étais jamais là pour les juger) et arrivaient plus facilement à me poser des questions. Les thèmes principalement abordés étaient la programmation impérative et l'utilisation du langage et du compilateur C. Ces 8 séances ont été très importantes pour moi: elles ont constitué ma première expérience d'enseignant et ont consolidé la motivation que j'avais *a priori* pour ce métier.

2.3. Pédagogie par Projets

Depuis le début de ma thèse, j'ai pris à coeur de participer à l'encadrement de projets pendant lesquels les étudiants doivent programmer des logiciels de taille importante, en groupe.
Cet aspect de l'enseignement, où les étudiants sont plongé dans "*une vie de programmeur*" est indispensable à mes yeux afin que chaque étudiant perçoive les aspects *organisationnels* liés à leur futur métier. Dans ces projets, il doivent toujours travailler en groupe (généralement 4 personnes ou plus) ce qui impliquent qu'ils doivent:

- organiser leur travail (mise en pratique indispensable des notions de génie logiciel sur les spécification 'haut-niveau' de leurs programmes)
- gérer le coté *humain* d'un travail d'équipe.

Bien évidement, ces séances de programmation intense permettent aussi d'encrer les notions fondamentales apprises pendant l'année et de leur faire prendre les "bonnes" habitudes de programmation (clarté du code, commentaires, etc...).

2.3.1 Projet de Programmation en Licence d'Informatique

Durant deux années (2002-2003 et 2003-2004), j'ai participé à l'encadrement du projet de fin d'année de la troisième année de licence d'informatique, sous la responsabilité de Mr Laurent Mounier. En 2002-2003, j'ai encadré 2 groupes de 6 étudiants. Malheureusement, suite à des fermetures prolongées des bâtiment de l'UFRIMA (dues à des grèves des personnels ITA de l'Université), le projet n'a pas pu être terminé. En 2003-2004, j'ai encadré 3 groupes de 6 étudiants. J'ai aussi participé à l'amphi de présentation des jeux.

Présentation Le projet se déroule sur 4 semaines (et à temps complet) pendant lesquelles les étudiants, répartis en groupe de 6 doivent développer un logiciel de taille "importante". Le logiciel à produire est un jeu à 2 joueurs choisis parmi une quinzaine de jeu proposés par l'équipe enseignante. Il doit contenir: 1- Une interface graphique; 2- Un algorithme "classique" non trivial pour faire jouer la machine; 3- Un arbitre gérant les parties (tour de jeu, désignation du gagnant, des coups interdits).

Les trois premiers jours du projet sont consacrés à la programmation d'un jeu simple, identique pour tous les groupes qui leur permet d'étalonner leurs méthodes, d'évaluer l'ampleur du travail (et la nécessité de mieux s'organiser). Puis, le "vrai" projet commence. Il est ponctué de plusieurs cours magistraux expliquant certains points généraux ou techniques utiles à la réalisation du logiciel (2 cours d'Interface Homme-Machine, 1 cours de stratégie de jeu). Chaque groupe se voit assigné un tuteur, membre de l'équipe enseignante, qui sert avant tout de conseiller et d'aide. L'évaluation se fait sur présentation oral avec démonstration ainsi que sur plusieurs documents à produire pendant le projet (manuel d'utilisateur, livret de conception, ...).

Commentaires Les difficultés principales de ce projet résident dans la taille du logiciel demandé et dans la nécessité de travailler en équipe. Les étudiants réalisent en général la grande utilité d'un code clair (bien écrit, avec des variables aux noms évocateurs) des spécifications, des commentaires. D'un point de vue encadrement, l'aspect "humain" est essentiel, tant pour résoudre d'éventuels conflits au sein des groupes que pour jouer un rôle de 'garde-fous' auprès des étudiants. Je me suis toujours présenté à eux non comme une critique "enseignante" de leur travail mais comme une aide tant technique qu'organisationnelle et comme un chef de projet, soucieux que le travail soit mené à bien et de la qualité du développement.

2.3.2 Projet C

Cet enseignement se déroule dans le cadre de la première année de l'ENSIMAG, à l'INPG. J'ai participé à l'encadrement du projet pendant l'année 2002-2003, sous la responsabilité de Mr Sylvain Boulmé.

Présentation Le projet consiste à écrire (en partie) un assembleur pour mini-pentium, compatible Gnu/Linux/Elf. Les étudiants utilisent le langage C et sont livrés à eux-mêmes pendant une semaine avec un support "humain" assuré par une équipe de tuteurs dont je faisais partie. L'évaluation de ce projet se fait sur une présentation d'une demi-heure avec démonstration.
Les principaux objectifs de l'enseignement sont: 1- l'apprentissage du langage C ; 2- de fournir une introduction au génie logiciel (développement d'un projet de taille importante); 3- la mise en pratique des enseignements de 1ère année (algorithmique, logiciel de base, théorie des langages); 4- enfin une préparation aux enseignements de 2ième année : compilation, projet Génie Logiciel et système d'exploitation.

Commentaires Le projet dure 1 semaine pendant laquelle mon rôle a consisté principalement à répondre aux questions des étudiants. Ces questions portaient aussi bien sur l'utilisation du langage C ou des aspects de développement du logiciel que sur des problème liés à l'assembleur développé (sur le codage des instructions, etc...).

2.3.3 Projet de Compilation

En 2002-2003, j'ai remplacé Mme Catherine Oriat pour organiser et encadrer le projet de compilation pour l'année spéciale de l'ENSIMAG. Ce projet dure une semaine pendant laquelle les étudiants travaillent en binôme à temps complet en libre service sur les machines de l'école. Le projet concerne la trentaine d'étudiants de l'"année spéciale"

Présentation N'ayant pu avoir de contact avec Mme Oriat (absente cette année là pour des raisons personnelles), j'ai du mettre en place un projet qui permettrai aux étudiants de valider leurs acquis du module compilation. Pour cela, j'ai repris le projet C proposé en première année à l'ENSIMAG dans lequel les étudiants doivent écrire un assembleur pour mini-pentium. Je l'ai orienté afin de permettre aux étudiants de voir toutes les étapes importantes du processus de compilation (analyse lexicale, analyse syntaxique et génération de code).
A partir d'un squelette d'assembleur, j'ai isolé les parties qui me paraissaient pertinentes pour comprendre ces 3 étapes et j'ai demandé aux étudiants de les développer. Étant donnée la durée courte du projet (seulement

une semaine) j'ai évidement insisté pour qu'ils réalisent l'ensemble de la chaîne de compilation pour quelques instructions seulement. L'extension au jeu d'instruction entier ne présente en effet aucun problème fondamental une fois qu'on a compris comment faire pour une instruction. Ce projet a aussi compris une partie cours (3h) durant laquelle j'ai rappelé aux étudiants les notions fondamentales des thèmes suivants: 1- analyse lexicale (expressions régulières, leur reconnaissance); 2- analyse syntaxique (grammaires, ambiguïtés; 3- construction d'une table des symboles; 4- langage d'assemblage qui devaient leur permettre d'accomplir leur projet. L'évaluation s'est faite par une démonstration du logiciel développé par chaque binôme.

Commentaires Cet enseignement m'a permis d'appréhender les difficultés principales liées à ce type d'activité. J'ai préparé un cours d'introduction dans lequel j'ai synthétisé plusieurs thèmes larges faisant tous partie du domaine de la compilation. Bien qu'utilisant les bases d'un projet préexistant, j'ai eu l'occasion de préparer un projet de taille raisonnable. Il a fallu spécialisé les documentations (certaines contenaient beaucoup d'informations qui s'avéraient parasites dans le contexte du projet). J'ai du aussi isolé les parties du code que les étudiants devraient réaliser et faire en sorte de leur fournir une base de projet claire, fonctionnelle et où les points d'intérêts étaient clairement identifiés. L'encadrement du projet lui-même m'a beaucoup intéressé, notamment par les cultures différentes des étudiants de cette troisième année "spéciale" de l'ENSIMAG. Ce sont en général des étudiants motivés qui ont une capacité d'abstraction plus importante (au même titre que, par exemple, les étudiants des DESS 'double compétence'). L'évaluation finale s'est faite sur démonstration des fonctionnalités implémentées par leur logiciel. Cette évaluation 'orale' a été nouvelle pour moi aussi.

Projets d'Enseignements et de Recherche: Construction d'Applications Embarquées

Les systèmes embarqués prennent une place de plus en plus importante. Que ce soit en informatique personnelle, en domotique ou dans des domaines à fort critère de sûreté, on délègue de plus en plus de tâches importantes à des ordinateurs. Cette utilisation grandissante doit indéniablement s'accompagner d'une évolution dans la manière de concevoir les systèmes informatiques afin qu'ils répondent à ces critères de sûreté.
Globalement, il faut distinguer différents niveaux de conception. Il nous faut tout d'abord des techniques de spécification et de conception haut niveau qui soient fondée mathématiquement et dont on peut décrire une sémantique formelle. Le besoin pour cette sémantique formelle des systèmes se fait ressentir lorsque l'on veut vérifier des propriétés sur les systèmes: on doit pouvoir définir à la fois les systèmes et leur propriétés de la façon la plus précise possible. L'aspect "haut-niveau" est indispensable si l'on veut que les personnes développant les applications puissent s'approprier ces techniques le plus facilement possible. Il faut donc fournir des langages à la fois clairs et assez expressif pour répondre aux besoins. Cette partition "langage haut-niveau/sémantique formelle" nécessite évidement des technologies de transformation efficaces qui permettent de générer des systèmes exécutables à partir des langage haut-niveau et garantissant la conservation de la sémantique.
Cette évolution dans la conception des systèmes doit se faire à mon sens selon deux voix parallèles et complémentaires que sont la recherche et l'enseignement. D'un point de vue recherche, la communauté scientifique académique travaillant dans le domaine des systèmes embarqués et de leur sûreté doit répondre aux attentes technologiques des applications finales en proposant de nouveaux formalismes, nouvelles technologies et nouvelles méthodologies qui permettent de mieux appréhender les problèmes mis en jeu.
D'un point de vue enseignement, il paraît crucial de former les ingénieurs de demain à ces nouvelles approches, puisque ce sont eux qui vont pour en tirer partie dans leur travail. Les cursus menant à ces métiers doivent donner les connaissances nécessaires pour la description haut niveau et sémantique des systèmes: 1- méthodes de conception haut-niveau; 2- la construction des systèmes à la fois matériels ou logiciels; 3- la compréhension des modèles parallèles et temporels des systèmes embarqués; ainsi que pour les méthodes de traductions vers des systèmes exécutables: 1- compilation; 2- théories des langages.
Il est pour moi tout à fait essentiel de définir des parallèles importants entre recherche et enseignement car chacun peut apporter à l'autre: l'enseignement peut aussi nourrir la recherche, il nous aide à prendre du recul par rapport à notre thématique.
Évidement, ce parallèle à établir est plus difficile lorsqu'on enseigne en premier cycle universitaire où le public n'a pas forcément vocation à poursuivre des études en informatique. Mais l'informatique doit, à mon avis, être présenté très tôt sous cet aspect formel, pour montrer aux étudiants qu'il représente une façon intéressante d'aborder et de résoudre des problèmes complexes. Le travail à fournir d'un point de vue purement pédagogique est alors plus important, mais tellement intéressant.

Informations complémentaires

4.1. Responsabilités collectives

Avril 2002
Aide à l'organisation pour la conférence ETAPS'02 à Grenoble.
Surveillance d'examens
Pour la quasi-totalité des enseignements détaillés au chapitre 2

4.2. Formations complémentaires

Formatoins Générales

Journalisme scientifique les 23, 24 et 25 Avril 2002.
Volume horaire: 24h.
Histoire des Sciences du 07 Mars au 02 Mai 2002.
Volume horaire : 24h (12 séances de 2h).

Formations Spécialisées

Specification par modèles et développement rigoureux.
Cours de DEA (Informatique Systèmes et Communications) UJF.
Du 15 octobre au 05 décembre 2001. 24h (6 séances).
Basic SPIN Tutorial, Tutorial de la Conférence ETAPS'02. Avril 2002.
Volume horaire : 6h.
Strategies for Program Transformations, Tutorial de la conférence ETAPS'02. Avril 2002.
Volume horaire : 6h.

Workshops

International Workshop on Synchronous Programming
– 2001: du 3 au 7 décembre 2001;
– 2002: du 25 au 29 novembre 2002;
– 2003: du 1er au 5 décembre 2003.
Volume horaire : 3*24 heures (sur 3*1 semaine)

Écoles

École Sémantique des Langages de Programmation.
Du 24 au 29 Mars 2002, Agay - France. Volume horaire : 30h (sur 1 semaine).
École Jeunes Chercheurs en Programmation (EJC'02).
Du 20 au 30 Mai 2002, Rennes - France. Volume horaire : 50h (sur 2 semaines).
École "Types" - Theory and Practice of Formal Proofs.
Du 2 au 13 Septembre 2002, Giens - France. Volume horaire : 50h (sur 2 semaines).

Bibliographie

[AL93] M. Abadi and L. Lamport. Composing specification. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[AL95] M. Abadi and L. Lamport. Conjoining Specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, november 1992.

[Bir88] R. S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computer Science*, pages 151–216. Springer-Verlag, 1988. NATO ASI Series, F55.

[Cas01] P. Caspi. Embedded control: From asynchrony to synchrony and back. In *Embedded Software, EMSOFT'01*, volume 2211. Lecture Notes in Computer Science, 2001.

[CMS02] M. Calder, S. Maharaj, and C. Shankland. A modal logic for full LOTOS based on symbolic transition systems. *The Computer Journal*, 45(1):55–61, ???? 2002.

[CS01] M. Calder and C. Shankland. A symbolic semantics and bisimulation for full lotos. In Kluwer Academic Publishers, editor, *FORTE'01 (International Conference on Formal Description Techniques for Networked and Distributed Systems)*, pages 184–200, 2001.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. *Lecture Notes in Computer Science*, 2211:148–??, 2001.

[FF00] R.B. Findler and M. Felleisen. Behavioral interface contracts for Java. Technical Report CS TR00-366, Department of Computer Science, Rice University, Houston, TX, August 2000.

[GAPT85] P.L. Guernic, A.Benveniste, P.Bournai, and T.Gauthier. Signal: A data flow oriented language for signal processing. Technical Report IRISA report 246, IRISA, 1985.

[Gau03] F. Gaucher. *Étude du débogage des systèmes réactifs et application au langage* LUSTRE. PhD thesis, Institut National Polytechnique de Grenoble, 2003.

[HB02] N. Halbwachs and S. Baghdadi. Synchronous modeling of asynchronous systems. In *EMSOFT'02*, Grenoble, October 2002. LNCS 2491, Springer Verlag.

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

[Hoa69] C. A. R. Hoare. An Axiomatic Basis of Computer Programming. *Communications of the ACM*, 12:576–580, 1969.

[Hoa89] C. A. R. Hoare. Communicating sequential processes. In *C. A. A. Hoare and C. B. Jones (Ed.), Essays in Computing Science, Prentice Hall*. 1989.

[Hol00] L. Holenderski. Compositional verification of synchronous networks. In *FTRTFTS: Formal Techniques in Real-Time and Fault-Tolerant Systems: International Symposium Organized Jointly with the Working Group Provably Correct Systems – ProCoS*. LNCS, Springer-Verlag, 2000.

[Mat98] R. Matesscu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[McM97] K. L. McMillan. A compositional rule for hardware design refinement. In *Proc. 9th International Computer Aided Verification Conference*, pages 24–35, 1997.

[Mey92] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[MM04a] F. Maraninchi and L. Morel. Arrays and contracts for the specification and analysis of regular systems. In *Fourth International Conference on Application of Concurrency to System Design (ACSD)*, Hamilton, Ontario, Canada, June 2004.

[MM04b] F. Maraninchi and L. Morel. Logical-time contracts for the development of reactive embedded software. In *30th Euromicro Conference, Component-Based Software Engineering Track (ECBSE)*, Rennes, France, September 2004.

[Mor02] L. Morel. Efficient compilation of array iterators for lustre. In Florence Maraninchi, Alain Girault, and ?ric Rutten, editors, *Electronic Notes in Theoretical Computer Science*, volume 65. Elsevier, 2002.

[OM92] J-P Sansonnet O. Michel, D. De Vito. 8_1/2 : Data-parallelism and data-flow. Technical report, LRI-CNRS, Université Paris-Sud, 1992.

[SBM04] C. Shankland, J. Bryans, , and L. Morel. Expressing iterative properties logically in a symbolic setting. In *10th International Conference on Algebraic Methodology And Software Technology AMAST'2004*, 2004.

[Var02] M. Vareille. Extension du concept de programmation par contrats aux système synchrones réactifse à travers le langage synchrone lustre. Master's thesis, UFRIMA-UJF Grenoble, 2002.

[Wad84] P. Wadler. Listlessness is better than laziness. In *Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming*, pages 45–52. ACM, ACM, august 1984.

[Wad90] P. Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 90.

[WB93] H.-Y. Wang and R.K. Brayton. Input Don't Care Sequences in FSM Networks. In *IEEE /ACM International Conference on CAD*, pages 321–329, Santa Clara, California, November 1993. ACM/IEEE, IEEE Computer Society Press.

[WB94] H.-Y. Wang and R.K. Brayton. Permissible observability relations in FSM networks. In Michael Lorenzetti, editor, *Proceedings of the 31st Conference on Design Automation*, pages 677–683, New York, NY, USA, June 1994. ACM Press.