Preface

Pierre-Louis Curien
(Laboratoire Preuves, Programmes et Systèmes, πr^2 team,
CNRS, Paris 7, and INRIA)

January 5, 2011

This text is both meant as a preface to this volume of Theoretical Computer Science dedicated to Jean-Yves Girard, and as a short essay in French (with an English summary) on the relation between proof theory and programming languages – a coming together in which Jean-Yves' works play a prominent role.

The idea of this volume grew out contemporaneously with the following events that were organised on the occasion of Jean-Yves' 60th birthday:

- Workshop on Linear Logic, Ludics, Implicit Complexity, Operator Algebras, held in Siena (Certosa di Pontignano), May 17-20, 2007;
- Journés Jean-Yves Girard, held in Paris (Institut Henri Poincaré), September 10-12, 2007, with the following list of speakers:
  - Patrick Dehornoy, Unprovability results involving braids
  - Thomas Ehrhard, Differential Linear Logic and the Pi-Calculus
  - Herman Ruge Jervell, Π^1_2-logic, β-completeness and the treatment of ordinals
  - Gérard Huet, Informatics in search of rigorous design principles
  - Yves Lafont, Homotopy of computation: The Minneapolis Program
  - Olivier Laurent, Is not not A equal to A?
  - Ieke Moerdijk, Trees, tensors and weak higher categories
  - Thierry Paul, From quantum to classical by letting the dimension diverge
  - Peter Selinger, Linear Logic in quantum computation
  - Bernard Teissier, Cognition and Foundations
  - Glynn Winskel, What is a process?

1 Proof theory and programming languages

The period of the 80's (of the last century) has seen a blossoming of fruitful contacts between mathematics (logic) and computer science (semantics) leading to the establishment of Curry-Howard isomorphism as a major backbone for the research in this area.

While Girard's system F [31], and independently (but later) Reynolds' polymorphism, were invented in the early seventies, it is not until the beginning of the 80's that the two communities actually met and cross-fertilized. Papers such as [29] played the role of "go-between", allowing for example Böhm's investigations on the representation of data types in the untyped λ–calculus to become type-aware [14].

A milestone was certainly reached when the first version of the Calculus of Constructions was presented, that took advantage of both Girard-Reynolds' polymorphism and of Martin-Löf dependent type theory [59, 60], which at around the same time was also beginning to attract the attention of Swedish computer scientists [66].

It is probably not by chance that linear logic arose in 1986, at a time when contacts were "stably" established between proof theory and semantics. The rise of linear logic followed Girard's revisitation of system F: in 1984, Girard constructed the first denotational model of polymorphism, reinventing Berry's stable functions for this purpose [32]. A simplification of Berry / Winskel's framework of dI-domains / event structures (see e.g. [5]) led to coherence spaces and to the seminal decomposition of implication into a linear implication and an exponential modality, leading to a procedural, resource conscious vision of logic. This resulted in the remarkably comprehensive article "Linear Logic", which was published by this very journal in 1987 [33], and has received the best cited paper award in 2005, on the occasion of the 30th birthday of the journal. This paper, which among others introduced the notion of proof net, and those which followed (on the geometry of interaction [36], on the constructive interpretation of classical logic [37], on light linear logics [38], and on ludics [39]), have exerted (and continue to have) a deep influence on many researchers in proof theory and in programming language semantics (in the large, including design and implementation issues).

Last but not least, Jean-Yves Girard is a man of courage, a creator, an explorer of unknown territories, now as much as yesterday.

Pour tout cela, nous pouvons lui tirer un beau coup de chapeau (for all these reasons, let us take our hat off to him)!

The rest of the preface consists of an expanded version of this section written in French (with apologies to those not familiar with Molière's tongue), followed by a brief description, back in English, of the contributions of the volume, classified in sort of historical order with respect to the lines of work to which they refer.

Théorie de la démonstration et langages de programmation

Le milieu des années 1980 a vu fleurir une rencontre, ou des *interactions*, pour reprendre un mot cher à Jean-Yves Girard, entre mathématiques et informatique, autour d'une correspondance, connue dans le jargon des spécialistes sous le nom d'isomorphisme de Curry-Howard. Ce court essai n'a pas l'ambition d'en retracer l'histoire, mais plutôt d'apporter quelques témoignages.

Certes, cette histoire commence plus tôt. Jean-Yves Girard a inventé le *système* F au tout début des années 70 [31]. Il s'agit d'une formalisation de la logique du second ordre qui permet d'en prouver la cohérence. Girard entrait ainsi dans l'arène avec un coup d'éclat, en résolvant ce problème ouvert connu sous le nom de conjecture de Takeuti.

Aux Etats-Unis, John Reynolds, spécialiste des langages de programmation, introduit indépendamment en 1974 les *types polymorphes*, c'est-à-dire la possibilité de donner un type très général à des fonctions opérant de la même manière sur des types qui sont tous instances d'un même type général. Reynolds présenta ces travaux lors d'un colloque international sur les langages de programmation, tenu à l'Université Paris 6 [71]. J.-Y. Girard, quant à lui, était rattaché ... à l'Université Paris 7 (sur le même campus que Paris 6), jusqu'à son départ à Marseille en 1992. Mais Reynolds ne connaissait pas le nom de Girard, et encore moins l'existence de ses travaux. La rencontre entre ces deux mondes ne se fera véritablement, à ma connaissance, qu'une dizaine d'années plus tard.

Entre temps, Girard a consacré plusieurs années à un travail de fond sur la complexité des preuves (théorie des dilatateurs, dont rend compte son livre [34]), et ne revient au système F qu'en 1984, date à laquelle il en construit le premier modèle dénotationnel, basé sur la notion de *fonction stable*, qu'il distille de son travail sur les dilatateurs, mais qui avait aussi été découverte dans un autre contexte en 1977 par Gérard Berry [11, 12]. Le rendez-vous était imminent, mais n'avait pas encore eu lieu. Girard ignorait le lien avec le travail de Berry jusqu'à ce que Berry et moi-même assistâmes à l'un de ses exposés.

Sur un plan plus international, un article de Fortune, O'Donnell et Leivant [29] a permis de jouer un rôle significatif d'"entremetteur". Leivant avait connaissance, et des travaux de Girard, et de ceux de Reynolds. Les auteurs de cet article se posaient à propos du système F des questions assez "informatiques", d'expressivité (que peut-on coder avec tel langage?), ou d'inférence de type (peut-on automatiquement calculer le type le plus général d'un programme? – par exemple, un programme qui inverse le sens d'une liste n'a pas besoin de connaitre la nature des éléments de la liste, et opère donc sur tous les types de la forme `List`(A), A quelconque). Voici ce que je sais sur l'influence de cet article (qui contenait une erreur mineure, corrigée dans [30]).

- C'est grâce à ce travail que Reynolds a pris connaissance des travaux de Girard. Il faut dire que le système F "dans le texte" (dans sa thèse d'Etat [31]) était difficile d'accès, et surtout à relier à des lignes de travaux issues de l'informatique.

- Travail de Böhm et Berarducci [14]. Böhm s'intéressait depuis longtemps (et totalement indépendamment de Girard ou de Reynolds) à la représentation de données et d'algorithmes dans le λ-calcul dit "pur" (c'est-à-dire sans types). La possibilité de typer tout cela dans le système F a dû leur apparaître a posteriori comme une belle confirmation de la pertinence de leurs travaux.

- Enfin, *last but not least*, c'est cet article que Gérard Huet avait confié à décortiquer à un jeune étudiant, Thierry Coquand, qui s'est alors empressé d'aller voir l'original. C'était l'époque du cours qui a mené au livre "Proofs and Types" [35], qu'ont suivi d'autres étudiants de cette génération, tels qu'Yves Lafont, Vincent Danos, Laurent Regnier,.... C'est, entre autres, de cette lecture, de l'enthousiasme de Huet pour l'implantation d'un dialecte du langage de programmation ML et pour les travaux du type Böhm-Berarducci qu'est né le calcul des constructions de Coquand et Huet. Bien sûr, il faut aussi invoquer la précocité scientifique et les qualités de synthèse de Coquand, qui avait aussi digéré les travaux autour du système Automath [23, 24], ainsi que ceux de Martin Löf [59, 60], lesquels s'acheminaient d'ailleurs aussi vers des applications à la programmation [66].

Avec le calcul des constructions et le développement logiciel du système Coq, dont les artisans principaux ont été, outre Huet et Coquand, Christine Paulin et Hugo Herbelin, avec aussi le système ALF développé à Göteborg à partir de la théorie des types de Martin-Löf, une communauté scientifique s'est formée, identifiée par l'acronyme TYPES, qui désigne un projet de recherche européen et une conférence annuelle. Acronyme porté également par un forum scientifique d'annonces et d'échanges scientifiques initié par Albert Meyer (`http://lists.seas.upenn.edu/mailman/listinfo/types-list`). Cette communauté met en œuvre l'isomorphisme de Curry-Howard, qui établit une correspondance entre les types (ou les spécifications) et les formules (ou les théorèmes), ainsi qu'entre les programmes et les preuves formalisées.

Les travaux de cette communauté portent tant sur des applications des mathématiques à l'informatique (développement de programmes certifiés, c'est-à-dire dont la correction est prouvée formellement avant exécution, extraction d'algorithmes et de programmes à partir de preuves de théorèmes ou de spécifications) que sur des applications de l'informatique aux mathématiques (démonstration entièrement vérifiée par ordinateur de théorèmes aussi célèbres que celui des quatre couleurs, voir `http://research.microsoft.com/~gonthier`). Citons ici les travaux de Jean-Louis Krivine [50], qui s'est attaqué à un vaste chantier: trouver le comportement algorithmique commun à toutes les preuves possibles d'un énoncé mathématique donné. Cette quête patiente et originale l'a mené à donner un sens mathématique à des instructions de langages de programmation dites "de bas niveau", telles qu'on les trouve dans les systèmes d'exploitation des ordinateurs.

Si, me semble-t-il, Girard n'était pas fasciné par le versant informatique de

Curry-Howard, en revanche, sa vision de la logique, résolument opératoire et procédurale, le plaçait de manière évidente en “pole position” pour nourrir la réflexion de cette communauté et au-delà. En sens inverse, la fréquentation du milieu informatique a certainement contribué au mûrisssement de la notion de formule comme ressource, qui est à la base de la *logique linéaire*.

Au départ, comme souvent, la découverte de la logique linéaire est le fruit d'un heureux hasard, d'une observation simple faite par Girard à propos de son modèle du système F (évoqué plus haut). Une fonction stable f admet une représentation économique, sous forme de paires (x, e'), où x est un point minimal tel que $e' \leq f(x)$, et où e' est un élément premier (c'est-à dire indécomposable en une borne supérieure de points plus petits) fixé du domaine d'arrivée de f. Cette représentation indique un début de symétrie entre entrées (x) et sorties (e'). Mais la symétrie ne devient convaincante que pour les fonctions telles que toutes ces paires (x, e') sont en fait de la forme (e, e'). Ces fonctions, Girard les a appelées linéaires, par analogie avec l'algèbre linéaire (dans cette analogie, les fonctions stables seraient les fonctions analytiques).

Arrive ensuite une profonde intuition logique. Nous sommes maintenant en présence de deux types: le type $A \to B$ des fonctions ordinaires (stables) de A dans B, et le type $A \multimap B$ des fonctions linéaires de A dans B. Par l'isomorphisme de Curry-Howard, il est naturel d'associer deux connecteurs logiques à ces types respectifs: l'implication ordinaire, et une nouvelle implication, l'implication linéaire, dont les conditions d'application sont plus restreintes. Le véritable génie fondateur de la logique linéaire est l'invention d'un autre connecteur, une modalité, permettant de relier les deux précédents par l'isomorphisme suivant:

$$A \to B \quad \cong \quad (!A) \multimap B \ .$$

La lecture procédurale est la suivante: pour prouver $A \to B$, on peut utiliser A comme lemme autant de fois que l'on veut, ou, ce qui revient au même, on utilise une et une seule fois $!A$ (lire “bien sûr A”). Ainsi, $!A$ est une ressource inépuisable, tandis que A, tel un euro dans votre poche, ne vaut qu'un euro. Ainsi, l'achat d'une boisson B coûtant deux euros se formalise par la formule $(A \otimes A) \multimap B$. Bien sûr, un riche emir en possession de $!A$ peut a fortiori s'acheter B: il lui suffira de détacher successivement deux copies de A (opération que l'on peut formaliser par l'implication linéaire $(!A) \multimap A \otimes (!A)$). La distinction entre linéaire et non linéaire amène à considérer deux formes de conjonction: $\otimes$ (que l'on vient d'utiliser) et &, également liées par un isomorphisme:

$$(!A) \otimes (!B) \quad \cong \quad !(A \& B) \ .$$

La négation, dans cette logique, est une involution (car débarrassée de toute interférence implicite avec la modalité !). De ce fait, la logique linéaire garde à la fois l'élégance de la logique classique (avec ses dualités, ses symétries), et le caractère constructif de la logique intuitionniste (élimination des coupures confluente, propriétés dites de la disjonction et de l'existence qui affirment que d'une preuve de A ou B on peut extraire une preuve de A ou une preuve de B et d'une preuve de $\exists x A(x)$ on peut extraire un n précis tel que $A(n)$).

La logique linéaire fut mise sur pied en quelques mois seulement, et donna lieu à un article dense, au retentissement considérable [33]. Cet article a d'ailleurs reçu le prix de l'article le plus consulté de la revue Theoretical Computer Science. Il présentait les axiomes et règles d'inférence de la logique linéaire, prouvait (avec un trou comblé récemment par Tortora de Falco et Pagani [68], en utilisant des techniques développées dans la thèse de Danos [21]) une version forte (confluente) de l'élimination des coupures, présentait une sémantique des formules (suffisante pour fournir une preuve simple de la cohérence de la logique), et une sémantique dénotationnelle (stable) des formules et des preuves, et, *last but not least*, introduisait la notion de *réseau de preuve*, remplaçant (quotientant) la notion d'arbre de preuve par un graphe de preuve. Il s'agit là peut-être de l'idée la plus novatrice:

- Les preuves ainsi représentées ressemblent à des circuits, dont la dynamique peut être décrite non plus par réécriture (comme dans le λ-calcul ou comme dans le processus d'élimination des coupures, où chaque étape transforme la preuve en une preuve de "complexité" moindre), mais par la circulation d'informations à travers le graphe. C'est cette interprétation (neuve en logique, mais qui était dèjà au cœur du modèle des algorithmes séquentiels de Berry-Curien [13]) qui a conduit ensuite Girard au programme de la *géométrie de l'interaction* [36], laquelle peut se formuler tant en termes concrets comme il vient dêtre suggéré qu'en termes mathématiques dans des algèbres d'opérateurs [40].

- Les réseaux de preuve sont plongés dans un monde plus vaste de "structures de preuve", et peuvent être caractérisés parmi ces dernières comme celles qui "convergent" ou "gagnent" contre un ensemble d'adversaires. Ces idées, entre autres, sont à la base de la ludique, développée par Girard au tournant du nouveau millénaire.

L'influence, directe ou diffuse, de la logique linéaire sur l'ensemble du milieu de la *sémantique* (qui recouvre le continuum allant de la conception à l'implantation des langages de programmation et des environnements de programmation, en passant par la certification de programmes, et sur un plan plus théorique par les modèles mathématiques des langages de programmation) est considérable. Il n'est pas ici question de dresser une liste exhaustive des travaux issus ou influencés explicitement ou implicitement par les idées de la logique linéaire. Le choix qui suit est lié à mon propre parcours, et à mes lectures.

- La première application informatique de la logique linéaire a été, à ma connaissance, la Machine Abstraite Linéaire de Lafont [51]. Cette machine tirait profit de ce qu'un type linéaire (i.e. qui n'est pas de la forme $!A$) permet de marquer les données de ce type comme "jetables" après usage, ce qui rend possible une gestion fine de la place mémoire au cours du calcul. D'autres travaux sur logique linéaire et gestion de la mémoire ont suivi, comme [44].

- Le lien entre formules et ressources en logique linéaire a permis des codages très précis et naturels du modèle de calcul distribué le plus ancien, les réseaux de Petri [61, 27]. Mais on peut aussi bien coder de manière très fidèle toutes sortes d'automates, dont par exemple les automates à deux compteurs, et c'est ce qui a permis de prouver l'indécidabilité de la logique linéaire propositionnelle du premier ordre [57].

- De nombreux travaux ont développé des langages de preuves, et donc par Curry-Howard des langages de programmation, pour des fragments ou variantes diverses de la logique linéaire (voir par exemple [72, 58]).

- On peut aussi partir de langages de programmation connus, et en inférer des types affinés avec des indications de linéarité qui peuvent ensuite être utilisés pour des optimisations telles que l'"inlining" (voir par exemple [4], voir aussi plus loin la rubrique "complexité implicite").

- La programmation logique (où les programmes sont exprimés comme des formules, dont on recherche les instances valides) a été revisitée à la lumière de la logique linéaire [70, 63]. C'est dans ce contexte qu'a été mise en valeur une autre idée, non explicite dans l'article initial de Girard, celle de *polarité*. Andreoli a observé que les connecteurs de la logique linéaire étaient soit actifs et irréversibles, soit passifs et réversibles. Il est alors possible d'organiser une formule en une cascade de connecteurs synthétiques regroupant de manière maximale les occurrences successives de connecteurs de la même polarité, et de programmer la recherche de preuves en exploitant cette polarisation, sans perte de complétude et avec un fort gain d'efficacité . La polarisation est aussi un point d'entrée de la ludique (voir plus bas), et aussi de la logique linéaire polarisée d'Olivier Laurent [53].

- Les réseaux de preuve, ou plus exactement une version locale de ces derniers, ont permis d'éclairer et de prouver la correction de l'algorithme de Lamping, qui implante la réduction optimale dans le λ-calcul au sens de Jean-Jacques Lévy [55, 56]. Ces travaux, dus à Gonthier (en collaboration avec Abadi et Lévy) [41, 42], ont été prolongés par Asperti, Laneve, Martini, Danos et Regnier. Ce riche ensemble de résultats est décrit dans le livre [7].

- En amont, la sémantique dénotationnelle a été profondément influencée par la logique linéaire. La quête de modèles très fidèles à la syntaxe a amené le développement des sémantiques de jeux [1, 2, 48] (pour des survols, voir [3, 20], voir aussi [62] pour un modèle de jeux de la logique linéaire complète). Elle a aussi amené à revoir de manière plus fine des modèles plus anciens du λ-calcul, comme celui des algorithmes séquentiels [13, 16] (cf. plus haut) ou des bistructures [19]. De nouveaux modèles ont été développés, comme ceux des hypercohérences, et des espaces de finitude (tous deux dus à Thomas Ehrhard) [25]. Ce dernier modèle – parfaite illustration de l'apport que peut fournir la sémantique dénotationnelle à la conception de langages – a suggéré l'ajout d'*opérateurs différentiels* au λ-calcul et aux réseaux de preuve.

- La représentation aisée des réseaux de Petri, le parallélisme inhérent aux réseaux de preuve (dans lesquels les entrées et les sorties ne sont pas distinguées), indiquent des liens avec les calculs distribués. Si le choix par Milner du symbole ! pour l'opérateur de réplication dans le π-calcul [64] est indépendant de la logique linéaire, des liens forts entre π-calcul et logique linéaire ont été établis:

  – soit par restriction du π-calcul à des fragments typés [49, 10],

  – soit par extension de la logique linéaire par des opérateurs différentiels (cf. plus haut) [26].

  La linéarité est aussi fortement présente dans les travaux de Winskel (modèles de préfaisceaux, modèles de "spans") [73].

Les années 90 ont aussi vu la mise à jour d'une autre importante extension de l'isomorphisme de Curry-Howard. Griffin [43] a montré comment la négation classique avait pour contrepartie des opérateurs dits de contrôle, qui permettent de capturer le contexte d'exécution d'un programme pour le restaurer ensuite le cas échéant, et permettre ainsi notamment de programmer des exceptions. Peu après, Girard fournissait le premier calcul des séquents avec élimination des coupures confluente pour la logique classique, sans référence aux opérateurs de contrôle, mais en faisant usage de polarités (cf. plus haut) [37]. L'analyse du contenu constructif de la logique classique a donné lieu à de nombreux travaux, faisant une part plus ou moins importante aux liens avec la programmation ($\lambda\mu$-calcul [69], $\lambda\overline{\lambda}\mu\tilde{\mu}$-calcul [18]), à la logique linéaire (travaux de Danos-Joinet-Schellinx [22]), ou aux polarités (travail de Laurent-Regnier [54], système L focalisé [65]).

La logique linéaire croise enfin le domaine de la complexité. En limitant le pouvoir d'expression (ou les règles d'inférence) des connecteurs dits exponentiels (! et son dual ?), on obtient des logiques dites légères, qui correspondent à des classes de complexité (polynomiale, élémentaire) [38]. Les travaux actuels de Patrick Baillot, Ugo dal Lago et d'autres portent sur les rapports entre cette complexité bornée et le développement d'algorithmes efficaces et naturels, ainsi qu'avec d'autres approches à la complexité implicite (dénomination qui recouvre les travaux où la complexité est automatiquement contrôlée par le langage dans lequel on s'exprime, un peu comme une forme de correction partielle pour un programme est automatiquement assurée par son bon typage).

Il est trop tôt pour évaluer l'impact des travaux ultérieurs de Girard autour de la ludique [39] et des algèbres d'opérateurs.

- La première est une tentative pour synthétiser les règles de la logique à partir de la seule dynamique (interactive) des calculs. La ludique, très apparentée à la sémantique des jeux (voir en particulier l'approche très opérationnelle adoptée dans [17]), a ceci de singulier qu'elle critique et dépasse le cadre confortable procuré par la théorie des catégories, où tout est "à isomorphisme près", rejoignant en cela les premiers travaux sur les modèles du λ-calcul [46, 8], dans lesquels l'interprétation des types était forgée au sein d'un modèle fixé non typé, et des remarques de Luca Cardelli sur l'impossibilité de donner une interprétation du sous-typage dans un cadre à isomorphisme près. La ludique apporte une nouvelle dimension, *spatiale*, à la lecture opératoire des formules. Ainsi, en ludique, il existe plusieurs notions de tenseur, selon la manière dont les preuves s'entendent pour partager l'espace.

- Le travail sur les secondes est une sorte de recherche du Graal. Outre que les algèbres d'opérateurs fournissent le cadre mathématique le plus général pour résoudre les équations de la géométrie de l'interaction, elles semblent receler des clés pour un traitement intrinsèque des classes de complexité.

La direction présente des travaux de Girard est marquée par une mathématisation accrue (tendance d'ailleurs partagée par l'évolution récente des travaux théoriques en sémantique, qui utilisent, ou développent davantage de notions de topologie et de géométrie), et par un investissement dans une réflexion épistémologique, à laquelle il attache une grande importance, et qui nourrit en retour son travail de mathématicien.

About the contributions of this volume

Strong normalisation. Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco exploit and refine the (multiset) relational semantics of linear logic in order to characterise the untyped normalisable MELL proofs and to compute the number of cut elimination steps leading to the cut free normal form (following a suitable strategy).

Luca Paolini, Elaine Pimentel, and Simona Ronchi Della Rocca revisit the question of strong normalisation in the (untyped) call-by-name λ-calculus and lazy λ-calculus. In an intriguing way, they take advantage of the fact that these respective calculi share the same syntax of terms with their call-by-value variants (and in fact any variant parametrised by a set of *values*, not necessarily the classical ones giving rise to the original call-by-value λ-calculus of Plotkin), and provide a characterisation of the strong normalisability property in terms of a property that makes sense in the (generalised) call-by-value framework.

Realisability. Realisability is pervasive in the study of logic and programming languages (notably in proofs of strong normalisation, or more generally of properties of programs). Benno van den Berg and Ieke Moerdijk construct realisability models for constructive set theories, by building on their previous work in algebraic set theory, thus showing how the former fits into the latter.

Proof nets. Paulin Jacobé de Naurois and Virgile Mogbil revisit the correctness of proof structures in linear logic. They provide a new correctness criterion for MELL (and then for MALL) and show that it is NL-complete, thus providing for the first time a lower bound for the complexity of this problem.

As for upper bounds (for deterministic computation), Stefano Guerrini presents the full details of his linear algorithm for checking the correctness of proof nets for MLL without units (originally presented at the LICS conference in 1999).

Paolo Tranquilli extends to exponentials Ehrhard-Regnier's framework of differential interaction nets, and provides a translation that extends their translation from Boudol's resource calculus (equipped with some "differential" reductions) to these nets with boxes. The extended source of this translation has terms admitting possibly infinitely usable resources.

Geometry of interaction. Jean Goubault-Larrecq proposes a new, "direct", construction of a categorical model of MLL, obtained by reverse-engineering of Danos-Regnier's weighted paths. This certainly deserves to be called a Geometry of Interaction (GOI) model, given its explicit construction that remains close to Girard's original ideas and framework. Goubault-Larrecq then goes on and proves strong negative results on the impossibility of modelling additive and exponentials in this model (actually, a family of models, parametrized by an inverse monoid with some additional properties). But an abstract construction that blends coherence spaces with any model of MLL yields a model for full linear logic.

On the other hand, Esfandiar Haghverdi follows the approach initiated by Abramsky and others consisting in identifying "GOI situations" in an abstract framework based on traced monoidal categories, the idea being that traces represent an abstraction of the actual GOI interaction (expressed e.g. at the level of paths, cf. above) that works in a feed-back loop fashion. In this paper, Haghverdi proves a general fixpoint theorem, showing that this general semantic framework can be extended from logic (total functions) to programming (with recursion).

Last but not least, Jean-Yves Girard delivers here his latest reformulation of the Geometry of Interaction in the framework of operator algebras, opening the way to brand new angles of attack in complexity theory.

Implicit complexity. Ugo Dal Lago and Martin Hofmann apply semantic realisability models that they recently found and investigated for their own value in a previous paper to give new proofs of soundness for several languages that have been proposed for bounded complexity – some based on linear logic, and some others not – and to guide the design of sound extensions of these languages.

Ludics. Kazushige Terui proposes a slightly abstracted version of ludics (without locations) which remains faithful to Girard's original framework in that it allows for an internal representation of both machines (automata of various sorts) and data in the same language, thus turning the heterogeneous notion of recognition of a word (or a term) by an automaton into the homogeneous interaction at work in ludics (or, equivalently, in an untyped version of game semantics on a universal arena), between two designs (or strategies).

An application of linear logic to planning. Max Kanovich and Jacqueline Vauzeilles pursue a line of work on the use of linear logic for formalising and solving planning problems. Here, they address planning under temporal uncertainty caused by actions with delayed effects. The formalisation takes the form of the search for specific winning strategies, which are encoded as LL proofs.

References

[1] S. Abramsky and R. Jagadeesan, Games and full completeness for multiplicative linear logic, Journal of Symbolic Logic 59, 543-574 (1994).

[2] S. Abramsky, R. Jagadeesan and P. Malacaria, Games and full abstraction for PCF, Information and Computation 163 (2), 409-470 (2000).

[3] S. Abramsky and G. McCusker, Game semantics, in Computational Logic, U. Berger and H. Schwichtenberg eds, Springer Verlag (1999).

[4] F. Alberti, Analyse statique typée des propriétés structurelles des programmes, Thèse de doctorat, Univ. Paris VII, mai 2005.

[5] R. Amadio and P.-L. Curien, Domains and Lambda-calculi, Cambridge University Press (1998).

[6] J.-M. Andreoli. Logic programming with focusing proofs in linear logic, Journal of Logic and Computation 2(3), 297-347 (1992).

[7] A. Asperti and S. Guerrini, The optimal implementation of functional programming languages, Cambridge University Press (1998).

[8] H. Barendregt, M. Coppo, and M. Dezani, A filter lambda model and the completeness of type assignment, Journal of Symbolic Logic 48, 931-940 (1983).

[9] J. Berdine, P. W. O'Hearn, U. Reddy, and H. Thielecke, Linear continuation passing, Higher Order and Symbolic Computation 15(2/3) 181-208 (2002).

[10] M. Berger, K Honda, and N. Yoshida, Sequentiality and the Pi-Calculus, in Proc.of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01), Lecture Notes in Computer Science 2044, 29-45, Springer (2001).

[11] G. Berry, Stable models of typed lambda-calculi, in Proc. International Colloquium on Algebra, Logic and Programming (ICALP' 78), Lectures Notes in Computer Science 62, 72-89, Springer (1978).

[12] G. Berry, Modèles complètement adéquats et stables des lambda-calculs typés, Thèse de Doctorat d'Etat, Université Paris VII (1979).

[13] G. Berry and P.-L. Curien, Sequential algorithms on concrete data structures, Theoretical Computer Science 20, 265-321(1982).

[14] C. Böhm and A. Berarducci, Automatic synthesis of typed lambda-programs on term algebras, Theoretical Computer Science 39, 135-154 (1985).

[15] T. Coquand and G. Huet, A calculus of constructions, Information and Computation 76, 95-120 (1988).

[16] P.-L. Curien, On the symmetry of sequentiality, in Proc. Mathematical Foundations of Programming Semantics 1993, Lecture Notes in Computer Science 802, 29-71 (1993).

[17] P.-L. Curien, Abstract Böhm trees, Mathematical Structures in Computer Science 8(6), 559-591 (1998).

[18] P.-L. Curien and H. Herbelin, The duality of computation, Proc. Int. Conf. on Functional Programming 2000, ACM Press.

[19] P.-L. Curien, G. Plotkin, and G. Winskel, Bistructures, bidomains, and linear logic, Milner Festschrift, MIT Press (1999).

[20] P.-L. Curien, Notes on game semantics, available from `www.pps.jussieu.fr/~curien` (2006).

[21] V. Danos, La logique linéaire appliquée à l'étude de divers processus de normalisation (principalement le λ-calcul), Thèse de Doctorat, Université Paris VII (1990).

[22] V. Danos, J.-B. Joinet, H. Schellinx, A new deconstructive logic: linear logic, J. of Symbolic Logic 62(3) 755-807 (1997).

[23] N. de Bruijn, The mathematical language Automath, its usage and some of its extensions, in Symposium on Automatic Demonstration, IRIA, Versailles, Dec. 1968, Lecture Notes in Mathematics 125, 29-61, Springer (1970).

[24] N. de Bruijn, A survey of the project Automath, in To H.B. Curry: essays on combinatory logic, lambda-calculus, and formalism, Hindley and Seldin (eds.), 479-490, Academic Press (1980)

[25] T. Ehrhard, Hypercoherences: a strongly stable model of linear logic, Mathematical Structures in Computer Science 3, 365-385 (1993).

[26] T. Ehrhard and O. Laurent, Interpreting a Finitary Pi-Calculus in Differential Interaction Nets, Information and Computation 208 (6), 606–633 (2010).

[27] U. Engberg and G. Winskel, Petri nets as models of linear logic, in Proc. 15th Coll. Trees in Algebra and Programming (CAAP), Lecture Notes in Computer Science 431, 147-161, Springer (1990).

[28] A. Filinski, Linear continuations. In Conference Record of the Nineteenth Annual Symposium on Principles of Programming Languages (POPL'92), 27-38, Albuquerque, New Mexico, ACM Press (1992).

[29] S. Fortune, D. Leivant, and M. O'Donnell, The expressiveness of simple and second-order type structures, Journal of the ACM 30(1), 151–185 (1983).

[30] S. Fortune, D. Leivant, and M. O'Donnell, Typing and computational properties of lambda expressions. Theoretical Computer Science 44, 51-68 (1986).

[31] J.-Y. Girard, Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur, Thèse d'Etat, Université Paris VII (1972).

[32] J.-Y. Girard,The system F of variable types, fifteen years later, Theoretical Computer Science 45, 159-192 (1986).

[33] J.-Y. Girard, Linear Logic, Theoretical Computer Science 50, 1-101 (1987).

[34] J.-Y. Girard, Proof-theory and logical complexity I, Bibliopolis, Napoli, 1987.

[35] J.-Y. Girard, Y. Lafont, and P. Taylor, Proofs and Types, Cambridge University Press (1989).

[36] J.-Y. Girard, Geometry of interaction I: interpretation of system F, in Proc. Logic Colloquium '88, 221-260, North Holland (1989).

[37] J.-Y. Girard, A new constructive logic: classical logic, Math. Struct. in Computer Science 1, 255-296 (1991).

[38] J.-Y. Girard, Light linear logic, Information and Computation 143(2), 175-204 (1998).

[39] J.-Y. Girard, Locus solum: from the rules of logic to the logic of rules, Mathematical Structures in Computer Science 11(3), 301-506 (2001).

[40] J.-Y. Girard, Geometry of interaction V: logic in the hyperfinite factor, dans ce volume.

[41] G. Gonthier, M. Abadi and J.-J. Lévy, The geometry of optimal lambda reduction, in Proc. Principles of Programming Languages 1992.

[42] G. Gonthier, M. Abadi and J.-J. Lévy, Linear logic without boxes, in Proc. Logic in Computer Science 1992.

[43] T. Griffin, A formulae-as-types notion of control, in Proc. ACM Principles of Programming Languages (1990).

[44] J. Guzman and P. Hudak, Single-threaded polymorphic lambda calculus, in Proc. of the Fifth IEEE Symposium on Logic in Computer Science, Philadelphia, IEEE Computer Society Press (1990).

[45] M. Hasegawa, Linearly used effects: monadic and CPS transformations into the linear lambda calculus, in Proc. 6th International Symposium on Functional and Logic Programming (FLOPS2002), Aizu, Springer Lecture Notes in Computer Science 2441, 167-182 (2002).

[46] R. Hindley and G. Longo, Lambda-calculus models and extensionality, Zeit. Math. Logik Grund. Math. 26, 289-310 (1980).

[47] W. Howard, The formulas-as-types notion of construction, in To H.B. Curry: essays on combinatory logic, lambda-calculus, and formalism, Hindley and Seldin (eds.), 479-490, Academic Press (1980) (manuscript circulated since 1969).

[48] M. Hyland and L. Ong, On full abstraction for PCF, Information and Computation 163, 285-408 (2000).

[49] N. Kobayashi, B. Pierce, and D. Turner, Linearity and the pi-calculus, Proc. of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'96), 358-371, St. Petersburg Beach, Florida, ACM Press (1996).

[50] J.-L. Krivine, Réalisabilité classique, in Interactive models of computation and program behaviour, Panoramas et Synthèses, Société Mathématique de France (2010).

[51] Y. Lafont, The linear abstract machine, Theoretical Computer Science 59, 157-180 (1988).

[52] J. Lamping, An algorithm for optimal lambda calculus reduction. Proc. Principles of Programming Languages 1990, 16-30.

[53] O. Laurent, Syntax vs.Semantics: a polarized approach, Theoretical Computer Science, 343(1–2), 177–206 (2005).

[54] O. Laurent and L. Regnier, About translations of classical logic into polarised linear logic, Proc. LICS 2003, ACM Press.

[55] J.-J. Lévy, Réductions correctes et optimales dans le lambda-calcul, Thèse d'Etat, Université Paris 7 (1978).

[56] J.-J. Lévy, Optimal reductions in the lambda-calculus, in To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism, J. Seldin and R. Hindley eds, 159-191, Academic Press (1980).

[57] P. Lincoln, J. Mitchell, A. Scedrov, Decision problems for propositional linear logic, Annals of Pure and Applied Logic 56, 239-311 (1992).

[58] I. Mackie, Lilac - a functional programming language based on linear logic, Journal of Functional Programming, 4(4), 395-433 (1993).

[59] P. Martin-Löf, Lecture notes on the domain interpretation of type theory, in Proc. Workshop on Semantics of Programming Languages, Chalmers University of Technology (1983).

[60] P. Martin-Löf, Intuitionistic type theory, Bibliopolis (1984).

[61] N. Marti-Oliet and J. Meseguer, From Petri Nets to Linear Logic, in Proc. Category Theory and Computer Science 1989, Lecture Notes in Computer Science 389, 313-340, Springer (1989).

[62] P.-A. Melliès, Asynchronous games 4: A fully complete model of propositional linear logic, Proc. of the 20th Conference on Logic in Computer Science, Chicago, 2005.

[63] D. Miller, Overview of linear logic programming, in Linear Logic in Computer Science, T. Ehrhard, J.-Y. Girard, P. Ruet, and P. Scott eds., London Mathematical Society Lecture Notes 316, 119-150 Cambridge University Press (2004).

[64] R. Milner, J. Parrow, and D. Walker, A Calculus of Mobile Processes I and II, Information and Computation 100(1), 1–40 and 41–77 (1992).

[65] G. Munch-Maccagnoni, Focalisation and classical realisability, Proc. CSL 2009 LNCS 5771, 409–423, Springer.

[66] B. Nordström, K. Petersson, and J. Smith, Programming in Martin-Löf's Type Theory, Oxford University Press (1990) (out of print, but downloadable from `www.cs.chalmers.se/Cs/Research/Logic/book`).

[67] P. O'Hearn and D. Pym, The logic of bunched implications. Bulletin of Symbolic Logic 5(2), 215-244 (1999).

[68] M. Pagani, L. Tortora de Falco, Strong normalization property for second order linear logic, Theoretical Computer Science 411(2), 410-444 (2010).

[69] M. Parigot, $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction, in Proc. of the Int. Conf. on Logic Programming and Automated Reasoning, St. Petersburg, LNCS 624 (1992).

[70] D. Pym, J. Harland, A uniform proof-theoretic investigation of linear logic programming. Journal of Logic and Computation 4(2), 175-207 (1994).

[71] J. Reynolds, Towards a theory of type structure, Proc. Colloque sur la Programmation, April 9–11 1974, Paris, B. Robinet ed., Lecture Notes in Computer Science 19, 408–425, Springer (1974).

[72] P. Wadler, Linear types can change the world. In M. Broy and C. B. Jones, editors, IFIP TC 2 Working Conference on Programming Concepts and Methods, 561-581, North-Holland (1990).

[73] G. Winskel and M. Nygaard, Linearity in process languages, in Proc. Logic in Computer Science LICS'02, IEEE Press (2002).