RESUME

Pierre LESCANNE
Born March 22 1947, at Dakar Sénégal.

Pierre LESCANNE born March 22, 1947, in Dakar Senegal.

2001-2004 Director of the master in Computer Science of Lyon and deputy director of the doctoral school mathematics and Computer Science of Lyon.

1997-.... Professor at Ecole Normale Superieure of Lyon.

1998-2001 Chair of Department of Mathematics and Computer Science at ENS Lyon.

1987-1998 Directeur de recherche at Centre National de la Recherche Scientifique.

1993-1996 Director of the CNRS National Research Consortium *Programmation*,

1993-1997 Director of *Centre Charles Hermite*, Centre Lorrain de Comp´etence en Mod´elisation et Calcul à Haute Performance.

1989-.... Member of the editorial board of the journal *Applicable Algebra in Engineering, Communication and Computing* published by Springer Verlag.

1986-1997 Manager of INRIA Project *(EURECA)* in Nancy,

1986-1992 Deputy Director of Centre de Recherche en Informatique de Nancy.

1992-1994 Vice Pr´esident of SPECIF, (Soci´et´e des Personnels Enseignants et Chercheurs de France).

July 1987 Scientific Consultant at Institute for New Generation Computer Technology, Tokyo, Japan.

July-August 1985 Visiting Scientist at Computer Science Laboratory of SRI-International, invited by Dr Joseph Goguen.

July-August 1984 Visiting Scientist at Laboratory for Computer Science du MIT, invited by John Guttag.

1983-1984 Professor at University de Nancy 2.

1980-1987 Charg´e de Recherche at Centre National de la Recherche Scientifique.

1980-1982 Visiting scientist at Laboratory for Computer Science of Massachusetts Institute of Technology, invited by Professor John Guttag.

September 1979 Docteur d'Etat.

1974-1980 Attach´e de Recherche Agr´eg´e at CNRS.

1972-1974 Maître-Assistant at the University of Nancy 2.

1971-1972 Military duty.

1968-1971 Assistant professor at University of Nancy 1.

July 1969 Agr´egation of mathematics

Domains of research

Formal Methods, Theoretical Computer Science, Logic.

Themes of research

- Automated Deduction, more precisely theorem proving in equational theories by rewriting techniques. Implementation of the software REVE and of the software ORME.
- Higher order automated deduction and its applications to protocols and electronic commerce.
- Functional and object oriented programming and models of computation.

Publications

- co-author of 1 book
- 19 international publications in journals
- 30 communications at international conferences
- 14 communications at national conferences
- 2 softwares : REVE and ORME
- Member of the editorial board of the journal *Applicable Algebra in Engineering, Communication and Computation,* published par Springer Verlag.

Awards, participations and membership

- Eureca was awarded of m´edaille d'argent du CNRS in 1987,
- Nominator for the Kyoto Prize.
- Member of comit´e national du Centre National de la Recherche scientifique (CNRS) 1983-1986,
- Member of the National Committee of Universities.
- Program committee member of European Symposium on Programming – Nancy F (1987), Copenhagen DK (1990)–, Rewriting Techniques and Applications – Dijon F (1985), Bordeaux F (1987) chairman –, Logic and Algebraic Programming, Symposium of Theory of Computer Science – Gaussig DDR (1988)–, Colloquium Current Concepts In Programming Languages in TAPSOFT – Barcelona E (1989)–, Logic in Computer Science – Montreal USA (1993) –, RTA 96, TLCA'97, PPDP-99, FOSSACS'01 (Genova (2002), RTA 2004.
- Organizer of the 2nd conference on Functional Programming and Computer Architecture (1985).
- Guest editor of Theoretical Computer Science and of Journal of Logic Programming,
- Member of the board of Association Française d'Informatique Th´eorique,
- Member of comit´e des projets of INRIA-Lorraine,
- Pr´esident of research committee of Soci´et´e des Personnels Enseignants et Chercheurs d'Informatique de France (SPECIF),
- Member of Association for Computing Machinery (ACM), European Association for Theoretical Computer Science (EATCS) and Association for Automated Reasoning (AAR), Association Française d'Informatique th´eorique (AFIT) (member of the board).
- Referee for Transaction on Programming Languages and Systems , Information and Computation, Journal of the ACM, Journal of Computer and System Science, Acta Informatica, Theoretical Computer Science, Techniques et Sciences de l'Informatique, Journal of Symbolic Computation etc.

Supervised PhD

Françoise Bellegarde *Utilisation des systèmes de réécriture d'expressions fonctionnelles comme outil de transformation de programmes itératifs*, Thèse d'Etat (1985).

Fernand Reinig *L'Ordre de Dcomposition : un Outil Incrmental pour Prouver la Terminaison Finie des Systmes de Rcriture quationnels* Thse de troisime cycle (1981)

Isabelle Gnaedig : *Preuves de terminaison des systmes de rcriture associatifs commutatifs : une mthode fonde sur la rcriture elle-même,* Thse de troisime cycle. (1986).

Ahlem Ben Cherifa : *Preuves de terminaison de systmes de rcriture bases sur des calculs lmentaires sur des polynômes.* Thse de l'Universit Nancy I (1986).

Azzedine Lazrek : *tude et ralisation de mthodes de preuve par rcurrence en logique quationnelle.* Thse de l'Institut National Polytechnique de Lorraine (1988).

Jocelyne Rouyer : *Calcul formel en gomtrie algbrique relle appliqu la terminaison des systmes de rcriture,* Thse de l'Universit Nancy I (1991).

Joseph Rouyer : *Dveloppement d'algorithmes dans le calcul des constructions.* Thse de l'Institut National Polytechnique de Lorraine (1994).

Stefan Krischer : *Mthodes de vrification de circuits digitaux*. Thse de l'Institut National Polytechnique de Lorraine (1994).

Boutheina Chetali *Vérification formelle des systmes parallles décrits en* UNITY *l'aide d'un outil de démonstration automatique* Thse de l'Universit´e Henri Poincar´e Nancy I (1996),

Zine-el-Abidine Benaissa *Les calculs de substitution explicites comme fondements des langages de programmation fonctionnels.* Thse de l'Universit´e Henri Poincar´e Nancy I (1997).

François Briaud *Unification d'ordre supérieur et substitutions explicites.* Thse de l'Universit´e Henri Poincar´e Nancy I (1997).

Barbara Heyd *Application de la théorie des types et du démonstrateur COQ la vérification de programmes parallles.* Thse de l'Universit´e Henri Poincar´e Nancy I (1997).

Frédéric Lang *Modles de la β-réduction pour les implantations.* Thse de l'ENS de Lyon (1998).

Frederic Prost *Interpretation of staic analysis in type theory* Thse de l'ENS de Lyon (1999)

Romain Kervarc *Les substitutions explicites dans le cube de Barendregt et dans les squents*. Second year.

MY FIVE MOST SIGNIFICATIVE PAPERS

Let me first recall few facts. I defended my first thesis (thse de troisime cycle) in 1971 and I defended my Thse d'Etat at the University of Nancy in 1979 both under the supervision of Professor Claude Pair. After taking teaching positions at the University of Nancy, I entered CNRS in 1974, which I left in 1997 to joint Ecole normale suprieure de Lyon. During the years 1980-1982, I staid at MIT in professor John Guttag group. My main contribution during the period 1970-1980 was the textbook *Theory of Programs* [Liv78][1] (in French) with co-authors, my main contribution was on fixed-point, verification and denotational semantics including continuations. In *Mathematical Reviews* Jean Vuillemin wrote

> *This is an excellent book on the "Theory of programming". The author takes us, in a unified manner, through the various theories and lessons of the field. He introduces fixed-points and program schemes ; from there we get to verification and construction of programs. The author then puts all this together, and constructs a programming language, together with various expressions of its semantics. This book could serve as the basis for a graduate course on the subject.*

Below I present my most significative papers[2] also the most cited papers of CiteSeer `http://citeseer.nj.nec.com/`. The papers are on the Web at `http://perso.ens-lyon.fr/pierre.lescanne/5_most_significative.htm`.

The presentation will be subdivided in three sections according to three main topics : *computer environments for rewriting, disequations* and *explicit substitutions.*

It should be noticed that in computer science some conferences have an impact equivalent to the best journals [Pat04]. This is the case of the *ACM Conference on Principles of Programming Languages* nicknamed POPL. This is also why two such communications at POPL appear in my top 5.

Computer environments for rewriting

In my career as a computer scientist I have worked on foundation of computer science, but I have also implemented algorithms in software which have been reused by other persons. This was the case of REVE an environment for proving property of term rewriting systems that I developed at MIT in 1980 and 1981 in the group of Professor John Guttag.

Many mathematical theories are presented by identities (or equalities) as are also theories associated with some abstract data types, which are tools used in the design of software. To prove that an equality is a consequence of old ones, the best way is to orient the given equalities into simplifying rules (called

[1] Bibliographic references are given at the end of this chapter.

[2] ‘(: −) I am talking about my scientific papers, since the note that made me known among the community of French young scientists is probably my Web page "How to apply ?" [Les90c] which tells young researchers how to prepare an application for an academic position.

rewrite rules) and to try to simplify the sides of the equality to be proved till one gets a trivial equality. But usually orienting the given equalities into rules is not easy. First, one has to be sure that the simplifying process terminates in all case, avoiding divergence and one has to be sure that the most simplified form of an expression is unique. Donald Knuth [KB70] has designed a procedure that starts from a set of equalities and provides a set of rules which is equivalent to the set of equalities, which terminates always and which provides always a unique most simplified expression. The so called Knuth-Bendix procedure relies on a method for proving that a set of rules terminates. Actually there is no universal method for proving that a set of rules terminates, so several methods exist with each a specific domain of application.

Computer Experiments with the REVE Term Rewriting System Generator *10th ACM Symposium on Principles of Programming Languages*, January 24-26 1983, Austin Texas, pp 99-108. In this paper, two appendices propose the full development of two examples : an equational specification of an abstract data types for sets and an automatic proof of an inductive lemma about the Fibonacci numbers. The main originality of REVE is its tool for automatically proving the *uniform termination* (also known as noetherianity or strong normalization) of the rewriting systems being built. The method based on the *recursive decomposition ordering* [Les90b] is *incremental*, which means REVE builds the proof of termination as it discovers new rules. The *status* of an operator (introduced for the first time) tells how the ordering should consider it and is also incrementally introduced. In this paper, I announce a very important result, namely the first proof done by REVE of a non trivial mathematical result by a computer with no help (or almost no help) from a human. This result says that the presentation by the unique equation

$$x/((((x/x)/y)/z)/(((x/x)/x)/z)) = y$$

proposed by Higman and Neuman [HN52] is actually this of group. In [Les84] I present a full description of the proof with other results obtained in algebras. Whereas the proof of Higman and Neuman uses second order argument i.e., makes quantifications over sets, REVE uses first order.

REVE was later developed by the MIT group and gave birth to a new tool called the LARCH PROVER which was used to handle huge proofs related to the foundation of computer science, for instance a practical concurrent garbage collector, a very sophisticated, subtle and error prone piece of software. Later I wrote a new software called ORME [Les90a], based on a rigorous approach mostly for a didactic purpose.

Termination of Rewriting Systems by Polynomial Interpretations and its Implementation *Science of Computer Programming, vol. 9, n 2, (1987), pp 137-160.* This work has been done with a Tunisian student, named Ahlem Ben Cherifa who works now in a business company in her country[3]. The paper describes an automatic implementation of a method for proving termination of rewriting systems based on polynomial interpretations. The key point is to fully automatically prove that a polynomial is positive for every value. The proposed heuristics is simple and works well. It is the basis of many tools developed these days. Since I extended this method to handle elementary interpretations (polynomials + exponentials) [Les95] and I proposed a method based on transformation of rewrite systems [BL90].

Disequations

Solving problems containing only formal equations is fundamental in symbolic computation, in computer aided theorem proving and in constraint solving. It is also an important theoretical issue.

[3]Her first name Ahlem means *dream* (alias *rve* in French).

Equational problems and disunification *J. of Symbolic Computation (1989), vol. 3 and 4, pp. 371-426.* This paper is a collaboration with my colleague Hubert Comon who is now professor at the ENS de Cachan. In equational theories and in rewriting, one has often to solve equations. Working with one of my first master students Jean-Jacques Thiel [Thi84, LLT90], we saw that one has also to solve disequations. Whereas equations are questions of the form $s \stackrel{?}{=} t$, *disequations*, a terminology that Hubert and I made popular, are questions of the form $s \stackrel{?}{\neq} t$. In this paper we proposed a new algorithm for solving equational problems, i.e., problems where the only predicate is $=$, this means they contain equations, disequations and quantifications. The algorithm is based on inference rules and we show that those rules form a complete procedure for finding solutions of equational problems if they exist. As a by product we give a algorithm for deciding the validity in the Herbrand universe of first order formulae that contain only $=$.

Explicit substitutions

λ-calculus was designed to formalize the concept of function. This description is intentional, that is that roughly speaking it tells how a function yields a result when applied to a value. It relies on the concept of substitutions which is defined in the epi-theory of the λ-calculus ($\epsilon\pi\iota$ is "around" in Greek).

From lambda-sigma to lambda-upsilon, a journey through calculi of explicit substitutions *21st ACM Symposium on Principles of Programming Languages (POPL), 16-19 January 1994, Portland, Oregon, pp 60-69.* The fact that a main concept namely substitution was not defined in the λ-calculus stroke me. In addition, with my first-order culture, I disliked the fact that the λ-calculus had to speak about bound variables, capture avoiding and like (since I partly changed my mind). At the beginning of the nineties, researchers proposed the concept of "explicit substitutions" which with de Bruijn indices *internalizes* the substitution and moreover makes the calculus *a true first order calculus*. But I found this proposal too complex and oriented toward one approach. Therefore I suggested many possible systems and eventually one named $\lambda\upsilon$ or *lambda-upsilon*. $\lambda\upsilon$ is extremely simple as it is made of eight tiny and straightforward rewrite rules. This paper with a rather provocative title contributed to open a new active and fecund field which sets a new foundation of λ-calculus and logic and which applies to the understanding of functional programming language implementation. Its impact was both among logicians and among specialists in semantics of programming languages.

Lambda-upsilon, a calculus of explicit substitutions which preserves strong normalisation *J of Functional Programming, Vol. 6, n 5 (September 1996).* This paper was written with my Nancy PhD students Zine-El-Abidine Benaissa, and Daniel Briaud and my Nancy colleague Jocelyne Rouyer. In 1994 a question was raised namely whether calculi of explicit substitution represent faithfully λ-calculus. Among others, people wanted to know whether strong normalization was preserved, where *strong normalization* of a term is the property that says that all reductions starting at that term terminates. In 1995 two main results were announced

1. The calculus of explicit substitution $\lambda\mathsf{x}$ (due to Roel Bloo and Kristoffer Rose [BR95], see also [DL03, LDL$^+$04]) and the calculus of explicit substitution $\lambda\upsilon$ preserve strong normalization.
2. The calculi of explicit substitution $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$ do not preserve strong normalization (Mellies [Mel95]).

Therefore, people realize that *preservation of strong normalization* is an important property of calculi of explicit substitution which is not shared by all other formalisms. This started many researches for new calculi which enjoy preservation of strong normalization and more properties in order to ensure that the λ-calculus is faithfully extended of as mentioned above. The method proposed in the paper was the source of many others.

Bibliography on the most significative publications

[BL90] F. Bellegarde and P. Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computation*, 1(2) :79–96, 1990.

[BR95] Roel Bloo and Kristoffer Høgsbro Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95 – Computer Science in the Netherlands*, pages 62–72, November 1995.

[DL03] Daniel Dougherty and Pierre Lescanne. Reductions, intersection types, and explicit substitutions. *Mathematical Structures in Computer Science*, 13(1) :55–85, 2003.

[HN52] G. Higman and B. H. Neuman. Groups as groupoids with one law. *Publ. Math. Debrecen.*, 2 :215–221, 1952.

[KB70] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[LDL+04] St´ephane Lengrand, Dan Dougherty, Pierre Lescanne, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. Intersection types for explicit substitutions. 2004.

[Les84] P. Lescanne. Term rewriting systems and algebra. In R. Shostak, editor, *Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, LNCS. Springer-Verlag, 1984.

[Les90a] P. Lescanne. Implementation of completion by transition rules + control : ORME. In H´elène Kirchner and W. Wechler, editors, *Proceedings 2nd International Conference on Algebraic and Logic Programming, Nancy (France)*, volume 463 of *LNCS*, pages 262–269. Springer-Verlag, 1990.

[Les90b] P. Lescanne. On the recursive decomposition ordering with lexicographical status and other related orderings. *Journal of Automated Reasoning*, 6 :39–49, 1990.

[Les90c] Pierre Lescanne. Comment candidater ? `http://perso.ens-lyon.fr/pierre.lescanne/TEXTS/candidater.html`, 1990.

[Les95] P. Lescanne. Elementary interpretations in proofs of termination. *Formal Aspect of Computing*, 7 :77–90, 1995.

[Liv78] C. Livercy. *Théorie des programmes*. Dunod, Paris, 1978. Livercy stands for Jean-Pierre Finance and Monique Grandbastien and Pierre Lescanne and Pierre Marchand and Roger Mohr and Alain Qu´er´e and Jean-Luc R´emy. Available at `http://perso.ens-lyon.fr/pierre.lescanne/publications.html`.

[LLT90] A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness and ω-completeness. *Information and Computation*, 84(1) :47–70, January 1990.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In M. Dezani and G.Plotkin, editors, *TLCA'95*, volume 902 of *LNCS*, pages 328–334. Springer-Verlag, 1995.

[Pat04] David A. Patterson. The health of research conferences and the dearth of big idea papers. *Communications of the ACM*, 47(12) :23 – 24, December 2004.

[Thi84] J.-J. Thiel. Stop losing sleep over incomplete data type specifications. In *Proceeding 11th ACM Symp. on Principles of Programming Languages*, pages 76–82. ACM, 1984.