CONFER

CONcurrency and Functions:

Evaluation and Reduction

Basic Research Action

Project Number: 6454

Periodic Progress Report
October 1, 1993

Contents

1 Overview **3**

2 Executive summary **5**

3 Management **9**

3.1 Consortium level 9
3.2 CWI 10
3.3 University of Edinburgh 12
3.4 ECRC 14
3.5 ENS 18
3.6 Imperial College 21
3.7 INRIA-Rocquencourt 23
3.8 INRIA-Sophia 25
3.9 Università di Pisa 26
3.10 SICS 30

4 Deliverables **33**

4.1 Workshop 1 33
4.2 Workshop 2 34
4.3 Software deliverables 36

5 Progress **37**

5.1 Foundational models and abstract machines 37
5.2 Calculi 51
5.3 Logics for Concurrency and λ-calculus 66
5.4 Programming Languages 79

6 Appendices **89**

Chapter 1

Overview

This report contains the First Periodic Progress Report for ESPRIT BRA Nr. 6454 (CONFER).

The report contains 4 main parts: management at consortium level and at each site, deliverables (programs of each CONFER workshops 1 and 2, software deliverables), a progress report describing the technical work achieved during the first year, and appendices listing CONFER publications.

Further information may be requested from the coordinator:

Jean-Jacques Lévy
INRIA, Rocquencourt
bat.8, Domaine de Voluceau
78153–Le Chesnay, Cedex
France
tel: +33-1-39-63-56-89
fax: +33-1-39-63-53-30
e-mail: `Jean-Jacques.Lévy@inria.fr`

This document has been compiled from input from all of the partners in the CONFER project. Lone Leth and Bent Thomsen from ECRC greatly helped in the writing and the assembly of the document. Alessandro Giacalone from ECRC prepared the area report on Programming Languages and Simon Gay and Rajagopal Nagarajan from Imperial College prepared the area report on Logics for Concurrency and the λ-calculus.

Chapter 2

Executive summary

The overall objective of the CONFER action is to create both the theoretical foundations and the technology for combining the expressive power of the functional and the concurrent computational models. The action is organized around four main areas of work:

- Foundational Models and Abstract Machines
- Calculi
- Logics for Concurrency and the λ-calculus
- Programming Languages

The first year of CONFER has been very successful. The objectives set forth in the work plan have been achieved, and significant results have been obtained beyond these objectives.

In the area of Foundational Models and Abstract Machines the state of the art in basic calculi with bound variables, ranging from the λ-calculus to higher-order communication systems, has been advanced significantly. Work in the area is often based on the concept of Lafont's interaction nets. The area includes more general schemes such as combinatory reduction systems and abstract reduction systems. The notion of action structures developed by Milner gives an algebraic setting which covers many of the known calculi of communication systems. It seems to allow connections with the very atomic operations of the λ-calculus. Many of the frameworks of this area of CONFER are graph reduction systems; the most highly developed are for functional calculi and provide insight into the optimality of computation strategies, but there are also graphical treatments of the π-calculus.

In the calculi area, a great deal of work has focused on comparing new calculi with existing ones. This has yielded a better understanding of the expressive power of existing calculi. During just one year the area of calculi with name passing has matured to a state where sound and complete axiomatisations for bisimulations exist. Furthermore, a verification tool has emerged. This is remarkable since in other areas of concurrency theory these developments have taken considerably longer time to develop.

The notion of bisimulation has been extensively studied. Several approaches have been compared, and some of them have been shown to lead to the same notion. Decisive steps have been taken to advance the state of the art in calculi to account for phenomena such as true concurrency and physical distribution, which are of paramount importance for the programming language area.

In the area of Logics for Concurrency and the λ-calculus, Interaction Categories have been developed as a new foundation for semantics of sequential and concurrent computation. A significant number of studies showing this have been carried out. Several sort and type systems have been developed. These are important for both the correctness and the optimization of concurrent/functional programs. The work on Linear Logic and Optimality transfers techniques from the λ-calculus paradigm to concurrency. The work done clearly reflects the fruitful interplay between logic, concurrency, and functional computation.

In the area of Programming Languages several prototypes have been developed: Prototype compiler for λ-calculus, based on graph reduction, Portable, unobtrusive garbage collection for multiprocessor systems, Lilac: a prototype functional programming language based on Linear Logic, Typed higher-order programming language based on π-calculus, all deriving from the substantial work on calculi, foundational models and logic. Important work results for compile time optimisations have been obtained on Termination properties of unfolding extended to programs with non-determinism. Clearly some of the above results and prototypes are necessarily of a rather preliminary nature. More concrete results should be expected for the last phases of the BRA. One exception is the Facile programming language, which is already being experimented with in quite significant applications. However, it should be noted that the Facile project was already in progress before the beginning of CONFER and also that the dimensions and scope of the project are wider than what can be considered as strictly relevant to the CONFER BRA

Looking at the summaries of the work in chapter 5 on progress, it is evident that related work done at other sites in the consortium is referenced often. Concrete collaboration between sites is taking place. This reflects one of the essential ideas of the CONFER project, namely crossfertilization of ideas among fellow researchers in the consortium.

During Year 1 of CONFER two workshops have been held. The first was organised by Gérard Boudol and took place in January 1993 at INRIA-Sophia Antipolis and had 13 presentations and 40 participants. The second workshop was organised by Davide Sangiorgi and took place in May 1993 at the University of Edinburgh with 15 presentations and 45 participants. The action has produced a large number of reports and several of these have been published at conferences and international workshops.

Several Ph.D.'s are in preparation in the action.

We would like to point out that the results achieved so far is the result of the involvement of a large research community at each site, involving not only the researchers strictly supported by the CONFER funding. Furthermore, a number of researchers outside the consortium are contributing to the effort and using or planning to use the results coming out of the CONFER action. To this regard, it is worth mentioning that the LOMAPS BRA project (which in part will focus on data flow analysis and optimisations for concurrent functional languages) will use the results being produced in the CONFER context.

Chapter 3

Management

3.1 Consortium level

The main management of the project is done at INRIA, Rocquencourt, and at ECRC, Munich.

The following activities at the consortium level have taken place during the first year of the CONFER project:

Two workshops have been held. The first took place in January 1993 at INRIA-Sophia Antipolis and had 13 presentations and 40 participants. The second workshop took place in May 1993 at the University of Edinburgh with 15 presentations and 45 participants.

At the second workshop a management meeting was held to plan for the first annual review.

A third workshop will take place at the end of September at CWI, prior to the annual review on October 1st, 1993. At the first workshop it was decided to create a CONFER ftp site to facilitate easy dissemination of results within and outside the consortium.

Imperial College volunteered to set up and maintain this service. It has been operational for several months (`theory/CONFER` at `theory.doc.ic.ac.uk`). It was also decided to maintain a "friends of CONFER" mailing list, used for broadcasting relevant information to researchers outside the consortium.

To further disseminate information the (technical) coordinators produced an overview of the project at the beginning of the action. This report was published in the Bulletin of EATCS, Number 45, October 1992, pp.158-185.

3.2 CWI

3.2.1 Research directions

The major line of research of CWI has been in the area of higher-order term rewriting, especially with respect to its syntactic properties. A second line concerns an extension of the process algebra ACP with combinators as in CL (Combinatory Logic) in order to eliminate bound recursion variables in favour of a purely first-order equational style.

Higher-order rewriting was introduced to treat first-order rewriting and rewriting with bound variables as in λ-calculus, π-calculus in a uniform framework. Under the restraint of (weak) orthogonality important properties such as confluence and uniqueness of normal forms can be proved. As an application, π-calculus without matching construct was shown to be a weakly orthogonal Combinatory Reduction System (CRS).

Several proposals for a uniform framework for higher-order rewriting have been made in recent years. A study has been made to compare two of the main proposals: Combinatory Reduction Systems (CRSs) and Higher-order Rewrite Systems (HRSs). Although these two formats seem rather different, CRSs being type-free (though able to model typed systems) and HRSs employing simply typed λ-calculus as meta-language, they have been proved to be equally expressive and able to simulate each other in a direct sense. An important difference between CRSs and HRSs is that in the first, there is a distinction between variables and meta-variables. The latter may be instantiated to terms; the former serve quite different purposes, like for instance the names in π-calculus.

A second line of research consisted in developing Process Algebra with Combinators. This study seems very much in the centre of CONFER goals, combining concurrency with function evaluation and reduction as in Combinatory Logic (CL). It is a quite different point of view than that of other calculi in CONFER and that of CRSs: here bound variables are entirely eliminated (as CL does for λ-calculus). The bound variables meant here are the recursion variables in a system of equations describing e.g. a protocol. The verification now becomes purely first-order equational, and does not have to resort to conditional equations as usual.

3.2.2 Persons and exchanges

The group at CWI involved in the project consists of Jan Willem Klop, Femke van Raamsdonk, Fer-Jan de Vries. Femke van Raamsdonk, Ph.D. student of Jan Willem Klop, is working on higher-order term rewriting. Part of the work is subcontracted to Jan Bergstra (University of Utrecht, University of Amsterdam); this work pertains to combining process algebra with combinatory logic and a current study of local ports in process algebra. There is a close cooperation by Jan Willem Klop and Femke van Raamsdonk with Vincent van Oostrom, Ph.D.student at the Free University Amsterdam, who has co-authored two CONFER deliverables and is writing a thesis under supervision of Jan Willem Klop that is relevant to the project (provisory title: 'Confluence for abstract and higher-order rewriting').

The main connections with other sites concern π-calculus (as object of study in the CRS framework) and the Interaction Systems of Andrea Asperti and Cosimo Laneve. (Interaction Systems are a special case of CRSs.) Up to this date no personnel exchanges have been made but a visit of Laneve to CWI has been arranged following the present project meeting. Furthermore we intend to profit (regarding the topic of π-calculus) from Davide Sangiorgi's stay at CWI in October 1993.

3.2.3 Perspectives, work in progress

"Confluence for weakly orthogonal CRSs", (V. van Oostrom, F. van Raamsdonk). The authors recently found a simple proof for confluence of weakly orthogonal CRSs. This is important because many interesting calculi are only weakly orthogonal, e.g. second order typed lambda calculus with beta and eta rule. Also the CRS version of pi-calculus has some (innocent) critical pairs, i.e. it is weakly orthogonal.

"Modular term graph rewriting and cyclic lambda graph rewriting", (J.W. Klop, Z. Ariola). It remains to be seen whether this study fits in CONFER; it originated in ESPRIT BRA Semagraph, now terminated (but continued as Working Group). It may fit in CONFER, in view of notions of bound variable, hiding of node names, encapsulating boxes. Especially in cyclic λ-graph rewriting there are some curious phenomena regarding bound variables. The relevance for our project is that several notions in this study 'resonate' with some that are present in calculi studied in the project; and second that in due time, implementations of calculi in the project may employ term graph rewriting.

3.2.4 Publications, technical reports

(1) Combinatory Reduction Systems: introduction and survey, J.W. Klop, V. van Oostrom, F. van Raamsdonk. CWI Report CS-R93xx (to appear); to be published in Theor. Comp. Sci.

(2) Comparing Combinatory Reduction Systems and Higher-order Rewrite Systems, V. van Oostrom, F. van Raamsdonk. Report IR-333, Free University Amsterdam, Aug. 1993; to appear in Proceedings of HOA '93 (Intern. Workshop on Higher Order Algebra, Logic and Term Rewriting, September 1993, Amsterdam).

(3) Process Algebra with Combinators, J.A. Bergstra, I. Bethke, A. Ponse, Report University of Amsterdam (to appear Sept. 93)

3.3 University of Edinburgh

3.3.1 Research directions

The Edinburgh group has primarily conducted research in the areas of π-calculus and action structures.

Action structures have been introduced as a mathematical structure which can underlie concrete models of computations. Uniform ways of building a process calculus from an action structure have been investigated. It has been shown that different formalisms, like λ-calculus, π-calculus, Petri Nets and certain higher-order calculi, can be recovered in such a way. This, for instance, allows a common treatment of behavioural equivalences.

Studies have been carried out on the development of the semantics of π-calculus, the main focus being the notion of (interleaving) bisimulation, and on the comparison of π-calculus with other formalisms, especially higher-order process calculi and the λ-calculus. The last item includes the study of the encoding of λ-calculus evaluation strategies and dialects into the π-calculus, and studies on the definability and the use of types in process calculi.

The theoretical work developed has constituted the foundational basis for PIC, an experimental programming language based on π-calculus, and on MWB, a tool to reason about behavioural equivalences of π-calculus processes.

3.3.2 Person and exchanges

The Edinburgh group involved in the project comprises Robin Milner, Davide Sangiorgi, Benjamin Pierce, Yoram Hirsfeld, David Turner and Peter Sewell. David Walker (Warwick) is an associated member. Davide Sangiorgi is full-time employed within the project.

David Turner and Peter Sewell are PhD students, under the supervision of Robin Milner whose theses are relevant to the CONFER project and are under completion in this period (provisory titles "Types and polymorphism in the π-calculus" and "Axiomatisation of higher-order finite state processes", respectively).

The study of π-calculus bisimulations and the development of the software tool MWB have been carried out in a collaboration with SICS. Benjamin Pierce has visited INRIA-Rocquencourt in the period October 1992—April 1993. Davide Sangiorgi has visited INRIA-Rocquencourt in the period October 1992—November 1992. These visits have had a strong impact on the work on PIC and on works (P4), (P6) and (P7) listed below. Davide Sangiorgi has visited Pisa in June 1993 to discuss the feasibility of the development of causal-sensitive semantics in calculi for mobile processes.

3.3.3 Publications

The results obtained by the group have given rise to the following publications (note that here we only list works for which the acceptance notice has already arrived):

P1.) *An action structure for synchronous π-calculus*, Milner, R. Proc. FCT Conference, Szeged, Hungary, LNCS 710, 1993.

P2.) *Action calculi, or concrete action structures*, Milner, R. Proc. MFCS Conference, Gdansk, Poland, LNCS 711, 1993.

P3.) *Algebraic Theories for Name-Passing Calculi*, Parrow, J. and Sangiorgi, D., To appear in the Proc. REX Summer School 1993, LNCS, Springer Verlag.

P4.) *Typing and Subtyping for Mobile Processes*, Pierce, B. and Sangiorgi, D., Proc. 8th LICS Conference, IEEE Computer Society Press, 1993.

P5.) *A Theory of Bisimulation for the π-calculus*, Sangiorgi, D., Proc. CONCUR '93, LNCS 715, Springer Verlag, 1993.

P6.) *From π-calculus to Higher-Order π-calculus — and back*, Sangiorgi, D., Proc. TAPSOFT '93, LNCS 668, Springer Verlag, 1993.

P7.) *An investigation into Functions as Processes*, Sangiorgi, D., to appear in the Proc. Ninth International Conference on the Mathematical Foundations of Programming Semantics (MFPS'93).

The papers (P1-3) represent invited papers.

3.4 ECRC

3.4.1 Outline

At the scientific level ECRC's involvement in CONFER in the first year has been centered around the development of the Facile language, its implementation, formal foundation, and its usage for development of a medium/large scale application.

A distributed implementation of the Facile environment has been realised by modifying and extending the Standard ML environment implemented at AT&T Bell Laboratories and Princeton University. The current implementation allows the development of applications that operate on a network of SPARC and Sun-3 workstations running UNIX and, to a limited extent, the Mach operating system.

A proposal for extending the formal foundation of Facile to cover the constructs for distributed programming has been worked out. An analysis of some of the constructs in Facile has been based on the notion of the CHemical Abstract Machine (CHAM), and a new polymorphic type system based on the notion of effects has been proposed. Furthermore, cooperation has been initiated with SICS on the issue of temporal logics that incorporate higher-order process passing for specifying Facile programs.

The medium/large scale application we have developed is a desktop conferencing tool (called Calumet) that supports meetings based on a slide presentation metaphor among users in different physical locations. The distributed part of the system, which handles all communications, has been built entirely with Facile.

At the administrative level ECRC has involvement in project management at the consortium level assisting Jean-Jacques Lévy in the technical coordination of the action. A visible result of this activity is the EATCS publication giving an overview of the CONFER project (R3).

3.4.2 Persons

The following ECRC personnel is engaged in the action: Alessandro Giacalone, Andre Kramer, Tsung-Min Kuo, Lone Leth and Bent Thomsen. Francois Cosquer has worked on the action until leaving ECRC at the end of 1992. Jean-Pierre Talpin has been involved in the action since 1st of September 1993.

Pierre Cregut (visitor), Sanjiva Prasad, Fritz Knabe (Ph. D. student), Philippe Marchal and Chris Crampton have also contributed to the development of Facile and the Calumet system.

3.4.3 Reports and Publications

A set of four reports has been produced as deliverables during Year 1 of CONFER:

ECRC/M1/R1: "Some Facile Chemistry", Technical report ECRC-92-14, 1992, by L. Leth and B. Thomsen. The first version of this technical report was finished before the official start of CONFER. A Journal version has been prepared and is currently under revision for publication. This version will be considered a deliverable for CONFER.

ECRC/M1/R2: "Some Issues in the Semantics of Facile Distributed Programming", Technical report ECRC-92-32, 1992, by B. Thomsen, L. Leth and A. Giacalone. The first version of this technical report was finished before the official start of CONFER. The version appearing in proceedings of the 1992 REX Workshop on "Semantics: Foundations and Applications", LNCS 666, Springer-Verlag, 1992 was prepared during the start of CONFER and will thus be considered a deliverable for CONFER.

ECRC/M1/R3: "Esprit Basic Research Action 6454-CONFER: CONcurrency and Functions: Evaluation and Reduction", description of the CONFER project in Bulletin of EATCS, Number 45, October 1992, pp.158-185, by J.-J. Lévy, B. Thomsen, L. Leth and A. Giacalone.

ECRC/M1/R4: "Polymorphic Sorts and Types for Concurrent Functional Programs", Technical report ECRC-93-10, 1993, by B. Thomsen.

The papers ECRC/M1/R1, ECRC/M1/R2 and ECRC/M1/R4 have been placed at the CONFER ftp site at Imperial College.

3.4.4 Software

The main thrust of work at ECRC during Year 1 of CONFER has been on the construction of two software systems:

Facile The implementation of Facile will be demonstrated at the first annual review, October 1st, 1993, at CWI, Amsterdam. We hope to be able to make the first release of Facile (Facile Antigua) freely available to the research community in the very near future. J. Glauert, at University of East Anglia, and E. St. James, at BULL, France, have received and installed earlier versions of Facile for testing.

Calumet We also hope to be able to demonstrate Calumet at the first CONFER annual review, with a CWI/ECRC connection.

We would like to point out that the above pieces of software are clearly not the results of the work delivered by just 1.5 FTE ESPRIT funding. It is the result of the involvement of the whole group at ECRC working on Facile and Calumet. It is our policy to make the results of the entire group available whenever possible.

3.4.5 CONFER exchanges

Members of the group have taken part in the two CONFER workshops held in Year 1. In the first workshop Alessandro Giacalone, Tsung-Min Kuo, Lone Leth and Bent Thomsen took part. Tsung-Min Kuo gave a talk about his work on introducing a notion of subtyping in Facile, and Bent Thomsen presented his work on a new polymorphic type system for Facile based on effect systems. At the second workshop Lone Leth and Bent Thomsen took part. Bent Thomsen gave an overview of the Facile Antigua implementation and a short presentation of the Calumet system.

On the 21st of May Lone Leth and Bent Thomsen visited the CONFER group at Imperial College and discussed the work on interaction categories.

Mads Dam from SICS visited ECRC from 5th to 9th of July, 1993. This visit was part of the cooperation which has been initiated with SICS on the issue of temporal logics that incorporate higher-order process passing for specifying Facile programs.

David Matthews affiliated with the Edinburgh CONFER group visited ECRC 13th and 14th of July, 1993. During his visit potential collaboration was discussed.

Roberto Amadio from CNRS and INRIA-Lorraine, Nancy, France, joined the group on the 6th of September, 1993, for a six month visit. He will be working on the formal foundation of Facile.

3.4.6 Non-CONFER visitors and visits

The group has been visited by a number of people outside the CONFER project: Scott Nettles, CMU/Fox project. Bernhard Steffen, RWTH Aachen, Germany. Jean-Pierre Talpin, ENS Paris. Mathias Felleisen, Rice University, USA. Jorge Cuellar, Siemens. Hanne Riis and Flemming Nielson, Aarhus University. Greg Morisset, CMU/Fox project, USA. Peter Lee, CMU/Fox project, USA. Dominique Bolignano, BULL. Jeannette Wing, CMU/Fox project, USA. Kohei Honda, Keio University, Japan.

Alessandro Giacalone, Andre Kramer and Bent Thomsen visited ANSA, Cambridge on the 6th of May. Bent Thomsen and Lone Leth visited Andy Pitts and Luke Ong at Cambridge University 19th and 20th of May, 1993. Two talks on Facile were given.

3.4.7 Perspective

Research progresses in all aspects of the groups involvement with CONFER. On *Sorts & Types* we expect to develop and implement a new type inference algorithm for Facile based on a polymorphic type system taking communications and other side effects into account. This type system will also have a notion of subtyping needed for programming in an object oriented style. done in the LOMAPS action.

On *Abstract Machines, Primitive Constructs* we expect to complete the work on Chemical Abstract Machines for Facile covering aspects of the distributed implementation.

The work on applications has pointed out that it will be beneficial to look at other communication paradigms such as broadcasting and asynchronous point-to-point.

On *Dynamic Behaviour* important results have recently been obtained in supporting the dynamic but type safe connection between different applications, which may have been independently compiled and activated. The result is important because this is going to be a rather frequent scenario in pervasively networked computing environments, and also because it appears to provide a solution to the more general question of “posting” and accessing resources on a network in a flexible and reliable fashion.

On *Programming Languages* we expect to finish a report on programming in Facile and a report on the definition of Facile Antigua release. A report describing the Calumet system is also in preparation. We have started experiments using tools developed in the CONCUR/CONCUR2 actions for formal specification and verification of essential parts of the Calumet system. In the area of distributed computing, continuing efforts are directed at making the implementation increasingly efficient and at experimenting with certain constructs needed to manage distributed applications (e.g. delay/time-out operators, operators for controlling the physical locations of processes, exception handling). An activity has recently begun to render communication between Facile processes more reliable through the introduction of low-level protocols that increase the tolerance of the system to partial hardware and software failures.

The CONFER work on Facile is (or will be) bringing results to the SEMAGRAPH working group, the LOMAPS and the COORDINATION ESPRIT BRA projects. The work in these projects is of a rather different nature from the work in CONFER. In the context of the SEMAGRAPH working group some results have been obtained in translating Facile into a low-level process model. It has been demonstrated how this relates to term-graph rewriting and results on execution in the graph rewriting language Dactl have been obtained. This work has recently proved to have a large potential for intersection with the work pursued at Edinburgh on programming with the π-calculus. The group at ECRC involved in CONFER will also take part in the LOMAPS BRA project. The focus of LOMAPS will be on data flow analysis and optimisations for concurrent functional languages and will thus use the result being produced in the CONFER context. A group at ECRC working on coordination systems is going to be involved in the COORDINATION BRA project. This group will use Facile for implementing the LO programming model.

3.5 ENS

3.5.1 Outline

The group at LIENS together with the associated researchers from Marseille, Nancy and Paris has produced significative contributions in the following research areas:

- Sequentiality and Game Semantics.
- Relationships among λ-calculus, π-calculus and Chocs.
- Optimal Reduction in the λ-calculus.

The group carries on mainly theoretical research in all areas of the CONFER project and collaborates actively with the other sites. In particular the work on calculi is carried on in strict collaboration with ECRC and INRIA-Sophia (Amadio is visiting ECRC for six months starting from September 1993 and G. Boudol is directing Lavatelli's thesis). The work on games is carried on in collaboration with Imperial College (Abramsky, Jagadeesan, Lamarche (formerly at LIENS)) and the work on optimal reduction is carried on in collaboration with INRIA-Rocquencourt (Asperti, Gonthier, Lévy) and INRIA-Sophia (Laneve).

3.5.2 Persons

The following researchers are engaged in the action: Pierre Louis Curien (CNRS, LIENS), Roberto Amadio (CNRS-INRIA, Nancy), Vincent Danos (CNRS, Paris VII), Carolina Lavatelli (PhD Student, LIENS) and Laurent Regnier (CNRS, Marseille).

3.5.3 Reports and Publications

A set of four reports has been produced as deliverables during Year 1 of CONFER:

1. R. Amadio, On the Reduction of Chocs Bisimulation to π-calculus Bisimulation, in Proc. CONCUR 93, E. Best (ed.), SLNCS 715. Also appeared as Research Report Inria-Lorraine 1726.
2. P.-L. Curien, On the Symmetry of Sequentiality, presented at the Conference on Mathematical Foundations of Program Semantics, to appear in the Proceedings of the Conference.
3. V. Danos and L. Regnier, Local and Asynchronous Beta-Reduction, in Proc. IEEE-LICS 93, Montreal.
4. C. Lavatelli, Non deterministic lazy λ-calculus vs. π-calculus, Technical Report LIENS 93-15, September 1993.

Deliverables 1 and 4 relate to the *Calculi* area. Deliverable 2 relates to the *Logics for concurrency and lambda calculus* area. Deliverable 3 relates to the *Foundational models and abstract machines* area.

3.5.4 Description of Technical Contributions and Related Work

1. Roberto Amadio has pursued the analysis of the relationship between Chocs and π-calculus. These are two natural extensions of CCS where, respectively, processes and channels are transmissible values. In previous work he had proposed a formalization of the notion of bisimulation for Chocs. His new contribution is a more effective way to reason about this notion by means of an embedding of Chocs into a richer calculus endowed with a notion of 'activation' channel which is christened $Chocs_t$. t is the name of a new internal action which is produced by a synchronization on an activation channel, such a synchronization has the effect of forcing the execution of an idle process. In first approximation transitions in $Chocs_t$ may be understood as sequences of synchronizations along activation channels followed by an 'observable' transition. There is a simple definition of bisimulation for $Chocs_t$ which satisfies natural laws and congruence rules, moreover the synchronization trees associated to $Chocs_t$ processes are finitely branching. $Chocs_t$ is proposed as an intermediate step towards the definition of a tool for the verification of Chocs bisimulation.

   Davide Sangiorgi from the University of Edinburgh has independently obtained a reduction of a restricted form of Chocs where sums are 'guarded' to the standard π-calculus (using weak bisimulation). Although these works share similar motivations their technical approaches are quite distinct.

2. Pierre-Louis Curien has proposed a symmetric account of sequentiality, by means of *symmetric algorithms*, which are pairs of sequential functions, mapping data to data, and output exploration trees to input exploration trees, respectively. The framework of *sequential data structures* is used, which is indeed a reformulation of a class of Kahn-Plotkin's concrete data structures. Sequential data structures and symmetric algorithms are the objects and morphisms of a symmetric monoidal closed category, which is also cartesian, and is such that the unit is terminal.

   The category obtained is a full subcategory of categories of games considered by Lamarche, and by Abramsky-Jagadeesan, respectively. This work, while finding its roots in the study of sequentiality, presents striking correspondences with game-theoretic concepts, introduced by Blass in the early seventies in a very different context.

3. Vincent Danos and Laurent Regnier have presented a new way of computing lambda-terms. They have provided an embedding of lambda-terms into the so-called *virtual nets*, which are graphs labelled by some coefficients of the *dynamic algebra*. In particular they have defined and studied a notion of *virtual reduction* which is a graph reduction based on the labelling of the virtual net. A single step of virtual reduction consists in composing two edges of the graph so that the product of their labels is non null in the dynamic algebra. In some sense, virtual reduction is the computation of the transitive closure of the virtual net. It is shown to be confluent and preserving the *execution formula* of Girard. This last property makes virtual reduction a good candidate for computing lambda-terms.

The notion of virtual reduction comes from Girard's program on the *geometry of interaction*. It is strongly linked with the sharing reduction techniques and the work of Abadi, Gonthier and Lévy on Lamping's *sharing graphs*. This relation is at the moment under study in collaboration with Asperti and Laneve. In some sense virtual reduction is another way, may be more abstract, to present the sharing graphs.

4. Carolina Lavatelli has examined and compared various lambda calculi with parallel and convergence testing. The λ_j-calculus, a lazy calculus augmented with a non-deterministic choice operator and a convergence testing combinator, has emerged as a suitable language to be encoded into the π-calculus. The substitution process in λ_j is managed in a semi-explicit way via the use of closures for variables and abstractions. The semantics associated to both λ_j and π are based on contextual testing preorders. An encoding of λ_j into π is defined and it is proved adequate with respect to those semantics. However, the encoding is not fully-adequate. Standard examples show that π is still more discriminating than λ_j.

   This work builds on previous results by Milner, Sangiorgi and Thomsen, among others, which analyse the expressive power of π-calculus. In these studies it was observed that the standard translation of the lazy λ-calculus into π-calculus is adequate (but not fully adequate) w.r.t. to suitable notions of bisimulation. Later, Sangiorgi showed that adding to the lazy λ-calculus a non-deterministic operator and providing it with a strong notion of testing equivalence (which behaves very much like a bisimulation equivalence) is enough to get full adequacy. Lavatelli's work shows that whenever the semantics is based on Morris' contextual testing equivalence then non-determinancy is not the only extra-feature which distinguishes λ-calculus from its π-calculus embedding.

3.5.5 Perspective

Roberto Amadio will collaborate with the group at ECRC on the formal definition of the Facile semantics. In collaboration with a student (O. Ait-Mohamed) he has also started to investigate the design of a verification tool for mobile calculi based on the HOL system. Pierre-Louis Curien will spend a sabbatical year at the University of Beijing, starting from September 1993. There he will pursue his study on the relationship between sequentiality and game semantics. The latter in particular has recently provided a new perspective on the full abstraction problem for PCF. Vincent Danos and Laurent Regnier will pursue the analysis of the pragmatic value of their notion of local and asynchronous reduction. Carolina Lavatelli will continue to develop her PhD thesis on the relationship between λ-calculus and π-calculus from the point of view of a testing semantics. This work is carried on under the direction of G. Boudol and P.L. Curien.

3.6 Imperial College

3.6.1 Research directions

Research done at Imperial College has covered all four areas of the CONFER project, but the main focus has been on Logics for Concurrency and λ-calculus with the main thread being the close correspondence between proofs and processes.

Interaction Categories have been proposed as a foundation for semantics of computation. The main examples of Interaction Categories are **SProc** and **ASProc** which are categories for synchronous and asynchronous processes. Much work has been done on investigating applications of these categories which has two benefits: to see how useful the framework is for semantics; and to obtain feedback on what structures are essential for the categories in the first place. Such application work is carried out in the research effort on real-time synchronous languages such as LUSTRE, SIGNAL and ESTEREL and on the development of a type system for concurrency.

Related to the work on types is the research on sort inference for the π-calculus which uses a weaker notion but is able to handle the notion of *mobility* more directly than it is currently possible in the Interaction Category framework. Also connected is the work on the translation of the π-calculus into Interaction Nets. The true concurrency semantics developed for the π-calculus also fits in nicely in this context.

On the functional side, there is the work on Linear Logic which discusses the notion of a term calculus for Intuitionistic Linear Logic (ILL) and shows that the internal language for autonomous categories corresponds to this term calculus. A prototype functional programming language based on this term calculus, called LILAC, has been implemented. Research has also been done on optimal reduction in the λ-calculus using interaction net implementations. There is additional work in this area — on proof nets for the multiplicative fragment of ILL; and on non-determinism in a functional setting.

3.6.2 Person and exchanges

The group involved in the project who are attached to this site comprises Samson Abramsky, Simon Gay, François Lamarche, Ian Mackie, Luke Ong and Rajagopal Nagarajan. Radhakrishan Jagadeesan, who has recently left the college, was also one of the active participants.

Simon Gay, Rajagopal Nagarajan and Ian Mackie are PhD students, under the supervision of Samson Abramsky. Their theses are directly relevant to the CONFER project. Respective provisional titles:

- Linear Types for Communicating Processes
- Specification and Verification of Typed Concurrent Programs
- Linear Logic and Functional Programming

Ian Mackie is visiting Jean-Jacques Lévy and his associates at Ecole Polytechnique, Paris and INRIA, Rocquencourt from September to December of this year to do collaborative work. François Lamarche has arrived at Imperial from Ecole Normale Supérieure after the start of CONFER. He has influenced and has been influenced by the work at ENS, especially that of Pierre-Louis Curien. Luke Ong has paid regular visits to Imperial and has played an active rôle in the joint seminars between Imperial College and University of Cambridge. The work on semantics of concurrency has strong connections with research at the University of Edinburgh and we have had a fruitful exchange of ideas. Our site has had a significant representation in the CONFER workshops held so far and this has resulted in active collaboration.

3.6.3 Publications

The results obtained by the group have given rise to the following publications (note that here we only list works for which the acceptance notice has already arrived):

[1] S. Abramsky. *Interaction Categories* (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods.* Springer-Verlag Workshops in Computer Science, 1993. To appear.

[2] S. Abramsky. *Interaction Categories and communicating sequential processes.* In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare.* Prentice Hall International, 1994. To appear.

[3] S. J. Gay. *A sort inference algorithm for the polyadic π-calculus.* In *POPL 93.* ACM Press, 1993.

[4] S. J. Gay and R. Nagarajan. *Modelling* SIGNAL *in Interaction Categories.* In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods.* Springer-Verlag Workshops in Computer Science, 1993. To appear.

[5] I. Mackie. Lilac: *A Functional Programming Language Based on Linear Logic.* To appear in the *Journal of Functional Programming.*

[6] I. C. Mackie, L. Román and S. Abramsky. *An Internal Language for Autonomous Categories.* Journal of Applied Categorical Structures 1993 (to appear)

[7] L. Ong. *Non-determinism in a functional setting.* In Proceedings of LICS 1993.

3.7 INRIA-Rocquencourt

3.7.1 Outline

The group at INRIA-Rocquencourt is involved in the study of syntactic properties of various λ-calculi. It had some results in the theory of optimal reductions, and is interested in abstracting properties of computing agents with bound variables (interaction systems, abstract reduction systems).

Implementations of these computing models have also been considered through $\lambda\sigma$-calculi and within ML (lazy or eager). A portable real-time realistic parallel garbage collection for CAML-light has been proved correct.

3.7.2 Persons and exchanges

The group at Rocquencourt is formed by Andrea Asperti, Damien Doligez, Georges Gonthier, Thérèse Hardin, Jean-Jacques Lévy, Luc Maranget, Paul-André Melliès and Didier Rémy. In CONFER, Jean-Jacques Lévy is the project coordinator.

Damien Doligez and Paul-André Melliès are PhD students from Ecole Normale Supérieure. Andrea Asperti is now professor at University of Bologna. Thérèse Hardin is professor at University of Paris 6. Other researchers are full members of INRIA. Georges Gonthier and Jean-Jacques Lévy have part-time teaching positions at Ecole polytechnique.

Work on optimal reductions is by Andrea Asperti, Georges Gonthier, Jean-Jacques Lévy and Luc Maranget; $\lambda\sigma$-calculus by Thérèse Hardin and Jean-Jacques Lévy and Luc Maranget; implementation of lazy ML by Luc Maranget; implementation of optimal reductions by Andrea Asperti; garbage collection by Damien Doligez (implementation and proof) and Georges Gonthier (proof).

Visitors of INRIA Rocquencourt were Cosimo Laneve (3 months) from Pisa, Benjamin Pierce (7 months) and Davide Sangiorgi (3 months) from Edinburgh, Ian Mackie (2 months) from Imperial College. Zena Ariola, who is now working with Jan-Willem Klop on a regular basis, spent 91 and 92 at INRIA Rocquencourt.

3.7.3 Publications

[INRIA/Rocq/M1/1] A. Asperti, *Linear Logic, Comonads, and Optimal Reductions*, Fundamenta Informaticae, Special Issue devoted to "Categories in Computer Science", Polish Academy of Sciences (invited paper). To appear.

[INRIA/Rocq/M1/2] A. Asperti, C. Laneve, *Paths, Computations and Labels in the λ-calculus*, Proc. of the 5th International Conference on Rewriting Techniques and Applications, RTA'93, Montreal. June 1993.

[INRIA/Rocq/M1/3] A. Asperti, C. Laneve, *Interaction Systems*, HOA'93. International Workshop on Higher-Order Algebra, Logic and Term Rewriting. Amsterdam, September 1993.

[INRIA/Rocq/M1/4] A. Asperti, C. Laneve, *Optimal Reductions in Interaction Systems*, Proc. of the 4th Joint Conference on the Theory and Practice of Software Development, TAPSOFT'93, Orsay (France). April 1993.

[INRIA/Rocq/M1/5] A. Asperti, C. Laneve, *Interaction Systems I: the theory of optimal reductions*, Rapport Technique 1748, INRIA-Rocquencourt. Submitted to Mathematical Structures in Computer Science. 1992.

[INRIA/UB/M1/6] A. Asperti, C. Laneve, *Interaction Systems II: the practice of optimal reductions.* Technical Report UBLCS-93-12, Laboratory for Computer Science, University of Bologna. Submitted to Theoretical Computer Science. 1993.

[INRIA/Rocq/M1/7] P.-L. Curien, T. Hardin, A. Rios, *Strong normalisation of Substitutions*, MFCS, Prague, 1992.

[INRIA/Rocq/M1/8] P.-L. Curien, T. Hardin, *Yet yet a counterexample for λ-calculus+SP*, Journal of functional Programming. to appear.

[INRIA/Rocq/M1/9] G. Gonthier, M. Abadi, J.-J. Lévy, *Linear Logic without Boxes*, 7th LICS, Santa Cruz, 1992.

[INRIA/Rocq/M1/10] G. Gonthier, M. Abadi, J.-J. Lévy, *Linear Logic without Boxes*, Accepted for the special LICS issue of Information & Computation, to be submitted to this journal.

[INRIA/Rocq/M1/11] G. Gonthier, J.-J. Lévy, P.-A. Melliès, *An abstract standardisation theorem*, 7th LICS, Santa Cruz, 1992.

[INRIA/Rocq/M1/12] T. Hardin, *η-reduction for the languages of Explicit Substitutions*, Algebraic and Logic Programming Conference, Volterra, Italy, August 1992.

[INRIA/Rocq/M1/13] T. Hardin, *From Categorical Combinators to $\lambda\sigma$-calculi: a quest for confluence*, INRIA Report 1777 - November 1992.

3.8 INRIA-Sophia

3.8.1 Outline

The INRIA group at Sophia Antipolis has done research mainly in the areas of Calculi and Foundational Models and Abstract Machines. The common aim of the group is to study and develop theoretical frameworks dealing with parallel computations. A special emphasis has been put on the lambda-calculus and related systems. We have pursued the work on optimal computations, obtaining a new characterization of the notion of family of redexes, which allows us to prove the correctness of the implementation of optimal computations. We also have introduced a refinement of the lambda-calculus, offering a sharp control on the process of substitution.

3.8.2 Persons

The following persons are involved in the project at Sophia Antipolis: G. Berry (Research Director, Ecole des Mines de Paris), G. Boudol (Research Director, INRIA), I. Castellani (Researcher, INRIA).

The group also includes C. Laneve, who has a post-doc position at INRIA from January 1st, 1993, and Ch. Retore, who will have a similar position, starting from October 1st, 1993. These positions are funded by the CONFER project.

3.8.3 Publications, Interactions

The INRIA group at Sophia Antipolis organized the first CONFER Meeting in January 1993. This was attended by 40 participants, mostly from the various sites of the project. On this occasion, C. Laneve gave a talk on "Interaction systems". G. Boudol and I. Castellani attended the second CONFER workshop in Edinburgh.

G. Boudol made two one-week visits in Paris, Ecole Normale Superieure, in March and May, 1993. He is working there with a PhD student, C. Lavatelli, who is also a member of the project. C. Laneve has written several joint papers with A. Asperti (University of Bologna), and is currently working with him and L. Régnier (LMD, Marseille. Régnier is formally associated with the ENS group in Paris).

G. Boudol has given an invited talk at the REX workshop, "A Decade of Concurrency" (Amsterdam, June 1993), on "The Chemical Abstract Machine", and an invited talk at the CONCUR'93 Conference (Hildesheim, August 1993), on "The Lambda-Calculus with Multiplicities". C. Laneve presented his work on "Optimal Reductions" at the TAPSOFT'93 Conference (Orsay, April 1993), and he gave a talk on "Path, Computations and Labels in the Lambda-Calculus" at the RTA'93 Conference (Montreal, June 1993).

3.9 Università di Pisa

3.9.0.1 Outline

The research activities of Pisa site in the first year of the project have been focussed on the following subjects:

- Concurrency and Optimality,
- Mobility in the cc Paradigm,
- Explicit Substitution for Mobile Processes,
- History Dependent Behaviours.

Studies have been carried out on the analysis of the relationships between strategies for optimal reduction and concurrency. A complete axiomatization of permutation equivalence has been provided.

It has been shown that cc-languages get higher order power even with a very poor underlying constraint system. This includes the study of the encoding of λ-calculus evaluation strategies into the cc framework.

An alternative but equivalent formulation of the π-calculus has been introduced. In this formulation name instantiation is handled explicitly. It is proved that π-calculus behavioural equivalences are retrieved and axiomatized by giving the description of the corresponding strategies for name instantiation.

Finally, it has been shown that many interesting issues of process calculi can be captured and explained by exploiting the history dependent paradigm.

3.9.0.2 Perspective

The alternative formulation of the π-calculus where name instantiation is handled explicitly provides the basis to understand and design abstract machines for the π-calculus. This work can constitute the theoretical basis to develop semantic-based verification tools for the π-calculus. Moreover, the issue of defining non interleaving semantics for the π-calculus can take advantage of the explicit handling of name instantiation. Further works are planned on these subjects.

Finally, the CHARM (Concurrency and Hiding in Abstract Rewriting Machine) framework presents some similarities with the Action Structures framework. We plan to investigate the relationships between CHARM and Action Structures.

3.9.0.3 Pisa CONFER Members

Ugo Montanari, GianLuigi Ferrari, Cosimo Laneve, Paola Quaglia.

3.9.0.4 Personnel Changes and Visitors

Cosimo Laneve originally from Pisa is now in Sophia Antipolis (INRIA) for one year. Paola Quaglia joined the PhD program of the Dipartimento di Informatica, Università degli Studi di Pisa (Supervisor: Prof. Ugo Montanari). Dr. J. Meseguer visited Pisa for one week in June.

3.9.0.5 Interactions with other CONFER Sites

Besides the partecipation to the CONFER workshops, Montanari, Ferrari and Quaglia visited Edinburgh in May 93 (before the second CONFER workshop). This visit has been found very useful for understanding and comparing the relative approaches. Davide Sangiorgi (Edinburgh) has been invited and visited Pisa in December 92, and June 93. Roberto Amadio (INRIA) visited Pisa in December 92.

3.9.0.6 PhD Thesis

Cosimo Laneve discussed his PhD thesis (Supervisor Prof. Ugo Montanari). The thesis studies the problems of optimality and concurrency in a class of higher order rewriting systems called *Interaction Systems*.

3.9.1 List of Reports

[D/Pisa/M1/1] Ferrari, G., Montanari, U., Quaglia, P., *The π-calculus with Explicit Substitutions*, Submitted for publication, 1993.
[D/Pisa/M1/2] Ferrari, G., Montanari, U., *Observing Time-Complexity of Concurrent Programs*, 1993.
[D/Pisa/M1/3] Laneve, C. *Distributive Evaluations of λ-calculus*, To appear in Acta Informaticae.
[D/Pisa/M1/4] Laneve, C., *Optimality and Concurrency in Interaction Systems*, PhD Thesis TD-8/93, Dipartimento di Informatica, Università di Pisa, 1993.
[D/Pisa/M1/5] Laneve, C., Montanari, U., *Mobility in the cc Paradigm.* Preliminary version in MFCS'92, LNCS 629, 1992.
[D/Pisa/M1/6] Laneve, C., Montanari, U., *Axiomatizing Permutation Equivalence*, Submitted for publication 1993. Preliminary version in ALP'92, LNCS 632, 1992.

3.9.2 Short Abstracts of Reports

[D/Pisa/M1/1] Ferrari, G., Montanari, U., Quaglia, P., *The π-calculus with Explicit Substitutions.*
A new formulation of the π-calculus, where name instantiation is handled explicitly, is presented. Behavioural equivalences originally developed for the π-calculus are retrieved by giving the description of the corresponding strategies for name instantiation, and, in each case, fully abstract semantics, with respect to the appropriate notion of observation of process behaviours, are obtained. The explicit handling of name instantiation allows us to take advantage of the SOS meta-theory developed for "static"

process calculi. Thus, axiomatic characterizations of behavioural equivalences can be automatically derived by analyzing the syntactic structure of the SOS inference rules. This paper makes a first step towards the design of abstract machines for concurrent programming languages based on the π-calculus. Here, we deal with the π-calculus, however the techniques we develop can be applied to value passing process calculi as well.

[D/Pisa/M1/2] Ferrari, G., Montanari, U., *Observing Time-Complexity of Concurrent Programs.*
We develop a semantic framework to describe and reason about the time-complexity of concurrent programs. To this purpose, we introduce a modular approach to the semantics of concurrent languages where the functional part of programs is handled together with the complexity part. We show that the time-complexity analysis of concurrent programs can be naturally dealt with a specialized matrix calculus.

[D/Pisa/M1/3] Laneve, C. *Distributive Evaluations of λ-calculus.*
In this paper we address the problem of encoding evaluation strategies for the λ-calculus into prime event structures. In order for this to be possible the derivation spaces yielded by the evaluation mechanism must be prime algebraic cpo's. This requirement is not met by permutation equivalence, the standard concurrent semantics with which λ-calculus is equipped. We solve this problem by taking the coarsest congruence contained in permutation equivalence such that permutations of disjoint reductions are equated and the downward closure of every derivation is a distributive lattice. This equivalence, called *distributive permutation equivalence*, is characterized directly by restricting permutations of redexes to those sets U which are distributive, i.e. for every $u \in U$, the development of every $V \subseteq (U \setminus \{u\})$ does not duplicate or delete u. A simple consequence of our results is that the derivation spaces of the call-by-value λ-calculus are distributive lattices. Finally, we show that a sequential evaluation mechanism can not, in general, be effectively transformed into a maximally distributive one.

[D/Pisa/M1/4] Laneve, C., *Optimality and Concurrency in Interaction Systems.*
The thesis studies the problems of optimality and concurrency in a class of higher order rewriting systems: the Interaction Systems. On one side these systems provide the intuitionistic generalization of Lafont's Interaction Nets (that are linear), by keeping the idea of binary interaction and the syntactical bipartition of operators into constructors and destructors. On the other side, Interaction Systems are a suitable subclass of Klop's Combinatory Reduction Systems where the Curry-Howard analogy "still makes sense". Namely, it is possible to consider constructors and destructors of Interaction Systems respectively as right and left introduction rules of intuitionistic systems, interactions as instances of cut-rules and computations as eliminations of cut-rules. In the first part of the thesis, we generalize the standard theory of optimality to Interaction Systems. In particular we define the notion of optimal sharing by means of two different approaches and prove their equivalence. Then we provide an implementation extending Lamping's graph reduction technique for the λ-calculus and fulfilling the optimality criteria. In the second part of the thesis, we study permutation equivalence in the framework of

Interaction Systems. This equivalence formalizes the notion of parallel reduction, which is essential in the theory of optimality. Foremost we provide a complete axiomatization of permutation equivalence. Then, by taking λ-calculus as running example, we study the issues of implementing permutation equivalence on Winskel's Event Structures, a mathematical model of distributed systems.

[D/Pisa/M1/5] Laneve, C., Montanari, U., *Mobility in the cc Paradigm.*
We prove that cc-languages get higher order power even with a very poor underlying system of constraints (the signature consists of a constant and a concatenation operation). The turning point is to observe that the phenomenon of mobility is already present in the cc-paradigm as *mobility of variables.* Therefore, by simulating π-calculus channels in terms of *streams*, it is possible to rephrase Milner's encodings of *lazy* and *call-by-value* λ-calculus into the π-calculus. Our encodings distinguish between channels that are used once, just for book-keeping reasons, and those representing variables in Milners's encodings, thus yielding a clearer distinction between resources.

[D/Pisa/M1/6] Laneve, C., Montanari, U., *Axiomatizing Permutation Equivalence.*
We axiomatize *permutation equivalence* in term rewriting systems and Klop's orthogonal left-normal Combinatory Reduction Systems. The axioms for the former ones are provided by the general approach proposed by Meseguer. The latters need extra axioms modelling the interplay between reductions and the operation of substitution.

3.10 SICS

3.10.1 Outline

The group at SICS has conducted research in all four main areas of the CONFER project. We have studied axiomatisations of name-passing calculi, primitives of interaction, tableau-based model checking for a temporal logic adapted to the π-calculus, decision procedures for bisimulation equivalences in the π-calculus, program unfolding in a concurrent setting, and software verification tools for the π-calculus.

The overall aim of the group is to develop practically useful theories and tools for the specification and verification of communicating systems. Of particular relevance to the CONFER project are systems that involve elements of mobility, access control, or dynamic reconfiguration capabilities. Associated groups at SICS are involved in case-studies, using specification languages based on ideas from the π-calculus to model and verify slot allocation protocols for high-speed optical networks.

3.10.2 Personnel

The following SICS personnel is engaged in the action: Joachim Parrow, Mads Dam, Björn Lisper, Björn Victor. The group has been joined from July 1, 1993, by Lars-Henrik Eriksson. Up the the end of June 1993 the group had devoted 1.507 hours to the project.

3.10.3 Diffusion and Exchanges

The results of the group have been presented at several occasions: An invited tutorial at the REX'93 workshop and summer school (J. Parrow), an accepted paper at CONCUR'93 (M. Dam), and presentations at the 1st and 2nd CONFER workshops (M. Dam, B. Lisper, J. Parrow).

An extensive collaboration between Joachim Parrow and Davide Sangiorgi (Edinburgh) has been conducted through electronic mail. As part of the action M. Dam visited ECRC from July 5 to July 9 1993, and B. Victor visited Edinburgh from July 8 to August 28.

3.10.4 Publications, Reports

[L93A] B. Lisper. "Total unfolding: theory and applications". Accepted for publication in Journal of Functional Programming.

[P93] J. Parrow. "Interaction Diagrams". Draft paper, presented at REX'93 workshop and summer school.

[PS93] J. Parrow and D. Sangiorgi: "Algebraic Theories for Name-Passing Calculi", SICS Research Report R93:04, 1993.

[D93] M. Dam. "Model Checking Mobile Processes". In Proc. CONCUR'93, LNCS 715, pp. 22-36.

[L93B] B. Lisper. "Unfolding of Programs with Nondeterminism and Processes (tentative title)". In preparation (should be available in time for review).

3.10.5 Software

In cooperation with Faron Moller and Davide Sangiorgi, Björn Victor has begun the development of the Mobility Workbench (MBW) — a tool for manipulating and analyzing mobile concurrent systems described in the π-calculus [MPW92].

3.10.5.1 Perspective

Important research directions include the following areas: Primitives of interaction need to be investigated both from the points of view of theoretical expressiveness and of usefulness in practical applications. Temporal logics based proof systems and model checkers need to be extended and adapted to higher-level communication and programming primitives. Decidability, axiomatisation, and program transformation issues will be further addressed, and software specification and verification tools embodying progress in these areas are under ongoing development and need to be exposed to practical applications.

Chapter 4

Deliverables

4.1 Workshop 1

The workshop was organised by Gérard Boudol at INRIA Sophia-Antipolis. (40 participants)

Tuesday, January 19

9.00-10.00	Registration
10.00-10.30	Opening : J-J Lévy
10.30-11.00	Coffee Break
11.00-12.00	R. Amadio - A Uniform Presentation of CHOCS and Pi-Calculus - Environment Machines and Pi-Calculus
12.00-13.00	G. Ferrari - U. Montanari Nonincremental Observation Algebras for Time-Complexity
13.00-14.30	Lunch
14.30-15.30	R. Milner Action Structures and the Pi-Calculus
15.30-16.00	Coffee Break
16.00-17.00	D. Sangiorgi On the Representation of Functions as Processes
17.00-17.30	B. Pierce - D. Sangiorgi Typing and Subtyping for Mobile Processes
17.30-18.00	D.N. Turner Sorts and Polymorphism in the lambda-calculus

Wednesday, January 20

9.00-10.00	S. Abramsky Interaction Categories
10.00-10.30	R. Jagadeesan Game Semantics
10.30-11.00	Coffee Break
11.00-12.00	S. Gay - R. Nagarajan Working With Interaction Categories
12.00-13.00	G. Gonthier Linear Logic - without Boxes
13.00-14.15	Lunch
14.15-14.45	I. Mackie Linear Logic and Optimal Reductions in the lambda-calculus
14.45-15.30	L. Ong Non-Determinism in a Functional Setting
15.30-16.00	Coffee Break
16.00-16.45	L. Regnier A Local and Asynchronous Reduction of lambda-calculus
16.45-17.30	A. Asperti - C. Laneve Paths, Computations and Labels in the Lambda-Calculus

Thursday, January 21

9.00-9.45	B. Thomsen Polymorphic Sorts and Types for Concurrent Functional Programs
9.45-10.30	C. Laneve - A. Asperti Optimal Reductions in Interaction Systems
10.30-11.00	Coffee Break
11.00-12.00	M. Dam A temporal Logic for the Lambda-Calculus
12.00-13.00	T.M. Kuo A Simple Subtype System for FACILE
13.00-14.00	Lunch
14.15-15.00	B. Lisper Termination Properties of Program Rewriting
15.00	End of the Workshop

4.2 Workshop 2

The workshop was organised by Davide Sangiorgi at University of Edinburgh. (36 participants).

Monday May 24

9.30-10.00	Opening
10.00-10.45	D. Walker [Warwick,UK] (Higher-Order pi-calculus and object-oriented languages)
10.45-11.15	Break
11.15-12.00	J.W. Klop [CWI,Holland] (Pi-calculus as a Combinatory Reduction System)
12.00-12.30	R. Jagadeesan [Imperial College,UK] (Processes as "Sets of Functions")
12.30-13.00	I. Mackie [Imperial College,UK] (An Internal Language for Autonomous Categories)
13.00-14.15	Lunch
14.15-15.00	S. Abramsky, S. Gay, R. Nagarajan [Imperial College,UK] (Interaction Categories: Illustrative Examples)
15.00-15.45	Lamarche [Imperial College,UK] (On the Proof Net Problem for Additives in Linear Logic
15.45-16.15	Break
16.15-17.00	L. Ong [Cambridge,UK] Fair games are fully complete for multiplicative linear logic without the MIX-rule
17.00-17.45	P.-L. Curien [CNRS,Paris] (On the symmetry of sequentiality)
17.15-18.15	L. Regnier [CNRS,France] (Virtual reduction: a mechanization)

Tuesday May 25

9.30-10.15	B. Pierce [Edinburgh,UK] (programming in the pi-calculus)
10.15-11.00	L. Leth, B. Thomsen [ECRC, Germany] (implementation of FACILE)
11.00-11.30	Break
11.30-13.00	Discussion on implementation issues
13.00-14.15	Lunch
14.15-17:00	 Planning for the future of CONFER, chaired by J.-J. Lévy [INRIA-Rocquencourt, France] (half an hour break at 15.30 or so for the coffee)

Wednesday May 26

9.30-10.00	Y. Hirshfeld [Edinburgh,UK] (Concrete action structures)
10.00-10.45	J. Parrow [SICS, Sweden] (Algebraic theories for name passing calculi)
10.45-11.15	Break
11.15-12.00	U. Montanari [Pisa, Italy] (Late Bisimulation as History Dependent Bisimulation)
12.00-12.30	D. Sangiorgi [Edinburgh,UK] (A theory of bisimulation for pi-calculus)
12.30-13.00	possibly, some closing remarks or others things..
13.00-14.15	Lunch and closure of the meeting

4.3 Software deliverables

Several pieces of software have been constructed during the first year of CONFER. Some of them have already been made available via the CONFER ftp site at Imperial College while others are expected to be made available in the near future. This is rather impressive since software deliverables are only planned for at Milestone 3. The following is a listing of constructed software.

- Facile programming language
  ECRC — A. Giacalone, F. Cosquer, F. Knabe, A. Kramer, T.M. Kuo, L. Leth, S. Prasad, B. Thomsen. Also with contributions of P. Cregut, J.P. Talpin and C. Crampton.

- Prototype compiler for λ-calculus, based on graph reduction
  INRIA — A. Asperti.

- Portable, unobtrusive garbage collection for multiprocessor systems
  INRIA — D. Doligez, G. Gonthier, J.J. Lévy.

- Lilac: a prototype functional programming language based on Linear Logic
  Imperial College — I. Mackie.

- Typed higher-order programming language based on π-calculus
  University of Edinburgh — B. Pierce, D. Rémy, D. Turner.

- The Mobility Workbench (MBW) — a tool for manipulating and analyzing mobile concurrent systems described in the π-calculus
  University of Edinburgh — Faron Moller, Davide Sangiorgi, SICS — Björn Victor.

Some of these pieces of software will be demonstrated at the CONFER workshop at CWI in Amsterdam and at the Annual review.

As mentioned software deliverables are only due at Milestone 3 at which point the descriptions will be provided as stipulated in the technical annex.

Chapter 5

Progress

Reports are done along the 4 areas announced in page 4 of the technical annex.

5.1 Foundational models and abstract machines

The area covers properties of basic calculi with bound variables from the λ-calculus to higher-order communication systems. Work in the area is often based on Lafont's concept of interaction net. The area includes more general schemes such as combinatory reduction systems, action structures and abstract reduction systems. The area report includes short summaries of the following efforts:

- Explicit substitutions
  T. Hardin, INRIA Rocquencourt

- Sharing in Linear Logic and the λ-calculus
  A. Asperti, G. Gonthier, J.-J. Lévy, INRIA Rocquencourt
  I. Mackie, Imperial College
  C. Laneve, University of Pisa, INRIA Sophia,
  V. Danos and L. Régnier, ENS

- Interaction systems
  A. Asperti, INRIA Rocquencourt (University of Bologna),
  C. Laneve, University of Pisa, INRIA Sophia,

- Interaction diagrams
  J. Parrow, SICS

- π-calculus and interaction nets
  S. J. Gay, Imperial College

- Chemical Abstract Machines
  L. Leth, B. Thomsen, ECRC

- Action Structures
  R. Milner, University of Edinburgh
- Combinatory Reduction Systems
  J. W. Klop, F. van Raamsdonk, F. de Vries
- Abstract reduction systems
  G. Gonthier, J.-J. Lévy, P.-A. Melliès, INRIA Rocquencourt

5.1.1 Lambda sigma calculus

This work was initialised by Curien (1983) and Hardin (1986) with categorical combinators. Some side-effect of this work was the formal definition of the abstract machine of the INRIA implementation of ML (CAML). In 1990, an alternative framework, *explicit substitutions*, was proposed by Abadi et al. Basically it is the same except that expressions are two sorted (terms and environments). It is still difficult and complicated to prove confluence and termination. But these proofs are more natural. Explicit substitutions are a very traditional implementation model of beta conversion, since functional languages are usually implemented with stacks and environments. In his thesis, Maranget [35] gave full proofs of the correctness and the optimality of implementations of weak lazy λ-calculus, by using explicit substitutions.

Hardin treats the case of connections with the previous calculus of categorical combinators [13], the extensionality rule [29]. Curien and Hardin [25] have a easier counterexample to the well-known surjective pairing confluence problem, first discovered by Klop. A great part of the work has been done before the official start of the project. We have developed two works published during CONFER:

5.1.1.1 From Categorical Combinators to $\lambda\sigma$-calculi: a quest for confluence

This is work presented by Hardin [13]. The λ-calculus is known to be the theoretical base of functional programming languages. But the substitution, which describes the replacement of procedure parameters, belongs only to the meta-language and this is a major drawback when dealing with compilation. Therefore, extensions of the λ-calculus, able to manipulate explicitly substitutions, and still confluent, are required. The categorical Combinatory Logic, introduced in 1983, and the $\lambda\sigma$-calculi, presented in 1988 and 1989, answers to this question. We give here a survey of these theories, explaining their evolution from a confluence point of view. This report does not contain new results and remains rather informal, avoiding too technical details.

5.1.1.2 Strong normalisation of Substitutions

This is work presented at MFCS'92 [7]. The strong normalization of the σ-calculus, the subcalculus which computes substitutions, may be inferred from the strong normalization of a subcalculus of CCL [30]. One presents in this paper an independent proof of the termination of the σ-calculus. We hope that the method developed for this new proof may help to obtain the strong normalisation of the typed $\lambda\sigma$-calculus.

As future works, the η-reduction permit to look at higher-order unification in order to use efficiently methods such as narrowing. Is it possible to use partial substitution for this unification? This work is currently studied with C. Kirchner. Another interesting problem is to solve the strong normalisation in the case of the first-order typed calculus.

5.1.2 Sharing in Linear Logic and the λ-calculus

This work is a basis for exploring the intrinsic parallelism inside the λ-calculus, since β-rule is split in more atomic operations.

The problem is to minimise the number of steps when reducing lambda expressions. Lamping [34] and Kathail [33] recently gave new and very complicated algorithms. By connecting this problem and the geometry of interaction given by Girard, Gonthier[9] was able to develop much simpler sharing methods in the λ-calculus. Linear Logic deals with the explicit duplication of logical formulae, and it is the corresponding problem which arises with duplication of subterms in the λ-calculus. Gonthier[10, 11] developed the full theory of sharing in linear logic without additives.

There are two possible translations of the λ-calculus into linear logic since types may be $D = (!D) \multimap D$ or $D = !(D \multimap D)$. Gonthier [9] considered the second one, Asperti[1] gave a category theoretic version of the first one.

Mackie considered the trade-off between book-keeping operations and beta conversions, and tries to reduce the former. With a linear logic viewpoint, he considered λ-abstractions as boxes, but implemented λ-binders with a pointer to the list of occurrences of its bound variable. Global operations are made on boxes, except when a copy is to be done, in which case a normal form is computed on any path from the binder to its bound variables. Copying is done by interaction net rules. For instance, when $M = \Delta((\lambda xy.xy)I)$, where $\Delta = \lambda x.xx$ and $I = \lambda x.x$, then M reduces to normal form in 13 interactions (4 beta steps) instead of 54 in Gonthier's. Mackie's reduction strategy is not optimal in some cases, but is better than Wadsworth's old method.

Asperti and Laneve developed a new computational framework, the interaction systems. These are higher order rewriting systems generalizing Lafont's interaction nets, retaining the idea of binary interaction between constructors and destructors through distinguished, complementary ports. The interaction systems also form a subclass of combinatory reduction systems of Klop, containing the λ-calculus. A major part of the research consisted in adapting and extending to interaction systems the notion of optimal computations, as introduced by Lévy for the λ-calculus. In particular, it has been shown that the Lamping-Gonthier optimal implementation of the λ-calculus can be smoothly extended to interaction systems.

In the same theme, they have further studied Lévy's notion of family of redexes of a λ-term, characterizing this notion by means of paths in the term. The idea is that redexes in the same family are created by contractions on a unique common path in a suitable graphical representation of the initial term (where a bound variable refers to the corresponding abstraction). This provides new evidence about the common nature of redexes of the same family, and therefore also about the possibility of sharing their reduction. Then our characterization may be seen as an alternative viewpoint on graph reduction techniques of Lamping and Gonthier implementing optimal reductions.

5.1.2.1 Linear Logic without Boxes

This is work by Gonthier [10, 11]. The λ-calculus is not entirely explicit about the operations of erasing and duplicating arguments. These operations are important both in the theory of the λ-calculus and in its implementations, yet they are typically treated somewhat informally, implicitly. The proof nets of Linear Logic [27] provide a refinement of the λ-calculus where these operations become explicit; they are even reflected in the type system for proof nets (that is, in Linear Logic). Abramsky, Wadler, and others have suggested that this new expressiveness makes Linear Logic a good basis for principled and useful improvements in functional-programming systems.

In some sense, however, Linear Logic could go further. The usual formulation of proof nets involves boxes. The box is the unit for discarding and copying fragments of proof nets. It works as a synchronization mark. The disappearance, reproduction, opening, and movement of boxes remain global operations; full boxes are handled at once, not incrementally, so for example it is not possible to copy a box gradually, in little pieces. As Girard points out, boxes are a bridle to parallelism. They are also an obstacle to sharing: the box formalism does not support some sophisticated mechanisms for "partial sharing" of common subexpressions available in λ-calculus implementations such as Lamping's and Kathail's. These sharing mechanisms are essential for optimality in reductions, and we believe that they can be of practical value. Moreover, boxes complicate the proof theory of Linear Logic; with boxes, Linear Logic falls short of giving a fully local account of computation.

In this work one describes a translation of proof nets into a system of sharing graphs. Proof-net reduction is simulated with graph rewriting. Sharing graphs are interaction nets, in the sense of Lafont; hence rewriting is obviously Church-Rosser, and a naive implementation is straightforward. Everything in the graph system is entirely local. In particular, there are no boxes. Instead, brackets are included as nodes in the system, and they represent the boundaries of boxes. These brackets can propagate and interact with other nodes independently of one another, so boxes can disintegrate. Partial copying and partial sharing become possible.

After translation of LL proof nets into sharing graphs, 12 graph-reduction rules permit calculations on these graphs. It is even possible to reduce this set to 6 rules with a bus notation (as for hardware). Soundness of these rules is proved by the introduction of the so-called context semantics. Contexts (not to confuse with the ones of the λ-calculus) are values given to arcs of the sharing graphs, and the context semantics is the input-output relation between conclusions of the sharing graphs. The exact relation between contexts and Girard's Geometry fo Interaction is studied.

A first correctness proof is done through the definition of a read-back procedure from sharing graphs into nets. Each graph reduction step correspond to several (maybe none) proof nets cut steps. Another proof considers only normal forms of proof nets and show correctness with respect to normal forms. This method uses isomorphisms of context semantics and accessibility.

The optimality part is sketched in the LICS version, and is rather simple. In general, proofs of this work are very technical and rather difficult.

5.1.2.2 Linear Logic, Comonads, and Optimal Reductions

This is work by Asperti [1]. The paper discusses, in a categorical perspective, some recent works on optimal-graph reduction techniques for the λ-calculus. In particular, some of the control operators used in the implementation are nicely related to the two operations associated with the comonad "!" of Linear Logic. The rewriting rules can be then understood as a "local implementation" of naturality laws, that is as the broadcasting of some information from the output to the inputs of a term, following its connected structure.

> ... The presentation is intended to be informal, but the main points are discussed in an intuitively appealing way. Hence the paper contains not only improvements on the treatment of Gonthier, Abadi and Lévy, but also a much more accessible presentation ... (textually borrowed from the referee report).

5.1.2.3 Paths, Computations and Labels in the λ-calculus

This is work by Asperti and Laneve [2]. It provides a new characterization of Lévy's redex-families in the λ-calculus as suitable paths in the initial term of the derivation. The idea is that redexes in a same family are created by "contraction" (via β-reduction) of a unique common path in the initial term. This fact provides new evidence about the "common nature" of redexes in a same family, and about the possibility of sharing their reduction. From this point of view, our characterization underlies all recent works on optimal graph reduction techniques for the λ-calculus, providing an original and intuitive understanding of optimal implementations.

As a simple by-product, we prove that neither overlining nor underlining are required in Lévy's labelling.

5.1.2.4 Local and Asynchronous Beta-Reduction

Vincent Danos and Laurent Régnier [8] have presented a new way of computing λ-terms. They have provided an embedding of λ-terms into the so-called *virtual nets*, which are graphs labelled by some coefficients of the *dynamic algebra*. In particular they have defined and studied a notion of *virtual reduction* which is a graph reduction based on the labelling of the virtual net. A single step of virtual reduction consists in composing two edges of the graph so that the product of their labels is non null in the dynamic algebra. In some sense, virtual reduction is the computation of the transitive closure of the virtual net. It is shown to be confluent and preserving the *execution formula* of Girard. This last property makes virtual reduction a good candidate for computing λ-terms.

The notion of virtual reduction comes from Girard's program on the *geometry of interaction*. It is strongly linked with the sharing reduction techniques and the work of Abadi, Gonthier and Lévy on Lamping's *sharing graphs*. This relation is at the moment under study in collaboration with Asperti and Laneve. In some sense virtual reduction is another way, may be more abstract, to present the sharing graphs.

5.1.2.5 A lambda evaluator based on Interaction Nets

This work [16] in concerned with implementing the theory of optimal reduction in the λ-calculus. More specifically the work is oriented towards finding Interaction Net implementations of the λ-calculus (and hence functional programming languages) where we try to *share* as much as possible. This work is strongly related to the work of Gonthier, Abadi and Lévy [10] and also Asperti and Leneve [4], where a proven correct and optimal implementation is given.

A little analysis of the extant algorithms for implementing optimal reduction indicates a trade-off: optimality vs. book-keeping. Indeed the cost of maintaining such a complicated data-structure outweighs the gains from optimality. The global aim of our work is to find a middle ground: as much sharing as possible with as little book-keeping as possible.

Several algorithms have been developed and extended to handle features from real programming languages, for example recursion, data-structures and references. It is hoped that these algorithms will be implemented, (to assess the performance in comparison with "standard" implementation techniques), during the remaining part of the CONFER project.

Further related work is underway to use Linear Logic in a more direct way by exploring the dynamic semantics—the Geometry of Interaction—as an implementation technique. The basic notion considered here is that of data-flow. More specifically, pushing a *single* token around a fixed network:

- the network is a Linear Logic proof structure; and
- the token is a data-structure which records the history of the path (modulo *cuts*).

The path taken by the token is *deterministic*, i.e. at each choice point the token always has the information on which way to go next. It is hoped that very efficient implementation of the λ-calculus can be generated using such techniques.

This work is related to the work of Danos and Régnier [8] where a notion of "virtual reduction" is introduced.

5.1.3 Interaction Systems

This is work by Asperti and Laneve [3]. A new class of higher order rewriting systems, called Interaction Systems, is introduced, and their relation with intuitionistic logic is discussed. Interaction Systems provide a nice integration of the functional paradigm with a rich class of data structures (all inductive types), and basic control flow constructs such as conditionals and (primitive or general) recursion.

More specifically, Interaction Systems are the *intuitionistic* generalization of Lafont's Interaction Nets, from which they borrow the logical setting, the bipartition of operator into constructors and destructors, and the principle of binary interaction. From a different point of view, Interaction Systems (IS's) can be also regarded as the subclass of Klop's Combinatory Reduction Systems where the Curry-Howard (Proofs as Proposition) analogy still "makes sense". This means that we may associate every IS with a suitable "intuitionistic" system: constructors and destructors respectively correspond to right and left introduction rules, interaction is cut and computation is cut-elimination.

Optimal reductions in Interaction Systems [4] generalize the ones of λ-calculus and linear logic. Lamping's optimal graph reduction technique for the λ-calculus is generalized to a new class of higher order rewriting systems, called Interaction-Systems. This provides a uniform description, in Lamping's style, of other basic computational constructs such as conditionals and recursion.

The paper by Asperti and Laneve [4] mainly deals with the theoretical aspects of optimal reduction in Interaction Systems (family relation, labeling, extraction, and so on). This is the first significant generalization of the theory of optimal reduction in a supersystem of the λ-calculus. Full versions of the article are in [5, 6].

5.1.4 Interaction Diagrams

Interaction diagrams in [21] are graphic representations of concurrent processes with evolving access capabilities; in particular they illustrate the points of access and relations between them. The basic step of computation is the migration of an access point between processes. This paper explains interaction diagrams through a sequence of examples. Diagrams can be regarded as graphic counterparts of terms in the π-calculus and illuminate some interesting points on its construction.

5.1.5 π-Calculus and Interaction Nets

This work was presented at the Linear Logic Workshop held at Cornell University in June 1993 [9]; a paper is in preparation.

Milner's π-calculus [36] is a notation for communicating processes, intended to play a role in concurrency analogous to that of the lambda calculus in sequential programming. Processes communicate by sending or receiving names along channels; an essential feature is that channels are identified by names of the same nature as those that can be passed along them. It is thus a generalisation of CCS, of considerable power since the name-passing feature allows many of the effects of transmitting processes from one place to another to be achieved—instead of sending a process, one sends a name which gives access to that process.

Interaction Nets is a programming language developed by Lafont [32]. It is based on, but is a generalisation of, the proof nets of Classical Linear Logic. One programs by defining new connectives, and rules for rewriting cuts between them. Thus it is natural to view programs as proofs (in some logic defined by the connectives used in the program) and execution as cut elimination. More concretely, an Interaction Net program consists of a set of nodes, a graph built from the nodes, and rewriting rules which replace adjacent pairs of nodes by new graphs. The rewriting rules have a restricted form owing to the logical basis: each node has a distinguished "principal port", and it is only when two nodes are connected by an edge between their principal ports that a rewrite can take place.

In the area of types for concurrency, one aim is to view processes as proofs in a logic associated with the type system; this view would extend the Propositions as Types paradigm from sequential (functional) programming to concurrent programming. A translation of a fragment of the π-calculus into Interaction Nets has been developed, as a step towards a representation of processes as proofs. The fragment includes the name-passing feature of the π-calculus, and allows interesting processes to be defined. Under the translation, parallel composition of processes becomes connection between graphs; the subsequent communication between the processes becomes a graph rewrite. Work is in progress to develop connections between sorts (types) in the π-calculus and the structure of types (propositions) in Interaction Nets.

5.1.6 Some Facile Chemistry

In the paper [15], we use the CHemical Abstract Machine (CHAM) framework for discussing various semantics for the Facile programming language and for formalising (parts of) its implementations. We use these formal descriptions to argue (informally) about implementability and cost of implementation in terms of low level machinery needed to implement the given semantics.

We take the Facile language as source for discussion, but the results also apply to several new languages such as CML and Poly/ML. Characteristic for all these languages is that they combine ideas from the λ-calculus and process algebra, such as CCS, to support high level constructs for programming concurrent, parallel and/or distributed systems.

The paper may also be seen as a case study in comparing semantic descriptions using structural operational semantics and the CHemical Abstract Machine framework.

5.1.7 Action Structures

Actions structures were introduced by Milner [37], to provide a general framework for models of concurrent computation and of communicating systems. The motivation is that, although such models often differ in many details, they all share the notions of information flow, independence (or parallelism) and parametrisation. An action structure is a monoidal category with extra structure, and the three concepts mentioned are represented respectively by composition, tensor product and (part of the added structure) an indexed family of functors called *abstractors*. The arrows (morphisms) of an action structure are the actions, and may be elementary or complex; the second extra structural ingredient in an action structure is a pre-order over the arrows known as the *reaction* relation, which represents dynamics.

Action structures may be concrete or abstract. Within the period of CONFER, considerable progress has been made in the form of a class of concrete action structures known as *action calculi* [17]. These calculi are inspired partly by the Chemical Abstract Machine of Berry and Boudol; they differ mainly in the emphasis upon algebraic structure. Action calculi represent the mechanics of concurrency; for example there is a hierarchy of action calculi for different strengths of the π-calculus, and at least one interesting action calculus for Petri nets. Those for the π-calculus [18, 19] have thrown a useful new light upon the π-calculus, since they can be presented in graphical form; this graphical representation is known as π-*nets*. These nets elucidate several dynamic properties of π-calculus rather more clearly than a formal system. For example, they clarify the power of the *prefixing* or *guarding* operation which π-calculus inherits from CCS. There is a strong correspondence with Parrow's interaction diagrams [21]; this link will be followed up in CONFER.

In recent (unpublished) work, Sangiorgi's definition of the Higher-Order π-calculus [22] has been generalised by Milner to allow an *arbitrary* action calculus to be lifted to higher-order. In fact, this step opens up several future lines of research within CONFER. For it turns out that the simplest action calculus of all, when lifted to higher-order, becomes the typed λ-calculus; moreover, this is embedded in every other higher-order action structure. Therefore phenomena of reduction in λ-calculus can now be studied alongside the dynamics of any action calculus –for example, Petri nets. In particular, the "lifting" proceeds via a mechanism which is essentially explicit substitution in the sense of the $\lambda\sigma$-calculus [13]; thus, the question of strong normalisation for the typed version of this calculus – an apparently difficult open problem – now gains wider importance. Furthermore, since the action calculus for π-calculus has a graphical form, its higher-order version should also have a graphical form; an important task is to see how this relates to the various graphical presentations of λ-calculus which are cited in this report.

More practically, action calculi have a *molecular* representation which strongly suggests the notion of a program module; this opens the way for experimental language design based upon action calculi, once they are better understood.

5.1.8 Combinatory Reduction systems

CRSs combine first-order term rewriting with bound variables. The paper [14] gives a detailed introduction with several examples such as second order typed λ-calculus. A short proof of confluence is given, and the notion of superdevelopments is discussed. The confluence proof uses the method of P. Aczel, who first introduced the basic idea of CRSs. It is close to the well-known proof by Tait and Martin-Loef for λ-calculus, but uses another notion of parallel reduction. Whereas Tait and Martin Loef's parallel reduction corresponds to the classical notion of developments, Aczel's parallel reduction corresponds to what we call superdevelopments, a more liberal notion of reduction. Like developments, also superdevelopments are always terminating.

A detailed comparison is made between CRSs and Nipkow's HRSs. The two formats turn out to be roughly co-extensive, which is somewhat surprising since HRSs employ a type discipline whereas CRSs are type-free. Every CRS can be simulated by a HRS and vice versa; the difference is in some cases that the simulating CRS performs several rewrite step against one step in the simulated HRS. In the HRS framework typed λ-calculus is used as a meta-language to define substitutions and rewrite steps; the advantage is that typed λ-calculus is well-understood and is confluent and terminating, ensuring at once the well-definedness of several operations in a HRS. On the other hand, CRSs employ basically the Finite Developments property to define substitutions.

5.1.9 Abstract reductions systems

This is work by Melliès on standardisation and finite developments. Gonthier, Melliès, Lévy [12] gave an abstract version of the classical standardisation theorem of the λ-calculus. The critical notion was to define what is a standard reduction by only considering nesting of redexes and the residual relation between redexes. The proof of Klop for the λ-calculus was extended by only considering 3 axioms between nesting and residuals. The uniqueness of the standard reduction inside a "permutation class" (for instance only one to get the normal form) was recovered by introducing an axiom for stability in Berry's sense.

However the finite development theorem, which states that the order of contracting a given set of redexes is not relevant, was assumed as built-in inside the previous abstract reductions systems. Melliès has now an axiomatic version of this theorem, giving a very natural proof of the finite developments theorem inside the λ-calculus.

5.1.10 Interrelations between sites and to other areas

Lambda sigma calculus has been studied in ENS and Rocquencourt, and during frequent visits of Hardin to CWI.

Optimal reductions is a work shared by Bologna, ENS, Imperial College, INRIA-Rocquencourt and INRIA Sophia. Asperti and Laneve have worked together during 3 months at Rocquencourt. Mackie is also 3 month at Ecole polytechnique and INRIA Rocquencourt. Exchanges exist between Marseille(Régnier) and ENS. Part of the work has also been done at DEC System Research Center (Palo Alto).

Combinatory systems is a theme shared by Bologna, CWI, INRIA Sophia, INRIA Rocquencourt. It relates to work on processes in Edinburgh. Some of the future work was started by Ariola, who stayed 12 months at Rocquencourt.

Interaction Diagrams is resulting of a strong connection between Edinburgh and SICS.

Work by Gay is related to Area 3, since it is concerned with types for concurrency as well as being a study of the connection between two existing calculi.

Abstract reduction systems is now more particular to Rocquencourt, but there are potential exchanges with CWI and Bologna.

5.1.11 Future Work

As a general goal, this area of CONFER has to make the bridge between the λ-calculus and of the π-calculus at a very syntactical level. Many of the problems come from the absence of a standard setting for the treatment of bound variables. It begins to appear that action structures may provide such a setting, particularly the higher-order action calculi.

There will be a full paper with Abadi on sharing in the λ-calculus. It would be interesting to develop the theory with δ-rules, in a more general setting than Interaction Systems. Practical output of such evaluators has to be studied. This might be done by using $\lambda\sigma$-calculi.

With abstract reduction systems, it would be interesting to understand the basic properties which make strong normalisation (rather than termination of finite developments). An abstract theory of redexes families would also be of interest. This should also contribute to the understanding of higher-order action calculi.

The work on action structures is strongly related to Area 2, on Calculi; it was initiated partly to gain a better understanding of π-calculus, and in particular the general treatment of higher order action calculi is an enrichment of Sangiorgi's higher order π-calculus. The two will be studied together.

Graphical representations of π-calculus have been given by both J. Parrow and R. Milner. They are strongly related, but their differences are important and must be studied. If a graphical form of higher-order π-calculus cxan be found, this must be compared to graphical forms of reduction in λ-calculus, such as the interaction systems of A. Asperti and C. Laneve.

J.A. Bergstra and J.W. Klop study π-calculus as a Combinatory Reduction Systems. In a talk at the Edinburgh CONFER workshop it was shown how a subset of π-calculus can be modeled as an orthogonal CRS. Writing a note on this was suspended in favour of other developments.

V. van Oostrom, F. van Raamsdonk currently work on confluence for weakly orthogonal CRSs. The authors recently found a simple proof for confluence of weakly orthogonal CRSs. This is important because many interesting calculi are only weakly orthogonal, e.g. second order typed lambda calculus with beta and eta rule. Also the CRS version of π-calculus has some (innocent) critical pairs, i.e. it is weakly orthogonal.

J.W. Klop, Z. Ariola work on modular term graph rewriting and cyclic lambda graph rewriting. It remains to be seen whether this study fits in CONFER; it originated in ESPRIT BRA Semagraph, now terminated (but continued as Working Group). It may fit in CONFER, in view of notions of bound variable, hiding of node names, encapsulating boxes. Especially in cyclic lambda graph rewriting there are some curious phenomena regarding bound variables.

Simon Gay aims at bringing the Interaction Nets representation of π-calculus closer to the Proof Nets of Classical Linear Logic. It is known that Proof Nets can be implemented in Interaction Nets by means of a small set of basic nodes [24]. A representation of the nodes used in the π-calculus translation in terms of these basic nodes would be a step in the right direction.

5.1.12 List of Reports

There are two lists; first, the papers written as part of CONFER, then (below the asterisks) a list of previous work cited.

[1] A. Asperti, *Linear Logic, Comonads, and Optimal Reductions*, Fundamenta informaticae, Special Issue devoted to "Categories in Computer Science", Polish Academy of Sciences (invited paper). To appear.

[2] A. Asperti, C. Laneve, *Paths, Computations and Labels in the λ-calculus*, Proc. of the 5th International Conference on Rewriting Techniques and Applications, RTA'93, Montreal. June 1993.

[3] A. Asperti, C. Laneve, *Interaction Systems*, HOA'93. International Workshop on Higher-Order Algebra, Logic and Term Rewriting. Amsterdam, September 1993.

[4] A. Asperti, C. Laneve, *Optimal Reductions in Interaction Systems*, Proc. of the 4th Joint Conference on the Theory and Practice of Software Development, TAPSOFT'93, Orsay (France). April 1993.

[5] A. Asperti, C. Laneve, *Interaction Systems I: the theory of optimal reductions*, Rapport Technique 1748, INRIA-Rocquencourt. Submitted to Mathematical Structures in Computer Science. 1992.

[6] A. Asperti, C. Laneve, *Interaction Systems II: the practice of optimal reductions*. Technical Report UBLCS-93-12, Laboratory for Computer Science, University of Bologna. Submitted to Theoretical Computer Science. 1993.

[7] P.-L. Curien, T. Hardin, A. Rios, *Strong normalisation of Substitutions*, MFCS, Prague, 1992.

[8] V. Danos, L. Régnier, *Local and Asynchronous Beta-Reduction* 8th LICS, Montreal, 1993.

[9] S. J. Gay. π-Calculus and Interaction Nets. Talk given at the Linear Logic Workshop, Cornell University, June 1993.

[10] G. Gonthier, M. Abadi, J.-J. Lévy, *Linear Logic without Boxes*, 7th LICS, Santa Cruz, 1992.

[11] G. Gonthier, M. Abadi, J.-J. Lévy, *Linear Logic without Boxes*, Accepted for the special LICS issue of Information & Computation, to be submitted to this journal.

[12] G. Gonthier, J.-J. Lévy, P.-A. Melliès, *An abstract standardisation theorem*, 7th LICS, Santa Cruz, 1992.

[13] T. Hardin, *From Categorical Combinators to $\lambda\sigma$-calculi: a quest for confluence*, INRIA Report 1777 - November 1992.

[14] J.W. Klop, V. van Oostrom, F. van Raamsdonk, *Combinatory Reduction Systems: introduction and survey*, CWI Report CS-R93xx (to appear); to be published in Theor. Comp. Sci.

[15] L. Leth and B. Thomsen, *Some Facile Chemistry*, Technical report ECRC-92-14, 1992, The first version of this technical report was finished before the official start of CONFER. A Journal version has been prepared and is currently under revision for publication. This version will be considered a deliverable for CONFER.

[16] I. Mackie, *A lambda evaluator based on Interaction Nets* Paper in preparation. (missing pictures)

[17] R. Milner. *Action calculi, or concrete action structures*, Proc. MFCS Conference, Gdansk, Poland, 1993.

[18] R. Milner. *Action structures for the π-Calculus*, Report ECS-LFCS-93-264, Computer Science Dept, University of Edinburgh, 1993 (35 pages).

[19] R. Milner. *An action structure for synchronous π-calculus*, Proc. FCT Conference, Szeged, Hungary, 1993.

[20] V. van Oostrom, F. van Raamsdonk. *Comparing Combinatory Reduction Systems and Higher-order Rewrite Systems*, Report IR-333, Free University Amsterdam, Aug. 1993; to appear in Proceedings of HOA '93 (Intern. Workshop on Higher Order Algebra, Logic and Term Rewriting, September 1993, Amsterdam).

[21] J. Parrow. *Interaction Diagrams*, Rex lecture 1993, Draft version.

[22] D. Sangiorgi. *From π-calculus to Higher-Order π-calculus — and back*, Proc. TAPSOFT'93, Lecture Notes in Computer Science, Springer-Verlag, volume 668, pp. 151–166, 1993.

****

THE FOLLOWING REFER TO PREVIOUS WORK:

****

[23] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, *Explicit Substitutions*, ACM Conference on Principles of Programming Languages, San Francisco, 1990.

[24] M. Abadi and G. Gonthier. Linear Logic Without Boxes Talk given at the Linear Logic Workshop, Cornell University, June 1993.

[25] P.-L. Curien, T. Hardin, *Yet yet a counterexample for λ-calculus+SP*, Journal of functional Programming. to appear.

[26] P.-L. Curien, T. Hardin, J.-J. Lévy, Confluence Properties of Weak and Strong Calculi of Explicit Substitutions, submitted to JACM.

[27] J.-Y. Girard. *Linear Logic*, Theoretical Computer Science, 50, 1987, pp. 1–102

[28] G. Gonthier, M. Abadi, J.J. Lévy. *The geometry of optimal lambda reduction.* Proc. of the 19th Symposium on Principles of Programming Languages (POPL 92). 1992.

[29] T. Hardin, *η-reduction for the languages of Explicit Substitutions*, Algebraic and Logic Programming Conference, Volterra, Italy, August 1992.

[30] T. Hardin and A. Laville, *Proof of Termination of The Rewriting System Subst on C.C.L.* Theoretical Computer Sc., 46, pp 305–312, 1986.

[31] T. Hardin, J.-J. Lévy, *A Confluent Calculus of Substitutions*, France-Japan Artificial Intelligence and Computer Science Symposium, Izu, 1989.

[32] Y. Lafont. Interaction Nets. In POPL'90 Proceedings, ACM Press.

[33] V. Kathail. *Optimal interpreters for lambda-calculus based functional languages*, PhD, MIT, 1990.

[34] J. Lamping, *An algorithm for optimal lambda calculus reduction* 7th POPL, 1990.

[35] L. Maranget. *La stratégie paresseuse* PhD, Université de Paris 7, July 6, 1992.

[36] R. Milner. *The Polyadic π-Calculus: A Tutorial*, in **Logic and Algebra of Specification**, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[37] R. Milner. *Action structures*, Research Report LFCS–92–249, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992.

5.2 Calculi

This section gives a short summary of the work pursued in the context of the CONFER BRA in the area of Calculi. During the first year of the CONFER BRA a large effort has been put into the area of Calculi. This is reflected by the impressive number of 14 reports/publications documenting the effort so far. Looking at the summaries of the work it is evident that related work done at other sites in the consortium is referenced often. It is also worth mentioning that concrete collaboration between sites is taking place. This reflects one of the essential ideas of the CONFER project, namely crossfertilization of ideas among fellow researchers in the consortium.

Since the area of Calculi is rather large we have divided it into three subareas:

- Comparing name passing with agent passing
- Algebraic theories, model checking and tool
- Adding physical distribution

Each of these will be treated as area reports including summary, work done, interrelation between sites and other areas, future work and list of reports.

5.2.1 Comparing name passing with agent passing

5.2.1.1 Summary

One of the main objectives in the Calculi area is to understand the relationship between name passing, as found in the π-calculus and to some extent in the concurrent constraint paradigm (cc paradigm), and agent passing as found in the λ-calculus as well as in higher order process calculi such as CHOCS and HOπ. As the following listing shows this has been a very active field of research in the project and comprises the following contributions:

- The Lambda-Calculus with Multiplicities
  (G. Boudol)
- The Lazy Lambda Calculus in a Concurrency Scenario
  (D. Sangiorgi)
- An investigation into Functions as Processes
  (D. Sangiorgi)
- Non deterministic lazy λ-calculus vs. π-calculus
  (C. Lavatelli)
- From π-calculus to Higher-Order π-calculus — and back
  (D. Sangiorgi)
- On the Reduction of Chocs Bisimulation to π-calculus Bisimulation
  (R. Amadio)

- Mobility in the cc Paradigm
  (C. Laneve, U. Montanari)
- Process Calculi and Parallel Object-Oriented Programming Languages (D. Walker)
- Non-determinism in a Functional Setting
  (C.-H. L. Ong)

Work Done

5.2.1.2 The Lambda-Calculus with Multiplicities

G. Boudol has recently introduced and started to study a refinement of the λ-calculus [Bou93], based on the notion of multiplicity. This emerged from the study of Milner's encoding of the lazy λ-calculus of Abramsky into the π-calculus. Milner has shown that this encoding is adequate, in the sense that any two terms that can be separated by λ-calculus contexts can also be separated by π-calculus means. The converse is not true however, and this means that the π-calculus is strictly more powerful in some sense than the lazy λ-calculus. This topic was further investigated by Sangiorgi who showed that what is missing from the λ-calculus, to make it as powerful as the π-calculus, is essentially a non-deterministic choice operator. This full-abstraction result was established with respect to weak barbed congruence as the notion of semantic equality for both calculi.

There is however a canonical interpretation for the lazy λ-calculus, as shown by Abramsky. This interpretation has as domain a lifted continuous functions space, and the non-deterministic choice may be interpreted in this domain as the join, that is parallel composition of functions (see [Bou90]). Then one may wonder whether the π-calculus encoding is fully abstract with respect to the resulting canonical semantic equality. It turns out that this is not true, and the reason is quite instructive: in the encoding, one uses the replication of the arguments, that is an infinite parallel composition of them, to deal with their possible duplication during the reduction process. Then by providing only a limited number of copies of an argument, one is able to distinguish in the π-calculus some terms which are equated in the canonical interpretation, such as xx and $x(\lambda y.xy)$.

This suggests to introduce a refinement of the λ-calculus, where the arguments come with an explicit (finite or infinite) multiplicity, indicating how many copies of them are available. More generally, an argument in this refined calculus will be a multiset of terms, that is a bag of resources. To define the evaluation mechanism, and more generally the reduction process in this λ-calculus with multiplicities, it is quite convenient to use explicit substitutions, as proposed by Curien et al. In the paper [Bou93] it is shown that we thus obtain a strict refinement of the lazy λ-calculus. In particular, one is able to define a non-deterministic choice, in exactly the same way as it would be defined in the π-calculus. Moreover, some other new phenomena arise, like the possibility of deadlocks during the evaluation.

We show in the paper that the λ-calculus with multiplicities has a natural logical interpretation, which is a refinement of the semantics given by Coppo et al. by means of the intersection types discipline. Our logic is an affine logic, following Girard's terminology, that is we use the weakening rule while disallowing the contraction rule. In the logic the conjunction is managed in a multiplicative manner – again, following Girard's terminology. We show that the logical interpretation is adequate, by establishing that a term has a non-trivial interpretation if and only if its evaluation may terminate on a value.

5.2.1.3 The Lazy Lambda Calculus in a Concurrency Scenario

The use of λ-calculus in richer settings, possibly involving parallelism, is examined in terms of the effect on the equivalence between λ-terms. D. Sangiorgi concentrates on Abramsky's lazy λ-calculus [Abr89], following two directions. Firstly, the λ-calculus is studied within a process calculus by examining the equivalence $\overset{\sim}{\leftrightarrow}$ induced by Milner's encoding into the π-calculus. We start from a characterisation of $\overset{\sim}{\leftrightarrow}$ presented in [San92]. We derive a few simpler operational characterisations, from which we prove full abstraction w.r.t. Lévy-Longo Trees. Secondly, we examine Abramsky's applicative bisimulation when the λ-calculus is augmented with (well-formed) operators, that is symbols equipped with reduction rules describing their behaviour. In this way, the maximal discrimination between pure λ-terms (i.e. the finest behavioural equivalence) is obtained when all operators are used. We prove that the presence of certain non-deterministic operators is sufficient and necessary to induce it and that it coincides with the discrimination given by $\overset{\sim}{\leftrightarrow}$. We conclude that the introduction of non-determinism into the λ- calculus is exactly what makes applicative bisimulation appropriate for reasoning about the functional terms when concurrent features are also present in the language, or when they are embedded into a concurrent language.

5.2.1.4 An investigation into Functions as Processes

In [Mil92] Milner examines the encoding of the λ-calculus into the π-calculus [MPW92]. The former is the universally accepted basis for computations with functions, the latter aims at being its counterpart for computations with processes. The primary goal of this paper by D. Sangiorgi is to continue the study of Milner's encodings. We focus mainly on the lazy λ-calculus [Abr87]. We show that its encoding gives rise to a λ-model, in which a weak form of extensionality holds. However the model is not fully abstract: To obtain full abstraction, we examine both the restrictive approach, in which the semantic domain of processes is cut down, and the expansive approach, in which λ-calculus is enriched with constants to obtain a direct characterisation of the equivalence on λ-terms induced, via the encoding, by the behavioural equivalence adopted on the processes. Our results are derived exploiting an intermediate representation of Milner's encodings into the Higher-Order π-calculus, an ω-order extension of π-calculus where also agents may be transmitted. For this, it is essential the use of the fully abstract compilation from Higher-Order π-calculus to π-calculus studied in [San92].

5.2.1.5 Non deterministic lazy λ-calculus vs. π-calculus

C. Lavatelli has examined and compared various lambda calculi with parallel and convergence testing. The λ_j-calculus, a lazy calculus augmented with a non-deterministic choice operator and a convergence testing combinator, has emerged as a suitable language to be encoded into the π-calculus. The substitution process in λ_j is managed in a semi-explicit way via the use of closures for variables and abstractions. The semantics associated to both λ_j and the π-calculus are based on contextual testing preorders. An encoding of λ_j into π-calculus is defined and it is proved adequate with respect to those semantics. However, the encoding is not fully-adequate. Standard examples show that the π-calculus is still more discriminating than λ_j.

This work builds on previous results by Milner, Sangiorgi and Thomsen, among others, which analyse the expressive power of π-calculus. In these studies it was observed that the standard translation of the lazy λ-calculus into π-calculus is adequate (but not fully adequate) w.r.t. to suitable notions of bisimulation. Later, Sangiorgi showed that adding to the lazy λ-calculus a non-deterministic operator and providing it with a strong notion of testing equivalence (which behaves very much like a bisimulation equivalence) is enough to get full adequacy. Lavatelli's work shows that whenever the semantics is based on Morris' contextual testing equivalence then non-determinancy is not the only extra-feature which distinguishes λ-calculus from its π-calculus embedding.

5.2.1.6 From π-calculus to Higher-Order π-calculus — and back

D. Sangiorgi compares the first-order and the higher-order paradigms for the representation of mobility in process algebras. The prototypical calculus in the first-order paradigm is the π-calculus. By generalising its sort mechanism we derive an ω-order extension, called Higher-Order π-calculus (HOπ). We give examples of its use, including the encoding of λ-calculus. Surprisingly, we show that such an extension does not add expressiveness: Higher-order processes can be faithfully represented at first order. We conclude that the first-order paradigm, which enjoys a simpler and more intuitive theory, should be taken as basic. Nevertheless, the study of the λ-calculus encodings shows that a higher-order calculus can be very useful for reasoning at a more abstract level.

5.2.1.7 On the Reduction of Chocs Bisimulation to π-calculus Bisimulation

R. Amadio has pursued the analysis of the relationships between Chocs and π-calculus. These are two natural extensions of CCS where, respectively, processes and channels are transmissible values. In previous work he had proposed a formalization of the notion of bisimulation for Chocs. His new contribution is a more effective way to reason about this notion by means of an embedding of Chocs into a richer calculus endowed with a notion of "activation" channel which is christened $Chocs_t$. t is the name of a new internal action which is produced by a synchronization on an activation channel, such a synchronization has the effect of forcing the execution of an idle process. In first approximation transitions in $Chocs_t$ may be understood as sequences of synchronizations along activation channels followed by an "observable" transition. There is a simple definition of bisimulation for $Chocs_t$ which satisfies natural laws and congruence rules, moreover the synchronization trees associated to $Chocs_t$ processes are finitely branching. $Chocs_t$ is proposed as an intermediate step towards the definition of a tool for the verification of Chocs bisimulation.

Davide Sangiorgi from the University of Edinburgh has independently obtained a reduction of a restricted form of Chocs where sums are "guarded" to the standard π-calculus (using weak bisimulation). Although these works share similar motivations their technical approaches are quite distinct.

5.2.1.8 Mobility in the cc Paradigm

In this area the relevant research developments include the study of mobility in the paradigm of concurrent constraints. C. Laneve and U. Montanari prove that cc-languages get higher order power even with a very poor underlying constraint system (the signature consists of a constant and a concatenation operation). The turning point is to observe that the phenomenon of mobility is already present in the cc-paradigm as mobility of variables. Therefore, by simulating π-calculus channels in terms of streams, it is possible to rephrase Milner's encodings of lazy and call-by-value λ-calculus into the π-calculus. The encodings distinguish between channels that are used once, just for book-keeping reasons, and those representing variables in Milner's encodings, thus yielding a clearer distinction between resources.

5.2.1.9 Process Calculi and Parallel Object-Oriented Programming Languages

D. Walker has studied how to give semantics for parallel object-oriented programming languages by translations to process calculi: the (polyadic) π-calculus, the Higher-Order π-calculus of Sangiorgi, and most recently an extension of the (first-order) π-calculus with value-expressions and conditional agents. He has shown how these semantics are related to natural structural operational semantics. Finally work has begun on examining how the semantics, in conjunction with techniques from process calculus, can be used to reason about parallel object-oriented programs.

5.2.1.10 Non-determinism in a Functional Setting

The function paradigm in programming methodology is built on foundations which are both logically (via Curry-Howard Isomorphism) as well as computationally (via Church's Thesis) sound. The Lambda Calculus – a language of functions – is the prototypical sequential programming language. But is the function paradigm an adequate basis for the modern practice of parallel (and even distributed) computing? Can it be extended satisfactorily to a useful theoretical framework in which parallel and sequential, and even non-deterministic computations coexist in harmony?

As a first step, the pure untyped Lambda Calculus extended with an (erratic) choice operator is considered as an idealised non-deterministic functional language. Based on both the "may" and the "must" modalities of convergence, a behavioural relation called applicative bisimulation is proposed. Its relationship with a notion of testing equivalence is then investigated. In the style of Abramsky's work on Domain Theory in Logical Form, the denotational type that captures this computational situation is $\delta = P[[\delta \to \delta]_\perp]$ where $P[-]$ is the Plotkin powerdomain functor. A systematic programme which hinges on three distinct interpretations of δ, namely, process-theoretic, denotational and logical is then carried out. This work may be seen as a step towards a reapprochement between the algebraic theory of processes in Concurrency on the one hand, and the Lazy Lambda Calculus as a foundation for functional programming on the other.

5.2.1.11 Interrelations between sites and to other areas

G. Boudol is currently investigating the full-abstraction problem for the λ-calculus with multiplicities in collaboration with C. Lavatelli, ENS Paris. The logical interpretation of this calculus is also relevant to the "Logics" area of the Project.

With Milner's action structures π-calculus and Higher-Order π (HOπ) are put into a more general framework. This gives strong relationship to the area of Foundational models.

The work in [San93c, San93b] shows that HOπ is representable within π-calculus. But it also shows (for instance, in the study of the encodings of λ-calculus) that a higher-order calculus can be very useful for reasoning and programming at a more abstract level. The work by Pierce, Remy and Turner on a higher-order programming language based on π-calculus exploit HOπ and its translation to π-calculus as given in [San93c].

HOπ is a sorted calculus. Its sorts can be arbitrarily higher order, and are reminiscent of higher-order types in functional programming languages. This is also relevant to the "Logics" area of the Project.

C.-H. L. Ong is developing a systematic programme which hinges on three distinct semantic approaches, namely, process-theoretic, denotational and logical. In collaboration between C.-H. L. Ong and ECRC the effectiveness of this approach will be tested by using it to provide a semantic basis for the Facile language developed at ECRC.

5.2.1.12 Future work

C. Lavatelli, ENS Paris, and G. Boudol, INRIA Sophia-Antipolis, are currently investigating the full-abstraction problem for the λ-calculus with multiplicities. The question is to see whether the logical interpretation is fully abstract, and whether the encoding of the λ-calculus with multiplicities into the π-calculus is fully abstract. This would give another view on the expressive power of the π-calculus, compared with the one of the lazy λ-calculus (extended with parallel functions).

We are also starting the study of a typed version of the λ-calculus with multiplicities, still using an affine logic. Here the intention is to see whether some standard results (such as: strong normalization, subject reduction, principal typing) still hold.

Moreover, this could provide us with a good syntax for the Bounded Linear Logic of Girard et al. Another interesting point is to see whether this refined typing discipline, including some logical information about the multiplicities, could be used from an implementation point of view.

Further development of the theory of HOπ is expected. The ultimate goal is to exploit the full abstraction of the encoding of HOπ into the π-calculus [San93c] and the theory of the latter, to derive proof systems for HOπ.

Process algebra with local ports is studied by J.A. Bergstra, J.W. Klop, P. Rodenburg. A note is in preparation for distribution at the Amsterdam workshop. A detailed study is made within the framework of ACP, Algebra of Communicating processes, of a restricted subset of pi-calculus.

5.2.1.13 List of reports

[Ama93] R. Amadio: *On the Reduction of Chocs Bisimulation to π-calculus Bisimulation.* In Proc. CONCUR 93, Lecture Notes in Computer Science, Springer-Verlag, volume 715, 1993. Also appeared as Research Report Inria-Lorraine 1726.

[Bou93] G. Boudol: *The Lambda-Calculus with Multiplicities.* (Preliminary report), INRIA Research Report 2025. (1993). A preliminary version was presented in an invited talk at the CONCUR'93 Conference, Hildesheim, August 1993.

[LanMon92] C. Laneve, U. Montanari: *Mobility in the cc Paradigm.* Submitted for publication 1993. Preliminary version in MFCS'92, Lecture Notes in Computer Science, Springer-Verlag, volume 629, 1992.

[Lav93] C. Lavatelli: *Non deterministic lazy λ-calculus vs. π-calculus.* Technical Report LIENS 93-15, September 1993.

[Ong93] C.-H. L. Ong: *Non-determinism in a functional setting.* In Proc. of LICS 93, 1993.

[San93] D. Sangiorgi: *The Lazy Lambda Calculus in a Concurrency Scenario.* Submitted for publication. This is a revised and extended version of a paper in the Proc. of LICS 92, 1993.

[San93b] D. Sangiorgi: *An investigation into Functions as Processes.* In Proc. Ninth International Conference on the Mathematical Foundations of Programming Semantics (MFPS'93), 1993.

[San93c] D. Sangiorgi: *From π-calculus to Higher-Order π-calculus — and back.* In Proc. TAPSOFT'93, Lecture Notes in Computer Science, Springer-Verlag, volume 668, pp. 151–166, 1993.

[Wal93] D. Walker: *Objects in the pi-calculus.* To appear in Information and Computation, 1993.

[Wal93b] D. Walker: *Process calculus and parallel object-oriented programming languages.* In Proc International Summer Institute on Parallel Computer Architectures, Languages and Algorithms, Prague, July 1993 (Computer Society Press, to appear).

[Wal93c] D. Walker: *Algebraic proofs of properties of objects.* Submitted for publication, 1993.

5.2.2 Algebraic theories, model checking and tool

5.2.2.1 Summary

During just one year of the CONFER project the area of name passing calculi has matured to a state where sound and complete axiomatisations for bisimulations, and even a verification tool has emerged. This is remarkable since in other areas of concurrency theory these developments have taken considerable longer time to emerge. Moreover, the notion of bisimulation has been extensively studied. Several approaches have been compared, and some of them have been shown to lead to the same notion. The following contributions have addressed these issues:

- Algebraic Theories for Name-Passing Calculi
  (J. Parrow, D. Sangiorgi)
- A Theory of Bisimulation for the π-calculus
  (D. Sangiorgi)
- The π-calculus with Explicit Substitutions
  (G. Ferrari, U. Montanari, P. Quaglia)
- Model Checking Mobile Processes
  (M. Dam)
- The Mobility Workbench — a tool for the π-calculus
  (F. Moller, B. Victor)

Work Done

5.2.2.2 Algebraic Theories for Name-Passing Calculi

In a theory of processes the names are atomic data items which can be exchanged and tested for identity, but which admit no other functions or predicates. A well-known example of a calculus for name-passing is the π-calculus, where names additionally are used as communication ports. J. Parrow and D. Sangiorgi provide complete axiomatisations of late and early bisimulation equivalences in such calculi. Since neither of the equivalences is a congruence we also axiomatise the corresponding largest congruences. We consider a few variations of the signature of the language; among these, a calculus of deterministic processes which is reminiscent of sequential functional programs with a conditional construct. Most of our axioms are shown to be independent. The structure of the systems reveals the symmetries of the calculi and equivalences since they differ only by a few simple axioms.

5.2.2.3 A Theory of Bisimulation for the π-calculus

D. Sangiorgi studies a new formulation of bisimulation for the π-calculus, which we have called open bisimulation ($\sim$). In contrast with the previously known bisimilarity equivalences, $\sim$ is preserved by all π-calculus operators, including input prefix. The differences among all these equivalences already appear in the sublanguage without name restrictions: Here the definition of $\sim$ can be factorised into a "standard" part which, modulo the different syntax of actions, is the CCS bisimulation, and a part specific to the π-calculus, which requires name instantiation. Attractive features of $\sim$ are: a simple axiomatisation (of the finite terms), with a completeness proof which leads to the construction of minimal canonical representatives for the equivalence classes of $\sim$; an "efficient" characterisation, based on a modified transition system. This characterisation seems promising for the development of automated verification tools and also shows the call-by-need flavour of $\sim$. Although in the paper we stick to π-calculus, the issues developed may be relevant to value-passing calculi in general.

5.2.2.4 The π-calculus with Explicit Substitutions

The π-calculus is centered around the notions of name and name instantiation which emerge both in the transition semantics, and in the definition of behavioural equivalences. Name instantiation confers mobility to the language. However, in designing an abstract machine for the π-calculus one has to cope with the problem that name instantiation is a meta-level operation which performs a run-time modification of the program. In practice, name instantiation must be applied in a more controlled way. Moreover, since name instantiation is not a proper operation of the language, the π-calculus transition semantics does not fit in any of the studied formats of Structured Operational Semantics (SOS). Hence, the π-calculus theory cannot take direct advantage of the SOS meta-theory developed for "static" process calculi. To address these problems a new formulation of the π-calculus, where name instantiation is handled explicitly, is presented by G. Ferrari, U. Montanari and P. Quaglia. Behavioural equivalences originally developed for the π-calculus are retrieved by giving the description of the corresponding strategies for name instantiation. The explicit handling of name instantiation allows us to take advantage of the SOS meta-theory. Thus, axiomatic characterizations of behavioural equivalences can be automatically derived by analyzing the syntactic structure of the SOS inference rules. This is a first step towards the design of abstract machines (and programming environments) for concurrent languages based on the π-calculus.

5.2.2.5 Model Checking Mobile Processes

Other work in the area has focused on the development of a temporal logic for the polyadic π-calculus based on fixed point extensions of Hennessy-Milner logic. M. Dam adds features to account for parametrisation, generation, and passing of names. These include the use, following Milner, of dependent sum and product to account for (unlocalised) input and output, and explicit parametrisation on names using λ-abstraction and application. The latter enables a clean and uniform account of parametrisation for both input, output, and fixed points. A proof system and decision procedure has been developed based on Stirling and Walker's approach to model checking the modal μ-calculus using constants. One difficulty, for both conceptual and efficiency-based reasons, is to avoid the explicit use of the ω-rule for parametrised processes. A key idea, following Hennessy and Lin's approach to deciding bisimulation for certain types of value-passing processes, is the relativisation of correctness assertions to conditions on names. Based on this idea soundness, completeness and decidability is obtained for arbitrary π-calculus processes with finite control, π-calculus correlates of CCS finite-state processes, avoiding the use of parallel composition in recursively defined processes. To the best of our knowledge this represents the first decidability result in the verification of recursive (or replicative) agents in the π-calculus.

A preliminary version of this work was presented at the first CONFER workshop at INRIA-Sophia-Antipolis, and a later version was presented at CONCUR'93 [Dam93].

5.2.2.6 The Mobility Workbench — a tool for the π-calculus —

The Mobility Workbench (MWB) is a tool for manipulating and analyzing mobile concurrent systems described in the π-calculus [MPW92], developed by F. Moller and B. Victor in collaboration with D. Sangiorgi at the University of Edinburgh. It is written in Standard ML, and currently runs under the New Jersey SML compiler.

In the current preliminary version, the basic functionality is to decide the open bisimulation equivalence of Sangiorgi [San93d], for agents in the monadic π-calculus with the original positive match operator. This is decidable for π-calculus agents with finite control, correlating to CCS finite-state agents, which do not admit parallel composition within recursively defined agents.

The algorithm is based on the efficient characterization of the equivalence described in [San93d], and (necessarily) generates the state space "on the fly". Versions for both the strong and weak equivalences are implemented.

There are also commands for finding deadlocks and for interactively simulating an agent.

The syntax of the agents is kept close to the syntax used in Milner's tutorial on the polyadic π-calculus [Mil91]. An example:

```
MWB> agent P(x) = (^b)x(a).('a<u>.0 | b(v).0)
MWB> agent Q(x) = (^b)x(a).('a<u>.b(v).0 + b(v).'a<u>.0 + [a=b]t.0)
MWB> eq P(a) Q(a)
yes
```

5.2.3 Interrelations between sites and to other areas

The purpose of [ParSan93, San93d] is to understand the notion of (interleaving) bisimulation in calculi for mobile processes like the π-calculus. This notion is well-understood in CCS-like languages, but its generalisation to π-calculus lends itself to subtle variations. Such variations are essentially due to different approaches to the instantiation of the bound names of an input. Therefore, to some extent, they are reminiscent of the variety of evaluation strategies in λ-calculus, which are caused by different approaches to the instantiation of the bound variable of a λ abstraction.

A basic theme in [ParSan93, San93d] is the study of how the basic constructs of π-calculus — first of all the conditional operators — affects the theory of the various kinds of bisimulation (axiomatisation, definition of normal forms etc..).

A major motivation for investigating different semantics theories of the π-calculus is to find suitable directions for the development of software tools to reason about π-calculus processes. We are at the moment experimenting (collaboration among Bjorn Victor (SICS), Faron Moller and Davide Sangiorgi (EdU)) with a tool based on open bisimulation for the mechanical check of process bisimilarities.

Ongoing and future work in this direction includes implementation in the π-calculus workbench currently under development in a cooperation between SICS and Edinburgh; case studies to investigate the viability of temporal logics along these lines as practical program specification tools; the application of the decision procedure to obtain decidability of late strong bisimulation equivalence of finite-control agents. Additionally, cooperation has been initiated with ECRC on the issue of temporal logics that incorporate higher-order process passing.

5.2.4 Future work

In the work on logics for concurrency we would like to know what is the best logic theory for the various bisimulations.

Future work on π-calculus will for some part be based on the observation that there seems to be a connection between the variety of π bisimulations and the variety of λ-calculus evaluation strategies. For instance, the open bisimulation of [San93d] has the flavour of the call-by-need strategy of λ-calculus.

The work in [ParSan93, San93d] focus on "strong" bisimulations, which do not abstract away from internal details of processes. A natural development of the work is to look at "weak" bisimulations. We would like to continue the study of the primitive constructs of π-calculus, initiated in [ParSan93, San93d] by studying the expressiveness they provide.

Future work on temporal logics and model checkers for the π-calculus includes implementation in the π-calculus workbench currently under development in a cooperation between SICS and Edinburgh; case studies to investigate the viability of temporal logics along these lines as practical program specification tools; the application of the decision procedure to obtain decidability of late strong bisimulation equivalence of finite-control agents. Additionally, cooperation has been initiated with ECRC on the issue of temporal logics that incorporate higher-order process passing.

We intend to continue the experimentation with the software tool based on open bisimulation and mentioned above. Future work includes algorithm versions for the polyadic π-calculus; investigating the open bisimulation equivalence in the presence of a mismatch operator; case studies to investigate the relation to the late bisimulation equivalence; algorithms for finding minimal distinctions and matchings for which two open agents are equivalent; modal logics and model checking for the open bisimulation.

5.2.5 List of Reports

[Dam93] M. Dam: *Model Checking Mobile Processes.* In Proc. CONCUR'93, Lecture Notes in Computer Science, Springer-Verlag, volume 715, pp. 22-36, 1993.

[FerMonQua93] G. Ferrari, U. Montanari, P. Quaglia: *The π-calculus with Explicit Substitutions.* Submitted for publication, 1993.

[ParSan93] J. Parrow, D. Sangiorgi: *Algebraic Theories for Name-Passing Calculi.* Report ECS–LFCS–93–262, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1993. Extended Abstract to appear in Proc. REX '93 Summer school, Lecture Notes in Computer Science, Springer-Verlag.

[San93d] D. Sangiorgi: *A Theory of Bisimulation for the π-calculus.* Report ECS–LFCS–93–270, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1993. An Extended Abstract of this paper has appeared in Proc. CONCUR'93, Lecture Notes in Computer Science, Springer-Verlag, volume 715, 1993.

5.2.6 Adding physical distribution

5.2.7 Summary

One of the main motivations for studying concurrent and functional systems is the use in distributed systems. In the CONFER project two promising steps have been taken:

- Observing Time-Complexity of Concurrent Programs
  (G. Ferrari, U. Montanari)
- Some Issues in the Semantics of Facile Distributed Programming
  (B. Thomsen, L. Leth and A. Giacalone)

The first contribution studies a calculus capable of expressing communicating costs in distributed systems. The result is so far limited to a static language (i.e. without mobility). The second contribution comes from practical experience with the Facile programming language where physical distribution is an integral part of the language. It is a first step towards extending the Facile calculus to cover these aspects.

As may be seen from the section on future work this (sub)area looks promising and will probably be very active in the coming year.

5.2.8 Work Done

5.2.8.1 Observing Time-Complexity of Concurrent Programs

G. Ferrari and U. Montanari develop a semantic framework to describe and reason about the time-complexity of concurrent programs. To this purpose, we introduce a modular approach to the semantics of concurrent languages where the functional part of programs is handled together with the complexity part. We show that the time-complexity analysis of concurrent programs can be naturally dealt with a specialized matrix calculus.

In this area the relevant research developments include the analysis of history dependent behaviours in process calculi. To capture the phenomena of history dependency a new paradigm has been introduced. The basic idea is that the state of a system is a pair consisting of a process together with a second component expressing an abstraction of the past moves of the ongoing computation. Many interesting issues of process calculi can be captured and explained by exploiting the history dependent paradigm. As a case study, a semantic framework to describe and reason about time-complexity of concurrent programs has been developed. To this purpose, a modular approach to the semantics of process calculi, where the functional part of programs (the actions processes perform) is clearly separated from the complexity part (the histories), has been introduced. In this case, a history records the amount of computational resources consumed, from the beginning of the computation, by each of the active processing sites (where processes can be executed). This semantic framework can be applied to deal with broader notions of program complexity rather than time-complexity.

5.2.8.2 Some Issues in the Semantics of Facile Distributed Programming

The theoretical foundation of Facile is based on an operational semantics in terms of labelled transition systems, and a notion of observability of programs has been defined by extending the notion of bisimulation. The foundation can be viewed as an integration of the typed call-by-value λ-calculus with a model of concurrency derived from Milner's CCS. The implementation, obtained by extending the SML/NJ implementation of the ML language, supports polymorphic types as well as mobility of functions, processes and communication channels across a distributed computing environment. Thus a number of language constructs have been added or modified to handle certain issues that arise with real distribution. These include the need to control the locality of computation in a physically distributed environment, the potentially expensive implementation of certain operators and the need for a system to tolerate partial failures. In the paper listed below we discuss a possible approach for the operational semantics of these constructs that follows the Facile philosophy and some recent results in concurrency theory.

5.2.9 Interrelations between sites and to other areas

This work shows promising results for the work on programming languages and we hope to strengthen the link between Pisa, Edinburgh and ECRC on the subject of time analysis of distributed concurrent functional systems and developing true concurrency semantics for mobile systems.

5.2.10 Future work

Solving the problem of distributed semantics for distributed higher-order functional and concurrent systems may be difficult. As a starting point it may be better to concentrate on distributed semantics for mobile systems such as the π-calculus.

Up to now an intensive effort has been devoted to the study of the interleaving semantics for the π-calculus, but nothing has yet been done on the non-interleaving (or true-concurrency) side. Various behavioural equivalences which belong to the latter area have been formulated using special machineries of reference- or pointer-like objects; examples are the location bisimulation, causal bisimulation and ST-split semantics. These instruments are intrinsic ingredients of the π-calculus and it is reasonable to ask ourselves whether this fact can be exploited in any significant way.

5.2.11 List of Reports

[FerMon93] G. Ferrari, U. Montanari: *Observing Time-Complexity of Concurrent Programs.* 1993.

[ThoLetGia92] B. Thomsen, L. Leth, A. Giacalone: *Some Issues in the Semantics of Facile Distributed Programming.* Technical report ECRC-92-32, 1992. Also appearing in proceedings of the 1992 REX Workshop on "Semantics: Foundations and Applications", Lecture Notes in Computer Science, Springer-Verlag, volume 666, 1992.

[Abr87] S. Abramsky: *Domain Theory and the Logic of Observable Properties.* Ph.D. Thesis, University of London, 1987.

[Abr89] S. Abramsky: *The Lazy Lambda Calculus.* In Research Topics in Functional Programming, editor D. Turner, pp. 65-116, Addison-Wesley, 1989.

[Bou90] G. Boudol: *A Lambda-Calculus for Parallel Functions.* INRIA Research Report 1231, May 1990.

[Mil91] R. Milner: *The polyadic π-calculus: a tutorial.* Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Dept of Computer Science, University of Edinburgh, UK, October 1991.

[Mil92] R. Milner: *Functions as Processes.* Research Report 1154, INRIA, Sophia Antipolis, 1990. Final version in Journal of Mathem. Structures in Computer Science 2(2):119–141, 1992.

[MPW92] R. Milner, J. Parrow, D. Walker: *A calculus of mobile processes, Parts I and II.* Journal of Information and Computation, 100:1–77, September 1992.

[San92] D. Sangiorgi: *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* Ph.D. Thesis, Department of Computer Science, University of Edinburgh, 1992. Also published as ECS–LFCS–93–266.

5.3 Logics for Concurrency and λ-calculus

This section describes the work carried out in the "Logics for Concurrency and λ-calculus" area of the CONFER Basic Research Action. This work has been carried out at four sites: the Department of Computing, Imperial College, London; the Laboratory for the Foundations of Computer Science, Edinburgh University; the Dipartimento di Informatica, University of Pisa; and ECRC, Munich. The topics addressed fall into the following groups.

- Interaction Categories (S. Abramsky, S. J. Gay, R. Nagarajan)
- Sorts and Types in the π-calculus (S. J. Gay, B. Pierce, D. Sangiorgi)
- Sorts and Types in FACILE (L. Leth, B. Thomsen)
- Linear Logic (S. Abramsky, F. Lamarche, I. C. Mackie, L. Román)
- Semantics of π-calculus (R. Jagadeesan)
- Optimality and Concurrency (C. Laneve, U. Montanari)

Interaction Categories are a new foundation for semantics of sequential and concurrent computation; sorts and types are important in the context of correctness and optimization of concurrent/functional programs; the work on Linear Logic and Optimality transfers techniques from λ-calculus paradigm to concurrency. Therefore, the work done clearly reflects the fruitful interplay between logic, concurrency, and functional computation.

A summary of the work done on each of these topics appears in Section 2. A list of reports and publications appears at the end of this document, along with other references.

5.3.1 Work Done

5.3.1.1 Interaction Categories

Interaction Categories [Abr93a, Abr93b, Abr93c, Abr94] have been developed as a new paradigm for the semantics of computation. The categories standardly used for denotational semantics have *structured sets* as objects and *functions* as morphisms. This limits the really effective use of denotational methods to the sphere of functional computation. Interaction Categories have *specifications* as objects, *processes* as morphisms, and *interaction* as composition.

Potential applications include:

- A useful type discipline for concurrent programming, integrated with a compositional verification methodology.
- Foundations for programming languages incorporating both concurrency and higher-order polymorphic constructs.

We do not, as yet, offer definitive axioms for Interaction Categories. More to the point at this stage, we can provide a number of substantive examples (the first with hindsight), in each of which the above paradigm is clearly visible:

1. Concrete Data Structures and Sequential Algorithms

2. Geometry of Interaction Categories

3. Games and Strategies

4. Specifications and Processes (this work)

The examples (1)–(3) above apply to sequential and/or functional computation. The decisive example needed to make the Interaction Category framework compelling is (4), to encompass concurrency. This was first mooted in [Abr91] which explicitly proposed the Interaction Category paradigm, albeit couched in terms of sequent calculus rather than categories.

The key new example of an Interaction Category **SProc** for concurrent computation follows the picture:

Objects	Specifications
Morphisms	Processes
Composition	Synchronous composition + restriction.

SProc has a very rich structure. Firstly, it provides a model for full (second order) Classical Linear Logic, and hence also, quite automatically, for typed λ-calculi such as System F. It also supports a hierarchy of *delay monads* which express the temporal structure of the category. They allow asynchrony to be built on top of synchrony, as in Milner's original work on SCCS, but in a richer mathematical framework. The delay monads also satisfy distributive laws with respect to the exponentials, relating delay—extension in time—to replication—extension in space.

Much of the familiar process calculus material can be extracted from the structure of **SProc**. *Relabelling* and *Restriction* can be described in terms of a subcategory of "embeddings"; *Non-determinism* arises from the semi-additive structure of **SProc**; *simulations* appear as 2-cells.

Our development of **SProc** clearly exhibits a view of processes as "relations extended in time". This is made precise by the fact that **SProc** is a *bicategory of relations* in the sense of Carboni and Walters. Finally, we show how the compact closed structure of **SProc** supports a "Multi-Cut" rule which allows general process networks to be described within a typed framework.

The overall effect of this program is that process calculus in the CCS tradition is reconstituted in functorial form, and integrated with type theory and functional programming. Moreover, a notion of *specification structure* is introduced which provides a framework in which the Propositions-as-Types and Verification paradigms are combined smoothly in a single framework. This allows a new and conceptually satisfying treatment of the thorny old problem of fairness. There is also a specification structure supporting a compositional treatment of deadlock-freedom, combining ideas from concurrency theory (ready sets) and Linear realizability (orthogonality). In this fashion, we obtain a working version of the Propositions-as-Types paradigm for concurrency, achieving the goal originally set forth in [Abr91].

In addition to **SProc** another Interaction Category, **ASProc**, of asynchronous processes has been developed. One reason why a synchronous model was developed first is that there are problems with identity morphisms in a world of complete asynchrony. The key to **ASProc** is that a certain amount of simultaneity is also needed—the identities still output data in the same instant as that in which it is received. **ASProc** shares a great deal of structure with **SProc**, and this provides helpful pointers towards an axiomatisation of Interaction Categories. The main difference is that non-determinism arises from *weak* biproducts rather than biproducts (as in **SProc**), and this gives the resulting combinators more of a flavour of CSP than CCS. The entire Linear type structure is present, however, and specification structures can still be used. Thus a lot of work can be done in quite a general way before deciding which category is the most appropriate for a specific example.

5.3.1.2 Interaction Categories: Illustrative Examples

Following the development of Interaction Categories [Abr93a] and in particular the category **SProc** of synchronous processes, one area of work has been the study of existing concurrency formalisms and examples in the new framework. Initially, the dataflow model of concurrent computation was studied. Dataflow is well-suited to an analysis in terms of Interaction Categories because of the fact that processes are connected rigidly in a network of fixed topology. It is straightforward to represent individual nodes as morphisms in a category of processes and use categorical composition to connect them into a network. It has been established [Gay93b] that, subject to a condition on the formation of feedback loops (this condition is enforced in real dataflow languages), the **SProc** semantics of dataflow agrees with the standard Kahn-style least fixed point semantics.

The established dataflow languages SIGNAL and LUSTRE have also been considered [GN93a, GN93b]—they are especially suitable because they are based on a synchrony hypothesis. Each language has its own particular features, which have enabled the appropriateness of the structure of Interaction Categories to be tested. In LUSTRE a stream of data may contain non-data values, or delays, and the clock of a stream is another stream which determines when the delays appear. A clock may also contain delays, and thus a stream can have multiple levels of clock, each with its own corresponding delay. This structure is modelled very well by the delay monads of **SProc**—a delay action is introduced into the alphabet of a type by applying a functor, and if this is done several times the different delay actions are naturally distinguished from one another. In SIGNAL, clocks are not nested, but have the subtlety that they need not be fully specified by the programmer. Rather, clock constraints are given, demanding that two streams have the same clock or that one stream delays at least as often as another one. Such clock constraints can be transformed into additional nodes inserted into the network, so that the laws of composition in **SProc** automatically perform the clock calculation which is done by the SIGNAL compiler.

Work has also begun on using **SProc** to give a semantics for the synchronous real-time language ESTEREL [AGN93]. This has been approached by way of the existing "electrical semantics" of ESTEREL, which converts programs into circuits; these circuits, like dataflow networks, are in a convenient form for modelling in Interaction Categories.

Moving on from dataflow and synchronous languages, **SProc** has also been used to model standard asynchronous concurrency examples, for example Milner's cyclic scheduler [AGN93]. This involves using the delay monads to allow the explicit introduction of idle actions at suitable points.

5.3.2 Typed Processes

This work [Abr93d] pursues the goal of typed concurrency along the lines suggested, initially in the work on Proofs as Processes [Abr91] and later in far more detail with the development of Interaction Categories. The central idea of Proofs as Processes is that the interface of a process can be described by a linear (in the sense of Linear Logic [Gir87]) type; the work on Interaction Categories reinforces this view by exhibiting several categorical models of concurrency which have a linear type structure.

Types have played an important rôle in sequential programming, and the typed λ-calculus not only provides an elegant connection between logic and computation via the Curry-Howard Isomorphism but also serves directly as the foundation for the many typed functional languages in practical use. When we turn to concurrency, however, the picture is very different. No canonical calculus for concurrent processes has yet emerged, and the many calculi which do exist have largely been developed without reference to types. There is the more limited notion of *sort*, but this does not seem to offer the same possibilities for a tight connection with a logic in the spirit of the Propositions as Types paradigm.

We have addressed this issue by developing a typed process calculus based on Interaction Categories and gaining experience with its use to define processes of interest arising out of standard examples from the literature. Moreover, we have begun to explore the possibilities for carrying out verification in the same Interaction Category framework. A cyclic scheduler [Mil89], for instance, has been constructed and verified using these ideas.

The combinators of our calculus are inspired by both CCS and the Interaction Category structure (which match rather well in any case). They are presented in the form of typed introduction rules, as is familiar from the typed λ-calculus, and include summation, parallel composition (with and without restriction), prefixing and recursion. While Interaction Categories have not yet been axiomatised, our calculus can be given a semantics using a certain amount of structure which is present in both **SProc** and **ASProc**—namely the Linear type structure, weak biproducts, and a unit delay functor with the unique fixed point property. The fact that this amount of structure supports a useful calculus points towards an axiomatisation based on this structure or something close to it.

As for verification, initially we have considered the safety properties built into **SProc** and **ASProc**. We state the safety of a process as an equality between two morphisms, one being the process itself, and the other being the composition of the process and a "safety morphism". A safety morphism is an identity morphism "cut down" to allow only safe behaviour. It is then possible to manipulate assertions about safety by means of equational reasoning. This has been illustrated in [Abr93d] by taking a safety specification for the cyclic scheduler, decomposing it into safety specifications for the individual cells used to implement the scheduler, stating the safety of the cells categorically, and recombining the corresponding safety equations to prove safety of the whole scheduler. Current work aims to extend this general approach to other forms of specification, by means of the specification structures of Interaction Categories.

5.3.2.1 A Sort Inference Algorithm for the Polyadic π-Calculus

Milner's Polyadic π-Calculus is a notation for communicating processes, intended to play a role in concurrency analogous to that of the λ calculus in sequential programming. Processes communicate by sending or receiving names along channels; an essential feature is that channels are identified by names of the same nature as those that can be passed along them. Milner introduces the notion of sort: the sort of a name determines the form of data which can be passed along the channel of that name, ie how many names may be passed, and also what their sorts may be. A well-sorted process is one in which whenever two channels are connected together, their sorts are the same. This means that whenever a message is passed along a channel, the sender and the receiver agree about what the message looks like and what can subsequently be done with it. In a programming environment, sorts help the programmer to write correct programs, in the same way that types help the functional programmer. An obvious question is whether, given a π-calculus process, a sorting scheme can be derived which makes the process well-sorted. This question is left as an open problem in Milner's original work on the π-calculus.

An algorithm has been developed [Gay93a] for deriving sorting schemes of the above form. It is based on a sequent calculus style system for constructing well-sorted processes; the construction of a well-sorted process also exhibits the sorting which it respects. This is very much in the style of Hindley-Milner type inference in functional language systems. In addition to the sequent calculus system, a concrete algorithm for unifying sortings (the non-trivial step in constructing sortings for well-sorted processes) is presented. The sort inference algorithm as a whole is shown to have polynomial complexity in the syntactic size of the process.

5.3.2.2 Typing and Subtyping for Mobile Processes

We already noted that the π-calculus is a process algebra that supports process mobility by focusing on the communication of channels and that Milner's presentation of the π-calculus includes a type system assigning arities to channels and enforcing a corresponding discipline in their use.

We extend Milner's language of types by distinguishing between the ability to read from a channel, the ability to write to a channel, and the ability both to read and to write [PS93]. This refinement gives rise to a natural subtype relation similar to those studied in typed λ-calculi.

The greater precision of our type discipline yields stronger versions of some standard theorems about the π-calculus. These can be used, for example, to obtain the validity of β-reduction for the more efficient of Milner's encodings of the call-by-value λ-calculus, for which β-reduction does not hold in the ordinary π-calculus.

We define the syntax, typing, subtyping, and operational semantics of our calculus, prove that the typing rules are sound, apply the system to Milner's λ-calculus encodings, and sketch extensions to higher-order process calculi and polymorphic typing.

5.3.2.3 Sorts and Types in FACILE

In this work [Tho93] we present a sort and type system for a simple variant of Facile where constructs for channel creation, sending and receiving are functions of polymorphic type as opposed to syntactic constructs in the original definition of Facile. The sort and type system is inspired by the type and effect discipline developed by Talpin and Jouvelot. The type system is polymorphic in the style of Damas and Milner, and the sort system is inspired by the type system for Nielson's TPL and it is similar to the sort system for CHOCS developed by Thomsen. The sort system is used to control type generalisation in presence of channel creation and communication. Regions (lacking a better name) are used to abstract sets of possible aliased channels. The observable communications of an expression range over the regions that are free in its type environment and its type. Thus communications with processes which are "local" to the expression can be discarded during type reconstruction. Apart from providing information for safe polymorphic generalisation sort and type systems may be of independent interest for concurrent functional programming since sorts and types capture essential static information about a programs dynamic behaviour, in particular its communication potential, and may thus prove useful for compile-time optimisations as well as for run-time placement of processes and channels in a distributed environment. Furthermore, sort and type information may prove valuable in module systems where the sort can be seen as enriching the signature with statements about the dynamic behaviour of encapsulated entities.

5.3.2.4 True concurrency semantics for the π-calculus

We have developed [Jag93] a true concurrency semantics for the π-calculus, viewing processes as sets of functions. The sets of functions for a process can be viewed operationally as the denotation of a dataflow network that implements the process. The novel feature of this description is that it is "name-free", in the spirit of the Geometry of Interaction, and communication is captured by the feedback formula in Abramsky's generalised Kahn Principle. Thus, substitution of names is modelled equationally. When restricted to CCS, the semantics coincides with augmented pomset traces, providing an independent justification for the "true-concurrency" content of the interpretation.

5.3.2.5 On Additive-Multiplicative Intuitionistic Linear Logic

We present a solution [Lam93] to the proof net problem for intuitionistic linear logic with the connectives $\otimes, \multimap, \&$. One way of stating this is to say that we give a "non-syntactical" description of the free monoidal closed (without unit) category with binary products over a countable number of generating objects. Our fragment of intuitionistic linear logic can be treated with one-sided sequents, just like full linear logic, with the use of the Danos-Régnier polarities *Input* and *Output*. In particular formulas can be negated (reversing the polarity) and the connectives $\oplus$ and $\wp$ actually appear in the syntax. Notice that the formulas of type *Output* are exactly the ones that appear as objects of the category and those of type *Input* their negations, the ones that should normally appear on the left side of the sequent.

Now somebody who claims having solved the proof net problem for any fragment of linear logic should have a clean solution to the following points:

- representation of proofs, in particular the existence of normal forms, i.e. two proofs representing the same morphism of the free category must have identical normal forms

- cut elimination, that is, it must follow naturally from the representation used for proofs.

We claim our solution is successful on both these counts. Let us say a little more about representation: a proof P of sequent $\vdash X_1, \ldots, X_n$ is given by a quadruple $(\overline{P}, \leq, \preceq, \rho)$ where $\overline{P}$ is a set with two order structures $\leq, \preceq$ and ρ a map from P to the forest (disjoint union of trees, seen as an ordered set) of subformulas of $X_1, \ldots, X_n$, the X_i being minimal and the atomic subformula being maximal for the ordering. We require that ρ respect the predecessor partial function given by both tree structures on P and the sequent; this maps fails to be an isomorphism only because of the "additive contractions" that appear because of the &-rule. The ordering $\preceq$ is a form of *causality* that can be defined only in intuitionistic linear logic. Basically it is identical with $\leq$ on formulas (i.e. elements of P seen as occurrence of a subformula in the proof) of polarity *Output*, but its *reverse* on subformulas of polarity *Input*, while in an axiom link the atomic subformula of polarity *Input* is immediately above the other one. We claim that such structures satisfying some simple correctness conditions represent proofs, and that there is an additional normal form criterion that picks out a unique representation among the ones representing the same morphism of the category. One important point is that a correct representation has a trivial automorphism group, which says that it is uniquely defined except for trivial issues of naming the elements of its underlying set. An interesting dividend of this approach is that if in a representation we keep only the atomic subformulas we obtain a *games semantics* in which the two players (determined by polarity) discard atomic formulas. Cut-elimination is obtained by matching the subtrees related to both cut formulas and constructing a new representation by induction following the $\preceq$-order.

5.3.2.6 An Internal Language for Autonomous Categories

The main theme of this work [Mac93] is to find the *internal language* of Symmetric Monoidal Closed (autonomous) categories; analogous to the λ-calculus being the internal language for Cartesian Closed Categories.

Previous attempts, most notably the work of Barry Jay, to define an internal language for autonomous categories have taken the language to be the freely generated class of terms with the basic ingredients of the monoidal structure, taking the congruence which identifies the basic terms. However this construction requires that for each additional term we add, we must show that it preserves the congruence. In many cases this is equivalent to showing that a particular diagram commutes, hence resort back to diagrams.

Our work differs in that we give a language in which we can actually define the basic ingredients of the monoidal structure. We define a language and show the standard results, e.g. Subject Reduction, Church-Rosser and Strong Normalisation theorems. We then generate a category for this language and show that it is *the* internal language for autonomous categories. As an application the Coherence theorems of Kelly and Mac Lane are proved in a very simple way.

The main theorem of the paper comes from the following definitions. We define:

1. An autonomous theory **LTC** — our candidate for an internal language, which is the proof expressions of the Multiplicative fragment of Intuitionistic Linear Logic. We show the standard results for this language mentioned above.

2. A Categorical interpretation of this proof system in an Autonomous category $\mathcal{C}$. If $\Gamma \vdash t : A$ is a valid proof, then we give a recipe for generating $[\![\Gamma \vdash t : A]\!]$. This is shown to be a sound model for the language, *i.e.* if $t = t'$ is a valid equation in the language, then $[\![t]\!] = [\![t']\!]$ in $\mathcal{C}$.

3. A construction of a category $C(LTC)$ from the autonomous theory, showing that this does in fact yield an Autonomous category.

4. A construction of an internal language $L(\mathcal{C})$ for an Autonomous category $\mathcal{C}$, showing that this language has the right structure to be an autonomous theory.

We then state the main result:

Theorem 5.3.1 *There is a full and faithful autonomous functor F giving an equivalence of categories:*

$$F : \mathcal{C} \simeq C(L(\mathcal{C})) : F^{-1}.$$

This result tells us that our language is *the* internal language for Autonomous categories.

With this result we have the capability to show the coherence theorems, which basically states that "every diagram built up from the basic ingredients commutes". The proof of this result is made complicated by the fact that each edge of a diagram could be made up of arbitrary many morphisms. The technical difficulty is that we cannot use an induction principle. However with our language we can restate the theorem as follows:

Theorem 5.3.2 *In the free autonomous category two arrows are equal if they have the same binary type.*

The proof of this theorem becomes straightforward since we can now reduce the term to normal form (which exists, and has the same type because of the results stated above) and perform induction on the term formation. Thus a very simple proof technique to show these results, since the work was done once and for all in setting up the language.

5.3.2.7 Optimality and Concurrency

In this area the relevant research developments include a detailed analysis of the relationships between strategies for optimal reduction and concurrency. To analyse the issue of optimality and concurrency a new class of higher order rewriting systems, called *Interaction Systems*, has been introduced. Interaction systems both generalize Lafont's Interaction Nets, and form an interesting subclass of Klop's Combinatorial Reduction Systems (where the Curry-Howard analogy "still makes sense"). The theory (and the strategies) of optimal reduction has been generalized to interaction systems. For instance, optimal implementations of interaction systems are obtained by extending the Lamping-Gonthier graph reduction techniques. Moreover, the problem of describing concurrent computations is addressed by studying the notion of *permutation equivalence.* A complete axiomatization of permutation equivalence is provided. Meseguer's *Concurrent Rewriting* provides the formal basis and most of the axioms. To obtain completeness, some extra axioms modelling the interplay between the operation of substitution and the rewrite rules must be added. Finally, a new concurrent semantics for permutation equivalence is defined. The *Distributive Permutation Equivalence* is the coarsest equivalence cointained into permutation equivalence which yields prime algebraic domains as derivation spaces.

In [Lan] we address the problem of encoding evaluation strategies for the λ-calculus into concurrent processes, i.e. prime event structures. In order to fulfill this aim a necessary condition is that the derivation spaces yielded by the evaluation be prime algebraic cpo's. This requirement is not met by Lévy's permutation equivalence (which equips the λ-calculus with a concurrent semantics). We solve this problem by taking, among those equivalences which reduce disjoint sets of redexes and which are contained into permutation equivalence, the coarsest one such that the downward closure of every derivation is a distributive lattice. This equivalence, called *distributive permutation equivalence*, is characterized directly by restricting permutations of redexes to those sets U which are distributive, i.e. for every $u \in U$, the development of every $V \subseteq (U \setminus \{u\})$ does not affect the individuality of u. A simple consequence of our results is that the derivation spaces of the call-by-value λ-calculus are distributive lattices. Finally, we show that a sequential evaluation mechanism can not, in general, be effectively transformed into a maximally distributive one.

The thesis [Lan93a] studies the problems of optimality and concurrency in a class of higher order rewriting systems: the Interaction Systems. On one side these systems provide the intuitionistic generalization of Lafont's Interaction Nets (that are linear), by keeping the idea of binary interaction and the syntactical bipartition of operators into constructors and destructors. On the other side, Interaction Systems are a suitable subclass of Klop's Combinatory Reduction Systems where the Curry-Howard analogy "still makes sense". Namely, it is possible to consider constructors and destructors of Interaction Systems respectively as right and left introduction rules of intuitionistic systems, interactions as instances of cut-rules and computations as eliminations of cut-rules. In the first part of the thesis, we generalize the standard theory of optimality to Interaction Systems. In particular we define the notion of optimal sharing by means of two different approaches and prove their equivalence. Then we provide an implementation extending Lamping's graph reduction technique for the λ-calculus and fulfilling the optimality criteria. In the second part of the thesis, we study permutation equivalence in the framework of Interaction Systems. This equivalence formalizes the notion of parallel reduction, which is essential in the theory of optimality. Foremost we provide a complete axiomatization of permutation equivalence. Then, by taking λ-calculus as running example, we study the issues of implementing permutation equivalence on Winskel's Event Structures, a mathematical model of distributed systems.

In [Lan93b] we axiomatize *permutation equivalence* in term rewriting systems and Klop's orthogonal left-normal Combinatory Reduction Systems. The axioms for the former ones are provided by the general approach proposed by Meseguer. The latters need extra axioms modelling the interplay between reductions and the operation of substitution.

5.3.3 Interrelations between sites and to other areas

Sangiorgi and Pierce's work on types in the π-calculus is related to Gay's work on sort inference, and to a greater degree, to the work that David Turner at Edinburgh is developing in his thesis. Turner generalises Milner's sorting discipline, showing that principle types (or sorts) exist and giving an algorithm for finding them. He also extends Milner's system to allow polymorphism. Further, he shows that there exists a precise correspondence between λ calculus types and π-calculus sorts (exactly, the most general sorts) in Milner's encoding of the (lazy) λ calculus into the π-calculus. There is no paper available on Turner's work yet, but it was presented at the first CONFER workshop. Turner's work is also related to Gay's, but goes further in terms of the nature of the sorts being considered. All of this is connected to the work by Leth and Thomsen on sorts and types for FACILE.

Mackie, Román and Abramsky's work on an internal language for autonomous categories is strongly related to Mackie's development of the functional programming language Lilac (described in the Area 4 report). Lilac is based on the linear term calculus which forms the internal language for autonomous categories, with some more user-friendly syntax and additional programming constructs. The work by Lamarche is in the same realm.

The Interaction Categories work described in this report is a paradigm for semantics of concurrency and as described in the work on typed processes gives a useful type discipline for concurrent programs. This is clearly related to the work on sorts and types. Here it must be mentioned that we cannot as yet handle mobility in the Interaction Category framework; but recent work by Montanari on encoding the π-calculus into CCS may open up some possibilities. Both Interaction Categories and Milner's work on Action Structures (which is described in the Area 2 report) are approaches to concurrency which make use of categorical structure (particularly a tensor product structure). The exact relationship between the two approaches has not yet been understood, and is an intriguing subject for future study.

5.3.4 Future Work

Future work on Interaction Categories will include the following:

- Identifying the essential common structure between the various categories, and developing an axiomatisation of Interaction Categories.
- Continuing development of a language for processes, and gaining more experience of its use in describing real examples.
- Continuing the work on verification, and extending it by means of Specification Structures to cover more refined properties than safety, such as deadlock-freedom and liveness.

On other topics, items that are on the agenda include:

- Sorts and types for FACILE - covering the full language and implementing the type inference algorithm as well as integrating it in the Facile compiler.
- Obviously the next step in Lamarche's work is to incorporate the exponentials in this approach to proof nets. He has succeeded in giving a very similar criterion for the multiplicative-exponential fragment of intuitionistic linear logic, but a unified treatment of all three kinds of connectives at once is much more difficult.
- Further work on Typing and Subtyping for mobile processes would concentrate on establishing the theory firmly, and on analysing how the typing/subtyping discipline relate to formalisms in the literature for process types.

5.3.5 List of Reports

Imperial College, London

[Abr93a] S. Abramsky. Interaction Categories (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*. Springer-Verlag Workshops in Computer Science, 1993. To appear.

[Abr93b] S. Abramsky. Interaction Categories I: Synchronous processes. Paper in preparation, 1993.

[Abr93c] S. Abramsky. Interaction Categories II: Asynchronous processes. Paper in preparation, 1993.

[Abr93d] S. Abramsky, S. J. Gay, and R. Nagarajan. Constructing and Verifying Typed Processes. Paper in preparation, 1993.

[Abr94] S. Abramsky. Interaction Categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*. Prentice Hall International, 1994. To appear.

[AGN93] S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction Categories: Illustrative examples, 1993. Abstract of talk given at the CONFER Workshop, University of Edinburgh, UK.

[Jag93] R. Jagadeesan. Processes as sets of functions. Abstract of talk given at the CONFER Workshop, University of Edinburgh, UK.

[Gay93a] S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *POPL 93*. ACM Press, 1993.

[Gay93b] S. J. Gay. On iteration and interaction. Draft paper, 1993.

[GN93a] S. J. Gay and R. Nagarajan. Modelling SIGNAL in Interaction Categories. In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*. Springer-Verlag Workshops in Computer Science, 1993. To appear.

[GN93b] S. J. Gay and R. Nagarajan. Synchronous dataflow in Interaction Categories, 1993. Unpublished Manuscript, 1993.

[GN93c] S. J. Gay and R. Nagarajan. Working with Interaction Categories, 1993. Abstract of talk given at the CONFER Workshop, INRIA Sophia Antipolis, France.

[Lam93] F. Lamarche. On Additive-Multiplicative Linear Logic Submitted to the proceedings of the 1993 Workshop on Linear Logic, Mathematical Sciences Institute, Cornell University.

[Mac93] I. C. Mackie, L. Román and S. Abramsky. An Internal Language for Autonomous Categories. Journal of Applied Categorical Structures 1993 (to appear)

University of Edinburgh

[PS93] B. Pierce and D. Sangiorgi. Types and subtypes for mobile processes. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1993.

University of Pisa

[Lan] C. Laneve. Distributive Permutation Equivalence in the λ-calculus. To appear in Fundamenta Informaticae. Technical Report [D/Pisa/M1/3]

[Lan93a] C. Laneve. Optimality and Concurrency in Interaction Systems. PhD Thesis TD-8/93. Dipartimento di Informatica, Università di Pisa, 1993. Technical Report [D/Pisa/M1/4]

[Lan93b] C. Laneve and U. Montanari. Axiomatizing Permutation Equivalence. Submitted for publication 1993. Preliminary version in ALP'92, LNCS 632, 1992. Technical Report [D/Pisa/M1/6]

ECRC, Munich

[Tho93] B. Thomsen. Polymorphic Sorts and Types for Concurrent Functional Programs. Technical report ECRC-93-10, 1993.

Other References

[Abr91] S. Abramsky. Proofs as processes. Unpublished Lecture Notes, 1991.

[Gir87] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

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

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

5.4 Programming Languages

This section provides a brief summary of the work pursued in the context of the CONFER BRA in the area of programming language design, implementation and experience with applications. As we focus this report on concrete work on programming languages rather than abstract formalisms, the results we present are necessarily of a rather preliminary nature. More concrete results should be expected for the last phases of the BRA. One exception is the Facile programming language. However, it should be noted that the Facile project was already in progress before the beginning of CONFER and also that the dimensions and scope of the project are wider than what can be considered as strictly relevant to the CONFER BRA.

The report includes short summaries of the following efforts:

- Facile programming language
  ECRC — A. Giacalone, F. Cosquer, F. Knabe, A. Kramer, T.M. Kuo, L. Leth, S. Prasad, B. Thomsen. Also with contributions of P. Cregut, J.P. Talpin and C. Crampton.

- Prototype compiler for λ-calculus, based on graph reduction
  INRIA — A. Asperti.

- Portable, unobtrusive garbage collection for multiprocessor systems
  INRIA — D. Doligez, G. Gonthier, J.J. Lévy.

- Lilac: a prototype functional programming language based on Linear Logic
  Imperial College — I. Mackie.

- Typed higher-order programming language based on π-calculus
  University of Edinburgh — B. Pierce, D. Rémy, D. Turner.

- Termination properties of unfolding extended to programs with non-determinism
  SICS — B. Lisper.

5.4.1 The Facile programming language

5.4.1.1 Summary

Facile is a higher-order, functional/concurrent programming language that supports the implementation of distributed applications. The distributed implementation of Facile environment has been realised by modifying and extending the Standard ML environment implemented at AT&T Bell Laboratories and Princeton University. The current implementation allows the development of applications that operate on a network of SPARC and Sun-3 workstations running UNIX and, to a limited extent, the Mach operating system. Facile is conceived to support the development of systems exhibiting a high degree of mobility, that is systems that may evolve dynamically in terms of structure, communication and computation capabilities. Both processes and communication channels may be dynamically created and are treated as first-class values in the same way as functions. In particular, they can be communicated over channels, also between processes executing at different physical locations. The notion of process in Facile is quite powerful, since each process has its own environment and runs a full ML program. However, its implementation is very light-weight, so that large numbers of processes can potentially be executed concurrently.

We hope to be able to make the first release of Facile –the Facile "Antigua" release– freely available to the research community in the very near future.

5.4.1.2 Work in progress

Continuing efforts are directed at making the implementation increasingly efficient and at experimenting with certain constructs needed to manage distributed applications (e.g. delay/time-out operators, operators for controlling the physical locations of processes, exception handling). An activity has recently begun, to render communication between Facile processes more reliable through the introduction of low-level protocols that increase the tolerance of the system to partial hardware and software failures. Important results have been recently obtained in supporting the dynamic but type safe connection between different applications, which may have been independently compiled and activated. The result is important because this is going to be a rather frequent scenario in pervasively networked computing environments, and also because it appears to provide a solution to the more general question of "posting" and accessing resources on a network in a flexible and reliable fashion. Finally, implementation strategies are being investigated to enable applications developed with Facile to operate transparently over networks that may include heterogeneous processors as well as heterogeneous software platforms. A crucial issue in this context is of course that of efficient communication of code (functions, processes) across different processors.

A particular effort is currently directed at experimenting with the development of advanced applications. This work began in mid-1992 as an internal activity directed at experimenting with the integration of distributed computing and graphical user interface technologies developed at ECRC. So far, the work has been focusing on a class of distributed, interactive multiple-users systems –e.g. tele-conferencing. The first demonstration system implemented was Calumet: a desktop conferencing tool that supports meetings based on a slide presentation metaphor among users in different physical locations. The distributed part of the system, which handles all communications, has been built entirely with Facile and consists of less than two thousand lines of source code. Calumet also utilizes the Tube system: a sophisticated, object oriented environment for designing and implementing graphical user interfaces, that provides support for defining constraints on user interface objects as well as advanced dialogue management. Calumet was initially designed to operate in local area networks. During the summer of 1993 the distributed part was redesigned and reimplemented –in less than one month, and without touching the user interface implementation– to enable the system to operate on wide area networks and to render it tolerant to partial failures.

5.4.2 A prototype compiler for λ-calculus based on graph reduction

5.4.2.1 Summary

Andrea Asperti and Cosimo Laneve have developed a prototype compiler for lambda calculus, based on the graph reduction technique described in [12], and then simplified in [9, 10].

This reduction technique is optimal in the sense of Lévy [13]. This means, that it is able to share all β-redexes that belong to a same *family* (Lévy has argued that this is the maximal sharing we may expect during reduction).

No traditional implementation of λ-calculus (e.g. Wadsworth's graph rewriting, combinatory logic, supercombinators, environment machines, continuation passing style, ...) is able to profit of all the sharing in λ-expressions, thus this new technique has a great practical interest, not only from the point of view of efficiency, but also of memory occupation space.

Consider for instance Wadsworth's graph rewriting technique for the evaluation of functional expressions. Every time you have a redex $r = (\lambda x.M)N$ you should start with duplicating the functional part $F = \lambda x.M$. Indeed, F could be shared by other terms, and since we are going to instantiate it, we must eventually work on a new copy. If F contains a redex, this will be duplicated as well. The fact of reducing F first, does not help that much. The "redex" inside F could be only a "virtual" one (see [7] for the formal notion of "virtual" redex). Suppose for instance to have in F a subterm like (yP), where y is bound externally to r. The subterm (yP) is not a redex, but its duplication can be as useless and expensive as the duplication of an actual redex. Moreover, the problem of sharing cannot be simply solved by the choice of a suitable reduction strategy. Indeed, Lévy has proved that there are terms, where *every* order of reduction would duplicate work. A typical example is provided by the following term:

$$P = (\lambda x.xIx\ldots x)\lambda y.((\lambda x.x\ldots x)(ya))$$

where I is the identity, a is some constant, and the two sequences of x have both length n. We have two redexes in P. If the outermost is reduced first, we eventually create n residuals of the inner one. Conversely, if we start reducing the innermost redex, n

copies of (ya) are created, and this will duplicate work later on, when I is passed as a parameter to y. In conclusion, any reduction strategy is at least linear in n.

An optimal compiler is able to get (a suitable representation of) the normal form of P in a constant number of β-reductions!

5.4.2.2 The compiler

Lamping's graph reduction technique is too complex to be discussed here. We just mention that the graph rewriting rules needed for the computation can be expressed in the form of an Interaction Net [11], and can be easily implemented in a quite efficient way.

The prototype compiler developed by A.Asperti and C.Laneve implements Lamping's technique via a sort of "lazy" strategy: reduction is pursued up to the weak head normal form of the term, by iterating the reduction of the redex-family of the leftmost outermost redex in the term.

The leftmost outermost redex is looked for by maintaining an auxiliary stack for the main spine of the term (in a way similar to typical supercombinators implementations, such as the G-machine).

One of the main problems of the implementation has been the read-back procedure, that is the procedure for translating the complex internal representation of the final lambda term into a readable form. This has been solved by implementing the readback algorithm in [8].

The source C code is available by anonymous ftp, in the CONFER/asperti subdirectory at `theory.doc.ic.ac.uk`. The file, in compressed tar format is named `opt_impl.tar.Z`.

5.4.2.3 Extensions and future work

The current version of the compiler is still very preliminary. In particular, the following features are under investigation:

- Garbage Collector. This does not seem to be a difficult problem: standard techniques can be adopted.
- Data Types and Control Primitives. The current version only deals with pure λ-terms. It looks urgent to integrate the compiler with an interesting set of data types (booleans, integers, lists, ...), and some control flow constructs (conditionals, recursion, etc.). The authors already developed the theoretical background for such extensions [4, 6, 8], and the actual implementation does not look too problematic.
- Optimizations. The complex book-keeping required by Lamping's technique is a big draw-back to the actual performance of the compiler. It seems to be possible to drastically reduce the number of *control operators* floating in the graph, but no theoretical result has been proved yet.

5.4.3 Portable, unobtrusive garbage collection for multiprocessor systems

Abstract

Doligez-Leroy[POPL'93] developed a garbage collector which has been implemented in CAML-light (INRIA version of ML) on a parallel shared-memory architecture (Encore). This real-time collector has a fancy synchronising protocol which makes it by far 10 times better than the best known parallel collector by H.-J. Boehm which has a high latency. It is so efficient that this algorithm is also used for the uni-processor version.

The algorithm relies on 2 observations. First, in ML, few objects are mutable. So the garbage collector can have efficiently a mark-and-sweep collector on mutable objects in global memory space and several generations for non-mutable objects of the local space of each processor. Non-mutables variables may be replicated without any consistency problems. Secondly, standard parallel garbage collectors do not treat hidden variables such as processor registers, which are not accessible by other processors. The DL real-time algorithm has a sophisticated marking strategy, different from the standard Dijkstra-Lamport which allows very little overhead on register load/store operations, small latency on the collector which runs in parallel.

Doligez-Gonthier[POPL'94] proved the previously mentioned algorithm. The proof starts by the translation of the true C program into a TLA/Unity language (TLA is Lamport's Temporal Logic of Actions). The proof led to the discovery of rare but very important mistakes, which were fixed by rewriting the algorithm with a finer analysis on overheads, livelocks and memory fragmentation. The proof is immediate, after the right translation into TLA/Unity. Moreover, the assumptions of the garbage collector make it independent of the hardware.

The present algorithm is the first efficient, portable, real-time parallel garbage collector.

5.4.4 Lilac: a prototype functional programming language based on Linear Logic

Summary

A prototype functional programming language based on Linear Logic has been developed at Imperial College to conduct experiments to determine the usefulness of Linear Logic proof terms (Linear Lambda Calculi) as a programming language. This preliminary version consists of a small environment where programs can be edited and executed. The 3000 lines of code is written in Standard ML.

The main aim of this research is to look closer into the notion of reduction in the λ-calculus by using more refined calculi with explicit substitutions, copying and discarding. Linear Logic proof terms provide a solid basis for a more refined implementation:

- Substitutions are made explicit: indeed the progress of a substitution through a term appears as explicit steps in the cut-elimination process.

- Copying an argument corresponds to the *contraction* rule of the logic. Knowing that an argument is not copied tells us that we can perform In-place updates safely.
- Discarding an argument corresponds to the weakening rule of the logic. Knowing that an argument is discarded tells us that it is not strict in that argument.

The philosophy behind Lilac is a very explicit language where the programmer is very much aware of the resource manipulations of the algorithm. Lilac is a very small strongly typed functional programming language without many of the advanced features found in languages such as Standard ML. The design philosophy was to keep things simple—a syntactically sparse language in the same spirit as Miranda[1] for example.

The main features of Lilac are:

- Script style: the current program is held in a script—a collection of function definitions.
- Pattern matching: on both built-in data structures, and the linear patterns.
- Lists: eager lists and lazy lists (streams).
- General Recursion and Iterators (for the natural numbers).
- Currying.

EXAMPLES:

A small script is shown below to give the flavour of the language. The functions are shown together with their types generated by Lilac.

```
fun $square\ (!x@!y) = x * y$ ;
    $square : !nat \multimap nat$
fun $S\ f\ g\ (y@z) = fy(gz)$ ;
    $S : (!\alpha \multimap \beta \multimap \gamma) \multimap (!\alpha \multimap \beta) \multimap !\alpha \multimap \gamma$
fun $K\ x\ \_ = x$;
    $K : \alpha \multimap !\beta \multimap \alpha$
fun $fst\ \langle x, \_\rangle = x$ ;
    $fst : \alpha \,\&\, \beta \multimap \alpha$
funrec $length\ (\_ : t) = let\ length\ be\ !len\ in\ 1 + len(t)$
       $length\ [\,] = let\ length\ be\ \_\ in\ 0$ ;
       $length : !(list(!\alpha) \multimap nat)$
funrec $sum\ (h : t) = let\ sum\ be\ !s\ in\ h + s(t)$
       $sum\ [\,] = let\ sum\ be\ \_\ in\ 0$ ;
       $sum : !(list(nat) \multimap nat)$
```

[1]Miranda is a trademark of Research Software Ltd.

5.4.5 A typed programming language based on the π-calculus

Summary

The π-calculus offers an attractive basis for concurrent programming languages. It is small, elegant, and well understood, and it supports, via simple encodings, a wide range of high-level constructs such as structured data, higher-order programming, concurrent control structures, and objects. Moreover, familiar type systems for the λ-calculus have direct counterparts in the π-calculus, yielding strong, static typing for high-level languages defined in this way.

We have developed a statically-typed higher-order concurrent programming language, called PICT. PICT extends the π-calculus with structured data (such as integers, tuples and records) and enables the communication of higher-order processes (c.f. [18]). PICT's static type system is derived from Milner's original work on sorts for the π-calculus (c.f. [17]).

We are currently investigating the different programming styles which can be used in PICT, with a view to further refining our language design. In particularly, we are interested in integrating work we have done separately on sequential object-oriented languages.

A compiler and interpreter for PICT has been implemented using the CAML-light compiler. A separate runtime system and bytecode interpreter has also been written in C (for reasons of efficiency). Both implementations emphasize simplicity and portability, and are designed to work in the standard Unix environment. This work is closely related to implementation efforts at other CONFER sites, since the runtime support required for PICT is very similar to that used by Poly/ML and Facile.

5.4.6 Termination properties of unfolding extended to programs with non-determinism

Source-to-source program transformations are of importance for improving the performance of high-level programs. A particular transformation method is called *partial evaluation*: the idea is to create specialized codes for functions that are called with some part of the argument known. The increased information about the input can then be used to simplify the code, so it is "tailored" to the particular situation. Thus, partial evaluation offers the possibility to write highly reusable code, with a high level of abstraction, and still have the benefits of the performance improvements given by specialized code. Partial evaluation can be done as a stand-alone process, but it can also be used automatically by compilers as an optimizing preprocessing phase.

A technique used extensively by partial evaluators is *unfolding*, i.e. replacing a function call with the definition, where actual parameters are substituted for formal parameters. If the actual parameters are (partially) instantiated, simplifications can often be carried out, which means transferring work from runtime to compile-time. Much of the performance gains in partial evaluation comes from this. On the other hand, unfolding introduces some hazards: one of these is the risk of *nontermination* due to infinite unfolding. Any partial evaluator must have some means to control this.

Partial evaluation is best developed for purely functional languages and logic programming languages, and it is also being developed for languages with imperative features. Languages with nondeterminism pose theoretical problems, however, and little has been known about how to resolve them. (This is one of the "challenging problems" in the area, as listed in [19].) Thus, it is also hard to treat programs with processes, since processes introduce nondeterminism in general. Still, it is desirable to be able to apply partial evaluation to such programs, since the possible benefits are still there.

The work carried out at SICS is concerned with the termination properties of unfolding. Thus, it is directly applicable to program transformation software that uses unfolding to improve performance. Previous work [20] concerns the purely functional case: a model of symbolic evaluation is developed, certain properties of unfolding are proved like *correctness* (the unfolded version does the same thing as the original program), and criteria are developed that ensure *termination* of unfolding. When these criteria are met, termination of symbolic unfolding is guaranteed provided that the original program terminates with "real" arguments. Some syntactical tests are developed and proved correct.

As part of the CONFER project, the results regarding unfolding are extended to programs with nondeterminism. In a preliminary report [21], an operational model for nondeterminism in applicative programs is developed. Nondeterminism is expressed through a nonconfluent term rewriting system. A theory for semantics-preserving unfolding in presence of nondeterminism are developed. Semantics-preserving unfolding means that the unfolded programs should be able to produce exactly the same results (no more, no less) as the original program. The term rewriting system is factored into a deterministic (confluent) part and a nondeterministic (nonconfluent) part, and unfolding is performed only with respect to the deterministic part. Then, termination properties carry over directly from the previous work. It is then proved that under certain conditions on the rewrite system the unfolding is indeed semantics-preserving, and some examples are given.

5.4.7 List of reports

[1] L. Leth and B. Thomsen. *Some Facile Chemistry.* Technical report ECRC-92-14, European Computer-Industry Research Centre, 1992. Journal version submitted and currently under revision for publication.

[2] B. Thomsen, L. Leth and A. Giacalone. *Some Issues in the Semantics of Facile Distributed Programming.* Technical report ECRC-92-32, European Computer-Industry Research Centre, 1992. Also in proceedings of the 1992 REX Workshop on *Semantics: Foundations and Applications.* LNCS 666, Springer-Verlag, 1992.

[3] *Polymorphic Sorts and Types for Concurrent Functional Programs.* Technical report ECRC-93-10, European Computer-Industry Research Centre, 1993.

[4] A. Asperti, C. Laneve *Interaction Systems I: the theory of optimal reductions.* Rapport Technique 1748, INRIA-Rocquencourt. Submitted to Mathematical Structures in Computer Science. 1992.

[5] A. Asperti *Linear Logic, Comonads and Optimal Reductions.* Acta Informaticae, Special Issue devoted to "Categories in Computer Science", Polish Academy of Sciences (invited paper). To appear.

[6] A. Asperti, C. Laneve *Optimal Reductions in Interaction Systems.* Proc. of the 4th Joint Conference on the Theory and Practice of Software Development, TAP-SOFT'93, Orsay (France). April 1993.

[7] A. Asperti, C. Laneve *Paths, Computations and Labels in the λ-calculus.* Proc. of the 5th International Conference on Rewriting Techniques and Applications, RTA'93, Montreal. June 1993.

[8] A. Asperti, C. Laneve *Interaction Systems II: the practice of optimal reductions.* Technical Report UBLCS-93-12, Laboratory for Computer Science, University of Bologna. Submitted to Theoretical Computer Science. 1993.

[9] G. Gonthier, M. Abadi, J.J. Lévy. *The geometry of optimal lambda reduction.* Proc. of the 19th Symposium on Principles of Programming Languages (POPL 92). 1992.

[10] G. Gonthier, M. Abadi, J.J. Lévy. *Linear Logic without boxes.* Proc. of the 7th Annual Symposium on Logic in Computer Science (LICS'92). 1992.

[11] Y. Lafont. *Interaction Nets.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[12] J. Lamping. *An algorithm for optimal lambda calculus reductions.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[13] J.J.Lévy. *Réductions correctes et optimales dans le lambda-calcul.* Thèse de doctorat d'état, Université de Paris VII. 1978.

[14] D. Doligez, G. Gonthier. *Portable, Unobtrusive Garbage Collection for Multiprocessor Systems.* To appear in Proceedings of POPL'94.

[15] I. Mackie. *Lilac: A Functional Programming Language Based on Linear Logic.* To appear in the *Journal of Functional Programming.*

[16] B. Pierce, D. Rémy, D.N. Turner. *A Typed Higher-Order Programming Language Based on the Pi-Calculus.* Draft, 1993.

[17] R. Milner. *The polyadic π-calculus: a tutorial.* Technical report ECS–LFCS–91–180. LFCS. Dept. of Computer Science, Edinburgh University. 1991.

[18] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD Thesis. Technical report CST–9-93. Dept. of Computer Science, Edinburgh University. 1992.

[19] N.D. Jones. *Challenging Problems in Partial Evaluation and Mixed Computation.* In *Partial Evaluation and Mixed Computation*, Proceedings of the IFIP TC2 Workshop (Gammel Avernæs, Denmark, October 1987). PP.1-14. D. Bjørner, A.P. Ershov and N.D. Jones Editors. North-Holland, 1988.

[20] B. Lisper. *Total Unfolding: Theory and Applications.* To appear in *Journal of Functional Programming.*

[21] B. Lisper. *Unfolding of Programs with Nondeterminism.* Preliminary report, 1993.

Chapter 6

Appendices

In separate documents, there are copies of transparencies of workshops 1 and 2, and of the corresponding printed papers.

Below is a list of reports and publications mentioned in this document.

CONFER-1	J.W. Klop V. van Oostrom F.van Raamsdonk	*Combinatory Reduction Systems: introduction and survey* CWI Report CS-R93xx (to appear) to be published in Theor. Comp. Sci.
CONFER-2	V. van Oostrom F.van Raamsdonk	*Comparing Combinatory Reduction Systems and Higher-order Rewrite Systems.* Report IR-333, Free University Amsterdam, Aug. 1993; to appear in Proceedings of HOA '93 (Intern. Workshop on Higher Order Algebra, Logic and Term Rewriting, September 1993, Amsterdam)
CONFER-3	J.A. Bergstra I. Bethke A. Ponse	*Process Algebra with Combinators* Report University of Amsterdam (to appear Sept. 93)
CONFER-4	R. Milner	*An action structure for synchronous π-calculus* Proc. FCT Conference, Szeged, Hungary, LNCS 710, 1993.
CONFER-5	R. Milner	*Action calculi, or concrete action structures*, Proc. MFCS Conference, Gdansk, Poland, LNCS 711, 1993.
CONFER-6	J. Parrow D. Sangiorgi	*Algebraic Theories for Name-Passing Calculi*, To appear in the Proc. REX Summer School 1993, LNCS, Springer Verlag.
CONFER-7	B. Pierce D. Sangiorgi	*Typing and Subtyping for Mobile Processes*, Proc. 8th LICS Conference
CONFER-8	D. Sangiorgi	*A Theory of Bisimulation for the π-calculus* Proc. CONCUR '93, LNCS 715, Springer Verlag, 1993.
CONFER-9	D. Sangiorgi	*From π-calculus to Higher-Order π-calculus — and back*, Proc. TAPSOFT '93, LNCS 668, Springer Verlag, 1993.
CONFER-10	D. Sangiorgi	*An investigation into Functions as Processes*, to appear in the Proc. Ninth International Conference on the Mathematical Foundations of Programming Semantics (MFPS'93).
CONFER-11	D. Walker	*Process calculus and parallel object-oriented programming languages* In Proc International Summer Institute on Parallel Computer Architectures, Languages and Algorithms, Prague, July 1993 (Computer Society Press, to appear)

CONFER-12	L. Leth B. Thomsen	*Some Facile Chemistry* Technical report ECRC-92-14, 1992.
CONFER-13	B. Thomsen L. Leth A. Giacalone	*Some Issues in the Semantics of Facile Distributed Programming* Technical report ECRC-92-32, 1992.
CONFER-14	J.-J. Lévy B. Thomsen L. Leth A. Giacalone	*Esprit Basic Research Action 6454-CONFER:* *CONcurrency and Functions:* *description of the CONFER project* Bulletin of EATCS, Number 45, October 1992, pp.158-185
CONFER-15	B. Thomsen	*Polymorphic Sorts and Types for Concurrent Functional Programs*, Technical report ECRC-93-10, 1993.
CONFER-16	R. Amadio	*On the Reduction of Chocs Bisimulation to π-calculus Bisimulation*, Proc. CONCUR 93, E. Best (ed.), SLNCS 715.
CONFER-17	P.-L. Curien	*On the Symmetry of Sequentiality* Conference on Mathematical Foundations of Program Semantics, to appear in the Proceedings of the Conference.
CONFER-18	V. Danos L. Regnier	*Local and Asynchronous Beta-Reduction* in Proc. IEEE-LICS 93, Montreal.
CONFER-19	C. Lavatelli	*Non deterministic lazy λ-calculus vs. π-calculus* Technical Report LIENS 93-15, September 1993.

CONFER-20	S. Abramsky	*Interaction Categories* (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods.* Springer-Verlag Workshops in Computer Science, 1993. To appear.
CONFER-21	S. Abramsky	*Interaction Categories and communicating sequential processes.* In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare.* Prentice Hall International, 1994. To appear.
CONFER-22	S. Abramsky S. Gay R. Nagarajan	*Constructing and Verifying Typed Processes* ICDOC Technical Report, October 1993 (To appear).
CONFER-23	S. J. Gay	*A sort inference algorithm for the polyadic π-calculus.* In *POPL 93.* ACM Press, 1993
CONFER-24	S. J. Gay R. Nagarajan	*Modelling* SIGNAL *in Interaction Categories* In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods.* Springer-Verlag Workshops in Computer Science, 1993. To appear.
CONFER-25	I. Mackie	Lilac: *A Functional Programming Language Based on Linear Logic.* To appear in the *Journal of Functional Programming.*
CONFER-26	I. C. Mackie L. Román S. Abramsky	*An Internal Language for Autonomous Categories* Journal of Applied Categorical Structures 1993 (to appear)
CONFER-27	L. Ong	*Non-determinism in a functional setting* In Proceedings of LICS 1993

CONFER-28	A. Asperti	*Linear Logic, Comonads, and Optimal Reductions* Fundamenta Informaticae, Special Issue devoted to "Categories in Computer Science", Polish Academy of Sciences (invited paper). To appear.
CONFER-29	A. Asperti, C. Laneve	*Paths, Computations and Labels in the λ-calculus* Proc. of the 5th International Conference on Rewriting Techniques and Applications, RTA'93, Montreal. June 1993
CONFER-30	A. Asperti, C. Laneve	*Interaction Systems* HOA'93. International Workshop on Higher-Order Algebra, Logic and Term Rewriting. Amsterdam, September 1993
CONFER-31	A. Asperti, C. Laneve	*Optimal Reductions in Interaction Systems* Proc. of the 4th Joint Conference on the Theory and Practice of Software Development, TAPSOFT'93, Orsay (France). April 1993.
CONFER-32	A. Asperti, C. Laneve	*Interaction Systems I: the theory of optimal reductions* Rapport Technique 1748, INRIA-Rocquencourt. Submitted to Mathematical Structures in Computer Science. 1992
CONFER-33	A. Asperti, C. Laneve	*Interaction Systems II: the practice of optimal reductions* Technical Report UBLCS-93-12, Laboratory for Computer Science, University of Bologna. Submitted to Theoretical Computer Science. 1993
CONFER-34	P.-L. Curien, T. Hardin, A. Rios	*Strong normalisation of Substitutions* MFCS, Prague, 1992
CONFER-35	D. Doligez, G. Gonthier	*Portable, Unobtrusive Garbage Collection for Multiprocessor Systems* To appear in Proceedings of POPL'94
CONFER-36	G. Gonthier, M. Abadi, J.-J. Lévy	*Linear Logic without Boxes,* 7th LICS, Santa Cruz, 1992. Accepted for the special LICS issue of Information & Computation, to be submitted to this journal.
CONFER-37	G. Gonthier, J.-J. Lévy, P.-A. Melliès	*An abstract standardisation theorem* 7th LICS, Santa Cruz, 1992.
CONFER-38	T. Hardin	*From Categorical Combinators to $\lambda\sigma$-calculi: a quest for confluence,* INRIA Report 1777 - November 1992

CONFER-39	G. Boudol	*The Lambda-Calculus with Multiplicities* (Preliminary report), INRIA Research Report 2025. (1993). presented in an invited talk at the CONCUR'93 Conference, Hildesheim, August 1993.
CONFER-40	G. Boudol	*The Chemical Abstract Machine* invited talk at the REX workshop, "A Decade of Concurrency" (Amsterdam, June 1993)
CONFER-41	G. Ferrari U. Montanari P. Quaglia	*The π-calculus with Explicit Substitutions* Submitted for publication 1993.
CONFER-42	G. Ferrari U. Montanari	*Observing Time-Complexity of Concurrent Programs* 1993.
CONFER-43	C. Laneve	*Distributive Evaluations of λ-calculus*, To appear in Acta Informaticae.
CONFER-44	C. Laneve	*Optimality and Concurrency in Interaction Systems*, PhD Thesis TD-8/93, Dipartimento di Informatica, Università di Pisa, 1993.
CONFER-45	C. Laneve U. Montanari	*Mobility in the cc Paradigm* Preliminary version in MFCS'92, LNCS 629, 1992
CONFER-46	U. Montanari	*Axiomatizing Permutation Equivalence*, Submitted for publication 1993. Preliminary version in ALP'92, LNCS 632, 1992
CONFER-47	B. Lisper	*Total unfolding: theory and applications* Accepted for publication in Journal of Functional Programming.
CONFER-48	J. Parrow	*Interaction Diagrams* Draft paper, presented at REX'93 workshop and summer school
CONFER-49	J. Parrow D. Sangiorgi	*Algebraic Theories for Name-Passing Calculi* SICS Research Report R93:04, 1993.
CONFER-50	M. Dam	*Model Checking Mobile Processes* In Proc. CONCUR'93, LNCS 715, pp. 22-36.
CONFER-51	B. Lisper	*Unfolding of Programs with Nondeterminism and Processes* In preparation (should be available in time for review)