INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Action MIRÓ

Systèmes à Objets, Types et Prototypes :

Sémantique et Validation

Lorraine, Sophia Antipolis

THÈME 2A

Table des matières

Miró est un projet commun à l'INRIA Lorraine, à l'Université Henri Poincaré, Nancy 1 et l'École des Mines de Nancy.

1 Composition de l'équipe

Responsable scientifique
Luigi Liquori [CR INRIA Lorraine à partir de septembre 2001]
Responsable permanent
Dominique Colnet [Maître de Conférences U.H.P. Nancy 1]
Personnel Inria
Olivier Zendra [CR INRIA Lorraine à partir de décembre 2001]
Ingénieur expert
Philippe Ribet [Ingénieur expert ODL-INRIA, à partir de juin 2001]
Chercheurs doctorants
Alberto Ciaffaglione [Doctorant Protheo, co-tutelle INPL-Univ. Udine, de mars 2001 à décembre 2002]
Benoît Sonntag [Doctorant CCH-INRIA]
Stagiaires
Vincent Croizier [Prof. agrégé, DEA U.H.P. Nancy 1 de janvier à juillet 2001]
Sylvain Salvati [DEA I.N.P.L. de janvier à juillet 2001]
Collaborateur extérieur
Furio Honsell [*Ad Honorem*, Président des Universités de Udine, Italie]
Assistante de projet
Danielle Marchand [U.H.P. Nancy 1, à temps partiel]
Projet CertiLab[1]
Joëlle Despeyroux [CR INRIA Sophia-Antipolis]
André Hirschowitz [Professeur, Université de Nice-Sophia]
Nathalie Bellesso [INRIA Sophia-Antipolis, à temps partiel]

2 Présentation et objectifs généraux

2.1 Pourquoi Miró

Johachim Miró est, à notre humble avis, un grand peintre *à objets* ; en fait, ses tableaux de la période 1940-1960 sont basés sur l'utilisation de nombreux objets géométriques : points, points colorés, carrés, cercles, fenêtres, lignes, courbes, etc. Tous ces éléments sont très chers aux amateurs de la programmation par objets. On peut même aller jusqu'à imaginer que ce peintre, qui s'est installé en France à Montmartre au début des années 50, a certainement inspiré les concepteurs de Simula, de Smalltalk, et la communauté d'informaticiens qui ont étudié les aspects théoriques du paradigme à objets.

[1] L'avant-projet CertiLab a formellement fusionné avec l'avant-projet Miró au comité des projets INRIA Lorraine du 3/12/01.

2.2 Objectifs généraux de Miró

L'un des objectifs de Miró consiste à explorer la possibilité de "concilier" la programmation à objets et la programmation fonctionnelle, tout en gardant l'esprit de l'une et l'élégance mathématique de l'autre.

Les langages à objets ont acquis une importance prépondérante dans les applications informatiques à grande échelle. Cette utilisation a rendu nécessaire l'étude formelle de ces langages pour à la fois mieux en cerner les caractéristiques fondamentales et aussi pour pouvoir définir de nouveaux langages à objets et concurrents, capables de combiner une plus grande expressivité avec une sécurité d'utilisation. C'est dans ce dernier cadre que notre recherche se situe.

Plus particulierement, nous nous intéressons à la certification des outils développés autour de ces langages (interprètes, compilateurs, ...), avec comme assistant à la preuve privilégié, le système Coq.

En complément de ces axes de recherche principaux, nous étudions la théories des types, la recherche de nouveaux systèmes améliorant l'activité de la preuve formelle, et la compilation efficace des langages de programmation à objets.

Ainsi, notre programme de recherche se focalise donc essentiellement sur les trois points suivants :

1. l'étude, la définition, et l'implantation certifiée d'un langage de programmation à classes, appelé SmallTalk2K, et d'un langage à prototypes, appelé FunTalk, langage intermédiaire du compilateur de SmallTalk2K, que nous definissions plus loin ;
2. l'étude de l'efficacité et de la sûreté des systèmes à objets, notamment du compilateur SmallEiffel et du système Isaac que nous definissions plus loin ;
3. la théorie des types pour les langages à objets et pour les assistants de preuves, le logiciel certifié, la réécriture et les calculs formels $(\lambda, \pi, \varsigma, \rho, \ldots)$ à la base des langages de programmation à objets, fonctionnels et concurrents.

3 Fondements scientifiques

Cette section présente les fondements scientifiques de Miró, le langage à objets Eiffel et son compilateur GNU SmallEiffel, les calculs à objets purs, dits *à prototypes*, les calculs de réécriture, et notamment le ρ-calcul, l'interaction entre les langages à objets et les langages fonctionnels, la théorie des types, les preuves formelles, ainsi que les assistants à la preuve.

3.1 Les langages et calculs à objets

3.1.1 Le langage Eiffel et le compilateur GNU SmallEiffel

Eiffel concrétise une certaine idée du développement de logiciels, qui repose sur le fait qu'il peut être effectué comme un ouvrage d'ingénieur. Ayant pour but d'atteindre un certain niveau de qualité logicielle, au terme d'un processus rigoureux de production et d'enrichissement de composants réutilisables, scientifiquement spécifiés, paramétrables, communiquant sur la base de contrats clairement définis et organisés selon des classifications multi-critères systématiques.

Si la notion de classe en programmation à objets trouve sa raison d'être dans une implantation de types de données abstraits, il ne suffit pas qu'une classe soit connue au travers des seules opérations qui lui sont applicables, mais aussi au travers des propriétés formelles de ces opérations. En Eiffel, le rôle des *assertions* est multiple : elles aident à produire du code correct et robuste, favorisent une documentation de haut niveau, fournissent un support de mise au point et servent de base au traitement des exceptions. C'est sur ce langage, Eiffel, que nous nous appuyons dans le cadre du projet SmallEiffel.

Le projet SmallEiffel [MNC+91,CCZ97,ZCC98] [17, 6], démarré en 1994 par D. Colnet, a donné lieu en 1995 à la première diffusion dans le domaine public d'un prototype de compilateur Eiffel gratuit. Depuis 1995, ce compilateur est utilisé par les étudiants de la licence d'informatique de l'U.H.P. Nancy 1 ainsi que par les élèves de l'ESIAL (École Supérieure d'Informatique et d'Application de Lorraine). Depuis sa création, SmallEiffel en est à sa $26^{ième}$ version et ne cesse de s'améliorer et de se diffuser au plan international.

La distribution actuelle de SmallEiffel comporte : un traducteur Eiffel vers C ANSI, un traducteur Eiffel vers la machine virtuelle Java (*bytecode*), un indenteur de programme, un extracteur d'interfaces paramétrable (HTML, TEX, etc.), une bibliothèque de base richement garnie ainsi qu'un bon nombre d'outils supplémentaires comme par exemple un dé-compilateur de *bytecode* Java.

Grâce à la technique originale d'inférence de types développée, plus de 80% des appels des méthodes sont couramment liés statiquement (la vitesse d'un exécutable Eiffel est comparable à celle d'un programme écrit en C++ ou en C).

De par sa qualité reconnue au plan international, SmallEiffel a été promu logiciel GNU par R. Stallman, auteur d'`Emacs` et `gcc`, président fondateur de la FSF (Free Software Foundation), qui nous a fait l'honneur de venir nous rencontrer au LORIA afin de concrétiser le label GNU. Ainsi, depuis janvier 1998, SmallEiffel est officiellement *SmallEiffel, The GNU Eiffel Compiler*.

3.1.2 Les calculs à prototypes et leurs systèmes de types

Parmi les langages à objets, les langages à prototypes [NTM99] (ou langages à délégation) constituent un axe de recherche important de l´avant-projet Miró ; en particulier, L. Liquori s'est intéressé aux aspects théoriques de ces langages, notamment aux *calculs à objets*, calculs où la création des nouveaux objets est déléguée aux objets eux-mêmes (calculs à délégation ou *delegation-based*). Ces calculs sont très importants pour fournir des sémantiques dénotationnelles, ou des systèmes de types sophistiqués pour les langages à prototypes.

Le *Lambda Calcul des Objets (LCO)* défini par Fisher, Honsell et Mitchell [11] à l'université de Stanford, et le *Calcul à Objets (OC)* défini par Abadi et Cardelli [1] au Centre de Recherche

[MNC+91] G. Masini, A. Napoli, D. Colnet, D. Léonard, K. Tombre, *Object-Oriented Languages*, Academic Press, Lyon, 1991.

[CCZ97] S. Collin, D. Colnet, O. Zendra, « Type Inference for Late Binding. The SmallEiffel Compiler. », *in : Joint Modular Languages Conference, LNCS, 1204*, Springer Verlag, p. 67–81, 1997.

[ZCC98] O. Zendra, D. Colnet, P. Coucaud, « With SmallEiffel, The GNU Eiffel Compiler, Eiffel joins the Free Software community. », *GNU Bulletin 25*, 1998.

[NTM99] J. Noble, A. Taivalsaari, I. Moore (éditeurs), *Prototype-Based Object-Oriented Programming: Concepts, Languages, and Applications*, Springer Verlag, 1999.

Digital de Palo Alto en sont les principaux représentants. En particulier, les aspects de typage et d'implantation desdits calculs ont été développés dans [14, 15, 12, 24].

Dans les calculs à délégation, les objets sont définis directement à partir d'autres objets, en utilisant ces derniers comme prototypes. Les seules opérations sur les objets sont l'extension d'un objet avec une variable ou une méthode (*object extension*), et la surcharge d'une variable ou d'une méthode (*object overriding*). L'objet modifié hérite de toutes les propriétés du prototype. Plusieurs modèles fonctionnels et impératifs avec différents systèmes de typage ont été présentés [Mic90,Pal93,PJ97,Fis96,BF98,Rém98,RS98].

Les calculs LCO et OC conduisent à divers systèmes de types qui empêchent l'erreur `message-not-found` à l'exécution, qui apparaît quand un objet reçoit un message qui n'est pas présent dans son interface. Les systèmes de types pour LCO et OC sont très puissants : en particulier ils permettent la *mytype method specialization*, c'est à dire la possibilité de spécialiser les types des méthodes héritées (une telle possibilité est déjà offerte par le langage Eiffel, avec le type `like Current`, qui malheureusement a un système de types qui ne prévient pas l'erreur `message-not-found`). Avec le calcul à prototypes on peut modéliser des langages comme Self [16], Obliq [Car95], Kevo [Tai83], Emerald [RTL+91], Cecil [Cha93], et Omega [Bla91].

Les calculs à prototypes peuvent aussi être utilisés comme "langages cibles" pour implanter et étudier des propriétés formelles des langages à classes, puisque les classes peuvent être vues comme des objets capables de recevoir le message `new` de création d'un objet.

[Mic90] J. C. Michell, « Toward a Typed Foundation for Method Specialization and Inheritance », *in: Proc. of POPL*, The ACM Press, p. 109–124, 1990.

[Pal93] J. Palsberg, « Efficient Inference of Object Types », *in: Proc. of LICS*, p. 186–195, 1993.

[PJ97] J. Palsberg, T. Jim, « Type Inference for Simple Object Types is NP-Complete », *Nordic Journal of Computing 4*, 3, 1997, p. 259–286.

[Fis96] K. Fisher, *Type System for Object-Oriented Programming Languages*, thèse de doctorat, University of Stanford, 1996.

[BF98] V. Bono, K. Fisher, « An Imperative, First-Order Calculus with Object Extension », *in: European Conference for Object-Oriented Programming, LNCS*, 1445, Springer Verlag, p. 462–497, 1998.

[Rém98] D. Rémy, « From Classes to Objects via Subtyping », *in: Proc. of ESOP, LNCS, 1381*, Springer Verlag, p. 200–220, 1998.

[RS98] J. Riecke, C. Stone, « Privacy via Subsumption », *in: Electronic proc. of FOOL-98*, 1998. Also in *Theory and Practice of Object Systems*.

[Car95] L. Cardelli, « Obliq: A Language with Distributed Scope », *Computing Systems 8*, 1, 1995, p. 27–59.

[Tai83] A. Taivalsaary, « Kevo, a prototype-based object-oriented language based on concatenation and module operations », *rapport de recherche n°LACIR 92-02*, University of Victoria, 1983.

[RTL+91] K. R. Rajendra, E. Tempero, H. M. Levy, A. P. Black, N. C. Hutchinson, E. Jul, « Emerald: a general-purpose programming language », *Software Practice and Experience 21*, 1, 1991, p. 91–118.

[Cha93] G. Chambers, « The Cecil language specifications and rationale », *rapport de recherche n° 93-03-05*, University of Washington, Dept. of Computer Science and Engineering, 1993.

[Bla91] G. Blashek, « Type-safe OOP with prototypes: the concepts of Omega », *Structured Programming 12*, 12, 1991, p. 1–9.

3.2 Programmation fonctionnelle contre programmation à objets : un conflit sans cause

Mots clés : Self, O-Haskell, Claire, ML2000, Moby, Obliq, Ocaml.

Glossaire :

Self Langage à objets où tout est objet (pas de classe, rien que des objets) [16].

O-Haskell Version à objets de Haskell[PHA+97,Nor99], qui est un langage fonctionnel pur, c'est-à-dire sans la possibilité d'avoir accès à la représentation interne des structures en mémoire (pas d'effet de bord).

Claire Langage fonctionnel, logique et à objets développé chez Bouygues Telecom par Y. Caseau et F. Laburthe [CL99].

ML2000 Nouvelle définition du langage ML. Il contient en *built-in* la notion d'objets et de *type objet* [Gro99].

Moby Langage de programmation à prototypes en cours de développement chez AT & T [FGM+00,FR99,FR00]. Ce langage devrait être à la base de la nouvelle définition de ML2000.

Obliq Langage de programmation à prototype développé chez Digital/Compaq. Ce langage est conçu pour la mobilité du code [Car95].

Ocaml Langage fonctionnel et à objets développé à l'INRIA Rocquencourt [Cri00].

Les langages à objets sont quasi universellement considérés comme intrinsèquement impératifs : en fait ils ont une notion d'état, c'est-à-dire de variables propres à chaque instance d'objet. Néanmoins :

- une grande part de la littérature scientifique sur la théorie des langages à objets à été développée au travers de petits langages à objets fonctionnels purs (voir par exemple, à ce sujet, le récent papier sur *Featherweight Java* de A. Igarashi, B. Pierce et P. Wadler [IPW99]) ;
- des concepts comme l'héritage, la liaison dynamique, l'envoi de message, la surcharge de

[PHA+97] J. Peterson, K. Hammond, L. Augustsson, B. Boutel, W. Burton, J. Fasel, A. Gordon, J. Hughes, P. Hudak, T. Johnsson, M. Jones, E. Meijer, S. Peyton-Jones, A. Reid, P. Wadler, « Haskell 1.4: A Non-strict, Purely Functional Language », 1997.

[Nor99] J. Nordlander, *Reactive Objects and Functional Programming*, thèse de doctorat, Dept. of Computing Science, Chalmers University of Technology, Göteborg, Sweden, 1999.

[CL99] Y. Caseau, F. Laburthe, « CLAIRE: Combining Objects and Rules for Problem Solving », *in : Proc. of ICLP*, T. M. Press (éditeur), p. 245–259, 1999.

[Gro99] T. M. W. Group, « Principles and a Preliminary Design for ML2000 », 1999.

[FGM+00] K. Fisher, D. Grossmann, D. MacQueen, R. Pucella, J. Reppy, J. Riecke, S. Weirich, « The Moby Programming Language », 2000, `http://www.cs.bell-labs.com/who/jhr/moby/index.html`.

[FR99] K. Fisher, J. H. Reppy, « The Design of a Class Mechanism for Moby », *in : Proc. of PLDI, SIGPLAN Notices, 34*, The ACM Press, p. 37–49, 1999.

[FR00] K. Fisher, J. H. Reppy, « Extending Moby with Inheritance-Based Subtyping », *in : Proc. of ECOOP, LNCS, 1850*, Springer Verlag, p. 83–107, 2000.

[Cri00] Cristal, « The Objective Caml », 2000, `http://pauillac.inria.fr/ocaml/`.

[IPW99] A. Igarashi, B. Pierce, P. Wadler, « Featherweight Java: A minimal core calculus for Java and GJ », *in : Object-Oriented Programming, Systems, Languages, and Applications*, The ACM Press, p. 132–146, 1999.

méthodes et l'extension dynamique des objets ne sont pas du tout en contradiction avec le paradigme fonctionnel lui même.

L' un des objectifs de Miró consiste à explorer la possibilité de "concilier" le monde objet et le monde fonctionnel, tout en gardant l'esprit de l'une et l'élégance mathématique de l'autre.

3.3 La théorie de la réécriture

Mots clés : Réécriture, ρ-calcul, sémantique, théorie des types, ELAN.

Glossaire :

ρ-calcul Calcul de réécriture [Cir00] à la base du système ELAN [BKK+98].

Un intérêt récent de Miró est l'étude du ρ-calcul, un calcul de réécriture développé au sein du projet Protheo. Nous souhaitons utiliser le ρ-calcul pour formaliser et intégrer le paradigme à objets et la réécriture. Le ρ-calcul intègre les propriétés complémentaires de la réécriture du premier ordre et du λ-calcul ainsi que des caractéristiques permettant d'exprimer le non-déterminisme. Ce calcul est suffisamment puissant pour décrire non seulement la réécriture avec des règles conditionnelles mais aussi leur contrôle. Ainsi, le ρ-calcul permet la description des langages basés sur la réécriture. La syntaxe du ρ-calcul est très simple :

$$\begin{array}{lll} t & ::= & a \mid X \mid t \to t \mid t \bullet t \mid \qquad \text{termes simples} \\ & & null \mid t, t \qquad\qquad \text{séquences} \end{array}$$

où a dénote une constante, X dénote une variable, $t \to t$ une fonction, $t \bullet t$ une application, $null$ dénote l'ensemble vide, et t, t une séquence. Dans le ρ-calcul, l'application est une opération décrite au même niveau de calcul que l'abstraction, c'est-à-dire les règles de réécriture. Nous obtenons ainsi un calcul similaire au λ-calcul avec motifs de S. P. Jones [PJ87] où l'abstraction est faite non seulement par rapport à une variable mais en considérant aussi le contexte de la variable.

Le ρ-calcul peut être employé pour donner une sémantique à l'application des règles du langage ELAN [BKK+98], un environnement de programmation et de prototypage reposant sur la logique de réécriture associée à la notion de stratégie. Le ρ-calcul est un excellent candidat pour étudier l'expression du paradigme à objets en un calcul de réécriture. Nous en faisons un de nos objectives ; en fait, l'interprétation dénotationnelle de l'appel d'une méthode m sur un objet obj dans la programmation à objet (Kamin [13]) peut être simplement répresenté en ρ-calcul en terme de filtrage et application. La figure 1 présente, par exemple, le codage en ρ-calcul d'un simple objet $Point$ et une simple classe $PointClass$ [1] (on présente soit un codage fonctionnel, soit un codage impératif, grâce au ρ-term $kill_m$, qui sélectionne une méthode dans l'objet et l'élimine).

[Cir00] H. Cirstea, *Calcul de Réécriture : Fondements et Applications*, Thèse de Doctorat, Université Henri Poincaré - Nancy I, 2000.

[BKK+98] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, « An Overview of ELAN », *in : Proc. of WRLA*, *15*, Electronic Notes in Theoretical Computer Science, 1998.

[PJ87] S. Peyton-Jones, *The implementation of functional programming languages*, Prentice Hall, Inc., 1987.

$$
\begin{array}{rcl}
Point & \triangleq & val \to S \to v(1\ 1), \\
 & & get \to S \to S.val, \\
 & & set \to S \to v(X\ Y) \to (S.val := S' \to v(X\ Y)) \\
PClass & \triangleq & new \to S \to (val \to S' \to (S.preval)\bullet S', \\
 & & \qquad\qquad get \to S' \to (S.preget)\bullet S', \\
 & & \qquad\qquad set \to S' \to (S.preset)\bullet S'), \\
 & & preval \to S \to S' \to v(1\ 1), \\
 & & preget \to S \to S' \to S'.val, \\
 & & preset \to S \to S' \to v(X\ Y) \to (S'.val := S'' \to v(X\ Y)) \\
 & \text{où} & \\
obj.m & \triangleq & obj\bullet m\bullet obj \quad \text{Kamin } \textit{self-application} \\
(a.m := b) & \triangleq & (m \to b, a) \ \textit{update} \text{ fonctionnel} \\
(a.m := b) & \triangleq & (m \to b, kill_m\bullet a) \ \textit{update} \text{ impératif grâce à } kill_m \\
kill_m & \triangleq & (X, m \to Z, Y) \to X, Y \\
 & \text{et} & \\
PClass.new & \mapsto\!\!\!\!\!\to & Point
\end{array}
$$

FIG. 1 – Codage d'un objet *Point* et d'une classe *PointClass* en ρ-calcul.

3.4 Les preuves formelles et logiciel certifié

Mots clés : Preuve formelle, logiciel certifié, Calcul des Constructions (CC).

Glossaire :
Calcul des Constructions (CC) Système de types dépendants, polymorphes et d'ordre supérieur pour le λ-calcul, introduit par T. Coquand et G. Huet [CH88].

Dans le domaine du logiciel certifié, on distingue en général la validation d'outils généraux sur les langages de programmation (tels que des interpréteurs, ou des compilateurs), de la validation des programmes eux-mêmes.

Dans le premier cas, on parle de *preuve de propriétés de langages*. Deux exemples typiques de ce domaine sont la preuve de conservation des types d'un langage lors de son exécution, et la preuve de correction d'un compilateur. La première propriété assure une certaine cohérence entre le système de vérification de type et les règles d'évaluation d'un langage. Ce type de preuve, essentiel pour les concepteurs du langage, donne par ailleurs au programmeur une certaine confiance dans les outils qu'il utilise.

La *preuve de correction de programmes* consiste généralement à prouver certaines propriétés d'un programme, typiquement la validité d'un invariant. Cependant, dans le cas idéal où l'on

[CH88] T. COQUAND, G. HUET, « The Calculus of Constructions », *Information and Computation 76*, 2/3, 1988, p. 95–120.

peut extraire un programme à partir de sa spécification, la preuve de correction de programmes devient *programmation certifiée*.

Ce thème de recherche est en plein essor, l'informatique étant de plus en plus utilisée dans des domaines critiques pour la vie humaine (comme le nucléaire, l'industrie aéronautique ou ferroviaire), ou bien dans des domaines où les enjeux financiers sont importants (comme le spatial, les télécommunications ou le domaine bancaire). Les preuves demandées sont encore très peu automatisées. L'avant-projet Miró veut investir dans ce domaine de recherche stratégique.

Les éléments logiciels présents dans les appareils modernes ont une importance cruciale pour le bon fonctionnement de ces appareils. Dans de nombreux cas, il n'est plus possible de recourir uniquement à des campagnes de tests pour assurer leur correction et des techniques de programmation certifiée doivent être développées et mises à la portée des ingénieurs.

De nombreuses équipes de recherche mettent au point des techniques qui permettent de développer des logiciels validés vis-à-vis de spécifications, ce qui permettrait théoriquement de réduire le risque d'erreur à zéro (en pratique des erreurs peuvent également se glisser dans les spécifications). Ces techniques sont plus ou moins coûteuses et plus ou moins générales. Avec le logiciel Coq, fondé sur une théorie typée d'un grand pouvoir d'expression, on se trouve à un bout du spectre : on peut aborder des problèmes variés et complexes mais avec un coût de développement très important [2]. Nos recherches visent à diminuer ce coût, d'une part par la recherche de théories typées et de méthodes de formalisation permettant des spécifications et des preuves plus concises, d'autre part par le développement de bibliothèques d'exemples.

3.5 Les assistants à la preuve

Mots clés : Coq, PCoq, PVS, HOL.

Glossaire :

Coq Assistant à la preuve fondé sur le Calcul des Constructions (co-)Inductives [HKPM97].

PCoq Interfaces graphiques pour Coq permettant le *proof by pointing* c'est-à-dire utiliser la structure des formules logiques pour aider l'utilisateur à effectuer les étapes du raisonnement en les désignant à la souris [BT98,LBRP99].

PVS Système de vérification consistant d'un langage de spécifications et d'un assistant à la preuve [SOR93].

HOL Assistant de preuves fondé sur la logique d'ordre supérieur [oCCL93].

[2] Les techniques de *Model Checking* sont à l'autre bout du spectre.

[HKPM97] G. Huet, G. Kahn, C. Paulin-Mohring, « The Coq Proof Assistant - A tutorial, Version 6.1 », *rapport technique n° 204*, INRIA, 1997, Version révisée distribuée avec Coq. `http://coq.inria.fr/`, `http://www.inria.fr/rrrt/rt-0204.html`.

[BT98] Y. Bertot, L. Théry, « The CtCoq System: Desing and Architecture », *rapport technique n° 3540*, INRIA, 1998, `http://www.inria.fr/rrrt/rr-3540.html`.

[LBRP99] P. Lequang, Y. Bertot, L. Rideau, L. Pottier, « The Pcoq System », *rapport technique*, INRIA, 1999, `http://www-sop.inria.fr/lemme/pcoq/`.

[SOR93] N. Shankar, S. Owre, J. M. Rushby, *PVS Tutorial*, Computer Science Laboratory, SRI International, Menlo Park, CA, 1993.

[oCCL93] U. of Cambridge Computer Laboratory, *The HOL System*, 1993.

Les assistants à la preuve sont très utilisés depuis ces dernières années pour essayer de démontrer des propriétés très difficiles comme, par exemple, la normalisation forte d'un calcul ou la conservation des types pour un langage de programmation. Depuis peu, on s'intéresse aussi à la production de *logiciels sûrs*. Dans ce domaine, les systèmes basés sur une théorie typée vérifiant l'isomorphisme de Curry-Howard sont, non seulement à priori les plus fiables (puisque leurs fondements sont connus), mais aussi les plus puissants, en terme d'expressivité. Dans le cas où l'on peut extraire un programme à partir de sa spécification, la preuve de correction de programme devient *programmation certifiée*. La spécification d'un langage de programmation se prête bien à être formalisée à l'aide de propriétés mathématiques directement transposables dans les démonstrateurs de théorèmes. La preuve de correction d'un compilateur (qui est lui-même un logiciel) représente l'exemple par excellence de l'utilité des démonstrateurs de théorèmes.

Parmi les assistants à la preuve, l'un des plus prometteurs semble être le système Coq : avec son système de types basé sur les types (co)-inductifs, il permet la spécification des langages (exprimés à l'aide des grammaires inductives) et de leurs systèmes de types. Avec Coq, ont été développées récemment beaucoup de bibliothèques : la preuve des propriétés des programmes impératifs, la modélisation de la *scope extrusion* dans le π-calcul, ainsi que la normalisation forte d'une restriction de Coq lui même (voir le papier "*Coq in Coq*" de B. Barras [Bar96]), etc.

Nous visons l'utilisation du système PCoq, interface homme machine dédiée au système Coq, développée dans le projet Lemme. PCoq est la version actuelle du système CtCoq, développée avec une technologie Java. Il permet le *proof by pointing* c'est-à-dire que l'environment utilise la structure des formules logiques pour aider systématiquement l'utilisateur à effectuer les étapes de base du raisonnement en les désignant à la souris. On focalisera notre intérêt :

- sur les aspects théoriques, voir fondateurs, de ces outils [vBLRU97,SDF01] [22, 23],
- sur les aspects d'utilisation de ces outils, car l'un des objectifs principaux de Miró est la construction d'un compilateur pour un langage à objets sûr et certifié.

3.6 FunTalk et SmallTalk2K

Participant : Avant-projet Miró.

Mots clés : SmallTalk, FunTalk, Eiffel, Self, LCO, OC.

Nous avons pour objectif de définir un nouveau langage à objets inspiré à la fois du langage Eiffel pour ce qui concerne les aspects génie logiciel, et des langages à prototypes pour ce qui concerne les aspect formels. En outre, les aspects dynamiques du langage SmallTalk nous semblent appropriés pour augmenter l'expressivité du langage.

Il est bien connu que le langage SmallTalk a été le premier espace de travail ou *système* complètement orienté vers les objets. SmallTalk était plus qu'un langage puisqu'il avait déjà

[Bar96] B. Barras, « Coq in Coq », *Rapport Technique n°3026*, INRIA, Roquencourt, 1996, `http://www.inria.fr/rrrt/rr-3026.html`.

[vBLRU97] S. van Bakel, L. Liquori, S. Ronchi della Rocca, P. Urzyczyn, « Comparing Cubes of Typed and Type Assignment System », *Annals of Pure and Applier Logics 86*, 3, 1997, p. 267–303.

[SDF01] C. Schürmann, J. Despeyroux, F.Pfenning, « Primitive Recursion for Higher-Order Abstract Syntax », *Theoretical Computer Science 266*, 1-2, 2001, p. 1–57.

un environnement composé d'un *Browser de classes* (pour "naviguer" dans les librairies), un *Workspace* (pour tester les applications), et un *Debugger*, eux-mêmes écrits en SmallTalk !

Beaucoup de concepts de SmallTalk ont été repris dans d'autres langages comme Java, C++ et Eiffel par exemple. Malheureusement et heureusement, cela dépend du point de vue dans lequel on se place, le langage SmallTalk est un langage interprété, donc lent, mais aussi très expressif, car il n'est pas typé. Il est largement utilisé dans les projets où la vitesse d'exécution n'est pas cruciale comme, par exemple, dans le prototypage d'applications.

Depuis plus d'une dizaine d'années, plusieurs équipes ont essayé de trouver un système de typage sûr pour SmallTalk sans trop restreindre l'expressivité du langage. Dans la théorie des types, on connaît très bien l'équation :

- langages typés : pas expressifs mais sûrs ;
- langages non typés : très expressifs mais pas sûrs.

Par certains côtés, Eiffel est le langage qui se rapproche le plus de SmallTalk du point de vue de son expressivité même si son système des types n'est pas sûr.

3.6.1 Les caractéristiques du langage SmallTalk2K

L'absence de typage statique confère une très grande expressivité au langage SmallTalk : il permet la création dynamique de classes et la définition réflexive du système. On souhaite conserver une grande partie de l'expressivité de SmallTalk dans le langage SmallTalk2K. Bien sûr, le système de types que nous allons définir a pour objectif de rejeter statiquement certains *mauvais* programmes (voir *e.g.* celui de la figure 2, écrit en Eiffel). Notre objectif consiste à conserver les acquis du langage Eiffel avec un système de typage sûr.

Puisque le système de types sera fortement inspiré du travail de recherche sur les langages à prototypes, on pourra ajouter la modification dynamique des instances comme dans les langages à délégations. Cette possibilité permettra, par exemple, d'écrire en SmallTalk2K l'affectation suivante (on utilise une syntaxe à la Java et on assume une simple classe `POINT`) :

```
point = new POINT(2,4);
point.set(int dx,int dy) = {new body pour set};
point.set(3,5);
```

Cela correspond à changer dynamiquement le comportement d'une seule instance de la classe `POINT` qui, après l'affectation de `set`, aura un comportement pour la méthode `set` *différent* de toutes les autres instances de `POINT`. Une telle caractéristique est déjà présente dans des langages à prototypes comme Self par exemple, et a été récemment explorée dans le cadre du langage Java[DDDCG01].

Nos objectifs ici sont de concevoir la sémantique et un compilateur d'un langage *intermédiaire* à prototypes, fortement typé, appelé FunTalk. La définition de SmallTalk2K pourra être exprimée à travers ce langage. Un programme écrit en SmallTalk2K pourrait être pré-compilé vers un programme en FunTalk équivalent en utilisant, par exemple, les transformations en *continuation passing style (CPS)* (voir par exemple [Rep00]).

[DDDCG01] F. Damiani, S. Drossopoulou, M. Dezani-Ciancaglini, P. Giannini, « Fickle: Dynamic Object Reclassification », *in : Proc. of ECOOP, LNCS*, Springer Verlag, 2001.

[Rep00] J. Reppy, « Local CPS conversion in a direct style compiler », *rapport technique*, Bell Labs, Lucent Technologies, 2000.

Le langage SmallTalk2K pouvant être vu comme une couche à classes au dessus du langage bas niveau à prototypes FunTalk , nous envisageons la possibilité de rendre disponible dans SmallTalk2K quelques primitives propres au langage FunTalk (par exemple la modification ou l'extension dynamique des méthodes dans un objet, c'est-à-dire dans une instance de classe). La sémantique statique et dynamique de FunTalk pourra être inspirée des calculs à prototypes OC et LCO en version impérative.

3.6.2 Phase formelle de conception de FunTalk/SmallTalk2K

Le phase formelle de la construction d'un langage (avec son compilateur) sûr et certifié est un objectif très ambitieux.

On prévoit les étapes suivantes :

- étude des langages Ocaml, Haskell, Obliq, Self, Kevo, Emerald, Cecil, et Moby, ainsi que des calculs O_1, O_2, O_3 basées sur OC [1], et LCO [11] (à la base de Moby) ;
- description d'une syntaxe, d'une sémantique à réécriture et d'une stratégie d'évaluation pour le langage FunTalk ;
- description d'un système de types pour FunTalk inspiré des systèmes de types pour les calculs LCO et OC. Ce système devra prévoir la *mytype method specialisation*, c'est-à-dire le `like Current` d'Eiffel, une solide notion de *subtyping* et/ou *matching* [4] [AC95] et la possibilité de typer statiquement les méthodes binaires ;
- démonstration du théorème de conservation des types et de complétude *i.e.* le slogan de Milner (*"well-typed programs cannot go wrong"*) pour FunTalk ;
- description d'une syntaxe pour le langage SmallTalk2K et sa traduction en langage intermédiaire FunTalk via des transformations en *continuation passing style*.

Pendant le développement théorique du langage FunTalk, on étudiera la possibilité de formaliser et de prouver les théorèmes fondamentaux, *e.g.* conservation des types et complétude, avec l'assistant de preuves Coq. L'extraction de programmes à partir de preuve en Coq devrait nous aider à la production d'un premier compilateur FunTalk certifié.

3.6.3 Phase de conception et d'implantation

Nous prévoyons les étapes suivantes pour la construction des compilateurs et environnements pour SmallTalk2K et FunTalk :

- construction (ou extraction si possible) d'un premier compilateur pour FunTalk et/ou d'une éventuelle machine virtuelle capable d'interpréter le code FunTalk compilé ;
- auto-compilation du compilateur pour FunTalk. A partir d'un compilateur pour FunTalk (écrit dans un langage quelconque ou extrait par un assistant de preuve type Coq, pas forcément le plus efficace mais, on l'espère, certifié "sûr" [Ber98]) on construira un compilateur optimisant pour FunTalk ;
- compilation du langage SmallTalk2K en FunTalk ;

[AC95] M. Abadi, L. Cardelli, « On Subtyping and Matching », *in : Proc. of ECOOP, LNCS, 952*, Springer Verlag, p. 145–167, 1995.

[Ber98] Y. Bertot, « A certified compiler for an imperative language », *rapport technique n° RR-3488*, INRIA, 1998, `http://www.inria.fr/rrrt/rr-3488.html`.

– évaluation du langage et de son environnement sur des programmes non triviaux.

3.7 Efficacité et sûreté du compilateur SmallEiffel

Mots clés : Eiffel, typage sûr, covariance, liaison dynamique, analyse globale, prédiction de type, auto-compilation.

Participants : Dominique Colnet, Olivier Zendra.

Glossaire :

Typage sûr Dans ce contexte, le fait que les indications de types écrites dans le programme soient respectées lors de l'exécution.

Analyse globale Le fait de considérer le programme dans son ensemble (*whole system analysis*) par opposition à la compilation séparée.

Auto-compilation Compilation du compilateur en utilisant le texte source du compilateur lui-même (*bootstrap*).

Si l'on fait abstraction du *principe de cohérence globale* de B. Meyer [Mey92], qui n'est implanté dans aucun compilateur Eiffel, on peut dire que le langage Eiffel n'est pas un langage avec un système de types sûr. Considérons par exemple le programme Eiffel de la figure 2. Dans cet exemple on définit deux classes `POINT` et `PIXEL` qui possèdent une méthode `is_equal` spécialisée dans `PIXEL`. La classe `MAIN` crée deux instances de `POINT` et une de `PIXEL`. L'affectation `p := r` est acceptée par le compilateur puisque `r` qui est de type `PIXEL` peut être aussi considérée comme une expression de type `POINT`. Malheureusement, l'appel `p.is_equal(q)` produit une erreur lors de l'exécution car c'est la version spécialisée de `is_equal`, définie dans `PIXEL`, que l'on exécute (liaison dynamique oblige). En effet, cette redéfinition suppose que l'argument `q` possède la méthode `status`. Comme le type dynamique de `q` est `POINT`, l'invocation de `status` avec une instance de `POINT` provoque une erreur. Le même problème, induit en fait par la redéfinition covariante de `is_equal`, se produit également à l'issu d'un passage d'argument (fonction `breakit`) (voir aussi le papier "*On Binary Methods*" [BCC+96]).

Dans [7], D. Colnet et L. Liquori ont défini un dialecte d'Eiffel, appelé Match-O en raison de l'ajout d'un nouveau type appelé "type `match`". Match-O n'a pas encore une sémantique formelle, mais son nouveau système de types, inspiré des travaux de K. Bruce *et alt.* [BSvG95, Bru97,BCC+96,Bur98] [3, 4] *rejette* le programme (erroné) de la figure 2 lors de la compilation. Ainsi, dans Match-O, on remplace systématiquement le type `like Current` par le type `match Current`. Intuitivement, le type `match Current`, permet d'exprimer le fait qu'une entité est exactement du même type que le receveur au regard du principe de liaison dynamique. Cette

[Mey92] B. Meyer, *Eiffel, The Language*, Prentice Hall, Englewood Cliffs, 1992.

[BCC+96] K. Bruce, L. Cardelli, G. Castagna, T. H. O. Group, G. Leavens, B. Pierce, « On Binary Methods », *Theory and Practice of Object Systems 1*, 3, 1996.

[BSvG95] K. Bruce, A. Shuett, R. van Gent, « PolyTOIL: a Type-Safe Polymorphic Object Oriented Language », *in: Proc. of ECOOP, LNCS, 952*, Springer Verlag, p. 27–51, 1995.

[Bru97] K. Bruce, « Increasing Java's expressiveness with `ThisType` and Match-bounded Polymorphism », *rapport de recherche*, Williams College, 1997.

[Bur98] J. Burstein, ***Rupiah:*** *an Extension to Java Supporting Match-bounded Parametric Polymorphism,* ***ThisType****, and Exact Typing*, Bachelor thesis, Williams College, 1998.

```
class POINT inherit ANY redefine is_equal end ;
creation set
feature
   x : REAL ; y : REAL ;
   set(a : real, b : real) is
      do
         x := a ; y := b ;
      end ;
   is_equal(other : like Current) : BOOLEAN is
      do
         Result := (x = other.x) and (y = other.y) ;
      end
end

class PIXEL inherit POINT redefine is_equal end ;
creation set
feature
   status : BOOLEAN ;
   set_status(s : BOOLEAN) is
      do
         status := s ;
      end ;
   is_equal(other : like Current) : BOOLEAN is
      do
         Result := (x = other.x) and (y = other.y)
                   and (status = other.status) ;
      end ;
end
```

```
class MAIN creation main
feature
   p : POINT ; q : POINT ; r : PIXEL ;
   main is
      do
         !!p.set(1,1) ; -- New POINT in variable p
         !!q.set(2,2) ; -- New POINT in variable q
         !!r.set(3,3) ; -- New PIXEL in variable r
         p := r ; -- Accepted because of subtyping.
         if p.is_equal(q)   -- CRASH ! (1)
         then ... else ... end ;
         breakit(r,q) -- Accepted because of subtyping.
      end ;
   breakit(p1, p2 : POINT) is
      do
         if p1.is_equal(p2) -- CRASH ! (2)
         then ... else ... end ;
      end ;
end
```

FIG. 2 – Un programme Eiffel qui laisse apparaître un problème de type lors de l'exécution.

nouvelle notation de type, que nous proposons d'ajouter dans Eiffel, permet à la fois de mieux décrire le type des entités manipulées mais aussi d'obtenir des programmes plus sûr. Une étude complète de l'intégration du type `match` par rapport à la définition de classes génériques est en cours. Nous pensons également tirer partie de l'expérience acquise en Eiffel pour la définition et la validation du langage SmallTalk2K/FunTalk.

3.7.1 Le principe d'analyse globale de SmallEiffel

Afin de *spécialiser* l'ensemble du programme, c'est à dire générer du code adapté autant que possible au contexte spécifique au programme compilé [5], nous avons adopté pour SmallEiffel une *approche globale* au système. Nous nous appuyons ainsi sur des algorithmes d'analyse globale, de prédiction de type et d'expansion en ligne (*inlining*) puissants [CCZ97] [17, 19, 20, 21].

L'*analyse globale* consiste à rassembler des informations sur l'ensemble du système compilé. Ceci va à l'encontre des pratiques actuellement les plus courantes pour la compilation des langages à objets, qui reposent sur la compilation séparée de fichiers sources indépendants, dont le principal inconvénient est qu'elle rend l'optimisation globale plus difficile.

L'analyse globale se focalise sur l'optimisation du système dans son ensemble. Elle a pour but de permettre au compilateur de préciser le *contexte* dans lequel chaque élément du code source (expression, instruction, variable, méthode, etc.) va être compilé. Ceci permet de passer

[CCZ97] S. COLLIN, D. COLNET, O. ZENDRA, « Type Inference for Late Binding. The SmallEiffel Compiler. », *in : Joint Modular Languages Conference, LNCS, 1204*, Springer Verlag, p. 67–81, 1997.

de la génération d'un code le plus général possible à la production de code plus spécialisé, et donc potentiellement plus efficace. Afin d'avoir une analyse globale aussi rapide que possible, il nous parait important de ne pas compiler tout le code source accessible (y compris dans les bibliothèques) sans discrimination, mais seulement le code indispensable au système, c'est à dire le code *vivant* de ce système.

En Eiffel, le point de départ d'un programme, appelé la *racine* de l'application, est spécifié par une paire composée d'une classe initiale et de l'une de ses procédures de création. La première étape de notre processus d'analyse statique globale calcule quelles parties du code source Eiffel sont *vivantes* (accessibles depuis la racine) ou *mortes* (inaccessibles). Ceci s'effectue pour l'instant dans SmallEiffel de façon totalement statique, sans analyse de flot. Le résultat de ce premier calcul est donc indépendant de l'ordre des instructions dans le programme.

Le processus de calcul du code vivant ainsi que de l'ensemble des types vivants est à la fois récursif et itératif. La récursivité est utilisée pour suivre le graphe d'appel de l'application. Le processus est répété itérativement en partant des méthodes vivantes de chaque type vivant jusqu'à être capable d'effectuer une itération complète sans ajouter ni nouveau type vivant ni nouvelle méthode vivante.

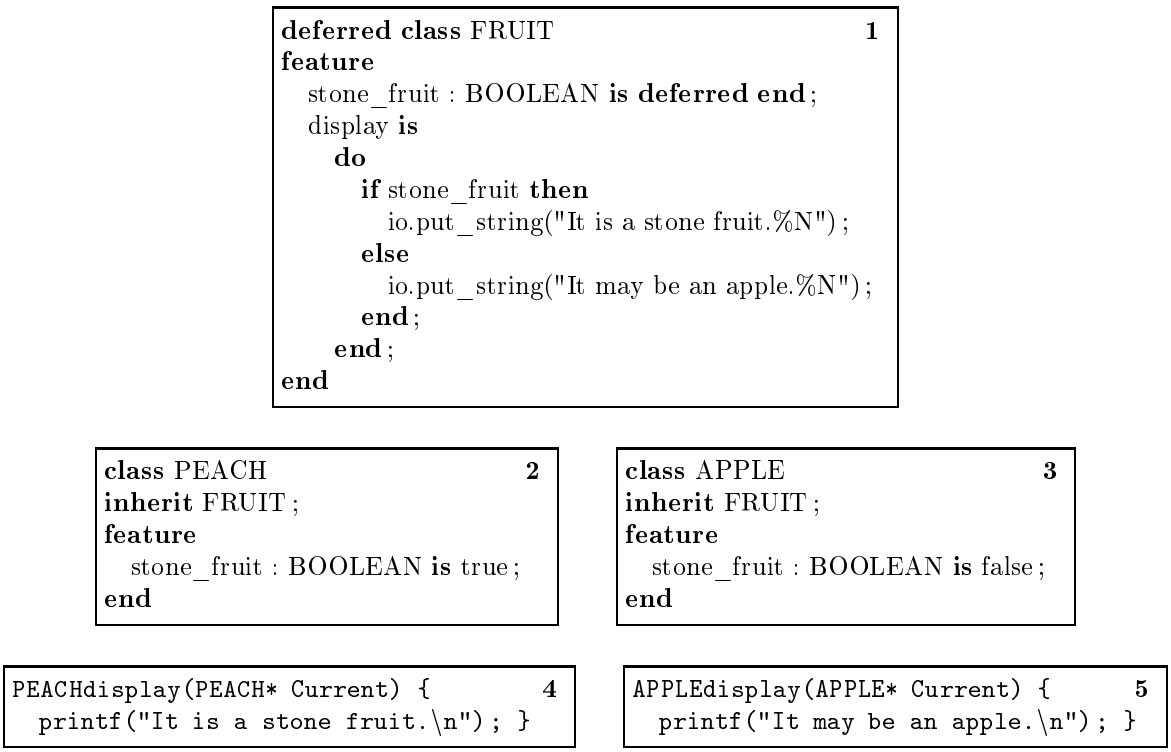
En fait, l'algorithme actuel de calcul du code vivant, même s'il donne des résultats plus qu'acceptables, n'est pas parfait. Par exemple, pour un appel `foo.bar`, lorsque le type statique FOO de `foo` ne fait pas partie des types vivants, on ne devrait pas considérer la définition de `bar` dans ce type comme vivante car, dans certains cas, ce code peut amener l'algorithme à considérer à tort qu'un autre type est vivant. Cette constatation fait apparaître un besoin criant de pouvoir *garantir* que le calcul du code vivant est correct. Une de nos perspective de recherche importante dans ce domaine est de formaliser le processus de calcul du code vivant dans le but de converger vers une nouvelle implantation approchant plus précisément la partie réellement vivante de l'application.

3.7.2 Spécialisation des méthodes vivantes selon le type du receveur

La spécialisation automatique du code en fonction du type du receveur est un point essentiel dans la stratégie de compilation de SmallEiffel [9]. Cette spécialisation consiste en quelque sorte à essayer de dérouler avant l'exécution le mécanisme de liaison dynamique, ce qui peut être considéré comme une forme d'évaluation partielle.

La figure 3 montre comment la méthode `display`, définie dans la classe FRUIT, héritée dans les classes PEACH et APPLE est automatiquement spécialisée lors de la traduction en code C. En effet, comme le prédicat `stone_fruit` prend la valeur vrai, dans la classe PEACH et la valeur faux dans la classe APPLE, l'expression conditionnelle est toujours vraie dans la classe PEACH et toujours fausse dans la classe APPLE. Ainsi, il ne subsiste aucune trace de l'expression conditionnelle au niveau du code généré. En outre, dans le véritable code généré par SmallEiffel, l'argument `Current` (`self` de SmallTalk ou `this` de Java) des fonctions `PEACHdisplay` et `APPLEdisplay`, destiné à référencer le receveur, est supprimé car non utilisé dans le code spécialisé.

Ainsi, toutes les méthodes vivantes d'un type vivant aussi sont spécialisées. En raison de la duplication-spécialisation du code dans tous les types vivants, la pseudo variable `Current` a toujours un seul et unique type dynamique, le type vivant dans lequel cette méthode est



```
deferred class FRUIT                                        1
feature
  stone_fruit : BOOLEAN is deferred end;
  display is
    do
      if stone_fruit then
        io.put_string("It is a stone fruit.%N");
      else
        io.put_string("It may be an apple.%N");
      end;
    end;
end
```

```
class PEACH                                                 2
inherit FRUIT;
feature
  stone_fruit : BOOLEAN is true;
end
```

```
class APPLE                                                 3
inherit FRUIT;
feature
  stone_fruit : BOOLEAN is false;
end
```

```
PEACHdisplay(PEACH* Current) {                              4
  printf("It is a stone fruit.\n"); }
```

```
APPLEdisplay(APPLE* Current) {                              5
  printf("It may be an apple.\n"); }
```

FIG. 3 – Spécialisation du code généré pour la méthode `display` définie dans la classe FRUIT. Les cadres **1**, **2** et **3** correspondent respectivement au source Eiffel des classes FRUIT, PEACH et APPLE. Les cadres **4** et **5** montrent le code C généré pour les classes PEACH et APPLE.

spécialisée. Tous les appels concernant `Current` sont donc des appels directs ne nécessitant *jamais* de code de liaison dynamique.

Le receveur étant en général l'objet le plus utilisé dans une méthode, cette optimisation concerne une très grande partie des envois de messages. Pour cette raison, les scores d'inférence de type atteignent des niveaux très élevés (de l'ordre de 90%).

Contrairement à ce que l'on pourrait craindre, cette technique de duplication-spécialisation des méthodes n'entraîne pas d'augmentation importante de la taille des exécutables. En fait, les exécutables produits par SmallEiffel sont souvent plus petits que ceux produits par les compilateurs du commerce, voire plus petits que les exécutables obtenus à partir de classes C++ (cf. par exemple [CCZ97]). Ceci est dû en partie au fait que la spécialisation des méthodes tend à élaguer du code, comme dans l'exemple de la figure 3. De plus, l'implantation de la liaison dynamique à l'aide de code de sélection dichotomique (*Binary Tree Dispatch*) permet le remplacement direct (*inlining*) de nombreuses méthodes. En particulier, toutes les fonctions d'écriture et de lecture d'attributs ne sont jamais ni appelées, ni même définies [17].

La technique de duplication-spécialisation est particulièrement bien adaptée à un contexte d'analyse globale où l'on ne considère que le code vivant. En particulier, le fait de travailler

[CCZ97] S. COLLIN, D. COLNET, O. ZENDRA, « Type Inference for Late Binding. The SmallEiffel Compiler. », *in : Joint Modular Languages Conference, LNCS, 1204*, Springer Verlag, p. 67–81, 1997.

sur un système fermé permet de spécialiser le code de toutes les *classes feuilles*, les classes sans sous-classes dans le système fermé. Plus généralement, nous appelons *type feuille* un type vivant sans sous type vivant dans le système fermé. Du point de vue spécialisation de code, le fait que le type d'une expression soit un type feuille n'est pas sans importance : comme dans le cas de `Current`, un envoi de message appliqué sur une expression dont le type est un type feuille correspond à un appel direct, sans liaison dynamique. La seule différence entre `Current` et une expression de type feuille consiste à envisager également le cas où l'expression ne référence aucun objet lors de l'exécution (`void`).

3.7.3 Optimisation et étude des interactions matériel-logiciel dans les langages à objets

Dans le cadre de son post-doctorat au sein de l'*Adaptive Computation Laboratory* de McGill University à Montréal, O. Zendra mène des recherches consistant globalement à étudier les interactions entre logiciels et matériel afin d'optimiser au mieux des programmes écrits dans des langages à objets. Ces recherches se focalisent plus particulièrement sur l'optimisation des programmes Java et les interactions entre Java Virtual Machine et processeur [30].

En effet, des échanges scientifiques enrichissants avaient déjà pu avoir lieu à diverses reprises entre, des membres de l'ACL (notamment le Prof. K. Driesen), ainsi que, dans une moindre mesure, des membres du Compilers and Concurrency Laboratory, anciennement ACAPS (Prof. L. Hendren) de McGill University, d'une part, et notre groupe développant SmallEiffel, le compilateur GNU Eiffel, au sein du projet ECOO puis Miró[3] de l'INRIA Lorraine, d'autre part. Ces échanges, bien que de qualité, avaient été généralement limités pour des raisons de contraintes temporelles, à des conférences et visites, et à des courriers électroniques. Ils avaient cependant fait apparaître très clairement la communauté de nos centres d'intérêts (compilation, optimisation, liaison dynamique) et la complémentarité de nos compétences et approches, les chercheurs de l'ACL de McGill se concentrant plus particulièrement sur le bas ou très bas niveau (architecture matérielle) alors que nous nous focalisons sur un niveau plus haut, lié à la sémantique des programmes.

3.8 Veille scientifique

Mots clés : Eiffel, Calculs pour la mobilité.

Participant : Avant-projet Miró.

3.8.1 Les langages concurrents et à objets

Comme d'autres projets de l'INRIA (*e.g.* Mimosa, Moscova, ...), l'avant-projet Miró s'intéresse aux calculs qui modélisent la mobilité, le réseau et ses problèmes de sécurité. Récemment, dans la littérature scientifique, on a assisté à la naissance d'une *plethora* de calculs comme, par

[3] Les participants de l'avant-projet Miró sont issus du projet ECOO.

exemple, les Ambients de Cardelli et Gordon [CG00,CGG99], le Join-calcul de Fournet, Gonthier, Levy, Maranget et Remy [FGL+96], le Blue-calcul de Boudol [Bou97], le Spi-calcul de Abadi et Gordon [MA97], et le Pi-calcul de Milner [MPW92]. En outre, des travaux récents ont montré comment le paradigme à objets et ses concepts de base peuvent facilement s'intégrer dans ces nouveaux calculs. On cherchera à être "à la page" par rapport à ces sujets de recherche et à ce que, éventuellement, les concepts de mobilité, réseaux et sécurité puissent s'intégrer d'une façon naturelle dans les calculs et/ou langages auxquels on s'intéresse.

3.8.2 L´évolution du langage Eiffel

La définition du langage Eiffel est encore en cours d'évolution. L'avant-projet Miró s'intéresse à cette évolution aussi bien au niveau des décisions de modification qu'au niveau de l'implantabilité de ces choix. La nouvelle définition prévoit l'ajout d'un nouveau type de base, type *Tuples*, qui donne la possibilité de manipuler des N-uplets typés de valeurs. Avec le type *Tuples*, il sera possible de typer plus strictement les fonctions ayant plusieurs résultats ainsi que les fonctions à nombre variable d'arguments. La méthode d'analyse globale devrait permettre de spécialiser chaque sorte de *Tuples* vivant. Dans la nouvelle définition d'Eiffel, un *agent* est, en quelque sorte, comparable à une fermeture, *closure*, d'un langage fonctionnel. L'objectif de cet ajout consiste essentiellement à pouvoir augmenter l'expressivité dans les assertions. Les agents permettent également une certaine forme d'évaluation partielle, comme dans les langages fonctionnels. De nouvelles possibilités d'introspection sont également en projet. Ces dernières n'étant que consultatives (*i.e.* combien de méthodes dans une classes, type des arguments d'une méthode, etc.), cela ne devrait pas remettre en cause l'hypothèse du système fermé qui est un point clef dans la stratégie de compilation de SmallEiffel.

3.9 Programme de travail

Compte tenu des effectifs actuels de l'avant-projet, bien conscients de nos limites, nous envisageons le programme de recherche suivant :

- à court terme, c'est à dire sur environ 2 ans :
  - description complète de la syntaxe, de la sémantique statique et dynamique du langage FunTalk ; démonstration de propriétés fondamentales sémantiques sur papier et début de leur formalisation dans un assistant de preuve ;

[CG00] L. Cardelli, A. D. Gordon, « Mobile Ambients », *Theoretical Computer Science 240*, 1, 2000.

[CGG99] L. Cardelli, G. Ghelli, A. D. Gordon, « Mobility Types for Mobile Ambients », *in : Proc. of ICALP*, *LNCS*, 1644, Springer Verlag, p. 230–239, 1999.

[FGL+96] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, D. Rémy, « A Calculus of Mobile Agents », *in : Proc. of CONCUR*, *LNCS*, *1119*, Springer Verlag, p. 406–421, 1996.

[Bou97] G. Boudol, « The π-Calculus in Direct Style », *in : Proc. of POPL*, The ACM Press, p. 228–241, 1997.

[MA97] A. D. G. M. Abadi, « A Calculus for Cryptographic Protocols: The Spi Calculus », *in : Proc. of ACM Conference on Computer and Communications Security*, The ACM Press, p. 36–47, 1997.

[MPW92] R. Milner, J. Parrow, D. Walker, « A Calculus of Mobile Processes I and II », *Information and Computation 100*, 1, 1992, p. 1–77.

  - construction d'un compilateur et peut-être d'une machine virtuelle jouet pour FunTalk (en utilisant *e.g.* SmallEiffel ou Self) ; syntaxe du langage SmallTalk2K ;
  - étude, à travers l'analyse globale à la base du compilateur SmallEiffel, de la possibilité de résoudre le problème de sécurité du langage Eiffel sans modifier la définition du langage ;
  - méta-théorie du ρ-cube et étude des différentes méthodes de formalisation et de preuves des langages et calculs à objets ;
  - étude des optimisations du langage Java, notamment en utilisant les techniques développées pour SmallEiffel ; étude de l'impact des architectures matérielle sur les performances des programmes à objets ;
  - maturation du langage à prototypes Lisaac ; étude de la portabilité du système d'exploitation Lisaac sur d'autres processeurs ; portage des produits GNU sur l'OS Isaac ;
- à moyen terme, soit environ 4 ans :
  - réalisation du compilateur et de la machine virtuelle pour FunTalk certifié (et peut-être extrait automatiquement) ; bootstrap d'un compilateur optimisé FunTalk écrit en FunTalk et début de sa certification semi-automatique ; validation de FunTalk avec exemples non triviaux ; certification partielle du compilateur optimisé et/ou de la machine virtuelle FunTalk ;
  - étude d'une logique possible pour le ρ-cube (peut-on envisager un isomorphisme de Curry-Howard ?) ;
  - étude et implantation d'une bibliothèque d'interfaçage homme-machine pour Eiffel, en utilisant le langage Eiffel lui-même et la programmation par contrat ;
  - étude d'optimisations portables au niveau des machines virtuelles Java ;
  - modélisation et prédiction de l'impact des architectures matérielles sur l'exécution des programmes à objets.

4 Faits marquants : actions menées en 2001

4.1 Intégration du projet CertiLab

Suite à l'expertise de J. Despeyroux en tant que relectrice de l'avant-projet Miró, le projet Certilab de l'INRIA Sophia-Antipolis a fusionné avec Miró le 3 décembre 2001. Miró est donc devenu un avant-projet bi-localisé entre Nancy et Sophia.

5 Domaines d'applications

5.1 Panorama

Mots clés : Palm Pilot, Palm-OS, Mobile phones, Mobile-OS, programmation sûre, logiciel certifié, code mobile, internet, télécommunications.

Les domaines concernés sont : les ordinateurs de poche et leurs systèmes d'exploitation, les téléphones mobiles et leurs systèmes d'exploitation, les télécommunications, les applications certifiées, les systèmes d'exploitation en général, le génie logiciel en général, la génération de

code pour les micro-contrôleurs synthétisables, et comme dit précedemment, tous les domaines touchant aux logiciels critiques.

5.2 Électronique embarquée

5.2.1 Palm Pilot et Palm-OS

L'évolution des technologies et du matériel permet aujourd'hui d'intégrer des outils informatiques performants dans des appareils divers et de petites tailles. Nous nous intéressons particulièrement à l'expansion massive des agendas électroniques. Actuellement nous observons un phénomène de diversité comparable à celui du début des ordinateurs personnels. Cette diversité se situe à tous les niveaux de cette nouvelle informatique. Nous constatons une variété au niveau des processeurs, des systèmes d'exploitation et de l'interface utilisateur qui reflètent les différentes possibilités du matériel : commande par clavier, écran tactile ou reconnaissance vocale. Dans ce domaine, deux grandes tendances se degagent aujourd'hui : les applications écrites en langage C pour les performances, ou dans un langage interprété pour la portabilité. La puissance qu'offrent les *palms* est certes impressionnante, mais reste un sujet de préoccupation pour le développeur. Il nous paraît donc indispensable de lier performance du code compilé spécifique à un processeur avec portabilité et standardisation de l'interface de programmation. La sûreté de ce genre d'application paraît être une priorité.

5.2.2 Logiciel Embarqué

Le centre de recherche de Nokia Network développe actuellement un langage de programmation adapté à des applications de télécommunications. Ce langage est essentiellement un langage fonctionnel, appelé *Shines*. Parmi les caractéristiques du langage Shines citons un ramasse-miettes très efficace, un traitement des exceptions, des co-routines, des threads, un typage statique (apparemment) sûr, la possibilité d'accéder au contrôle via opérateur en CPS, et des prototypes dans le style du langage Self. Le système de typage de Shines utilise des types d'ordre supérieur et du sous-typage. Un objet en Shines est représenté par un état et une suite des méthodes.

Les applications de télécommunications ont très souvent besoin de pouvoir modifier le code "à la volée", ce qui est très facilement modélisable avec des langages à prototypes. Nous pensons pouvoir utiliser notre langage FunTalk dans ce type d'applications et nous envisageons une prise de contact avec P. Mäenpää, membre du Working group européen TYPES et *Chief research engineer* chez Nokia Network, Helsinki.

5.3 Systèmes d'exploitation

Mots clés : Isaac, Self, prototype, Architecture à *Blackboards*.

Participants : Dominique Colnet, Benoît Sonntag.

Glossaire :
Isaac Système d'exploitation entièrement à objets dynamiques et prototypes [25, 26].
Lisaac Langage à prototype, conçu pour l'implantation des OS à objets.

Dans l'avant-projet Miró, nous explorons la possibilité d'intégration de concepts objets au coeur même des systèmes d'exploitation. Cette branche de recherche fait l'objet de la thèse de B. Sonntag. L'évolution des technologies et du matériel nous amène à revoir la conception du noyau des systèmes d'exploitation actuels. Nous pensons que l'introduction de concepts objets au coeur même des systèmes d'exploitation simplifierait et optimiserait l'utilisation des divers composants physiques d'un ordinateur. De manière générale, notre démarche consiste à repenser la définition d'un noyau système en intégrant des concepts objets, voir multi-agents.

Nous sommes aujourd'hui en mesure d'affirmer que ce type de démarche apporte un réel progrès aussi bien dans l'adaptation dynamique du système que dans la simplification et le pouvoir expressif du code. De plus, notre approche a des répercussions notoires pour l'utilisateur de ce type de système. L'ultime attente réside en sa capacité de s'adapter, et de s'auto-gérer pour concevoir automatiquement des liens entres les objets (liens de clientèle, mais aussi d'héritage), qui permettront de construire une application dynamiquement selon les actions et les types de données traitées par l'utilisateur à un instant donné.

Pour mener à bien nos objectifs, sans délaisser les performances globales du système, nous avons commencé par étudier les architectures de processeurs (notamment Intel x86). Cette étude avait deux objectifs : permettre une portabilité rapide du système et une utilisation approfondie des capacités du processeur pour la mise en place des mécanismes objets. Parmi les différents concepts des langages à objets existant, nous avons choisi de privilégier les prototypes, et de construire un langage à prototypes, Lisaac, permettant l'élaboration du système.

Parallèlement à ce développement, B. Sonntag a conçu et réalisé avec ce langage un système d'exploitation expérimental appelé *Isaac*, *entièrement objet* et accèdant directement le matériel sans aucune autre couche intermédiaire. Ainsi, il n'utilise le BIOS d'un PC qu'au moment du boot. Isaac possède aussi une interface graphique de haut niveau et pourra supporter l'ensemble des produits GNU via une simple compilation. Il est suffisamment avancé et compact pour servir de base à l'incorporation rapide de nouveau concepts et à l'étude de nombreux problèmes liés à l'interaction entre les objets et les systèmes d'exploitation. De plus, les premières réalisations menées par B. Sonntag semblent montrer qu'un tel système d'exploitation est extrêmement compact et se prête particulièrement bien aux systèmes embarqués.

Enfin, une formalisation du langage propriétaire à la base du système Isaac est en cours d'étude. Nous pensons que ce langage pourrait être formalisé avec une sémantique opérationnelle relativement simple car le langage a une taille raisonnable (une vingtaine de constructeurs).

5.4 Micro-contrôleurs synthétisables

Participant : Dominique Colnet.

Mots clés : Compilation, micro-contrôleurs synthétisable, FPGA, C ANSI.

Ce domaine d'application est en relation avec l'expérience acquise par D. Colnet dans le cadre du logiciel ProMic ; le savoir-faire ainsi accumulé pourra être utile à l'avant-projet Miró pour proposer des applications possibles, écrites en SmallTalk2K/FunTalk, orientées "firmware".

Un micro-contrôleur synthétisable, aussi appelé FPGA est un composant programmable, comportant un grand nombre de portes logiques. Ces portes peuvent être combinées les unes

avec les autres afin de réaliser un composant spécialisé pour une tâche donnée. Le composant peut-être reconfiguré plusieurs fois, parfois même dynamiquement, selon le fabricant de composants.

Les systèmes sur un composant (*System On Chip*) sont désormais une réalité et leur utilisation va se répandre. Cependant, il semble nécessaire de définir de nouveaux outils afin de pouvoir profiter pleinement de cette nouvelle technologie. Le logiciel ProMic constitue une première action de recherche dans cette thématique.

6 Logiciels

6.1 SmallEiffel : The GNU Eiffel Compiler

Participants : Dominique Colnet [Correspondant], Olivier Zendra.

Le résultat principal des recherches effectuées dans le cadre de la thèse de doctorat d'O. Zendra [19], a été le développement et la diffusion d'un compilateur pour le langage Eiffel nommé SmallEiffel. En effet, les buts éminemment pratiques de nos travaux nous ont dès le début amenés à nous poser la question de la validation expérimentale de nos idées. Il nous est fort naturellement apparu qu'une des meilleures façons de le faire consistait à les implanter dans un compilateur créé *ex nihilo*, dont les deux objectifs principaux étaient qu'il soit capable de compiler vite, tout en générant du code efficace. Afin de mettre nos travaux à disposition du plus grand nombre, nous avons choisi de diffuser SmallEiffel comme logiciel libre (`http://SmallEiffel.loria.fr`).

SmallEiffel nous a ainsi servi d'outil expérimental dans lequel nous avons pu implanter nos idées et valider nos hypothèses, ainsi que de vecteur de diffusion auprès de la communauté scientifique et du public. Mais il a aussi constitué un sujet d'observation, un banc d'essai significatif, extrêmement intéressant et plein d'enseignements, facilitant ainsi l'émergence de nouvelles idées. En effet, rappelons que SmallEiffel est complètement auto-compilé (*bootstrapped*) en Eiffel et représente dans sa version `-0.75` environ 110 000 lignes d'Eiffel pour environ 300 classes vivantes, ce à quoi il convient d'ajouter l'ensemble de la série de programmes utilisés pour valider le compilateur, soit environ 2500 classes représentant 250 000 lignes d'Eiffel.

Intégrant les techniques d'optimisation développées dans la thèse d'O. Zendra, SmallEiffel produit un code d'excellente qualité, dont les performances sont en général supérieures (voire très supérieures) à celles des autres compilateurs Eiffel disponibles dans le commerce. Il se compare également très favorablement aux compilateurs C++.

La très bonne qualité du code généré par SmallEiffel permet l'obtention d'exécutables de très petite taille, grâce notamment à la *compilation sur nécessité* et la *spécialisation des primitives*. De même, cette spécialisation est indéniablement à l'origine de la rapidité des exécutables produits. SmallEiffel étant écrit en Eiffel, ses exécutables bénéficient eux aussi des optimisations appliquées aux programmes compilés. Ses propres performances, en terme de mémoire et surtout de temps de compilation sont elles aussi excellentes, comparées aux autres compilateurs Eiffel ou à des compilateurs C++.

Ces qualités montrent donc l'efficacité des techniques sur lesquelles nous avons focalisé notre travail. Elles montrent également que les objectifs initiaux (processus de compilation

plus efficace, code généré plus efficace, portabilité) ont été atteints de façon fort satisfaisante.

6.2 Le logiciel ProMic

Participant : Dominique Colnet [Correspondant].

L'objectif du contrat Promic consistait à concevoir et à réaliser un *environnement de développement pour micro-contrôleurs synthétisables*, programmable en C. Une *machine virtuelle*[KLC99] à été définie. Celle-ci est comparable à une machine RISC dotée d'un grand nombre de *paramètres*, fournis au moment même de la configuration du FPGA : nombre de registres, taille d'une adresse RAM, taille d'une adresse ROM, hauteur de pile, taille d'un mot machine, nombre de ports d'entrée-sortie, etc. Son jeu d'instructions est composé d'un noyau dur d'instructions similaires à celles que l'on trouve habituellement dans les machines RISC, qui peut être complété par d'autres instructions à prendre en compte dans la génération du code cible. Cette machine virtuelle doit à la fois être une machine cible adaptée au langage source (le langage C), et être implantée dans le micro-contrôleur de façon efficace.

6.3 FunTalk/SmallTalk2K

Participant : Avant-projet Miró.

Grâce à la subvention de la Région Lorraine, nous avons commencé la définition du langage SmallTalk2k/FunTalk. Pour développer ce projet, nous avons proposé plusieurs sujets de DEA. De plus, A. Ciaffaglione un élève en thèse en co-tutelle INPL-Université de Udine, sous la direction de C. Kirchner (co-encradré par L. Liquori) et F. Honsell nous fournira, avec J. Despeyroux, les connaissances pour le développement de spécifications formelles de FunTalk en Coq. Les étudiants de DEA, V. Croizier et S. Salvati, ont conçu la syntaxe, la sémantique opérationnelle et le système de types du langage FunTalk. Un simple interprète non typé (réalisé en Eiffel) pour FunTalk a déjà été implanté [27, 29].

6.4 Isaac

Participants : Dominique Colnet, Benoît Sonntag [Correspondant].

Le *challenge* actuel du système Isaac est celui d'établir une passerelle fiable entre les outils présents sous UNIX et le système lui même. En effet, pour que notre système d'exploitation ne reste pas un système "jouet" et expérimental, il est indispensable d'établir des passerelles et de l'ouvrir par compatibilité avec d'autres systèmes déjà présents. Une première documentation du système Isaac se trouve sur la page web `http://www.IsaacOS.com`.

[KLC99] J. K. Limam, P. Vernel, D. Colnet, « Design of a customizable processor IP », *in : International Workshop on IP Based Synthesis and System Design*, p. 181–185, Grenoble, 1999. loria 99-R-380.

7 Résultats nouveaux

7.1 Résultats récents dans le domaine de la réécriture et des types

Participants : Horatiu Cirstea [Projet Protheo], Claude Kirchner [Projet Protheo], Luigi Liquori.

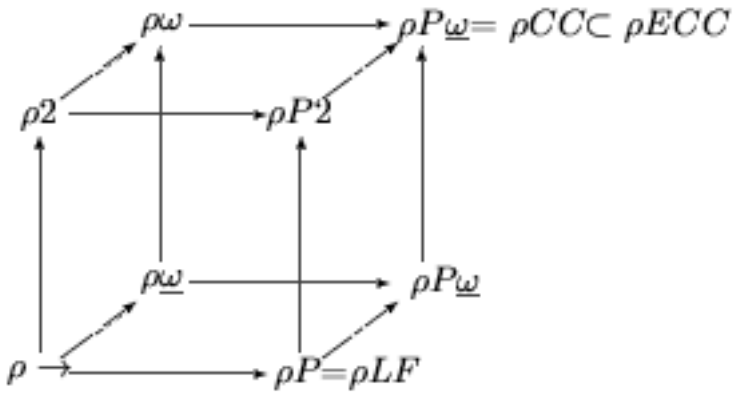


FIG. 4 – Le ρ-cube

En 2001, la collaboration avec des éléments du projet Protheo est devenu plus étroite. Nous avons publié deux articles à des conférences internationales ; dans [22], nous avons montré comment le ρ-calcul peut être utilisé en tant que *lingua franca* pour capturer différents paradigmes de programmation, comme, par exemple, la programmation à objets. En particulier, nous avons présenté un codage du LCO [11] et du OC [1] en ρ-calcul. La possibilité de codifier d'une façon naturelle le comportement de `self` (`this` de Java) dans le ρ-calcul permet une représentation des objets comme de simple ensembles, et des méthodes comme de simples règles de réécriture. Dans un certain sens, tous les formalismes et techniques propres des calculs à prototypes, type LCO et AC, peuvent être réproduit dans le ρ-calcul. En outre, dans [23], nous avons proposé le ρ-calcul dans un cadre de typage à la Church, en présentant huit systèmes des types placés dans un cube à la Barendregt (voir figure 7.1). Le plus puissant de ces systèmes s'inspire essentiellement du *Extended Calculus of Constructions* de Z. Luo [Luo90]. Une version étendue est en cours de rédaction avec G. Barthe du projet Lemme.

On pense que l'étude du ρ-calcul (typé ou non) est fondamentale et prometteuse pour explorer différentes sémantiques des langages à objets, et étendre le Calcul de Constructions (et peut-être aussi Coq) avec une solide notion de réécriture (voir aussi [BFG97]). Les recherches les plus fondamentales dans Miró seront menées dans ce sens.

[Luo90] Z. LUO, « ECC: An Extended Calculus of Constructions », *in : Proceedings of the Fourth Annual Conference on Logic in Computer Science*, IEEE Computer Society, p. 385–395, 1990.

[BFG97] F. BARBANERA, M. FERNANDEZ, H. GEUVERS, « Modularity of Strong Normalization in the algebraic-λ-cube », *Journal of Functional Programming 7*, 6, 1997, p. 613–660.

```
if (typeId < 3)
 then
  if (typeId < 2) then specialized code for type #1
                  else specialized code for type #2
  endif
 else
  if (typeId < 4) then specialized code for type #3
                  else specialized code for type #4
  endif
endif
```

FIG. 5 – Le code généré pour implanter la liaison dynamique avec sélection dichotomique.

7.2 Résultats sur la compilation efficace et sûre des langages à objets

Participants : Dominique Colnet, Luigi Liquori, Olivier Zendra.

7.2.1 Implantation de la liaison dynamique par sélection dichotomique

La connaissance de l'ensemble des types vivants offre d'importantes possibilités d'optimisations du code généré. Une des plus remarquables optimisations réalisée dans le cadre du projet SmallEiffel a consisté à implanter la liaison dynamique en générant du code de sélection dichotomique. Tous les types vivants étant connus, on associe statiquement à chacun de ces types un identifiant entier qui sert d'indicateur à l'exécution. La détermination du type dynamique de l'objet receveur est réalisée par du code de sélection dichotomique comme présenté à la figure 5. Le principal avantage de cette technique est d'éviter l'utilisation de tableaux de pointeurs de fonctions, *Virtual Function Tables* (VFT), comme en C++ par exemple. En outre, il est souvent possible de remplacer un appel de fonction par le code qui lui correspond (*inlining*) au cœur même du code de sélection dichotomique (voir [17]).

7.2.2 Match-O, un dialecte d'Eiffel probablement sûr

Les travaux commencés cette année ont abouti à la définition d'un dialecte du langage Eiffel, appelé Match-O, qui remplace le type `like Current` par un type plus précis, `match Current`. Nous avons auto-compilé un compilateur qui implante ce dialecte, accessible à `http://www.loria.fr/~colnet/Match-O/macho.tgz`.

7.2.3 Impact de l'aliasing et programmation par contrat

SmallEiffel fait un usage intensif de l'aliasing afin d'atteindre les meilleures performances possibles, tant en termes de mémoire que de vitesse d'exécution. Cette technique semble très appropriée à la compilation mais peut aussi s'appliquer à une large gamme d'applications. Les excellentes performances de SmallEiffel proviennent en grande partie de l'usage intensif de

l'aliasing dans l'implantation du compilateur. Nous avons montré dans [ZC00] comment il est possible de mettre en œuvre cette technique, de façon systématique et plus sûre.

7.3 Résultats sur le système d'exploitation Isaac

Participants : Dominique Colnet, Benoît Sonntag.

7.3.1 Définition et implantation du langage Lisaac

Le langage *Lisaac*, proche de Self, apporte un modèle uniforme pour la réalisation de nos concepts. Il se démarque par sa capacité à communiquer entres objets déjà compilés connus ou partiellement connus. Il possède les qualités de flexibilité de Self avec des outils directement liés à la mise en place de systèmes d'exploitation ou plus généralement à la programmation de bas niveau avec un langage de haut niveau. Nous avons développé un compilateur pour un langage spécialisé à base de prototypes, nommé Lisaac, qui nous sert à implanter notre système d'exploitation Isaac. Le compilateur Lisaac utilise des méthodes de compilation avancées comme l'analyse de flot pour développer l'algorithme du produit cartésien (CPA) de prédiction de type. Les premiers résultats sont encourageants et ont donné lieu à une première publication internationale présentant notre langage [25]. Les nouvelles possibilités qu'offre ce langage au niveau de la programmation de bas niveau permettent également d'uniformiser le modèle de notre système Isaac.

7.3.2 Maturation du système Isaac

Avec l'aide du stagiaire F.X. Kuntz, nous avons spécifié un nouveau format d'exécutable pour Isaac, lui permettant une meilleure flexibilité et une meilleure évolutivité. Ce format permettra aussi dans un futur proche l'intégration des produits GNU de manière transparente pour le système. Il est en cours d'implantation. Le gestionnaire du système de fichier d'Isaac a fait l'objet d'une reformalisation complète permettant une modularité et une bonne abstraction de la notion de fichier sous forme d'objet. L'interface utilisateur d'Isaac a également nécessité l'étude d'une forte abstraction des besoins en terme d'ergonomie. Notre modèle d'interface peut ainsi se décliner très simplement vers une représentation concrète diverse et appropriée à l'utilisateur. Sa représentation peut être graphique, vocale ou autre. Elle passe d'une représentation à une autre par simple modification d'héritage dynamique de l'interface abstraite.

7.4 Le contrat plan état-région (CPER)

Participants : Dominique Colnet, Luigi Liquori.

Nous sommes promoteurs du projet : *SmallTalk2K : Construction d'un Compilateur et d'un Environnement "Sûr" pour un Langage à Objet*, financé par la Région Lorraine pour l'année 2001 et 2002, avec 200 KF [8].

[ZC00] O. Zendra, D. Colnet, « Vers un usage plus sûr de l'aliasing avec Eiffel », *in : Colloque Langages et Modèles à Objets (LMO'2000)*, Hermes, p. 183–194, 2000.

7.5 Opérations développement du logiciel (ODL)

Participants : Dominique Colnet, Philippe Ribet.

Dans le cadre des Opérations Développement du Logiciel, le projet SmallEiffel a obtenu pour l'année 2001 un financement de l'INRIA de 100 KF et un ingénieur expert pour 12 mois. Le but principal de P. Ribet est l'étude et l'implantation d'une bibliothèque d'interfaçage homme-machine pour Eiffel, en utilisant le langage Eiffel lui même et la programmation par contrat.

8 Actions régionales, nationales et internationales

8.1 Équipes associées à l'étranger 2001-2002

Dans le cadre des relations internationales de l'INRIA, l'avant-projet Miró a repondu à l'appel à propositions pour la constitution d'équipes associées à l'étranger. Nous avons décidé de nous associer avec l'équipe *Semantics of Logic and Programming (SLP)* de l'Université d'Udine, équipe dirigée par le prof. F. Honsell (`http://www.dimi.uniud.it/SLP/gruppo.html`). Le titre de la proposition est *Formal semantics and certified tools for the language FunTalk* [28]. Cette proposition n'a pas été acceptée mais nous comptons la retravailler et la re-soumettre en 2002.

8.2 Accueils de chercheurs

L´avant-projet Miró a accueilli les chercheurs suivants :
- avril 2001 (30 jours) : Richard Walker, associate lecturer à l'Australian National University (ANU), de Canberra, Australie, pour effectuer un travail conjoint sur SmallEiffel, The GNU Eiffel Compiler,
- septembre 2001 (3 jours) : Gilles Barthe, CR INRIA, projet Lemme (*Reasoning about Javacard*),
- novembre 2001 (5 jours) : Dan Dougherty, full professor, Wesleyan University (*Normal forms and reduction for theories of binary relations*),
- décembre 2001 (4 jours) : Claudio Casetti, MdC, École Polytechnique de Turin (*Enhancing the TCP Protocol* et *Advanced TCP/IP*),
- décembre 2001 (3 jours) : Rossano Gaeta, associate professor, Università di Torino (*ATM*).

9 Diffusion de résultats

9.1 Diffusion du projet Miró

Les participants au projet Miró ont commencé un petit "tour de France" de projets de recherches, notamment à l'INRIA. Nous envisageons la visite de quelque projet INRIA corrélées à notre travail de recherche pour présenter, discuter et réviser le projet Miró à l'aide de séminaires informels, notre but étant de profiter au maximum des compétences de l'INRIA

pour mieux adapter notre programme de recherche et/ou trouver des collaborations. Voici pour mémoire la liste des rencontres et présentations effectuées à ce jour :

- mars 2001 : INRIA Roquencourt (Cristal & Moscova),
- juin 2001 : INRIA Sophia (Certilab & Lemme & Mimosa),
- juillet 2001 : École des Mines de Nantes (OCM, de P. Cointe),
- octobre 2001 : ENS Lyon (Plume).

9.2 Enseignement des permanents

- D. Colnet a été de 1999 à 2001 en délégation à l'INRIA. Néanmoins, il a animé à l'ESIAL (U.H.P. - Nancy 1) le cours *Programmation à objets*, où il utilise avec succès depuis 1994 le langage Eiffel et son compilateur SmallEiffel.
- L. Liquori a été responsable de 1999 à 2001 du cours de $2^{ème}$ année *Fondements de l'algorithmique et de la programmation* et il est responsable depuis 1999 du cours du $3^{ème}$ année *Réseaux et télécommunications* : en outre il participe depuis 2000 au un module de DEA IAEM Nancy ayant pour titre : *Sémantique des langages de programmation.* Il a également encadré en 2001 à l´École des Mines un projet de 2ème année sur CORBA (3 élèves) et 2 stages de fin d'étude de 3ème année. Il a également été tuteur de 3 élèves.
- O. Zendra a participé, durant son postdoctorat, à l'enseignement des modules *CS308-764.2001 : Run-time Support for Object-Oriented Programming Languages* et *CS 505 - High Performance Computer Architecture* à la School of computer Science de McGill University, Canada. Il y a également participé à l'encadrement de Matthew Holly, étudiant en *Master of Computer Science* travaillant sur les systèmes adaptatifs et l'évaluation des prédicteurs de branchements dans les micro-architectures.

9.3 Thèses et stages

Thèses et DEAs en cours dans l'avant-projet :

- A. Ciaffaglione, (doctorant INPL en cotutelle France-Italie) ; *Co-inductive reasoning on reals and objects using type theory* : le sujet de thèse porte sur le développement et la formalisation en Coq d'un nouveau langage à objets, FunTalk, et de son compilateur (encadrant L. Liquori 25% et C. Kirchner 25%) ;
- V. Croizier, DEA, 2001 : *FunTalk : Sémantique et Validation* (encadrants : D. Colnet 50% et L. Liquori 50%) ;
- S. Salvati, DEA, 2001 : *FunTalk : Sémantique et Démonstration* ; (encadrants : D. Colnet 20% et L. Liquori 80%) ;
- B. Sonntag, doctorant CCH-INRIA, 2001-2002 : *Intégration de concepts objets au coeur même des systèmes d'exploitation* (encadrant D. Colnet).

9.4 Participations à des jurys

- D. Colnet est membre du Département de Formations Doctorale (DFD) de Nancy. Il a aussi participé à la commission de spécialistes pour le concours d'ATER à Nancy. Il a également participé à la commission nationale de l'INRIA qui a évalué les opérations de développement logiciel (ODL) 2001-2002 à l'INRIA.

- L. Liquori a participé au jury de thèse de l'UHP-Nancy 1 de H. Dubois ; il a aussi participé à la commission de spécialistes pour le concours de MdC à l´École des Mines de Nancy 2001 et d'ATER à Nancy.
- O. Zendra est juge au sein du jury du concours international annuel *Eiffel Struggle* organisé par le consortium NICE, en coopération avec l'Eiffel Forum, l'Eiffel Liberty Journal (ELJ) et les principales sociétés fournissant des outils Eiffel.

9.5 Participation à des colloques, séminaires, invitations

- D. Colnet a présenté à l'université Pierre et Marie Curie de Jussieu le projet SmallEiffel, suite à une invitation ;
- L. Liquori a participé à la conférence ESOP/FOSSACS (Foundations of Software Science and Computation Structures, Genova, Italie, avril 2001) [22], à la conférence RTA (Rewriting Techniques et Applications, Utrecht, Hollande, mai 2001) [23], et au workshop WESTAPP (Utrecht, Hollande, mai 2001) [24]. L. Liquori à été invité à l'ENS-Lyon et à l'INRIA Rocquencourt pour faire un séminaire sur le ρ-calcul.
- B. Sonntag a participé à CFSE (la Conférence Française sur les Systèmes d'Exploitation, Paris, mai 2001) [26]. Il a également rendu visite à M. Muller du projet Compose de l'IRISA de Rennes. B. Sonntag a aussi animé en juin un séminaire au Centre Charles Hermite ayant pour but de présenter l'état d'avancement du système d'exploitation Isaac.
- Durant son postdoctorat, O. Zendra a participé à la conférence JVM (USENIX First Java Virtual Machine Research and Technology Symposium, Monterey, Californie, avril 2001), ainsi qu'à la conférence OOPSLA (ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, Tampa Bay, Floride, octobre 2001) où il a présenté ses résultats préliminaires [30]. Il été invité en juillet 2001 pour animer un séminaire sur l'optimisation dans les langages à objets, à Purdue University, Indiana, USA. Il a également animé des séminaires informels au sein de McGill University, centrés sur l'expérience aquise sur la compilation efficace des langages objets dans le cadre du projet SmallEiffel.

9.6 Divers

- L. Liquori est responsable des relations de l'École des Mines de Nancy avec les universités italiennes. En 2001 sont arrivés à Nancy 2 élèves en DEA d'informatique et 7 élèves en cursus scolaire terminale à l'École des Mines. L. Liquori participe aussi à la commission de choix des enseignants de l'École.
- D. Colnet et O. Zendra sont membres actifs du Nonprofit International Consortium for Eiffel (NICE), et participent à la définition des standards pour le langage Eiffel.

9.7 Référés d'articles

- L. Liquori a été rapporteur pour les conférences/journaux suivants : APPSEM (numéro spécial LNCS), FOSSACS-01 (LNCS), TLCA-01 (LNCS), RTA-01 (LNCS), STRATEGIES-01 (ENTCS), TOPLAS, MFCS.

– O. Zendra a été rapporteur pour deux articles soumis à la conférence ECOOP-01 (LNCS), ainsi que pour deux articles soumis en 2001 au journal Software - Practice & Experience (InterSciences, Wiley).

Remerciements : Pour la rédaction finale de ce rapport, l'avant-projet Miró remercie sincèrement Isabelle Gnaedig pour sa relecture attentive et constructive du document.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] M. Abadi, L. Cardelli, *A Theory of Objects*, Springer Verlag, 1996.

[2] K. Bruce, L. Cardelli, C. B. Pierce, « Comparing Object Encoding », *in : Proc. of TACS, LNCS, 1281*, Springer Verlag, p. 415–438, 1997.

[3] K. Bruce, « A Paradigmatic Object–Oriented Programming Language : Design, Static Typing and Semantics », *Journal of Functional Programming 4*, 2, 1994, p. 127–206.

[4] K. Bruce, « Subtyping is not a Good Match for Object-oriented Programming Languages », *in : Proc. of ECOOP, LNCS, 1241*, Springer Verlag, 1997.

[5] C. Chambers, D. Ungar, « Customization : Optimizing Compiler Technology for Self, a Dinamically-typed Object-Oriented Programming Language », *in : SIGPLAN-89 Conference on Programming Language Design and Implementation*, p. 146–160, 1989.

[6] D. Colnet, P. Coucaud, O. Zendra, « Compiler Support to Customize the Mark and Sweep Algorithm », *in : ACM SIGPLAN International Symposium on Memory Management*, p. 154–165, 1998.

[7] D. Colnet, L. Liquori, « Match-O, a Statically Safe (?) Dialect of Eiffel », *in : Technology of Object-Oriented Languages and Systems*, IEEE Computer Society, 2000.

[8] D. Colnet, L. Liquori, « SmallTalk-2K : Construction d'un Compilateur et d'un Environnement "Sûr" pour un Langage à Objet », 2000, Demande de Subvention à la Région Lorraine pour l'an 2001-2002 Soutien aux jeunes équipes de recherche et aux projets émergents.

[9] D. Colnet, O. Zendra, « Optimizations of Eiffel programs : SmallEiffel, The GNU Eiffel Compiler », *in : Technology of Object-Oriented Languages and Systems*, IEEE Computer Society, p. 341–350, 1999.

[10] D. Colnet, *Compilation, langages à objets et programmation par contrats. Expérience acquise dans le cadre du projet "SmallEiffel The GNU Eiffel Compiler"*, Thèse d'habilitation à diriger des recherches, Université Henri Poincaré – Nancy 1, 2000.

[11] K. Fisher, F. Honsell, J. C. Mitchell, « A Lambda Calculus of Objects and Method Specialization », *Nordic Journal of Computing 1*, 1, 1994, p. 3–37.

[12] P. D. Gianantonio, F. Honsell, L. Liquori, « A Lambda Calculus of Objects with Self-inflicted Extension », *in : Object-Oriented Programming, Systems, Languages, and Applications*, The ACM Press, p. 166–178, 1998.

[13] S. N. Kamin, « Inheritance in Smalltalk-80 : A Denotational Definition », *in : Proc. of POPL*, T. A. Press (éditeur), p. 80–87, 1988.

[14] L. Liquori, « An Extended Theory of Primitive Objects : First Order System », *in : Proc. of ECOOP, LNCS, 1241*, Springer Verlag, p. 146–169, 1997.

[15] L. Liquori, « On Object Extension », *in : Proc. of ECOOP, LNCS, 1445*, Springer Verlag, p. 498–552, 1998.

[16] D. Ungar, R. B. Smith, « Self : The power of simplicity », *in : Object-Oriented Programming, Systems, Languages, and Applications*, The ACM Press, p. 227–241, 1987.

[17] O. Zendra, D. Colnet, S. Collin, « Efficient Dynamic Dispatch without Virtual Function Tables. The SmallEiffel Compiler. », *in : Object-Oriented Programming, Systems, Languages, and Applications, 32(10)*, The ACM Press, p. 125–141, 1997.

[18] O. Zendra, D. Colnet, « Adding external iterators to an existing Eiffel class library », *in : Technology of Object-Oriented Languages and Systems*, IEEE Computer Society, p. 188–199, 1999.

[19] O. Zendra, *Traduction et optimisation dans les langages à objets*, Thèse de Doctorat, Université Henri Poincaré – Nancy 1, 2000.

Articles et chapitres de livre

[20] D. Colnet, O. Zendra, « Global System Analysis at Work », *Journal of Object-Oriented Programming 14*, 1, 2001, p. 10–13.

[21] O. Zendra, D. Colnet, « Coping with aliasing in the GNU Eiffel Compiler implementation », *Software - Practice and Experience 31*, 6, 2001, p. 601–613.

Communications à des congrès, colloques, etc.

[22] H. Cirstea, C. Kirchner, L. Liquori, « Matching Power », *in : Proc. of RTA, LNCS, 2030*, Springer Verlag, p. 168–183, 2001.

[23] H. Cirstea, C. Kirchner, L. Liquori, « The Rho Cube », *in : Proc. of ESOP/FOSSACS, LNCS, 2051*, Springer Verlag, p. 77–92, 2001.

[24] D. Dougherty, F. Lang, P. Lescanne, L. Liquori, K. Rose, « A Generic Object-calculus based on Addressed Term Rewriting Systems », *in : Proc. of Westapp, Fourth International Workshop on Explicit Substitutions : Theory and Applications to Programs and Proofs*, F. Kamareddine, V. van Oostrom (éditeurs), Logic Group Preprint series No 210. Utrecht University, the Netherlands, p. 6–25, 2001. A longer version is under major revision in *Journal of Functional Programming*, `http://www.loria.fr/~lliquori/PAPERS/jfp-02.ps.gz`.

[25] B. Sonntag, D. Colnet, « Lisaac : the power of simplicity at work for operating system », *in : Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific'2002), Sydney, Australia*, IEEE Computer Society, 2002. A paraître.

[26] B. Sonntag, « Utilisation de la segmentation mémoire du processeur à partir d'un langage de haut niveau », *in : Proc. of Conférence Française sur les Systèmes d'Exploitation, CFSE*, B. Folliot, P. Sens (éditeurs), ACM Press, p. 107–116, 2001.

Divers

[27] V. Croizier, « FunTalk : Formalisations et Validation », 2001, Stage de DEA.

[28] L. Liquori, M. Miculan, « Formal semantics and certified compilers for the language FunTalk », 2001, Proposition pour équipe associés, `http://www.loria.fr/~lliquori/PAPERS/miro-slp.ps.gz`.

[29] S. Salvati, « FunTalk : Formalisations et Démonstrations », 2001, Stage de DEA.

[30] O. Zendra, K. Driesen, F. Qian, L. Hendren, « Evaluation of the runtime performance of control flow structures for dynamic dispatch in Java », Poster and extended abstract. OOPSLA Conference Companion, The ACM Press, 2001.