

# Projet de Recherche PPS

## (Preuves, Programmes et Systèmes)

Décembre 2007

Le présent rapport décrit les activités de recherche menées dans le laboratoire Preuves, Programmes et Systèmes sur la période des 4 années universitaires de 2003 à 2007. Nos recherches couvrent les domaines de la programmation (incluant la réécriture et la concurrence) et des outils mathématiques pour l'aide à la conception, à la compréhension, à la preuve des programmes, des logiciels, des systèmes, des architectures logicielles, etc. Notre spectre est largement ouvert,

- entre théorie et pratique (fertilisation dans les deux sens),
- par la variété des logiciels concernés par nos recherches,
- par la variété des théories mathématiques mises en œuvre (théorie de la démonstration, théorie des catégories, algèbre homologique et homotopie, probabilités, ...).

De tous ces points de vue, le laboratoire a fait du chemin depuis sa création en janvier 1999. Il a quasiment triplé de taille depuis cette date, mais sans jamais déroger à une grande unité thématique, puisée dans une conviction commune qu'il n'y a pas de bons logiciels sans de solides fondations mathématiques. Rappelons et *soulignons* ici que le laboratoire n'est pas organisé en équipes, ni même en thèmes constitués. Les quatre thèmes qui sont présentés ici sont plutôt le résultat d'une volonté d'organiser notre rapport de recherche de manière structurée et reflétant le paysage actuel de notre recherche (il y avait 6 thèmes dans le rapport quadriennal précédent). La dernière section de ce rapport souligne certaines de nos synergies internes et externes. Le rapport intègre une dimension prospective, au cours du texte, et dans les sous-sections finales de chaque thème. La coordination de la rédaction a été assurée par :

- Thomas Ehrhard et Paul-André Melliès pour le thème 1 (structures mathématiques de la programmation),
- Delia Kesner et Alexandre Miquel pour le thème 2 (réécriture et preuves),
- Roberto Amadio et Vincent Danos pour le thème 3 (concurrence et modélisation),
- Giuseppe Castagna et Roberto Di Cosmo pour le thème 4 (outils logiciels).

**Conventions.** Au début de chaque thème, nous indiquons le sous-ensemble du laboratoire concerné (bien entendu, ces sous-ensembles se recouvrent largement). Les personnes sur poste permanent ont leur nom souligné. Nous n'indiquons pas les prénoms dans le corps du texte, sauf (et seulement la première fois qu'ils sont mentionnés) pour les coauteurs d'articles de membres du laboratoire.

## 1 Structures mathématiques de la programmation

**Participants.** Emmanuel Beffara, Chantal Berline, Antonio Bucciarelli, Daniel De Carvalho, Pierre Clairambault, Pierre-Louis Curien, Vincent Danos, Joachim de Lataillade, Ugo Dal Lago, Thomas Ehrhard (depuis 2005), Claudia Faggian (depuis 2006), Philippe Gaucher, Stéphane Gimenez, Russell Harmer, Thierry Joly, Olivier Laurent, Giulio Manzonetto, Paul-André Melliès,

François Métayer, Samuel Mimram, Vincent Padovani, Mauro Piccolo, Nicolas Tabareau, Christine Tasson, Daniele Varacca (depuis 2006).

La programmation est une activité tout à la fois pratique, et purement immatérielle. Programmer, en effet, c’est produire un objet formel : le programme, qui sera ensuite exécuté par une machine. Cette machine est elle-même (au-delà de son support matériel) un dispositif symbolique. Ses différents processeurs exécutent de manière coordonnée les instructions élémentaires d’un langage assembleur – dont les diverses instructions et leurs effets sur l’état courant de la machine sont décrits formellement.

Pour cette raison, l’étude de la programmation relève des mathématiques, au même titre que l’étude de la logique formelle. L’enjeu à la fois conceptuel et technologique de cette mathématique des langages de programmation est de mettre à jour et d’étudier des notions opératoires permettant de raisonner sur les programmes, et en particulier, de prouver formellement leur correction, c’est-à-dire, leur adéquation à une spécification.

Les langages de programmation ont été développés avec des motivations pragmatiques fortes. Ils fourmillent d’inventions et d’astuces qui rendent la programmation plus facile et plus conceptuelle. La branche de l’informatique fondamentale qu’on appelle “sémantique” vise à donner des bases mathématiques solides à cette activité symbolique. L’étude mathématique de la programmation trouve, pour nous, ses origines dans trois principales théories.

La première est la *théorie de la récursivité*, née dans les années 1930 de travaux de logiciens qui avaient besoin d’une définition mathématique des fonctions calculables. Elle a mis sur pied plusieurs formalismes très différents qui ont ensuite été montrés équivalents : le  $\lambda$ -calcul pur, les schémas de définition récursive de Herbrand-Gödel et les machines de Turing. Plus tard, dans les années 1960, le  $\lambda$ -calcul a servi de fondement aux langages de programmation fonctionnels (McCarthy), et à la sémantique dénotationnelle à base de domaines (Strachey, Scott). Cette dernière a ensuite connu de formidables développements, avec l’introduction des fonctions stables par Berry puis Girard, du concept de séquentialité (Milner, Vuillemin, Sazonov), de l’introduction des structures de données concrètes par Kahn et Plotkin pour la sémantique des réseaux de Kahn (un modèle du calcul concurrent), de l’introduction des techniques de relations logiques, analogue sémantique de la réalisabilité etc. Les machines de Turing quant à elles sont à la base de la théorie de la complexité algorithmique, car elles donnent une façon uniforme de compter les opérations élémentaires. Poussant l’approche “Herbrand-Gödel” vers l’ordre supérieur, Kleene en était arrivé à développer une approche qui préfigurait la sémantique des jeux, laquelle fut découverte à la même époque au cours de travaux de sémantique dénotationnelle (par Berry et Curien, dans la lignée des travaux de Scott). Cette approche, où un programme est représenté par l’ensemble des suites d’interactions qu’il peut avoir avec son environnement (ces suites sont des parties dans un jeu) a eu un grand succès dans les années 1990, car elle a permis de construire des modèles dénotationnellement complets du langage PCF, puis de diverses extensions impératives de ce noyau purement fonctionnel. Rappelons ici que le langage PCF est un  $\lambda$ -calcul simplement typé étendu avec les opérations arithmétiques usuelles, le branchement conditionnel (*if-then-else*), ainsi qu’un opérateur de récursion à tous les types.

La seconde est la *théorie de la démonstration*, à travers ce qu’on appelle la correspondance de Curry-Howard. C’est la constatation, limitée originellement à la logique intuitionniste, du fait que les démonstrations peuvent être vues comme des programmes (ou plus précisément, des  $\lambda$ -termes), et que l’élimination des coupures des preuves correspond à la beta-réduction des  $\lambda$ -termes. Cela mène à une compréhension approfondie des liens entre programmes et formules logiques (types) : typage, et, plus généralement, réalisabilité. Cette correspondance a été élargie à la logique classique, pour la première fois par Griffin, qui a montré que les principes classiques ont une interprétation opérationnelle (la construction *call-cc* de *Scheme*). La correspondance

de Curry-Howard a été considérablement élargie par l'introduction de la logique linéaire par Girard au milieu des années 1980. Cela a permis notamment d'introduire de nouvelles notions, asynchrones, de preuves (les réseaux) et a donné un point de vue nouveau sur la sémantique dénotationnelle, la plaçant dans le cadre mathématique plus riche des catégories monoïdales. La sémantique des jeux, déjà évoquée, trouve son autre origine dans la théorie de la démonstration, avec les intuitions interactives de Gentzen sur le calcul des séquents, les travaux de Lorentzen qui proposent une interprétation du calcul des séquents en termes de jeux, et le travail de Blass sur les jeux pour la logique linéaire qui, au début des années 1990 a beaucoup contribué à l'explosion de la recherche dans ce domaine. Ces travaux ont mené aux théorèmes de complétude de Hyland-Ong/Nickau (HO) d'une part, et de Abramsky-Jagadeesan-Malacaria (AJM) de l'autre, qui sont des manifestations du pouvoir expressif de la sémantique des jeux.

La troisième source de mathématisation de l'informatique qui nous intéresse ici n'est pas de nature logique, mais plus directement algébrique, topologique et catégorique. Elle résulte d'une volonté de généraliser la théorie de la réécriture, pour l'adapter à des objets plus généraux que les mots ou les termes. On est alors amené à considérer des objets de "dimension supérieure" (un mot est de dimension 1, un arbre de dimension 2), représentés comme des cellules (ou des composées de cellules) dans des graphes de dimension supérieure, comme les polygraphes de Burroni. On arrive ainsi également à représenter les réécritures, puis les standardisations (réécriture sur les réécritures), etc., comme des cellules de dimensions de plus en plus élevées. Outre des applications à la théorie de la réécriture, à l'algèbre (dans la lignée des travaux de Squier), ce genre d'approche homotopique s'applique également à la concurrence, en généralisant les automates à la dimension supérieure (une 2-cellule correspond alors à 2 transitions pouvant avoir lieu en parallèle). Il s'agit, dans cette dernière approche, de rendre compte des phénomènes d'exclusion mutuelle en étudiant la géométrie des "trajectoires" d'évolution de systèmes concurrents, vues comme des courbes dans des espaces de dimension supérieure (typiquement, autant que de processus élémentaires). On peut ainsi associer un invariant algébrique (une catégorie finie typiquement) à un système concurrent, sur lequel il est possible de détecter, de façon statique, les situations de blocages. Le problème est bien sûr de rendre cet invariant le plus "petit" possible. La théorie catégorique de l'homotopie (catégories modèles de Quillen) s'avère offrir un cadre naturel pour mener cette étude.

Ce thème s'articule en quatre sous-thèmes : jeux et concurrence, modèles du  $\lambda$ -calcul pur et de PCF, logique linéaire et concurrence, méthodes algébriques et homotopiques.

## 1.1 Jeux et concurrence

Les modèles de jeux permettent d'interpréter des programmes, même très fonctionnels et très abstraits, à base d'opérations élémentaires de très bas niveau, échangés entre un Joueur (le programme) et un Opposant (l'environnement). En cela, la sémantique des jeux s'apparente à une sémantique de trace (au sens de la théorie de la concurrence) des langages de programmation. La nouveauté est que ces sémantiques de traces sont intégrées dans un cadre à la fois :

- interactif : la dualité Opposant/Joueur est une notion primitive dans les jeux,
- modulaire : les stratégies décrivent des morphismes de catégories, qui peuvent donc être composés,
- algébrique : ces catégories de jeux et stratégies sont monoïdales ou cartésiennes fermées.

La grande ductilité des modèles de jeux et leur aspect "bas niveau" ont été mis à profit pour développer des modèles de différentes extensions des langages de programmation purement fonctionnels : exceptions, références, etc. Cette ligne de recherche lancée au milieu des années 1990 débouche aujourd'hui sur des outils d'analyse et de certification des programmes à la compilation – outils associant sémantique des jeux et *interprétation abstraite*, cf. notamment les travaux de

Ghica à Birmingham [AGMO04, DGL06]. De ce point de vue technologique, la sémantique des jeux arrive bien à maturité.

La sémantique des jeux est loin, cependant, d’avoir dévoilé tous ses secrets. Malgré des progrès rapides depuis quinze ans, la sémantique des jeux reste un sujet émergent, et plein de promesses pour l’avenir. Ainsi, un enjeu très actuel est de l’intégrer de manière harmonieuse au sein du champ plus large des mathématiques des langages de programmation : logique linéaire, théorie de la concurrence, théorie des types et quantification, théorie monadique des effets. La recherche de cette intégration est fructueuse, car elle nécessite de transformer le champ tout entier, par une lecture “dynamique” et “interactive” de ses concepts cardinaux.

Cette recherche part naturellement des concepts d’*innocence* et de *bon parenthésage* des stratégies, introduits par Hyland et Ong [HO00, Nic94] pour caractériser les stratégies interactives implémentables par un programme du langage PCF. Le théorème de complétude dénotationnelle qui découle de cette caractérisation indique que ces deux conditions purement combinatoires sur les stratégies reformulent la syntaxe du  $\lambda$ -calcul et des preuves. Après la description des progrès importants et variés réalisés sur ces deux conditions, nous indiquerons deux directions naturelles d’extension de la sémantique des jeux : à la théorie de la concurrence, et à la théorie de la quantification et des types dépendants.

### 1.1.1 Innocence et jeux asynchrones

Les stratégies innocentes sont des stratégies à information partielle, qui jouent en fonction de leur “vue” courante (et partielle) de la “trajectoire” d’interaction que constituent le Joueur et son Opposant depuis le début de la partie. Du point de vue syntaxique, la trajectoire décrit une exploration possible de l’arbre de Böhm (ou  $\lambda$ -terme) qu’interprète la stratégie innocente, tandis que la vue courante dont dispose la stratégie sur l’exploration transpose dans l’univers sémantique la notion syntaxique de branche de l’arbre de Böhm.

Une des questions les plus délicates de la sémantique des jeux est de comprendre comment deux stratégies à information partielle, autrement dit “amnésiques”, peuvent néanmoins interagir de manière cohérente, et produire par composition (dans la catégorie des stratégies innocentes) une nouvelle stratégie innocente, tout autant amnésique. La difficulté, en effet, est que Joueur et Opposant ont des vues hétérogènes, et apparemment incohérentes, de la trajectoire interactive en cours. Pour éclaircir cette question, Melliès a introduit la notion de *jeu asynchrone* où Joueur et Opposant s’affrontent sur des trajectoires dans un espace cubique  $n$ -dimensionnel. Dans ce cadre inspiré des travaux de Mazurkiewicz en théorie de la causalité concurrente (true concurrency) et de ses travaux en théorie de la réécriture, Melliès donne une caractérisation purement diagrammatique de l’innocence – qui s’assimile à une version interactive de confluence locale (Church-Rosser) en théorie de la réécriture.

De cette caractérisation découle le théorème fondamental, que les stratégies innocentes sont positionnelles [Mel06a]. Cela explique, en particulier, pourquoi les stratégies innocentes sont amnésiques et se composent néanmoins : de la trajectoire d’interaction, elles ne retiennent que la position courante, ou mieux : la classe d’*homotopie* par permutation de cellules 2-dimensionnelles. Du point de vue technique, la positionnalité assure encore qu’il existe un *foncteur logique* de la catégorie des jeux asynchrones et stratégies innocentes vers la catégorie des ensembles et relations, qui transporte toute stratégie vers l’ensemble des positions qu’elle atteint dans le jeu. Ici, on appelle foncteur logique un foncteur qui préserve tous les connecteurs de la logique linéaire : tenseur, produit, et modalité exponentielle. Ce résultat offre un point de synthèse entre sémantique des jeux et modèle relationnel de la logique linéaire – poursuivant le travail entrepris en collaboration avec Samson Abramsky sur les jeux concurrents [AM99], et réalisant en quelque sorte le projet des “timeless games” défini par Baillot, Danos, Ehrhard et

Regnier [BDER97] à la fin des années 1990. Cette approche positionnelle permet de surcroît à Melliès [Mel04, Mel05] de définir le premier modèle de jeux séquentiel de la logique linéaire toute entière (et non pas des fragments) grâce à l’identification dans le modèle des stratégies égales du point de vue de leur états terminaux (ou positions d’arrêt) dans l’espace défini par le jeu asynchrone.

Plus récemment, le modèle de jeux asynchrones a été étendu par Melliès et Mimram [MM07] à des jeux où l’interaction entre Joueur et Opposant n’est plus nécessairement alternée. Cette extension nécessite d’utiliser des notions avancées de la théorie de Mazurkiewicz – tel l’axiome du cube, bien connu en théorie de la concurrence, qui relie sémantique des traces et causalité asynchrone. Ce travail permet de reconstruire le modèle des jeux concurrents de Abramsky et Melliès [AM99] où les stratégies sont définies par des opérateurs de clôture, à partir d’un modèle où les stratégies sont définies par des ensembles de trajectoires séquentielles et non alternées, satisfaisant des propriétés de causalité très naturelles. Un résultat de complétude dénotationnelle est obtenu pour une variante de la logique linéaire multiplicative, en remplaçant la condition de bascule (switching conditions) ordinaire des jeux alternés [AJ94] par une condition d’ordonnancement (scheduling conditions) adaptée aux interactions non alternées. Cette nouvelle condition impose, par exemple, que tout partie jouée par une stratégie de formule  $A \otimes B$ , peut être jouée d’abord sur le jeu  $A$  puis sur le jeu  $B$  – ou d’abord sur le jeu  $B$  puis sur le jeu  $A$ .

### 1.1.2 Innocence et stratégies cellulaires

La présentation usuelle de la composition des stratégies innocentes (en termes de “parallel composition plus hiding”) dissimule une grande complexité liée à la gestion de l’information disponible à chaque pas de l’interaction entre le Joueur et son Opposant. Afin d’analyser cette situation, Harmer a introduit une nouvelle classe de stratégies : les stratégies cellulaires, qui rendent explicite cette gestion de l’information. Cette définition est inspirée par (et généralise) les termes cellulaires de Padovani ; en effet, un terme cellulaire est une stratégie à la fois cellulaire *et* innocente. Il a défini deux machines abstraites : la première (la DPAM) agit directement sur les stratégies innocentes et doit maintenir une structure de données compliquée pour gérer l’interaction ; la seconde (la CPAM) agit sur les stratégies cellulaires et n’a donc besoin d’aucune structure de données compliquée, mais est associée à une procédure de “cellularisation” tout-à-fait intéressante, qui transforme une stratégie innocente en stratégie cellulaire. De fait, la procédure de cellularisation explicite la gestion de l’information en l’intégrant aux stratégies et permet donc à la CPAM de simuler la DPAM [Har06]. Par la suite, Harmer a analysé les phénomènes de répétition dans l’interaction entre deux stratégies innocentes, ce qui l’a amené à une définition de stratégies affines, dont il a montré qu’elles forment une catégorie symétrique monoïdale fermée.

Dans cette direction, et sous la direction de Harmer, Clairambault a introduit un analogue des termes cellulaires de Padovani, dans le cadre de la sémantique des jeux. Ces termes sont au cœur de la question de la décidabilité de l’équivalence observationnelle de PCF et de ses restrictions. Rappelons en effet que l’égalité extensionnelle de deux programmes PCF n’est pas décidable – tandis qu’elle l’est dans la version “unaire” de PCF. L’approche de Clairambault lui a permis de refaire sémantiquement la preuve très difficile de décidabilité de Loader pour la version de PCF avec des booléens unaires.

### 1.1.3 Innocence et algèbre

Il existe aussi une approche purement algébrique de l’innocence, élaborée par Harmer, Hyland et Melliès [HHM07, Mel], que nous exposerons dans la troisième partie de ce thème.

### 1.1.4 Bon parenthésage et contrôle

Comme il a été dit plus haut, la catégorie des jeux et des stratégies innocentes et bien parenthésées fournit un modèle complet de PCF. Relâcher la condition de bon parenthésage sur les stratégies revient, du point de vue syntaxique, à étendre le langage PCF avec l'opérateur de contrôle **call-cc**. Poursuivant les travaux de Danos et Harmer [DH01] qui analysaient cette situation catégorique à travers deux contraintes : le *bon parenthésage* et la *rigidité* Harmer et Laurent [HL06] ont introduit une nouvelle contrainte : la *rigidité arrière* (la rigidité étant rebaptisée “rigidité avant”). Il est possible de caractériser en termes d'arbres de Böhm les propriétés syntaxiques correspondant à ces contraintes. Ainsi, **call-cc** viole le bon parenthésage, **if** viole la rigidité avant et les constantes comme le booléen **true** violent la rigidité arrière. De là découle une série de factorisations dans la catégorie des stratégies innocentes : toute stratégie  $\sigma$  qui viole la condition  $P$  peut s'écrire comme la composée de  $\sigma'$  et  $\tau_P$  où  $\sigma'$  satisfait  $P$  et  $\tau_P$  est une stratégie fixée qui viole  $P$  (les trois stratégies  $\tau_P$  sont des variantes de **call-cc**, **if** et **true**). De plus, si  $\sigma$  satisfait une des deux autres contraintes  $Q$ , alors c'est également le cas pour  $\sigma'$  – ce qui montre l'indépendance des conditions et définit donc sept sous-catégories de la catégorie innocente. Ces sous-catégories forment un cube pour l'ordre d'inclusion, et correspondent chacune à un sous-langage de PCF avec contrôle.

Si l'on restreint la catégorie innocente au cas d'un unique type de base ayant une unique valeur, on obtient des modèles du  $\lambda$ -calcul et du  $\lambda\mu$ -calcul simplement typés, avec un seul atome. Les stratégies innocentes, rigides avant et rigides arrière forment un modèle complet du  $\lambda\mu$ -calcul, celles qui sont de plus parenthésées forment un modèle complet du  $\lambda$ -calcul. L'unique atome est alors interprété par un jeu à deux coups : une question d'Opposant et une réponse de Joueur. Il est également possible de construire un modèle du  $\lambda$ -calcul à partir d'un jeu à un seul coup. Il n'y a plus alors de questions et de réponses et les stratégies innocentes (sans condition additionnelle) fournissent directement un modèle complet [DHR96].

### 1.1.5 Bon parenthésage et typage des ressources

L'illusion est répandue : le bon parenthésage serait une notion particulière à la sémantique des jeux, fondamentalement hétérogène, et irréductible aux concepts des autres champs de la sémantique des langages de programmation. En reposant la question des rapports entre sémantique des jeux et logique linéaire, Melliès et Tabareau ont montré que le bon parenthésage peut s'interpréter, en fait, comme une condition de *linéarité* sur les formules de jeux. Ainsi, le typage du contrôle : peut-on appliquer une opération de **call-cc** à cet endroit du programme ? est de ce fait ramené au typage des ressources : peut-on appliquer un glaneur de cellule à cette partie précise de la pile ? Cela rattache un concept clef, et jusque là énigmatique, de la sémantique des jeux, à une notion fondamentale au cœur de logique linéaire.

Ainsi, il est possible de voir le modèle  $\mathcal{M}_1$  des stratégies innocentes et bien parenthésées (caractérisant PCF) et le modèle  $\mathcal{M}_2$  des stratégies innocentes sans condition de parenthésage (caractérisant PCF avec opérateur **call-cc**) au sein d'un modèle de jeux  $\mathcal{M}$  plus général. Chacun des ces sous-modèles  $\mathcal{M}_1$  et  $\mathcal{M}_2$  est engendré par un choix différent de codage du type booléen :

$$\mathbb{B}_1 := \frac{O}{\neg} \frac{P}{\neg} (1 \oplus 1) \qquad \mathbb{B}_2 := \frac{O}{\neg!} \frac{P}{\neg} (1 \oplus 1).$$

Ce codage des booléens indique bien la dynamique de l'interaction : tout d'abord, Opposant interroge Joueur par une question, signalée dans le type par la négation  $\neg$  surmontée d'une étiquette  $O$  ; ensuite, Joueur donne sa réponse, signalée dans le type par la négation  $\neg$  étiquetée  $P$  ; enfin, Joueur choisit entre la valeur booléenne Vraie (unité 1 de gauche dans  $1 \oplus 1$ ) ou Faux (unité 1 de droite). Dès lors, il apparaît que la seule différence entre les deux variantes

de PCF est la modalité de ressource ! introduite entre les deux négations de  $\mathbb{B}_1$  pour définir la formule  $\mathbb{B}_2$ . Cette modalisation de la sous-formule  $\neg(1 \oplus 1)$  donne la possibilité au Joueur de ne pas répondre à la question posée par l’Opposant – ou d’y répondre plusieurs fois. On peut bien entendu hybrider  $\mathcal{M}_1$  et  $\mathcal{M}_2$  au sein du modèle  $\mathcal{M}$  et ainsi définir un système de types imposant une politique de contrôle – à savoir, si un opérateur de contrôle peut être appliqué (ou pas) dans certaines positions du programme, selon son type.

Le modèle de jeu lui-même est construit à partir d’une notion généralisée de parenthésage, appelée *multi-parenthésage*, fondé sur l’intuition que chaque coup du jeu peut ouvrir plusieurs canaux (parenthèses ouvrantes) que d’autres coups devront ensuite fermer (parenthèses fermantes). Cette extension permet, par exemple, d’imposer à tout programme de type  $(\mathbb{B} \otimes \mathbb{B}) \multimap \mathbb{B}$  d’interroger ses *deux* arguments booléens, avant de donner son résultat. Ce travail permet aussi de dégager une logique tensorielle, variante “dynamique” de la logique linéaire.

### 1.1.6 L-nets, jeux, structures d’évènements et concurrence

Faggian, en collaboration avec Maurel [FM05] puis avec Curien [CF05, CF06], a proposé un modèle de l’interaction à base de jeux, qui repose sur une représentation des stratégies de la ludique au moyen de graphes, appelés *L-nets*. Cette représentation parallèle des stratégies intègre des idées venant des réseaux de preuves de la logique linéaire – alors que la ludique telle que l’a définie Girard est fondée sur un calcul des séquents non typé abstrait : les *desseins*. Cette théorie présente également des liens avec les idées de Winskel sur la représentation de la concurrence dans les *structures d’évènements*, ainsi qu’avec les travaux d’Abramsky et Melliès sur les *jeux concurrents* et les *jeux asynchrones*. Par cette approche, la représentation séquentielle des stratégies de la ludique (au sens de Girard) apparaît comme un cas particulier dans la théorie de L-nets, qui proposent tout un spectre de degrés de parallélisme, allant du calcul des séquents aux réseaux de preuve. Cette idée de hiérarchie de parallélisme a été transférée à la théorie des réseaux de preuves par Faggian et Paolo Di Giamberardino [FDG06].

Plus récemment, Faggian et Piccolo [FP07a, FP07b] ont établi un lien entre cette approche à base de ludique parallèle et de structures d’évènements d’une part, et une version linéaire du  $\pi$ -calcul de l’autre. Ce travail s’inspire en partie du travail de Varacca et Nobuko Yoshida [VY06] sur le  $\pi$ -calcul linéairement typé. Les relations formelles entre ces travaux font l’objet de recherche active. En particulier, Faggian, Piccolo et Varacca cherchent à comprendre le côté “ludique” du  $\pi$ -calcul non typé. Dans cette approche, le caractère localisé de la ludique (les desseins sont des “preuves” mettant en jeu des *adresses* et non des occurrences de formules) est essentiel, car les adresses correspondent au noms de canaux sur lesquels se produisent les communications du  $\pi$ -calcul.

Dans une toute autre direction, Varacca a travaillé (avec Hagen Völzer et Ekkart Kindler) sur une définition abstraite de la notion de justice (fairness) [VVK05]. Un point tout-à-fait remarquable est que cette définition de justice est basée sur une notion de jeu de Banach et Mazur. Nous envisageons d’appliquer cette connection avec les jeux, afin de définir une notion de justice dans les modèles causaux, au moyen d’une notion adaptée de stratégie dans les jeux concurrents.

### 1.1.7 Quantification et types dépendants

Un des enjeux de la recherche en sémantique des jeux est de comprendre aujourd’hui comment intégrer la quantification, et plus généralement les types dépendants, dans ce cadre sémantique. Ainsi, le polymorphisme (ou quantification du second ordre) joue un rôle clef dans un langage de programmation comme CAML, où il offre la possibilité d’appliquer un même programme (typiquement, une fonctionnelle) à des programmes de types différents. Sous la direction de

Laurent, de Lataillade a étudié le polymorphisme à travers la sémantique des jeux, aboutissant à un modèle dans lequel les calculs sont plus simples à effectuer que dans les modèles existants [Hug00, AJ03, MO01]. Dans [DL07b], de Lataillade a utilisé ce modèle pour retrouver le résultat de Di Cosmo [DC95] sur les isomorphismes de types du système F à la Church. Dans un autre paradigme de calcul polymorphe, le système F à la Curry, les termes ne contiennent pas explicitement d’indication de type. L’égalité dans ce langage est donc plus riche que celle du système F à la Church. De Lataillade [DL07a] a donné une caractérisation (via la sémantique) des isomorphismes de types dans ce système : en effet, grâce à la sémantique des jeux, ces isomorphismes peuvent aussi être vus comme des invariants géométriques, synthétisés en enrichissant les équations de Di Cosmo d’une nouvelle égalité :

$$\forall X. A \simeq A[\forall Y. Y/X] \quad \text{si } X \notin \text{Neg}_A$$

En ce qui concerne l’articulation entre sémantique des jeux et quantification du premier ordre, il est naturel de partir du travail de Krivine sur la réalisabilité classique, et en particulier, une interprétation à base de jeux que Krivine en a donné. Procédant de la sorte, Laurent a étendu son modèle “à un seul coup” décrit plus haut, avec de nouveaux pointeurs, appelés  $\mu$ -pointeurs – qui s’avèrent être un cas particulier des pointeurs introduits par Laird [Lai01] pour représenter les exceptions locales. Laurent construit de la sorte un modèle complet du  $\lambda\mu$ -calcul. Il établit, de plus, que ces  $\mu$ -pointeurs codent exactement (à travers une fonction de pliage) les réponses du modèle “à deux coups”, ces  $\mu$ -pointeurs étant triviaux (toujours pointés vers le coup précédent) dans le cas de stratégies bien parenthésées (les stratégies du  $\lambda$ -calcul).

Du point de vue logique, les  $\mu$ -pointeurs représentent les axiomes. La porte est alors ouverte à l’extension au cas de plusieurs atomes (et de façon beaucoup plus satisfaisante que dans le modèle “à deux coups” [Lau05b]) et à la quantification du premier ordre. De fait, Laurent construit un modèle complet pour la logique classique du premier ordre en montrant qu’une stratégie correspond à un arbre de Böhm pour une extension à la Church du  $\lambda\mu$ -calcul. Une application de ce modèle est la caractérisation des isomorphismes de types pour la logique classique du premier ordre. Reste à traiter le symbole d’égalité dans cet esprit.

Plus généralement, il nous semble possible désormais d’étudier les phénomènes subtils liés aux types dépendants – cette notion générale de quantification en théorie des types – à partir des outils de la sémantique des jeux, nourrie des principes mis en place par Krivine en réalisabilité classique. Sous la direction de Harmer, Clairambault travaille actuellement dans cette direction.

## 1.2 Modèles du $\lambda$ -calcul pur et de PCF

Telle que développée par Scott, la sémantique dénotationnelle fournit des modèles pour le  $\lambda$ -calcul pur. Traditionnellement, ces modèles sont décrits par des ordres partiels complets (domaines), mais très souvent, ces domaines admettent une représentation plus simple et combinatoire (modèles de graphes).

### 1.2.1 Théories des modèles du $\lambda$ -calcul pur

Chaque modèle induit sur les  $\lambda$ -termes une théorie équationnelle, et l’étude et la classification de ces théories est un sujet très riche et difficile sur lequel Berline, Bucciarelli et Manzonetto travaillent, en collaboration avec Antonino Salibra de l’université de Venise. Ces théories forment un treillis complet dont ils ont étudié les propriétés. Berline et Salibra ont généralisé dans [BS06] la preuve sémantique classique de “facilité” du terme  $\delta\delta$ , montrant via les modèles de graphes l’existence des suites infinies de termes qui sont, indépendamment, “extrêmement faciles”. Ils utilisent ensuite ceci pour montrer que le treillis des  $\lambda$ -théories contient un “très gros” sous-treillis distributif.



Dans [Ber06], Berline fait un état des lieux des résultats et des questions ouvertes portant sur les théories de diverses classes naturelles de modèles, centrant son attention sur la classe des modèles de graphes, et propose une liste de nouvelles questions. Elle montre également comment les techniques de Berline et Salibra [BS06] peuvent s'appliquer pour montrer que les classes des théories représentées dans chaque classe de modèles connue sont aussi différentes que possible : si  $C$  et  $C'$  sont deux telles classes, alors soit  $C$  est incluse dans  $C'$  soit  $C - C'$  a la cardinalité du continu.

Berline, Manzonetto et Salibra ont progressé vers la résolution d'un des principaux problèmes du domaine : existe-t-il un modèle, qui ne soit pas un "modèle syntaxique", et dont la théorie est  $\beta$  (la  $\beta$ -équivalence) ou  $\beta\eta$  (la  $\beta\eta$ -équivalence) ? Ils ont conjecturé que la théorie d'un modèle qui "vit" dans la sémantique de Scott ou l'un de ses raffinements ne pouvait pas être récursivement énumérable (r.e.) et ont prouvé diverses instances de cette conjecture dans [BMS07], ainsi que de son extension aux théories inéquationnelles (qui semble plus abordable). Pour des raisons méthodologiques précises, ils se sont concentrés sur deux classes : celle des modèles *effectifs* et celle des modèles de graphes. La notion de modèle effectif qu'ils ont précisée est assez générale pour inclure tous les modèles introduits "individuellement" dans la littérature, et juste assez forte pour que l'interprétation d'un terme normal soit "décidable". En ce qui concerne les modèles effectifs, Berline, Manzonetto et Salibra ont montré que leur théorie ne pouvait être ni  $\beta$  ni  $\beta\eta$ , que leur théorie inéquationnelle n'était jamais r.e., et que si ils vivaient en sémantique stable ou fortement stable leur théorie (équationnelle) ne l'était pas non plus. En ce qui concerne la sémantique continue, ils ont montré que la théorie inéquationnelle d'un modèle de graphes n'est jamais r.e., et que si sa trame était "essentiellement finiment présentable" sa théorie (équationnelle) n'était pas r.e. non plus. Enfin, dans le même article, ils ont prouvé une sorte de théorème de Löwenheim-Skolem pour les modèles de graphes, résultat qu'ils ont généralisé depuis à toutes les classes de modèles de trames et qui permet, en ce qui concerne l'étude des théories, de se restreindre aux modèles à trame dénombrable.

On notera que Berardi et Berline avaient montré que, pour le lambda-calcul typé par le Système F (qui peut être vu comme un fragment riche du  $\lambda$ -calcul pur), il existe un modèle non syntaxique (en fait, il en existe beaucoup), dont la théorie est exactement la  $\beta\eta$ -équivalence [BB02]. Ce modèle, qui vit dans la sémantique continue, qui est un modèle de trame, qui n'admet d'analogue ni dans la sémantique stable ni dans la sémantique fortement stable, s'avère être effectif. Ce qui justifie entre autres la méthodologie ci-dessus.

Bucciarelli et Salibra ont montré dans [BS03] que la classe des théories de modèles de graphes admet un plus petit élément, en montrant qu'il existait des modèles dont la théorie est l'intersection de toutes les théories de la classe. Dans [BS04], ils ont poursuivi l'étude des  $\lambda$ -théories issues des modèles de graphes, et montré en particulier que  $\beta$  n'est pas une théorie de graphes, et que parmi les théories de graphes sensibles, celle des arbres de Böhm est maximale. Ils ont résumé et complété ces deux travaux et montré quelques propriétés additionnelles des théories de graphes dans [BS07]. Le problème de caractériser la plus petite théorie de graphes sensible reste ouvert.

### 1.2.2 Modèles non extensionnels du $\lambda$ -calcul pur

Bucciarelli, Ehrhard et Manzonetto ont étudié un modèle du  $\lambda$ -calcul pur dans une catégorie cartésienne fermée qui, pour être inhabituelle du point de vue de la théorie des domaines (le point de vue standard quand on étudie la sémantique du  $\lambda$ -calcul), n'en est pas moins complètement canonique du point de vue de la logique linéaire. Il s'agit de la catégorie dont les objets sont les ensembles, et où un morphisme d'un ensemble  $E$  vers un ensemble  $F$  est une relation de l'ensemble des multiensembles finis d'éléments de  $E$  vers  $F$  (cette catégorie est déjà présente dans

la *sémantique quantitative* de Girard, qui remonte au début des années 1980). Cette catégorie n’ayant pas assez de points, il n’était pas complètement clair qu’un modèle catégorique du  $\lambda$ -calcul pur puisse y être décrit avec les outils algébriques usuels (algèbres combinatoires), ce que Bucciarelli, Ehrhard et Manzonetto ont établi en toute généralité. Ils ont ensuite exhibé un objet réflexif *extensionnel* (c’est-à-dire satisfaisant la règle  $\eta$ ) dans cette catégorie, d’une simplicité extrême : c’est l’ensemble  $D$  dont les éléments sont les suites infinies de multiensembles d’éléments de  $D$ , presque tous vides (définition récursive). Un élément de  $D$  est un arbre fini, qui alterne des couches commutatives (où les branchements ne sont pas ordonnés) et des couches où les branchements sont indexés et donc ne peuvent être échangés.

Cet objet permet aussi d’interpréter le  $\lambda\mu$ -calcul pur, le  $\lambda\mu$ -calcul avec opération MIX considéré il y a quelques années par Danos et Krivine, les réseaux d’interaction différentiels purs (et donc, par transitivité, un bon morceau du  $\pi$ -calcul), etc. Ces premiers résultats sont présentés dans [BEM07].

On peut voir cet objet comme un analogue relationnel du  $D_\infty$  de Scott, et d’ailleurs, Bucciarelli, Ehrhard et Manzonetto ont montré qu’il a la même théorie équationnelle que  $D_\infty$ , à savoir  $\mathcal{H}^*$ . Ils ont établi que ce modèle est sensible pour une extension doublement parallèle du  $\lambda$ -calcul pur. Ces derniers résultats doivent encore être publiés.

### 1.2.3 Modèles de PCF

Dans le domaine de l’étude des modèles de PCF, Bucciarelli et Leperchey ont donné une caractérisation complète de la PCF-définissabilité relative des fonctions booléennes “sous-séquentielles”, [BL04]. Une nouvelle notion de morphisme d’hypergraphes (*timed morphism*) est à la base de la preuve de complétude. Bucciarelli, Leperchey et Padovani ont étudié dans [BLP03] le problème de la définissabilité relative pour la variante “unaire” de PCF, et ils prouvent que l’ordre partiel des degrés de définissabilité est non trivial. La décidabilité de cet ordre partiel reste un problème ouvert. Dans le même esprit, Bucciarelli et Tranquilli, en collaboration avec Lorenzo Tortora de l’Université Roma 3, étudient définissabilité et définissabilité relative, dans les espaces cohérents et hypercohérents, par les structures de preuves de MALL, la logique linéaire additive et multiplicative.

## 1.3 Logique linéaire et concurrence

Inventée par Girard au milieu des années 1980, la logique linéaire a révolutionné la théorie de la démonstration et l’informatique théorique. Il est utile de rappeler qu’elle a été découverte à l’occasion d’investigations sur la sémantique dénotationnelle du système  $F$ , un  $\lambda$ -calcul typé du second ordre mis au point par Girard 15 ans plus tôt (il s’agit plus précisément d’une formalisation du calcul propositionnel intuitionniste du second ordre, dans lequel une très vaste classe de fonctions calculables est représentable). A force de purifier et de simplifier son modèle, Girard était arrivé aux *espaces cohérents*, qui ont une symétrie nouvelle – par rapport à tous les domaines utilisés en sémantique jusque là – autorisant, dans ce cadre constructif, une négation involutive, plus proche de la dualité algébrique que de la négation classique. Transformant ces considérations sémantiques en syntaxe, Girard introduisit alors deux systèmes de déduction pour une nouvelle logique, dont les connecteurs correspondent aux constructions d’espaces disponibles dans cette sémantique, qui sont proches de celles de l’algèbre linéaire (mais moins dégénérées qu’elles, au sens par exemple où le produit direct n’est pas identique à son opération duale, la somme directe, et de même pour le produit tensoriel, même en dimension finie).

Ces systèmes logiques sont un calcul des séquents d’une part, et un analogue de la déduction naturelle, à savoir les *réseaux de preuves*, d’autre part. A cause de la dualité parfaite de la logique linéaire, les réseaux n’ont pas la structure arborescente des preuves de la déduction naturelle (qui,

rappelons-le, sont des  $\lambda$ -termes). Pour cette raison, vérifier qu'un réseau correspond vraiment à une preuve en calcul des séquents – on dit alors qu'il est séquentialisable, en général, il y a de nombreuses preuves en calcul des séquents qui correspondent à un même réseau séquentialisable – est une opération non triviale : il faut vérifier qu'un *critère de correction* de nature géométrique et globale, est satisfait, les plus connus étant ceux de Girard et de Danos-Regnier.

La logique linéaire introduit deux nouveaux connecteurs, les exponentielles, auxquels sont associés des règles logiques correspondant aux règles structurelles de la logique intuitionniste et de la logique classique (contraction et affaiblissement) ainsi que des règles propres à ces connecteurs, déréluction et promotion, cette dernière règle seule introduisant l'infini en logique (possibilité de dupliquer *ad libitum*, grâce aux contractions sur les formules promues).

Ce raffinement de la logique a eu de nombreuses conséquences. En jouant sur les règles associées aux exponentielles, la logique linéaire a permis d'introduire des systèmes logiques dont l'élimination des coupures a une complexité réduite : c'est le cas de ELL (réduction en temps élémentaire) ou SLL et LLL (temps polynomial, [Gir98, Laf04]). Elle a aussi permis de mieux comprendre le contenu calculatoire des preuves de la logique classique, en mettant en relief la *polarisation* des formules. C'est une façon canonique de lever l'ambiguïté de certaines paires critiques apparaissant dans l'élimination des coupures de LK, qui éclaire d'un jour nouveau les travaux précurseurs de Griffin [Gri90] sur le lien entre la construction call/cc du langage Scheme et la logique classique qui menèrent, parallèlement, au  $\lambda\mu$ -calcul de Parigot. La polarisation des formules en logique linéaire est également liée à la propriété de *focalisation* découverte par Andreoli [And92], qui a mené Girard à la ludique [Gir01]. De fait, la polarisation offre un point de vue logique sur la notion de *continuation* dans les langages de programmation, et joue donc un rôle essentiel en sémantique des jeux.

### 1.3.1 Systèmes à complexité bornée

Peu de travaux sur ELL, LLL et SLL portent sur l'étude de leur sémantique dénotationnelle [Bai04]. Laurent et Lorenzo Tortora de Falco [LTDF06] ont défini une notion de *clique obsessionnelle* qui permet de définir des modèles de ELL et SLL à partir de la sémantique relationnelle (ou cohérente) de LL. Ces modèles ne sont pas complets, mais travailler à l'intérieur d'un modèle de LL permet de poser la question de la *complétude relative* (satisfaite par ces deux modèles) : le modèle  $\mathcal{M}_0 \subseteq \mathcal{M}$  est relativement complet pour le système  $\mathcal{S}_0 \subseteq \mathcal{S}$  ( $\mathcal{M}$  étant un modèle de  $\mathcal{S}$  et  $\mathcal{M}_0$  un modèle de  $\mathcal{S}_0$ ) si toute preuve de  $\mathcal{S}$  dont l'interprétation est dans  $\mathcal{M}_0$  est une preuve de  $\mathcal{S}_0$ . L'application de ces techniques à LLL n'a pas encore pu aboutir.

Partant de la question de l'interprétation des connecteurs additifs pour construire des modèles du système ELL (notamment à partir de modèles de LL), Laurent a défini deux notions de modèles catégoriques de ELL (que l'on peut également décliner pour LLL) : catégories de Seely élémentaires (à la Seely) et catégories linéaires non linéaires élémentaires (à la Benton). L'interprétation des connecteurs additifs dans les catégories de Seely élémentaires est légèrement différente de ce que l'on rencontre habituellement en logique catégorique. Alors que le schéma traditionnel est donné par : formule  $\mapsto$  objet, preuve  $\mapsto$  morphisme, connecteur  $\mapsto$  foncteur, et règle  $\mapsto$  transformation naturelle, le connecteur ! est ici décomposé à l'aide d'un "pré-connecteur" noté  $\sharp$  dont l'interprétation dépend de la preuve en cours d'interprétation (une fois la preuve complètement interprétée,  $\sharp$  devient ! et on retrouve le cadre usuel). Ainsi  $\sharp A$  est donné par une combinaison arbitraire, et dépendante de la preuve, d'opérations  $\otimes$  et  $\&$  appliquées à  $A$ .

Dal Lago collabore actuellement avec Laurent sur une caractérisation en théorie des jeux de la complexité calculatoire, dans le contexte de la logique linéaire multiplicative et exponentielle. Plus spécifiquement, ils essayent d'enrichir et modifier la construction habituelle de jeux AJM sur les multiplicatifs et les exponentielles afin de prouver une correspondance forte entre certains

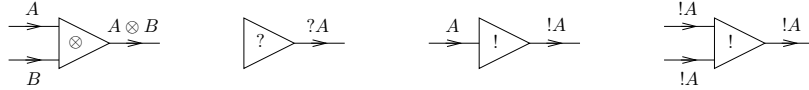
attributs de la sémantique d'une preuve et le temps nécessaire à la normaliser.

Dal Lago a collaboré avec Patrick Baillot et Paolo Coppola, démontrant que les termes qui sont typables en logique légère affine peuvent être réduits au moyen de l'algorithme abstrait de Lamping en temps polynomial [BCDL07]. La même propriété est vraie pour la logique élémentaire affine et le temps élémentaire. La preuve est basée sur la sémantique des contextes, un modèle de la géométrie de l'interaction qui s'est récemment avéré utile pour l'analyse des propriétés quantitatives des programmes et des preuves.

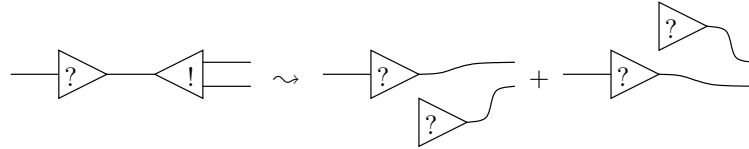
### 1.3.2 Logique linéaire différentielle et processus concurrents

Divers chercheurs ont proposé des formalisations de la concurrence au moyen de structures graphiques. On pense bien sûr aux *bigraphes* de Milner, mais aussi aux *réseaux concurrents* de Beffara et Maurel [BM06] (voir aussi le thème 3) ou aux *réseaux d'interaction multiports* de Mazza (de tels réseaux avaient déjà été considérés par Alexeiev). Mentionnons enfin les *diagrammes de solos* de Laneve, Parrow et Victor. Ces formalismes n'ont toutefois que des liens assez lâches avec la logique et ne proposent pas d'avancée réelle dans la direction d'une interprétation Curry-Howard de la concurrence.

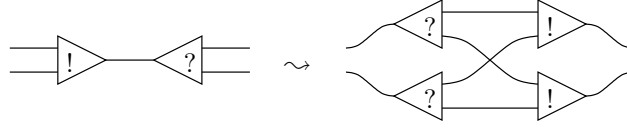
Partant de travaux antérieurs de Ehrhard et Laurent Regnier sur le  $\lambda$ -calcul et la logique linéaire différentiels, ainsi que de travaux de Laurent et Kohei Honda sur la traduction du  $\pi$ -calcul dans les réseaux de la logique linéaire, Ehrhard et Laurent ont proposé un codage d'un fragment "finitaire" du  $\pi$ -calcul dans les réseaux d'interaction différentiels [ER06]. Cette traduction exploite la nouvelle symétrie entre les deux connecteurs exponentiels de la logique linéaire qu'introduit la logique linéaire différentielle. Le formalisme des réseaux d'interaction, introduit par Lafont à la fin des années 1980, est particulièrement adapté à la présentation de cette "logique différentielle". Par exemple, voici les cellules correspondant aux règles tenseur, affaiblissement, codéréliction et cocontraction (pour chaque cellule, le port principal est le seul qui se trouve sur l'un des sommets du triangle).



Les fils *orientés* peuvent être typés par des formules de la logique linéaire comme on l'a fait ici (quand on change l'orientation d'un fil typé par  $A$ , son type devient  $A^\perp$ , la négation linéaire de  $A$ ; rappelons que cette négation est involutive). Calculer sur ces réseaux consiste à évaluer des *redex*, qui sont des structures consistant en deux cellules connectées par leurs ports principaux. Voici par exemple la réduction d'une déréluction contre une cocontraction

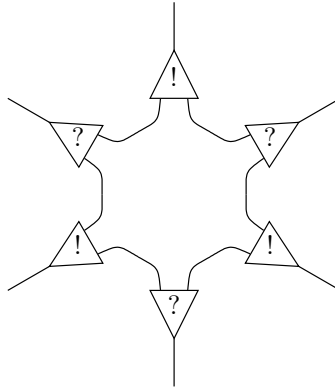


et on a une réduction parfaitement duale pour l'interaction d'une codéréluction avec une contraction. On voit apparaître une somme, qui a un sens algébrique clair dans la sémantique dénotationnelle, et qu'on peut interpréter ici comme du non déterminisme : la déréluction est "routée" soit à droite, soit à gauche, dans la cocontraction, et l'autre port de la cocontraction ne reçoit rien (affaiblissement). Les *redex structurels* correspondent à la configuration cocontraction/contraction et cocontraction/affaiblissement (et sa duale); ce sont des règles de réduction correspondant à l'axiomatique usuelle des bialgèbres, par exemple :

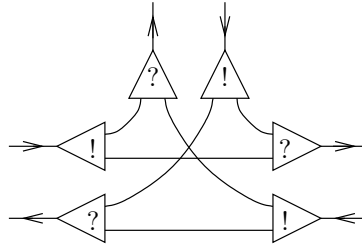


La traduction du  $\pi$ -calcul dans ces réseaux en exploite une version non typée (ou plutôt, typée au moyen de deux types,  $\iota$  et  $o$  vérifiant  $o = ?(o^\perp) \wp o$  ou, ce qui revient au même, en posant  $\iota = o^\perp$ , l'équation  $\iota = !o \otimes \iota$ . L'idée de base est que, à chaque nom libre d'un processus, deux ports, de types duaux, sont associés.

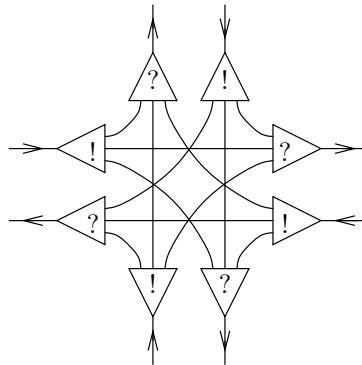
Cette traduction s'articule autour de la notion de zone de communication, une structure complètement symétrique construite à partir de contractions (règle standard de la logique linéaire) et de cocontractions (nouvelle règle introduite par le cadre différentiel). La zone de communication typique est la structure suivante



que l'on peut aussi dessiner ainsi en précisant les types (par l'orientation des fils) :

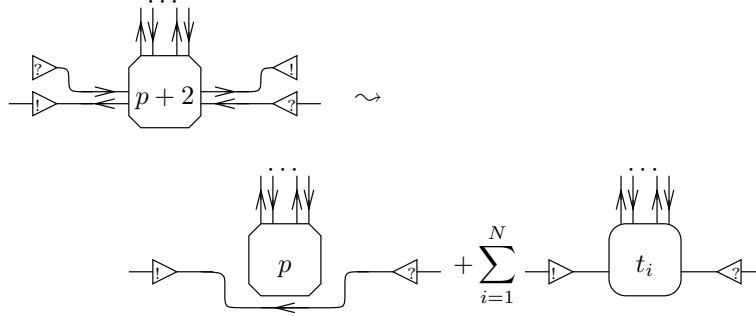


où les paires de ports à connecter à celles des réseaux à mettre en parallèle sont mises en évidence (on voit bien qu'elles portent des types duaux ; un fil orienté porte le type  $?\iota$ , par convention). Ces zones de communication ont de bonnes propriétés, par exemple, quand on connecte deux zones d'arité 3 (comme ci-dessus), on obtient une zone d'arité 4



en quelques étapes de réduction.

Les préfixes d'émission et de réception sont interprétés au moyen de la dérélction et de la codérélction respectivement. Les aspects séquentiels ainsi que la polyadicité du  $\pi$ -calcul sont représentés au moyen des connecteurs multiplicatifs de la logique linéaire. En interagissant avec les zones de communication, les dérélctions (resp. codérélctions) des préfixes d'émission (resp. de réception) sont routées, au cours de la réduction, jusqu'à la rencontre éventuelle avec une codérélction (resp. dérélction) correspondant à un préfixe du genre opposé ; on a en effet une réduction de la forme



Dans cette figure, les carrés biseautés correspondent à des zones de communication dont l'arité (nombre de bifils connectés) est le nombre qu'on y inscrit, plus 2. Après un certain nombre d'étapes de réduction, on voit que, dans le premier terme de la somme, la dérélction et la codérélction se retrouvent en contact direct. Elles se réduisent ensuite, produisant un simple fil reliant leurs entrées respectives. Les autres termes de la somme correspondent à d'autres routages possibles.

Ehrhard et Laurent ont montré que la “fonction” de traduction (qui est en fait une relation) est une bisimulation forte, pour des systèmes de transition associés au  $\pi$ -calcul et aux réseaux différentiels respectivement [EL07]. Dans la mesure où les réseaux d'interaction différentiels sont une extension très naturelle de la logique linéaire, ce travail suggère une véritable interprétation Curry-Howard de la concurrence. Ils ont également considéré le calcul des solos (un calcul concurrent complètement asynchrone dû à Laneve, Parrow et Victor), pour lequel la traduction dans les réseaux d'interaction différentiels est plus simple que dans le cas du  $\pi$ -calcul. Ils ont circonscrit un fragment, stable par réduction, de ce calcul, pour lequel les réseaux différentiels obtenus sont “acycliques” en un certain sens, et qui contient l'image de la traduction classique du  $\pi$ -calcul dans les solos.

Sous la direction de Ehrhard, Gimenez développe une approche des *sharing graphs* de Gonthier, Abadi et Lévy adaptée à l'approche différentielle ainsi qu'une théorie des définitions de “fonctions récursives” dans la logique linéaire. Ces travaux préliminaires trouveront leur application dans les futures interprétations différentielles de versions plus riches du  $\pi$ -calcul, avec réplcation et définitions récursives.

Dans sa thèse, De Carvalho (Institut de Mathématiques de Luminy, ATER à Paris 7 en 2006-2007), sous la direction de Baillot et Ehrhard, a notamment étudié les modèles catégoriques de la logique linéaire différentielle finitaire (sans promotion). Dans ce cas, il n'est pas nécessaire que les exponentielles soient des foncteurs ; il suffit que le “!” associe à chaque objet  $X$  une bigèbre munie d'une dérélction  $!X \rightarrow X$  et d'une codérélction  $X \rightarrow !X$ . Sous certaines hypothèses sur la catégorie ambiante, une version de la formule de Taylor permet, à partir de ce matériel minimal, de faire de ! une comonade ayant les structures requises pour interpréter la logique linéaire. Le modèle obtenu satisfait cependant des conditions plus faibles que celles requises par l'axiomatisation de Bierman [Bie95].

### 1.3.3 Sémantique vectorielle de la logique linéaire

Prolongeant son étude antérieure du modèle des espaces de finitude [Ehr05], qui est un modèle de la logique linéaire dans lequel les formules sont interprétées par des espaces vectoriels topologiques à topologie linéaire (qu'on appellera *espaces de Lefschetz* dans la suite), et les preuves par des vecteurs (ou plus généralement par des fonctions linéaires continues), Ehrhard a prouvé un théorème caractérisant la catégorie cartésienne associée comme une catégorie de préfaisceaux sur la théorie de Lawvere des polynômes. Ce résultat n'est, pour l'instant, obtenu que sous des hypothèses restrictives sur le corps des coefficients. Sous la direction de Curien et de Ehrhard, Tasson développe un modèle de la logique linéaire utilisant des espaces de Lefschetz généraux au lieu des espaces de finitude qui en sont des cas particuliers. Elle a prouvé que tout espace de Lefschetz est linéairement isomorphe à son bidual, mais que cet isomorphisme n'est pas continu en général. Elle développe dans ce cadre un analogue algébrique de la cohérence et de la totalité, notions combinatoires et discrètes mises en œuvre par les espaces cohérents de Girard. Dans ce cadre, la cohérence apparaît comme un sous-espace affine fermé d'un espace vectoriel interprétant le type.

### 1.3.4 Logique tensorielle et continuations linéaires

L'analyse des modalités de ressources en sémantique des jeux a conduit Melliès et Tabareau [MT07] à relâcher la logique linéaire, en une *logique tensorielle* où la négation ( $A \mapsto \neg A$ ) n'est pas nécessairement involutive. Ce relâchement est limpide du point de vue catégorique : il revient à remplacer les catégories  $*$ -autonomes (ou monoïdales fermées) de la logique linéaire par des catégories monoïdales munies d'une notion de négation tensorielle. La définition catégorique de modalité de ressources ! prend alors la forme usuelle d'une "adjonction monoïdale" décrite par Benton pour la logique linéaire.

Au final, on obtient une logique tensorielle qu'il faut comprendre comme une *extension polarisée* de la logique linéaire – à même de refléter dans un cadre logique (calcul des séquents, etc. . . ) le caractère dynamique de la sémantique des jeux. Point important, seule la négation est autorisée à changer la polarité d'une formule en logique tensorielle. Ce changement de polarité découle du fait que la négation signale un "transfert de contrôle" entre Joueur et Opposant, transfert qu'on retrouve interprété comme un coup en sémantique des jeux. Par cette lecture, il apparaît que le principe d'involution  $A \cong \neg \neg A$  de la logique linéaire ne respecte pas la temporalité de la sémantique des jeux. Il est donc naturel de le relâcher.

La logique tensorielle offre aussi un point de vue pertinent sur les logiques polarisées usuelles : systèmes LC de Jean-Yves Girard, ou LLP de Laurent. Contrairement à ces logiques, la modalité exponentielle ! de la logique tensorielle ne change pas la polarité d'une formule (nécessairement positive) à laquelle elle s'applique. De fait, Melliès et Tabareau montrent comment décomposer l'exponentielle de LC ou de LLP en une négation tensorielle (qui change la formule négative en formule positive) suivie de l'exponentielle tensorielle. Ainsi, cela permet de déduire un modèle de LLP à partir de tout modèle de logique tensorielle – par une construction de Kleisli analogue à la décomposition linéaire  $A \Rightarrow B := (!A) \multimap B$  de l'implication intuitioniste, familière en logique linéaire.

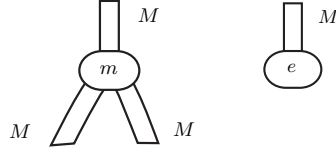
Pour finir, la logique tensorielle offre un point de rencontre fondateur entre théorie de la démonstration (logique linéaire), théorie des continuations linéaires (monades de continuation), et algèbre 2-dimensionnelle (théorie des algèbres relâchées ou "lax" d'une 2-monade). Par exemple, il apparaît que la disjonction multiplicative  $A \wp B$  de la logique linéaire définit, une fois transcrite en logique tensorielle, une loi monoïdale relâchée au sens de Leinster [Lei04] décrite par une famille d'opérations de disjonction  $n$ -aires  $[A_1 \wp \dots \wp A_n] := \neg(\neg A_1 \otimes \dots \otimes \neg A_n)$ . Cette disjonction  $n$ -aire décrit le type d'un programme à  $n$  points de contrôle de types  $A_1, \dots, A_n$  –

chacun de ces points de contrôle étant accessible par une commande `goto`. Le cas particulier d'un seul point de contrôle, décrit par l'opérateur unaire de disjonction  $[A]$  coïncide avec la monade de continuation ( $A \mapsto \neg\neg A$ ) usuelle.

## 1.4 Méthodes algébriques et homotopiques

### 1.4.1 Diagrammes de cordes

Une grande attention a été portée dans le laboratoire aux diagrammes de cordes, un formalisme introduit par Penrose pour décrire des morphismes de catégories monoïdales – dont nous noterons ici le produit tensoriel  $\otimes$  et l'unité  $I$ . Ainsi, un monoïde  $M$  est donné par deux morphismes  $m : M \otimes M \longrightarrow M$  et  $e : I \longrightarrow M$  que l'on représente ainsi dans les diagrammes de corde :



Les lois d'associativité et d'unité sont décrites par les égalités suivantes :

(1)

La notion duale de comonoïde, donnée par des morphismes  $d : C \longrightarrow C \otimes C$  et  $u : C \longrightarrow I$  est représentée en retournant les diagrammes du monoïde.

Ce langage diagrammatique est très proche de celui des réseaux de preuves de Girard. Ainsi, la loi fondamentale des bigèbres est donnée par l'égalité diagrammatique :

où l'on reconnaît l'une des égalités clefs des graphes de partage de Gonthier, Abadi et Lévy. De la même manière, une algèbre de Frobenius est donnée par un monoïde et un comonoïde liés par l'égalité suivante :

(2)

L'égalité diagrammatique (2) des algèbres de Frobenius décrit précisément une propriété de “mal-léabilité” attendue des canaux de communication dans toute algèbre de processus. Melliès [Mel07]



interprète dans cet esprit les zones de communications définies par Ehrhard et Laurent (voir ci-dessus). Toute bigèbre auto-duale définit une algèbre de Frobenius, dont la multiplication est donnée par le diagramme suivant :

(3)

Cette construction est analogue à (mais subtilement différente de) la construction bien connue de l’algèbre de Frobenius des matrices  $Mat(A) = A^* \otimes A$  sur un  $k$ -espace vectoriel  $A$ . L’algèbre de Frobenius définissant les zones de communication est obtenue en appliquant cette construction (3) à la bigèbre auto-duale  $H \otimes H^*$ , où  $H = !A$ . Point important : ce calcul ne s’effectue pas dans la catégorie sémantique  $\mathbb{C}$  elle-même, mais dans une catégorie  $Int(\mathbb{C})$  “entortillée” (torile) au sens de Joyal, Street et Verity [JSV96] dans laquelle la catégorie  $\mathbb{C}$  se plonge par la construction libre  $Int$ , liée à la Géométrie de l’Interaction [AJ92].

#### 1.4.2 Points fixes, traces et jeux de Conway

On aimerait mieux comprendre quand une catégorie cartésienne fermée déduite d’un modèle de logique linéaire dispose d’un opérateur de point fixe – cela afin d’interpréter la récursion. Sous la direction de Melliès, Tabareau a travaillé sur la structure des points fixes en sémantique des jeux [Tab06]. Il est naturel de penser que la notion d’*opérateur de trace* décrit la version linéaire, ou monoïdale, de la notion d’opérateur de point fixe. Or, toute catégorie entortillée dispose d’un opérateur de trace, défini grâce au dual  $U^*$  (à la fois à droite et à gauche) associé à tout objet  $U$ . La catégorie  $\mathbb{G}$  des jeux de Conway, introduite par Joyal [Joy77], est entortillée, en tant que catégorie compacte fermée. Elle dispose donc d’un opérateur de trace. De plus, la catégorie  $\mathbb{G}^-$  des jeux alternés où Opposant commence, se plonge dans la catégorie  $\mathbb{G}$  par un foncteur  $U$  monoïdal (au sens fort) qui dispose, de plus, d’un adjoint à droite. De tout cela découle que la catégorie  $\mathbb{G}^-$  est symétrique monoïdale fermée, et dispose d’un opérateur de trace. De plus, cet opérateur de trace dans  $\mathbb{G}^-$  se transporte dans la catégorie cartésienne fermée  $\mathbb{G}_!^-$  déduite par construction de Kleisli sur la comonade  $!$  des jeux. La propriété de transport est établie par une démonstration diagrammatique, élucidée en [Mel06b]. On obtient de la sorte une notion de point fixe paramétré très robuste dans la catégorie cartésienne fermée  $\mathbb{G}_!^-$  des jeux usuels.

Masahito Hasegawa (RIMS, Kyoto) était invité à PPS à l’époque de ces travaux. Il a tiré de cette construction l’observation suivante, de nature purement algébrique : une catégorie  $\mathbb{C}$  monoïdale tracée est fermée ssi le plongement canonique  $\mathbb{C} \longrightarrow Int(\mathbb{C})$  vers la catégorie entortillée libre  $Int(\mathbb{C})$  dispose d’un adjoint à droite. Cela montre que toute catégorie symétrique monoïdale fermée avec opérateur de trace est la “déformation” par adjonction d’une catégorie compacte fermée (et donc auto-duale).

#### 1.4.3 Innocence, monade et comonade

Harmer, Martin Hyland et Melliès [HHM07] réinterprètent la catégorie  $\mathbb{I}$  des jeux d’arène et des stratégies innocentes de Hyland et Ong comme une catégorie de Kleisli  $\mathbb{G}_\lambda$  construite au dessus d’une catégorie  $\mathbb{G}$  de jeux alternés. et de stratégies à information complète. Cette construction de Kleisli est basée sur une loi de distributivité  $\lambda : ?! \rightarrow !?$  au sens de Beck,

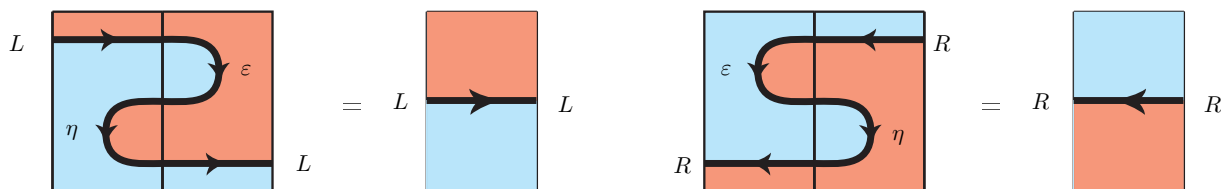
entre une monade  $?$  et une comonade  $!$  sur la catégorie  $\mathbb{G}$ . Une stratégie innocente  $A \rightarrow B$  dans la catégorie  $\mathbb{G}_\lambda$  est donc ici interprétée comme une stratégie à information complète  $!A \rightarrow ?B$  dans la catégorie  $\mathbb{G}$ . Ici,  $!A$  décrit le jeu dont les positions sont les “vues de l’Opposant” du jeu  $A$ , tandis que  $?B$  décrit le jeu dont les positions sont les “vues du Joueur” du jeu  $B$ . Cette reconstruction algébrique de la catégorie des stratégies innocentes capture donc bien l’idée qu’une stratégie innocente  $A \rightarrow B$  joue en fonction de sa vue courante dans le jeu  $A \Rightarrow B$  – un entrelacement d’une vue de l’Opposant dans le jeu  $A$ , et d’une vue du Joueur dans le jeu  $B$ . La loi de distributivité  $\lambda$  capture quant à elle les propriétés fondamentales qui font que deux stratégies innocentes se composent dans la catégorie  $\mathbb{I}$  définie par Hyland et Ong. Le fait que la catégorie  $\mathbb{I} = \mathbb{G}_\lambda$  est cartésienne fermée est aussi expliqué par cette construction.

#### 1.4.4 Syntaxe des jeux et diagrammes de cordes

Notre intérêt pour les diagrammes de corde est aussi motivé par le désir de rapprocher théorie des nœuds et théorie de la démonstration. Ainsi, Mellies donne une présentation algébrique de la logique tensorielle, qui s’appuie sur la théorie formelle des adjonctions  $L \dashv R$  dans le langage des 2-catégories [Str72]. Il est bien connu que les deux générateurs d’une adjonction  $L \dashv R$  sont l’unité  $\eta$  de la monade  $R \circ L$  et la counité  $\varepsilon$  de la comonade  $L \circ R$ , dessinées comme suit dans le langage des diagrammes de cordes :



Les deux relations qui régulent l’adjonction sont appelées “lois triangulaires” et s’expriment comme des déformations topologiques :

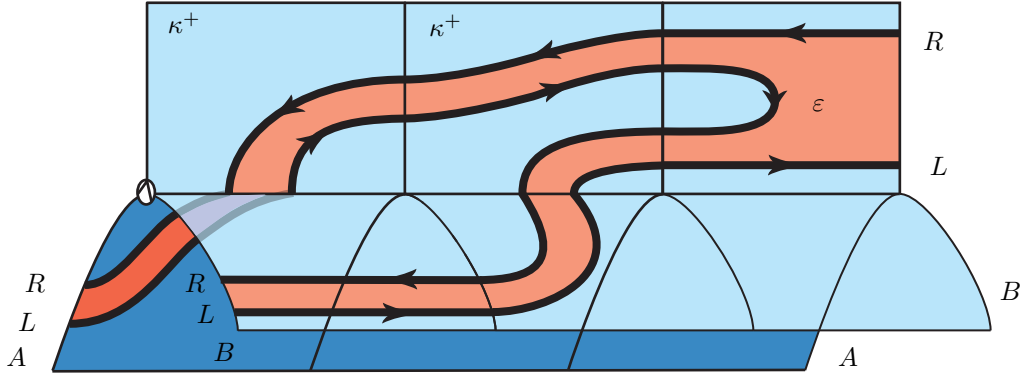


Dans ce cadre algébrique, la “sémantique des jeux” devient une “syntaxe de corde” décrivant la logique tensorielle. En effet, toute négation tensorielle définit une adjonction  $L : \mathbb{C} \rightleftarrows \mathbb{C}^{op} : R$  entre la catégorie sémantique  $\mathbb{C}$ , et sa catégorie opposée  $\mathbb{C}^{op}$ . De plus, il s’avère que toute preuve  $A \vdash B$  de logique tensorielle peut être décrite par générateurs et relations – de la même manière que tout nœud peut être présenté comme le composé de générateurs de tressage et de dualité. Parmi ces générateurs des preuves, on trouve les combinateurs  $\eta$  et  $\varepsilon$ , ainsi que des générateurs  $\kappa^+$  et  $\kappa^-$  décrivant des lois de distributivité entre conjonction et disjonction.

Point capital : on monte d’une dimension lorsque l’on passe des nœuds aux preuves : en effet, si les générateurs  $\eta$  et  $\varepsilon$  sont de dimension 2, les générateurs  $\kappa^+$  et  $\kappa^-$  sont de dimension 3 à cause de la présence des connecteurs  $\otimes$  et  $\oplus$  au sein des formules. Pour illustration, la preuve “par la gauche” du séquent

$$\neg\neg A \otimes \neg\neg B \vdash \neg\neg(A \otimes B)$$

est décrite par le diagramme de corde suivant :



La sémantique des jeux se trouve au cœur de cette syntaxe algébrique des preuves. En effet, la stratégie innocente associée à la preuve  $A \vdash B$  est décrite par les trajectoires des négations  $L$  et  $R$  (identifiées ici aux coups Joueur et Opposant) à l'intérieur du diagramme 3-dimensionnel – cela étant illustré par le diagramme précédent.

Pour établir ce résultat clef (la correspondance entre les preuves, vues comme des objets algébriques – compositions de générateurs modulo équations – et la notion usuelle de stratégie innocente), Melliès démontre un théorème de factorisation, qui énonce que toute preuve  $A \vdash B$  de logique tensorielle se décompose en une preuve  $A \vdash A^-$  extrayant une vue Opposant  $A^-$  de la formule  $A$ , une preuve  $B^+ \vdash B$  extrayant une vue Joueur  $B^+$  de la formule  $B$ , et une stratégie  $A^- \vdash B^+$  à information complète. Mimram travaille actuellement à une extension de cette approche purement algébrique aux jeux asynchrones non alternés décrits en [MM07]. Ce travail à l'interface entre théorie de la démonstration et réécriture sera décrit plus avant dans le prochain thème.

#### 1.4.5 Catégories $n$ -dimensionnelles

Métayer étudie les catégories de dimensions supérieures comme espaces de représentation du calcul. Par exemple, toute présentation d'un monoïde par un système de réécriture  $(S, R)$  donne une 2-catégorie ayant une seule cellule de dimension 0, pour cellules de dimension 1 les mots sur l'alphabet  $S$ , et pour cellules de dimension 2 les chemins de réécriture d'un mot à l'autre, modulo certaines équations de commutation. La généralisation de la réécriture aux dimensions supérieures se fait au moyen des polygraphes de Burroni [Bur93].

L'intérêt et la richesse des  $n$ -catégories tient essentiellement au fait qu'elles combinent de façon harmonieuse une structure géométrique qui évoque celle des CW-complexes, et une structure algébrique donnée par les lois de composition sur les cellules de toute dimension. Techniquement parlant, ce sont d'ailleurs les algèbres d'une monade sur la catégorie des ensembles globulaires, ces derniers ne contenant que les données de recollement des différentes cellules entre elles [Bat98].

La notion clé de résolution polygraphique [Mét03] permet de définir une homologie des  $n$ -catégories, et laisse entrevoir des invariants plus fins, de nature homotopique. C'est le point de départ d'une collaboration avec Yves Lafont (IML), visant une théorie homotopique du calcul. Parmi les résultats déjà obtenus, [LM07a] montre que l'homologie polygraphique évoquée plus haut coïncide avec l'homologie habituelle dans le cas particulier des monoïdes. D'autre part, [Mét07] caractérise les  $n$ -catégories librement engendrées par un polygraphe de façon abstraite par une propriété de relèvement par rapport à une classe de morphismes.

Plus récemment, Lafont, Métayer et Krzysztof Worytkiewicz ont montré que les notions précédentes s'insèrent en fait comme on pouvait le soupçonner dans une structure modèle sur la catégorie  $Cat_\omega$  des  $\omega$ -catégories : en particulier les résolutions polygraphiques sont des fibrations

acycliques de cette structure, et les objets cofibrants sont exactement les catégories librement engendrées par un polygraphe. Les équivalences faibles sont définies à partir de la notion d' $\omega$ -équivalence entre  $n$ -cellules, sorte d'égalité faible définie coinductivement sur la dimension. On remarque également que cette structure généralise de façon très précise la structure modèle sur  $Cat$  évoquée par Joyal et Tierney dans [JT91], et la structure construite par Lack sur  $Cat_2$  [Lac02, Lac04]. L'inclusion naturelle de ces catégories dans  $Cat_\omega$  possède en effet un adjoint à gauche qui détermine des adjonctions de Quillen entre  $Cat$ ,  $Cat_2$  et  $Cat_\omega$  munies de leurs structures de modèles respectives.

Cette clarification conceptuelle permettra, on l'espère, dans la période suivante, de retourner à la motivation initiale de cette recherche : comprendre les liens qui peuvent exister entre les invariants géométriques d'une structure et les propriétés du calcul dans les systèmes de réécriture qui la présentent. Il s'agit d'un vaste chantier, qui s'inscrit dans la lignée des travaux précurseurs de Squier sur les monoïdes [Squ87].

#### 1.4.6 Homotopie de la concurrence

Gaucher introduit dans [Gau03] une catégorie de modèles (cadre catégorique possible pour faire de l'homotopie, cf. plus haut) dont les objets modélisent le flot temporel d'un processus concurrent. Un flot temporel  $y$  est modélisé par une petite catégorie sans identité enrichie sur les espaces topologiques compactement engendrés. Les objets modélisent les états du système et les morphismes ses chemins d'exécution. Les équivalences faibles de cette catégorie de modèles sont les morphismes de flots temporels induisant une bijection sur les états et une équivalence faible d'homotopie sur les espaces de chemins. La propriété essentielle de cette catégorie de modèles est que le segment dirigé n'est pas équivalent à un point. Autrement dit, les équivalences faibles préservent bien la structure causale du flot temporel – propriété indispensable si l'on veut décrire la situation informatique.

La motivation principale de ce travail est l'étude de théories homologiques décrivant les branchements et les confluences non déterministes. En effet, l'homologie capture une série de propriétés cardinales de la structure causale du flot temporel. Ces théories homologiques sont définies dans [Gau05c] où une longue suite exacte est construite. L'espace d'états sous-jacent à un flot temporel n'est défini qu'à homotopie près [Gau05a]. Les équivalences faibles de [Gau03] définissent une partie de la classe des bisimulations fortes de haute dimension – classe qu'il reste à définir et à étudier correctement sur les flots.

Cependant, ces équivalences faibles ne permettent pas de prendre en compte la notion de "raffinement" de l'observation. La définition correcte de raffinement est introduite dans [Gau07], après un essai infructueux dans [Gau05a]. Il est alors démontré que les homologies des branchements et des confluences sont invariantes par raffinement de l'observation [Gau06b] et que l'espace sous-jacent est également invariant par raffinement de l'observation [Gau06c]. Dans [Gau05b], on démontre que le raffinement de l'observation ne peut être pris en compte par aucune structure de catégorie de modèles sur les flots. Mais on démontre dans [Gau06a] que l'on peut quand même construire pour le raffinement de l'observation un analogue du théorème classique de Whitehead inversant les équivalences faibles d'homotopie entre CW-complexes.

### 1.5 Perspectives

**Sémantique des processus concurrents et logique linéaire.** A travers le lien entre les objets logiques que sont les réseaux d'interaction différentiels, les jeux asynchrones et la ludique (L-nets) d'une part, et les calculs de processus de l'autre, une correspondance de Curry-Howard pour la concurrence semble prendre forme. Ce sera la mission principale du projet CHOCO (Curry Howard pour le Concurrency), sélectionné par l'ANR en 2007, que d'élucider plus avant

cette correspondance. Ce projet commencera en janvier 2008 et sera piloté par Ehrhard. Le projet durera trois ans.

**Sémantique des jeux et compilation.** La sémantique des jeux s'apparente à une compilation idéalisée d'un langage de haut niveau dans un langage de bas niveau. Cette analogie, déjà observée dans le cadre des algorithmes séquentiels (Berry et Curien) a été mise en œuvre par Danos, Regnier, Curien, Harmer pour construire des machines abstraites capables d'exécuter des stratégies. Il nous semble possible d'aller plus loin, et d'étudier de ce point de vue la compilation réelle : en particulier, nous disposons désormais pour cela d'une sémantique des types pour les langages machine [AMRV07] mise au point par Melliès et Vouillon en collaboration avec Andrew Appel et Christopher Richards – voir thème 2.

**Sémantique des jeux et interfaces APIs.** La sémantique des jeux offre un outil bien adapté pour analyser les primitives de concurrence dans les langages de programmation et les systèmes d'exploitation. Ainsi, Adam Bakewell, Dan Ghica, Harmer et Guy McCusker développent actuellement une sémantique des jeux pour un langage de “pre-emptive threads” idéalisé, pour les interfaces de programmation (APIs) décrites sous la norme POSIX. La première version du modèle intègre la création et terminaison dynamique de threads, notions typiques dans POSIX. Il sera intéressant de voir comment l'étendre dans un stade ultérieur avec des aspects nominaux, telle que la comparaison des noms de threads, en s'appuyant sur la technologie des jeux nominaux.

**Catégorification.** La procédure de “catégorification” consiste à relever une structure algébrique usuelle, en la structure catégorique correspondante. Exemple typique : la notion de “catégorie monoïdale” catégorifie la notion algébrique de “monoïde”. Est-il possible de donner une version “catégorique” de la notion usuelle d'algèbre de Frobenius ? La réponse est donnée par Street, qui observe que la notion de catégorie  $\ast$ -autonome coïncide (essentiellement) avec la notion d'algèbre de Frobenius dans une certaine bicatégorie de modules (ou distributeurs au sens de Bénabou). De fait, il apparaît qu'une notion “relâchée” d'algèbre de Frobenius  $F$ , où l'isomorphisme canonique  $F \cong F^{op}$  est remplacé par une adjonction  $F \rightleftarrows F^{op}$ , décrit exactement la structure de la négation en logique. Du coup, une preuve formelle est assimilée à une transformation naturelle entre différentes opérations tensorielles sur cette “catégorie de Frobenius”  $F$ . Ce thème offre un point de contact vivifiant et prometteur entre théorie de la démonstration (logique tensorielle), théorie des langages de programmation (continuations linéaires), et physique mathématique (cobordisme et catégorification).

**Sémantique des jeux, type dépendants et effets algébriques.** Un vaste chantier est ouvert pour opérer une synthèse attendue entre la sémantique des jeux, la théorie des types dépendants, et la théorie des effets algébriques – en particulier, exceptions et références. Par exemple, une bonne compréhension des mécanismes de la quantification existentielle est très utile pour comprendre les mécanismes symboliques régulant les modules de CAML (cf. les travaux de Montagu et Rémy) ou le “linking dynamique” (cf. les travaux d'Abadi, Gonthier et Werner [AGW04] sur la notation  $\varepsilon$  de Hilbert, très liés aux travaux de Krivine sur la réalisabilité classique de ZF). D'un autre côté, il existe une théorie algébrique des effets très étudiée, au parfum monadique, qu'il s'agira de combiner à la sémantique des jeux, et sa modalité de ressource, d'essence comonadique. En procédant dans cet esprit, nous espérons aussi tirer la sémantique des langages de programmation vers une sémantique des assistants à la démonstration – cela afin de rendre compte de leurs mécanismes internes (tactiques), d'indiquer des extensions possibles (effets) et de mieux comprendre comment en modulariser le code.

## 2 Réécriture et preuves

**Participants.** Roberto Di Cosmo, Stéphane Glondu, Mauricio Guillermo, Olivier Hermant, Philippe Hesse, Jean-Baptiste Joinet, Delia Kesner, Hélène Kraeutler, Jean-Louis Krivine, Olivier Laurent, Sylvain Lebresne, Yves Legrandgérard, Stéphane Lengrand, Séverine Maingaud, PaulAndré Melliès, Samuel Mimram, Alexandre Miquel, Vincent Padovani, Michel Parigot, Barbara Petit, Emmanuel Polonovski, Fabien Renaud, Paul Rozière, Jérôme Vouillon.

La théorie de la réécriture et la théorie de la démonstration sont intimement liées, comme en témoignent les nombreux travaux de recherche à cheval sur les deux domaines.

D'un côté, la réécriture combine des éléments de logique, d'algèbre universelle, de preuve automatique, de programmation fonctionnelle. . . Et si plusieurs systèmes calculatoires sont issus de l'algèbre, comme les systèmes de réécriture du premier ordre, de nombreux autres s'inspirent de la logique (intuitionniste, classique, linéaire) comme le  $\lambda$ -calcul ou, aujourd'hui, les réseaux de preuve ou d'interaction. En particulier, la réécriture d'ordre supérieur intègre, dans une syntaxe unique, des formalismes qui trouvent leur origine dans la théorie de la démonstration, comme le  $\lambda$ -calcul typé, ou dans l'algèbre, comme les systèmes de réécriture du premier ordre. On dispose ainsi d'un formalisme capable de modéliser en même temps les langages fonctionnels, équationnels ou orientés objets, aussi bien que des assistants à la démonstration, ou encore des langages concurrents.

De l'autre côté, la théorie de la démonstration est décrite et étudiée au moyen de systèmes de réduction qui spécifient la dynamique de ses objets. Ainsi, l'élimination des coupures est l'une des propriétés fondamentales que peuvent vérifier des systèmes logiques. Cette élimination des coupures découle en général d'une procédure de calcul (réécriture) qui transforme une preuve avec coupures en une preuve équivalente sans coupures. La correspondance de Curry-Howard a beaucoup contribué à la mise en évidence d'un lien profond entre logique et calcul : les propositions sont des types, les preuves sont des programmes, et la procédure de normalisation d'une preuve s'interprète comme un système d'évaluation/réécriture. La théorie de la démonstration se développe dans une telle symbiose avec les systèmes calculatoires qu'elle se confond même avec la sémantique des langages de programmation.

### 2.1 Réécriture

La théorie de la réécriture repose sur l'idée de manipuler et de transformer des objets tels que des expressions, des valeurs, des graphes, des propriétés et/ou des méthodes, apportant ainsi un complément aux paradigmes impératifs de la programmation où l'on décrit une séquence d'instructions gérant et transformant la mémoire d'une machine. Elle est utilisée pour modéliser un grand nombre de paradigmes de programmation comme les langages fonctionnels, logiques, équationnels et orientés objets aussi bien que des langages concurrents et distribués, des systèmes non déterministes ou des assistants à la démonstration. La réécriture peut aussi se voir comme un outil de spécification pour le raisonnement logique/mathématique, les protocoles de sécurité, la résolution de contraintes, les systèmes de transitions, les langages naturels, les machines abstraites, la recherche de preuves, la sémantique opérationnelle, la transformation de programmes, . . . Dans tous ces exemples, il y a une description donnant une information sur des valeurs, plutôt que des instructions pour produire des effets particuliers sur l'état de la mémoire.

Les activités autour de la réécriture à PPS mettent l'accent sur la modélisation des langages fonctionnels, l'étude de la sémantique opérationnelle dans certains systèmes logiques, les formalismes syntaxiques permettant la transformation d'objets d'ordre supérieur et la réécriture multidimensionnelle.

### 2.1.1 Modélisation de langages avec motifs

Le filtrage de motifs est une technique très naturelle pour définir des opérations par cas dans des langages de programmation, des langages permettant la manipulation de données XML ou des assistants à la démonstration.

Cependant, la modélisation traditionnelle de ces langages à travers des formalismes tels que le  $\lambda$ -calcul cache certaines de ces caractéristiques propres par l'intermédiaire de lourdes traductions. Ceci est à l'origine de plusieurs travaux mettant l'accent sur l'idée d'un filtrage primitif associé à des constructeurs de liaison ainsi qu'aux notions de sémantiques opérationnelles correspondantes.

Certains langages avec motifs ont été conçus sous l'inspiration de la correspondance de Curry-Howard. Cette correspondance n'apparaît pas uniquement au niveau de la syntaxe mais également au niveau de l'évaluation. Ainsi, les travaux [CK04] menés par Kesner en collaboration avec Serena Cerrito montrent comment le processus d'élimination des coupures – procédure de normalisation dans les systèmes logiques – trouve une interprétation très naturelle comme le processus de filtrage de fonctions définies par cas à l'aide de motifs. Par exemple, le filtrage de la paire  $\langle M, N \rangle : A \times B$  avec le motif  $\langle P, Q \rangle : A \times B$  se réduit tout naturellement au filtrage du terme  $M : A$  avec le motif  $P : A$  et au filtrage du terme  $N : B$  avec le motif  $Q : B$ . Dans un formalisme logique, ceci exprime tout simplement le remplacement d'une coupure sur la formule  $A \wedge B$  par des coupures sur les sous-formules  $A$  et  $B$ .

Kesner a poursuivi ces recherches en collaboration avec Julien Forest, afin d'incorporer aux langages fonctionnels avec motifs la possibilité de définir des nouvelles règles de calcul dites de réécriture. Cela a abouti à la définition d'un cadre très général, appelé ERSP ("Expression Reduction Systems with Patterns") [FK03], où l'on peut représenter sous une même syntaxe des programmes fonctionnels aussi bien que des preuves par réécriture d'ordre supérieur. L'une des originalités de ce formalisme se trouve dans la notion de filtre générique : le mécanisme de liaison n'est pas uniquement possible sur des motifs imbriqués, mais aussi sur des motifs quantifiés dénotés par des méta-variables.

Tous les travaux traditionnels dans le domaine de la modélisation de langages avec motifs séparent systématiquement l'ensemble des motifs de l'ensemble des termes. C'est donc en confondant ces deux ensembles que l'on arrive à donner aux motifs un vrai statut de *citoyen de première classe*. Dans cette direction, Kesner et Barry Jay [JK06] ont introduit le *Pure Pattern Calculus*, dont la simplicité et le pouvoir consistent tout simplement à considérer n'importe quel terme comme un motif potentiel. Cette approche apporte deux nouvelles formes de polymorphisme : le *polymorphisme de chemin* qui offre un traitement homogène pour la recherche de motifs sur n'importe quelle structure de données et le *polymorphisme de motifs* permettant la création dynamique et l'évaluation de motifs.

Les langages avec motifs (re)posent des questions auxquelles on sait déjà répondre dans le cadre du  $\lambda$ -calcul pur comme par exemple la propriété de *séparation*, qui exprime que deux termes qui normalisent face aux mêmes contextes d'évaluation ont la même forme normale (ou plus généralement, le même arbre de Böhm). Cette propriété initialement démontrée par Böhm dans le cas du  $\lambda$ -calcul pur a été récemment étendue par Saurin au  $\lambda\mu$ -calcul de Parigot dans le cas non typé. En collaboration avec Ariel Arbisser et Alejandro Ríos (Univ. de Buenos Aires), Miquel a défini dans [AMR06] une extension du  $\lambda$ -calcul avec des constructeurs et un mécanisme de filtrage suffisamment riche pour exprimer le filtrage à la ML tout en vérifiant la propriété de séparation. L'originalité de ce formalisme réside dans le traitement du filtrage qui se propage comme une substitution linéaire de tête et commute en particulier avec l'application. Arbisser, Miquel et Ríos ont en outre démontré que ce système est confluent, ainsi qu'une large classe de sous-systèmes (obtenus en combinant certaines des 9 règles de réduction primitives), à l'aide d'une technique originale de type *divide and conquer*, laquelle consiste à déterminer l'ensemble

de toutes les paires de sous-systèmes qui commutent à partir d'un petit ensemble de lemmes de commutation bien choisis.

Au cours de son stage de M2 sous la direction de Miquel, Petit a défini pour ce calcul un système de types au second ordre et a établi ses principales propriétés, à savoir la préservation du typage par réduction et la normalisation forte.

### 2.1.2 Ressources explicites

Le  $\lambda$ -calcul est souvent utilisé pour modéliser les langages de programmation fonctionnels, son unique règle d'évaluation  $(\lambda x.M)N \rightarrow_{\beta} M\{x := N\}$  fait appel à une notion de substitution externe au langage (méta-substitution) qui consiste à remplacer de manière globale toutes les occurrences libres de la variable  $x$  dans  $M$  par le terme  $N$ . Ceci s'éloigne de la situation que l'on retrouve dans l'implémentation des langages fonctionnels où la substitution n'est plus une notion atomique, mais au contraire une opération interne au langage (substitution explicite) dont l'évaluation se décompose en plusieurs étapes — à l'aide d'un ensemble de règles bien adaptées aux objectifs du langage en question. L'introduction de la notion de substitution explicite [ACCL91] a ouvert une nouvelle voie de recherche, car il était nécessaire de définir des calculs spécialement adaptés aux différentes exigences provenant du monde de la programmation fonctionnelle, de la recherche de preuves, de la vérification des machines abstraites, des logiciels d'aide à la démonstration, etc. Cela demande l'étude des propriétés de confluence, de normalisation, ou de préservation de normalisation forte (PSN), dont certaines ne sont pas évidentes à établir : en particulier, le résultat de Melliès en 1995, montrant que certains systèmes de substitutions explicites ne préservent pas la normalisation forte du  $\lambda$ -calcul, était assez inattendu. Mais il a servi en même temps de *challenge* à la recherche de solutions alternatives permettant une notion d'interaction entre substitutions explicites qui soit correcte vis-à-vis de la propriété de normalisation.

Les relations fondamentales qu'entretiennent le  $\lambda$ -calcul, la théorie des catégories et la théorie de la preuve se retrouvent de manière toute naturelle transposées dans le domaine des calculs avec substitutions explicites. Ainsi, l'un des premiers  $\lambda$ -calculs avec substitutions explicites, le  $\lambda\sigma$ -calcul [ACCL91], s'inspire des opérateurs catégoriques de Curien, tandis que des langages plus récents comme le  $\lambda_{ws}$  [DG01] s'interprètent naturellement dans un système logique intuitionniste avec affaiblissement explicite. La logique linéaire [Gir87] (LL) régule ce champ par la corrélation fructueuse qui apparaît entre la notion de ressource logique d'une part, et celle d'opérateur explicite de substitution d'autre part. Plus précisément, la logique linéaire propose un mécanisme de contrôle de ressources qui permet d'interpréter de manière plus fine et plus précise des mécanismes de déduction de la logique, qu'elle soit classique ou intuitionniste (cf. thème 1). Ainsi, Di Cosmo, Kesner et Polonovski [DCKP03] ont mis en évidence la relation entre le comportement de la substitution explicite dans le calcul  $\lambda_{ws}$  et le mécanisme d'élimination des coupures dans les réseaux de preuve de MELL, un fragment de la logique linéaire capturant, entre autres, le pouvoir de la logique intuitionniste.

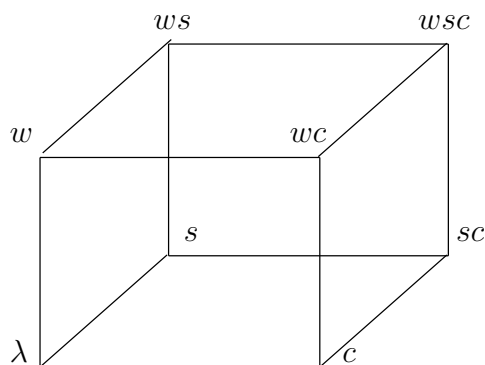
Cependant, la gestion du partage qui est propre aux règles multiplicatives de certains systèmes logiques comme MELL n'est pas reflétée dans le calcul  $\lambda_{ws}$ , où l'on contrôle uniquement les effacements. Kesner et Lengrand ont poussé encore plus loin la relation entre opérateurs explicites dans un langage fonctionnel et le contrôle de ressources en logique linéaire en définissant un calcul, appelé  $\lambda\mathbf{1x}\mathbf{r}$  [KL05, KL07], qui clarifie l'encodage de la logique intuitionniste dans la logique linéaire, en particulier en mettant sa notion de calcul (sa procédure de normalisation) en correspondance avec l'élimination des coupures dans les réseaux de preuve de MELL. Dans le cas général (non typé),  $\lambda\mathbf{1x}\mathbf{r}$  met en évidence le contrôle des ressources du  $\lambda$ -calcul, en particulier la duplication, l'effacement et la propagation de données. Il améliore aussi des résultats précé-



dents en offrant la possibilité de composer *sans restriction* toute paire de substitutions explicites consécutives, tout en préservant la terminaison des programmes. Cette dernière propriété stipule que l'évaluation, par les règles de calcul des substitutions explicites (proches de l'implémentation), d'un  $\lambda$ -terme dont la  $\beta$ -réduction termine, termine elle aussi. La compatibilité de cette propriété avec la composition des substitutions avait été mise en défaut par le contre-exemple de Melliès (pour le  $\lambda\sigma$ -calcul). La réponse apportée par le  $\lambda_{ws}$ -calcul avait le défaut de ne pas laisser interagir deux substitutions consécutives quelconques.

Bien que le calcul  $\lambda\mathbf{1xr}$  [KL07] apporte une solution au problème mis en évidence par le contre-exemple de Melliès, son nombre de règles de réécriture (19) et d'équations (6) le rend assez complexe. Il était donc nécessaire de bien comprendre les mécanismes essentiels de  $\lambda\mathbf{1xr}$  permettant de garantir des bonnes propriétés de confluence et de terminaison en même temps. Une réponse simple et concise a été proposée par Kesner [Kes07], qui a défini un nouveau calcul, appelé  $\lambda_{es}$ , lui aussi inspiré des réseaux de preuve de la logique linéaire, mais réduit seulement à un petit nombre de constructions syntaxiques et de règles d'évaluation. Ce langage donne un mécanisme de spécification pour la substitution simultanée, tout en utilisant une syntaxe minimaliste avec substitutions unaires, où un seul axiome équationnel suffit à exprimer la propriété de commutation entre deux substitutions unaires indépendantes.

Les calculs avec ressources explicites peuvent tous se voir comme des extensions syntaxiques du  $\lambda$ -calcul représentables dans les réseaux de preuve de la logique linéaire où l'on choisit de gérer de manière explicite ou implicite les opérations de substitution ( $s$ ), duplication/contraction ( $c$ ), et effacement/affaiblissement ( $w$ ). Au cours de son stage de M2 sous la direction de Kesner, Renaud a proposé une reconstruction formelle de tous ces langages en termes d'un *cube de ressources*, où chaque point représente une manière particulière de gérer les ressources du  $\lambda$ -calcul.



Des liens entre la notion de type intersection et celle de normalisation forte dans les calculs avec substitutions explicites ont été étudiés par Lengrand dans [LLD<sup>+</sup>04]. De nouvelles techniques pour montrer la propriété PSN dans des systèmes de réécriture ont été aussi présentées dans [Len04] et approfondies dans [Len05], qui expose également de manière constructive la théorie de la normalisation et ses techniques de base.

### 2.1.3 Formalisation de la réécriture d'ordre supérieur

La formalisation des systèmes de réécriture d'ordre supérieur est confrontée aux notions de variables liées et de substitution, avec les complications qu'elles apportent, comme l' $\alpha$ -conversion d'un côté, et la notion si complexe de substitution de l'autre côté. Des erreurs d'implémentation de langages d'ordre supérieur qui sont bien connues dans la communauté de développeurs de ces langages proviennent bien souvent de ces deux aspects incontournables. Plusieurs techniques

existent aujourd’hui pour mieux formaliser les mécanismes de liaison, plusieurs manières de formaliser la notion de substitution d’ordre supérieur ont été abordées.

Dans la première direction, Kesner, Eduardo Bonelli (Univ. La Plata) et Ríos ont proposé un formalisme pour spécifier des systèmes de réécriture d’ordre supérieur, appelé *Simplified Expression Reduction Systems à la de Bruijn* (*SERS<sub>dB</sub>*) [BKR05a] où les variables sont remplacées par des indices, dits “indices de de Bruijn” afin d’éviter les nombreux inconvénients de l’ $\alpha$ -conversion. Ces systèmes sont inspirés des *Expression Reduction Systems* [Kha90] ; un chapitre intitulé *Expression Reduction Systems and Extensions : An Overview* par John Glauert (Univ. East Anglia), Kesner et Zurab Khasidashvili (Intel, Haifa) est paru [GKK05] à l’occasion du *Klop Festschrift*.

Le pouvoir d’expression des systèmes d’ordre supérieur a été abordé en suivant la deuxième direction évoquée ci-dessus. Plus précisément, Kesner, Bonelli et Ríos ont démontré [BKR05b] qu’un système d’ordre supérieur peut s’exprimer par un système de réécriture du premier ordre modulo une théorie équationnelle. Cette traduction est possible grâce aux substitutions explicites, et permet d’établir formellement la relation entre les systèmes de réécriture d’ordre supérieur et ceux du premier ordre. Elle permet en même temps de raisonner sur les constructeurs “de liaison” utilisés dans les langages et formalismes d’ordre supérieur.

#### 2.1.4 Réécriture comme outil de raisonnement logique

Roy Dyckhoff, Kesner et Lengrand [DKL06] présentent un langage de termes de preuves muni d’une sémantique opérationnelle pour le calcul des séquents intuitionniste **G4**. La spécificité du système logique **G4** est de poser des contraintes sur la construction de preuves afin que la taille de celles-ci soit bornée (en fonction des propositions qu’elles prouvent). Contrairement à d’autres systèmes logiques, comme par exemple LJQ, l’espace de recherche de preuve est alors fini, et les règles d’inférence définissent une procédure de décision pour la logique intuitionniste propositionnelle. La sémantique opérationnelle de Dyckhoff, Kesner et Lengrand s’applique à un calcul plus large, non borné et avec coupures, et normalise toute preuve en une preuve du calcul **G4** borné et sans coupure. Ce calcul plus large encode facilement le  $\lambda$ -calcul simplement typé, et la faible normalisation de cette procédure d’élimination des coupures suffit alors à donner une preuve (nouvelle et directe) de la complétude de **G4**. Dyckhoff, Kesner et Lengrand montrent cependant le résultat plus fort de forte normalisation de la procédure et de plusieurs de ses variantes, qui permettent toutes de réduire des preuves longues en preuves courtes, sous la borne, en repérant les redondances inutiles. Ainsi l’encodage avec coupures du  $\lambda$ -terme “ $1 + 1$ ” est réduit en 1, preuve du type des entiers plus courte que (le codage de) 2. La définition d’une sémantique dénotationnelle pour le langage **G4** et la mise en relation de ses preuves et réductions avec celles du  $\lambda$ -calcul permettraient de comprendre d’autres propriétés dynamiques de ce langage.

#### 2.1.5 Réécriture multidimensionnelle

Sollicité en 2006 par Albert Burroni et les membres du projet INVAL, Padovani poursuit son étude sur le problème du calcul effectif du produit tensoriel de polygraphes, l’analyse algorithmique de ce produit ainsi que son implémentation pour diverses classes de polygraphes. Il s’agit d’un problème excessivement difficile, mais fondamental. Car, comme nous l’avons expliqué dans le thème 1, les polygraphes constituent un cadre d’étude naturel pour la réécriture de dimension supérieure, dans lequel les propriétés de l’exécution d’un programme sont déduites de propriétés d’objets géométriques. Intuitivement, le produit tensoriel de deux polygraphes peut être vu comme l’enchevêtrement des exécutions de deux machines travaillant en parallèle.

### 2.1.6 Réécriture et algèbres de jeux

Nous l'avons souligné dans notre premier thème, la sémantique des jeux offre un outil robuste et malléable pour décrire une grande variété de systèmes logiques, et de langages de programmation. Dans ce cadre, la caractérisation du comportement interactif des programmes d'un langage donné (typiquement, le langage PCF) est habituellement obtenue au moyen de conditions combinatoires subtiles sur les stratégies (typiquement, des conditions d'innocence). Il est apparu dans les travaux récents de Melliès [Mel] qu'il était possible de comprendre certaines de ces conditions combinatoires d'un point de vue différent, plus directement algébrique.

Mais pour cela, il faut mettre en place une méthodologie nouvelle, basée sur des techniques diagrammatiques à mi-chemin entre sémantique des jeux, théorie de la réécriture, et algèbre catégorielle. Point cardinal de cette approche : une description par *générateurs* et *relations* des conditions combinatoires bien connues (typiquement l'innocence) sur les stratégies définies en sémantique des jeux. Cette correspondance algèbre-sémantique offre un point de vue tout à fait nouveau et éclairant sur les théorèmes de complétude dénotationnelle établis dans le domaine.

Melliès a pu ainsi reconstruire [Mel] la condition d'innocence sur les stratégies alternées (cf. thème 1). Mimram [Mim07] s'est attaqué au problème tout aussi délicat de reconstruire algébriquement une catégorie de jeux semblable à celle décrite en [MM07], dont les stratégies sont innocentes, mais plus nécessairement alternées. Plutôt que d'aborder ce problème difficile de front, il a préféré analyser tout d'abord la structure des tours de quantificateurs universels et existentiels, définissant le fragment sans connecteur de la logique du premier ordre.

Ces tours de quantification définissent une certaine catégorie monoïdale. Mimram démontre par des méthodes de réécriture élaborées, inspirées des travaux de Lafont [Laf03], qu'il s'agit d'une catégorie monoïdale libre (ou PROP) définie par une certaine structure algébrique, satisfaisant des lois proches de celles des bigèbres. Cela démontre que, de la même manière que dans le cas alterné, les stratégies innocentes non alternées du modèle peuvent être générées par composition séquentielle et parallèle, à partir de stratégies atomiques en nombre fini – l'égalité entre ces stratégies étant elle-même finiment axiomatisable. Reste aujourd'hui à étendre ce modèle à des fragments plus importants de la logique du premier ordre.

## 2.2 Preuves

### 2.2.1 Systèmes de preuve

**Traductions du  $\lambda$ -calcul en logique linéaire.** Peu après l'introduction de la logique linéaire, Seely [See89] a montré que la traduction la plus simple donnée par Girard du  $\lambda$ -calcul dans LL (traduction en appel par nom) correspond à la construction d'une catégorie cartésienne fermée à partir d'un modèle de la logique linéaire via la construction de co-Kleisli. Hyland et Laurent [HL07] ont abordé la question de l'interprétation catégorique d'autres traductions du  $\lambda$ -calcul dans LL. Il apparaît que, pour les traductions étudiées (les traductions de Girard en appel par nom et en appel par valeur, la  $\mathbf{t}$ -traduction et la  $\mathbf{q}$ -traduction [DJS95]), la catégorie mise en valeur par Seely est centrale. Les catégories que l'on obtient sont notamment issues de l'application, à cette catégorie, de constructions de Kleisli pour des monades fortes dans l'esprit de [Mog91].

**Un langage de preuves pour LJQ.** Le calcul des séquents LJQ est une restriction du calcul des séquents pour la logique intuitioniste qui contraint et oriente la construction de preuve par un mécanisme appelé focalisation, qui joue un rôle central dans les développements récents en logique linéaire (cf. thème 1).. Lengrand et Dyckhoff ont proposé [DL06, DKL06] un langage de termes de preuves pour LJQ en lien avec la sémantique du  $\lambda$ -calcul en appel par valeur tel que

le présente Moggi. Par l'établissement d'une situation appelée "reflexion", Lengrand et Dyckhoff ont montré en particulier l'adéquation entre le  $\lambda$ -calcul de Moggi et sa sémantique telle que la donne Fischer, une adéquation mise en doute dans la littérature pour des raisons techniques mais non essentielles.

**Focalisation en logique linéaire.** Puisant fortement dans les idées utilisées par Girard dans son système LC [Gir91], et s'appuyant sur l'intégration de certaines de ces idées dans LL via la logique linéaire polarisée, Laurent a donné une preuve de la propriété de focalisation [Lau04] (et de quelques variantes) centrée sur l'élimination des coupures. Il s'agit principalement de la rédaction formelle d'idées faisant partie du "folklore linéaire".

**Traductions entre  $\lambda$ -calculs classiques.** Depuis la fin des années 80, de nombreux travaux ont mis en lumière le contenu calculatoire des preuves classiques. En particulier, la notion de terme de preuve mise en évidence par la correspondance de Curry-Howard a progressivement convergé vers une notion de calcul non confluent aux nombreuses symétries. Ces symétries correspondant à celles du calcul des séquents classiques, deux formalismes de termes de preuves pour le calcul des séquents sont apparus indépendamment : celui de Urban et celui de Curien-Herbelin. Le premier est prouvé fortement normalisant par l'adaptation de la technique dite *des candidats de réductibilité symétriques* de Barbanera et Berardi, construits par point fixe. Le second s'inspire des travaux sur les encodages de la logique classique dans la logique linéaire et de la notion de focalisation pour exprimer la symétrie entre l'évaluation en appel par nom et en appel par valeur de la programmation fonctionnelle.

Lengrand a défini [Len03] des traductions entre les deux formalismes, si bien que, d'un côté, il obtient une preuve de normalisation forte du calcul de Curien-Herbelin (inférée à partir de celle de Urban), et de l'autre, le calcul de Urban hérite des sémantiques en appel par nom et en appel par valeur exprimées par Curien et Herbelin. Polonovski [Pol04a] donne une preuve directe de la normalisation forte du calcul de Curien-Herbelin, par la techniques des candidats de réductibilités symétriques. De plus, il ajoute la notion de substitution explicite au calcul, dont il infère également la normalisation forte par des techniques adaptées aux substitutions explicites. Celles-ci sont développées en détails dans [Pol04b], systématisant par exemple l'inférence de la normalisation forte à partir de la propriété PSN.

**Existence constructive en logique classique.** Parigot a travaillé sur la constructivité en logique classique, où la question centrale est celle de l'existence constructive. En mathématique classique, prouver un énoncé existentiel  $\exists x A(x)$  n'assure pas l'existence concrète d'un objet  $a$  tel que  $A(a)$ . Le problème de l'existence constructive est précisément d'assurer d'une façon ou d'une autre l'existence d'un objet correspondant. Parigot a donné en 2004 une preuve simple et générique d'une caractérisation mathématique de l'existence constructive en logique classique due à Baaz et Fermueller. Il a depuis mis au point une logique de la constructivité  $K$  qui, contrairement à la logique intuitionniste, permet de manipuler la constructivité en restant dans un cadre de logique classique. La logique  $K$  est une extension de la logique classique avec des connecteurs constructifs (disjonction constructive et quantification existentielle constructive) qui permet de mélanger librement les connecteurs classiques et constructifs et d'utiliser le raisonnement classique pour prouver des propriétés constructives. Techniquement, la constructivité est gérée par des contraintes structurelles plus fines que celles utilisées pour définir la logique intuitionniste ou la logique linéaire. Depuis, Parigot étudie les propriétés logiques du système, le but étant ensuite de formaliser des théories mathématiques dans ce cadre.

**Distributivité linéaire.** Joinet étudie [Joi07] la complétude de la *distributivité linéaire* (également appelée distributivité faible) dans les réseaux de preuve pour la logique linéaire multiplicative (MLL). Pour cela il utilise l’axiome de distributivité comme une règle de réécriture (appelée *WD*). Cette règle ne préserve pas le type des réseaux, mais préserve néanmoins leur correction. La contribution spécifique de ce travail est de fournir une preuve directe de complétude de *WD* (tout réseau monoconclusion s’obtient par *WD* réécriture, à partir d’un tenseur  $n$ -aire d’axiomes “parisés”, à commutativité-associativité près du tenseur et du par) et de clarifier la relation entre les réseaux de preuve et les calculs de structures introduits par Guglielmi [Gug07].

**Un système de preuves pour le calcul des structures.** Le calcul des structures est un formalisme où les preuves sont des séquences (de réécriture) de formules plutôt que des arbres. Un tel langage permet non seulement de définir facilement une notion d’égalité/équivalence sur ces preuves (lorsqu’elles sont “moralement” les mêmes), mais aussi d’exprimer des représentations canoniques pour ces classes d’équivalence. Kai Brünnler et Lengrand ont défini [BL05] un langage de termes de preuves pour le calcul des structures. Ils ont également fourni une procédure de normalisation qui compile ces représentations canoniques par un système de règles convergent. Leur approche se généralise à tout système de réécriture linéaire, et traite notamment [BL05] le calcul des structures pour la logique classique, logique pour laquelle la notion d’égalité de preuves n’est pas encore bien identifiée.

### 2.2.2 Systèmes de types

**PTS en calcul des séquents.** Les *systèmes de types purs* (PTS) regroupent dans un même cadre un grand nombre de théories des types, parmi lesquelles certaines sont à la base d’assistants à la preuve tels que Coq, Lego/Plastic ou Agda/Alfa. A ce titre, les PTS peuvent constituer une alternative séduisante à la théorie des ensembles comme fondement des mathématiques, dans la mesure où ils prennent bien mieux en compte la notion de calcul.

En collaboration avec Dyckhoff et James McKinna (St Andrews), Lengrand a défini [LDM06] un calcul des séquents pour les PTS et a établi ses propriétés fondamentales. Ce calcul des séquents sert de support à l’expression de tactiques de recherche de preuve, comme celles implémentées dans Coq ou Lego/Plastic. Lengrand a développé dans sa thèse [Len06] cette base théorique en introduisant des méta-variables pour représenter des preuves inconnues à déterminer. Ces méta-variables donnent plus de flexibilité à l’organisation de la recherche de preuves et peuvent par exemple décrire des algorithmes qui énumèrent les programmes (termes) d’un type (d’une proposition) donné(e), similaires à ceux de Dowek et Muñoz. Cas particulier de ces algorithmes, l’*unification d’ordre supérieur* devrait aussi s’exprimer directement dans ce calcul des séquents avec méta-variables.

**PTS et théorie des ensembles de Zermelo.** Pour obtenir une meilleure estimation de la puissance théorique des PTS courants par rapport à la théorie des ensembles, Miquel a dégagé, au sein de la hiérarchie des sous-PTS du calcul des constructions avec univers, un PTS à 4 sortes capturant exactement la puissance théorique de la théorie des ensembles de Zermelo. Pour ce faire, il a montré que ce PTS constitue (via une représentation des ensembles en théorie des types sous forme de graphes pointés, dûs à Aczel) une extension conservative de la théorie de Zermelo intuitionniste avec axiome de clôture transitive et d’antifondation. La preuve de conservativité repose sur l’utilisation d’une forme très restreinte du schéma de remplacement prouvable dans la théorie de Zermelo, et suffisante pour interpréter le produit dépendant.

**La théorie des ensembles de Zermelo en déduction modulo.** La déduction modulo est une extension de la logique du premier ordre qui permet de donner un contenu calculatoire à n'importe quelle théorie du premier ordre dont les axiomes peuvent s'exprimer sous forme d'un système de réécriture sur les formules. Gilles Dowek (LIX) et Miquel ont donné une formulation de la théorie des ensembles de Zermelo en déduction modulo (sans axiome) de manière à ce que la notion de preuve induite vérifie la propriété de normalisation forte (en ce qui concerne le fragment intuitionniste). Cette formulation de la théorie des ensembles de Zermelo repose sur la représentation des ensembles sous forme de graphes pointés (cf. plus haut).

**Une présentation classique du système  $F_\omega$ .** Dans [LM07b], Lengrand et Miquel s'intéressent à l'élimination des coupures en logique classique et aux techniques qui en montrent la terminaison (forte). Pour un cadre illustrant ces problématiques, ils choisissent de partir du système  $F_\omega$ , une théorie des types hiérarchisée en deux couches : une couche supérieure formée par un calcul fonctionnel de prédicats simplement typé (de manière intuitionniste), une couche inférieure de termes de preuves pour ces prédicats. C'est cette notion de preuve, originellement intuitionniste, qu'ils rendent classique. La forte terminaison des deux couches est établie par deux adaptations de la technique des candidats de réductibilité de Girard : une reformulation autour du concept d'*orthogonalité* pour la couche supérieure, une notion de *candidats symétriques* pour la couche inférieure, adaptée à l'élimination des coupures de la logique classique. Comparant ces deux techniques, ils montrent par un contre-exemple que la notion d'orthogonalité ne permet pas de remplacer la construction (par point fixe) des candidats symétriques.

**Un système F avec exceptions.** Lebresne a défini dans le cadre de sa thèse une extension du système F avec des exceptions en appel par nom, et dont le typage est capable d'indiquer si un programme est susceptible de lever une exception ou non. Ce travail a permis de dégager une nouvelle construction de type  $A^\varepsilon$  (corruption du type  $A$  par l'exception  $\varepsilon$ ) regroupant les programmes de type  $A$  susceptibles de lever l'exception  $\varepsilon$  dans n'importe quel contexte d'évaluation. Lebresne a ensuite construit un modèle de réalisabilité pour interpréter ces différentes notions et montrer les propriétés de correction du calcul.

### 2.2.3 Réalisabilité et extraction de programmes

**Réalisation des preuves mathématiques.** Krivine a approfondi la correspondance preuves-programmes (*Curry-Howard*) dans le cadre de la *réalisabilité classique* qu'il a progressivement dégagé au cours de ces dernières années et qui est aujourd'hui bien établi. Il s'agit d'une importante extension de la méthode du "forcing" qui permet de construire de nouveaux modèles de l'Analyse (arithmétique de Peano du second ordre) et de la théorie des ensembles (Zermelo-Fraenkel). Ce nouveau cadre, dans lequel les programmes sont écrits en langage *impératif* de très bas niveau, offre aujourd'hui la seule méthode connue pour transformer en programmes la totalité des preuves mathématiques. Il a permis de trouver les instructions à associer à l'axiome de fondation, à l'axiome du choix dépendant, puis à l'axiome du choix complet, et enfin à d'autres axiomes comme l'hypothèse généralisée du continu. Les instructions utilisées sont connues depuis longtemps en programmation impérative : la boucle while, l'horloge, la signature, la lecture-écriture dans une mémoire globale. Mais on peut maintenant *typer* ces instructions, c'est-à-dire prouver des programmes qui les utilisent.

Un énorme travail reste à faire pour trouver la *spécification* associée à chaque théorème mathématique ; autrement dit, comprendre le comportement des programmes associés aux preuves d'un théorème fixé. Krivine a récemment étudié le cas particulier des *formules valides du calcul des prédicats* (c'est-à-dire des théorèmes démontrables sans aucun axiome) et a montré que ces

programmes implémentent une stratégie gagnante pour un jeu associé à la formule. Krivine et Legrandgérard ont observé que ce jeu spécifie un *protocole de communication*. Par exemple, le jeu associé à la formule valide très simple  $\exists x \forall y (Px \rightarrow Py)$  correspond exactement à *l’envoi d’un paquet avec acquittement*.

**Jeux en réalisabilité.** Guillermo a étudié l’interprétation de la réalisabilité classique développée par Krivine en termes de jeux. La théorie générale de la réalisabilité classique exprime que le terme associé à une preuve d’un théorème  $\varphi$  est en mesure de combiner des stratégies gagnantes pour les hypothèses de manière à produire une stratégie gagnante pour la conclusion. Le problème fondamental est de comprendre comment les preuves arrivent à combiner les stratégies. Guillermo a abordé ce problème dans le cas particulier du modus ponens en forme normale prénexe. Le résultat constitue une généralisation de l’exemple présenté par Coquand dans [BBC95]. Guillermo a aussi présenté une généralisation de la définition des jeux en réalisabilité qui subsume les définitions données dans [Kri03, Kri05]. Cela lui a permis de travailler avec des formules comportant des quantificateurs existentiels intuitionnistes – et ayant en conséquence un contrôle très précis sur le comportement de la pile – tout en gardant la souplesse du langage présenté dans [Kri05]. Guillermo a également développé une méthode de construction de modèles de réalisabilité – la méthode des fils – qui permet d’étudier le problème de la spécification dans des cas divers. Grâce à cette méthode, il a montré qu’un réalisateur de la loi de Peirce se comporte toujours de façon similaire à l’opérateur `call/cc`.

**Réalisabilité classique dans le calcul des constructions.** La théorie de la réalisabilité classique introduite par Krivine s’exprime naturellement dans le cadre de la logique du second ordre, et s’étend facilement à toute la logique d’ordre supérieur. Miquel a montré que le modèle de réalisabilité sous-jacent pouvait être étendu au calcul des constructions avec univers (le noyau du système Coq) avec entiers primitifs, tiers-exclu et principe de *proof-irrelevance*. Techniquement, cette extension repose sur l’introduction d’une notion de  $\Pi$ -ensemble qui n’est autre que la transposition de la notion usuelle de  $\omega$ -ensemble (ou “ $\omega$ -set”) de Kleene, dans le cadre de la réalisabilité classique. D’un point de vue pratique, ce travail ouvre la voie à un schéma d’extraction classique des preuves de Coq vers le  $\lambda_c$ -calcul de Krivine, schéma d’extraction en outre compatible avec l’interprétation krivinienne de l’axiome du choix dépendant.

**Réalisation du théorème de Herbrand.** Krivine a démontré que l’on pouvait réaliser l’axiome du choix dépendant en se donnant une instruction utilisant une horloge système. Il n’y a cependant à ce jour pas d’exemple de  $\lambda$ -terme issu d’une preuve qui utilise cet axiome de manière non triviale. Hesse a réalisé le théorème de Herbrand, qui indique qu’une formule est démontrable si et seulement si sa forme de Skolem l’est, dont la preuve nécessite l’axiome du choix dépendant pour construire une fonction de choix. Il reste encore à comprendre dans un cadre général le comportement du terme obtenu, notamment lorsqu’on applique le théorème de Herbrand aux formules dont la spécification s’interprète naturellement dans le cadre des protocoles de communication.

**Extraction du lemme de Dickson.** Rozière et Kraeutler ont travaillé sur l’extraction de programme en logique classique. Leur point de départ a été le lemme de Dickson, un exemple traité par Schwichtenberg et Berger [BBS02], et déjà repris par Raffalli [Raf04]. L’utilisation de la réalisabilité de Krivine a permis d’en donner une formalisation plus simple. Parallèlement, Rozière a repris dans PhoX (assistant à la preuve développé à Chambéry) des bibliothèques modifiées pour qu’elles soient plus adéquates pour l’extraction de lambda-termes ( $\lambda_c$ ).

**Réalisabilité et types rékursifs.** Melliès et Vouillon [MV05, VM04] ont démontré la sûreté (“safety”) d’un  $\lambda$ -calcul typé au système de types très riche, avec types rékursifs, polymorphisme, et sous-typage. Cette propriété est fondamentale dans les langages de programmation typés : elle garantit que tout programme typé se réduit sans jamais commettre d’erreur de syntaxe, selon l’adage souvent répété que : “well-typed programs cannot go wrong”. C’est la première fois qu’on obtient un tel résultat sans enrichir artificiellement le  $\lambda$ -calcul de constructions ad hoc, – comme cela est fait, par exemple, avec le test dynamique de type dans le modèle idéal [MPS86, APP91] défini par MacQueen, Plotkin et Sethi.

La propriété de sûreté est démontrée dans un cadre purement opérationnel, en construisant un modèle de réalisabilité de types, inspiré des travaux de Krivine. Chaque type rékursif est approximé dans le modèle par une suite convergente d’intervalles de types. Melliès et Vouillon démontrent que la suite converge en utilisant un argument subtil sur la longueur d’un chemin menant à une possible erreur de syntaxe. Cet argument transpose dans le cadre purement opérationnel de la réalisabilité un principe topologique énoncé dans les domaines : toute suite croissante admet une limite.

**Sémantique des types dans les langages de bas niveau.** Poursuivant dans cette ligne par une collaboration avec Appel (Princeton) et Christopher Richards, Melliès et Vouillon [AMRV07] ont donné une sémantique des types rékursifs pour un langage de bas niveau (langage machine) – ce travail débouchant sur le premier modèle du polymorphisme, du sous-typage et des types rékursifs dans un langage avec variables modifiables (“mutable references”). La description s’appuie sur un modèle de Kripke, où chaque monde  $w$  décrit le type des cellules mémoire actuellement allouées, et où la relation d’accessibilité  $wRw'$  étend le monde  $w$  en un monde  $w'$  où de nouvelles cellules mémoire sont allouées. Une étiquette de temps  $k \in \mathbb{N}$  est ajoutée au monde  $w$ , afin d’assurer que la relation d’accessibilité  $R$  est bien fondée. Cela assure que la logique modale associée valide la règle de Gödel-Löb, familière en logique de prouvabilité :

$$\frac{A, \triangleright B \vdash B}{A \vdash B} \text{ Gödel - Löb}$$

où  $\triangleright$  est la modalité de nécessité (nécessité dans le strict futur) et  $A$  est n’importe quelle formule nécessaire (c’est-à-dire toujours vraie dans le futur). On établit alors, par un argument purement sémantique, la correction du système de types pour le langage machine. L’argument requiert les types dépendants, et a donc été formalisé dans le système Coq. La correction du système assure que tout programme bien typé n’induit pas d’erreur de syntaxe au cours de son évaluation.

## 2.2.4 Développements en Coq

**Certification de l’extraction de programmes.** Le mécanisme d’extraction implémenté dans l’assistant à la preuve Coq permet de transformer automatiquement toute preuve constructive (ou toute fonction Coq) en un programme fonctionnel (exprimé en OCaml, en Haskell ou en Scheme) qui hérite des propriétés de correction établies au niveau de Coq. Depuis son arrivée à PPS, Letouzey (qui a conçu dans sa thèse le mécanisme d’extraction implémenté aujourd’hui dans le système Coq) a apporté quelques améliorations et corrections au code de l’extracteur.

Les difficultés liées à la complexité du mécanisme d’extraction suggèrent naturellement d’entreprendre sa certification en Coq. Un tel travail de formalisation recouvre deux aspects : d’une part la formalisation des aspects théoriques développés dans la thèse de Letouzey, et d’autre part la preuve que l’implémentation actuelle effectue la transformation en accord avec la description théorique. Dans le cadre de son stage de M2 (MPRI), Glondou a entrepris sous la direction de Letouzey une grande partie de la formalisation des aspects théoriques du mécanisme d’extraction.



**Développement de bibliothèques en Coq.** Letouzey a développé et intégré à Coq deux nouvelles bibliothèques :

- Le couple de bibliothèques **FSets/FMaps**, qui fournissent des structures d'ensembles finis et de tables d'association calquées sur le modèle des bibliothèques correspondantes en OCaml. Ces bibliothèques (initiées en collaboration avec Jean-Christophe Filliâtre et Pierre Courtieu) conservent le souci d'efficacité des algorithmes tout en apportant les garanties de correction issues des preuves formelles.
- La bibliothèque **QArith** qui traite des nombres rationnels en Coq (définition et propriétés des opérations dans  $\mathbb{Q}$ , structure d'anneau, etc.).

Letouzey a également contribué à l'expansion du module **List** (qui traite des opérations sur les listes) et du module **NArith** (qui traite de l'arithmétique sur les entiers en représentation binaire).

**Participation au projet ANR CompCert.** Le projet ANR CompCert coordonné par Leroy (INRIA) a pour but la réalisation de compilateurs certifiés. Dans ce projet sont développés notamment un compilateur C vers assembleur PowerPC ainsi qu'un compilateur ML vers assembleur (thèse de Dargaye). Au sein du projet, Letouzey s'est intéressé à la possibilité de produire de l'assembleur vers d'autres architectures. Il a également travaillé à l'intégration des bibliothèques **FSets/FMaps** et a adapté le code de l'extracteur de programmes de Coq pour pouvoir brancher sa sortie vers le compilateur certifié.

### 2.2.5 Varia

**Modèles du  $\lambda\mu$ -calcul pur et types intersection.** Les systèmes de types intersection permettent de définir des modèles du  $\lambda$ -calcul pur (*i.e.* non typé). En rapport avec la notion de modèles de filtres, certains de ces systèmes sont basés sur une notion de sous-typage [BCDC83] (avec par exemple  $A \cap B \leq A$ ). Laurent [Lau05a] a montré comment modulariser la preuve de correction de ces systèmes (et des modèles engendrés) en traitant notamment la relation de sous-typage comme une relation de déduction logique et en formulant cette relation non pas comme un système axiomatique mais à l'aide de règles de déduction se prêtant à une analyse par la théorie de la démonstration en termes d'élimination des coupures. L'objectif initial de pouvoir ainsi étendre ces techniques à la construction de modèles du  $\lambda\mu$ -calcul pur [Par92] n'a pour l'instant pas été atteint.

Cependant une approche directe à la question des types intersection pour le  $\lambda\mu$ -calcul (sans sous-typage cette fois) a permis à Laurent de construire un modèle de la  $\beta\mu\rho$ -égalité et ainsi une version classique du modèle de Engeler. L'introduction de sous-typage et l'intégration de ces deux lignes de travaux aurait pour objectif de construire un modèle extensionnel (validant la  $\beta\eta\mu\rho\theta$ -égalité).

Une motivation forte pour la construction de modèles du  $\lambda\mu$ -calcul pur est donnée par la réalisabilité classique de Krivine. En effet, bien que celle-ci soit principalement appliquée à partir de structures syntaxiques, elle pourrait l'être à partir de tout modèle dénotationnel (non nécessairement extensionnel) du  $\lambda\mu$ -calcul pur, offrant ainsi la possibilité de construire de nouveaux modèles de la théorie des ensembles.

**Polynômes chromatiques.** Le nombre de  $k$ -coloriages d'un graphe  $G$  est donné par un polynôme  $P_G(k)$  dépendant uniquement du graphe  $G$  et appelé le polynôme chromatique de  $G$ . En collaboration avec Pasacal Berthomé (LRI) et Kim Nguyen (LRI), Lebesne a développé un algorithme [BLN06] permettant de calculer le polynôme chromatique d'un graphe, algorithme qui repose sur les propriétés des graphes triangulés et des arbres de cliques associés.

## 2.3 Perspectives

La correspondance de Curry-Howard a jeté un pont solide entre la logique et le calcul. De manière surprenante, cette correspondance est aujourd’hui plus complète dans le sens qui conduit de la logique au calcul que dans le sens opposé. Si on sait maintenant traduire quasiment toutes les démonstrations mathématiques en programmes, de nombreux traits de programmation (notamment les traits impératifs) n’ont pas encore trouvé leur équivalent logique. Une perspective à long terme est d’intégrer à cette correspondance les paradigmes de calcul issus de la programmation (notamment ses traits non fonctionnels) mais aussi les paradigmes issus de la réécriture.

**Problème de la spécification.** La réalisabilité classique soulève des questions inédites dans la mesure où elle associe à chaque énoncé mathématique un contenu calculatoire bien plus riche que dans le cas intuitionniste (i.e. réalisabilité de Kleene). Par exemple, s’il est connu depuis longtemps que les entiers issus de la réalisabilité intuitionniste se comportent essentiellement comme des itérateurs (les entiers de Church), les entiers issus de la réalisabilité classique peuvent avoir un comportement calculatoire beaucoup plus complexe, qui est encore très mal compris. Le problème qui consiste à déterminer le comportement calculatoire commun à tous les programmes réalisant un même énoncé, connu sous le nom de *problème de la spécification*, reste une direction de recherche centrale en réalisabilité classique. Dans cette perspective, l’interprétation du comportement calculatoire des réalisateurs en termes de jeux tient un rôle privilégié.

Une question afférente est de déterminer dans quelle mesure le problème de la spécification est sensible au langage utilisé pour réaliser les formules classiques. Cette question est d’autant plus importante qu’il existe aujourd’hui de nombreux formalismes différents qui étendent la correspondance de Curry-Howard à la logique classique, et dont les spécificités ne sont pas encore comprises dans le cadre général de la réalisabilité classique. Une meilleure compréhension des traductions qui relient ces formalismes classiques sera essentielle pour aborder cette question.

**Extraction classique en Coq.** Les résultats présentés dans [Miq07] montrent qu’il est possible d’étendre la réalisabilité classique (initialement définie en logique du second ordre) au noyau fonctionnel du système Coq. Cependant, il reste encore beaucoup de travail à faire, notamment en ce qui concerne le traitement des types inductifs, pour que cette forme d’extraction classique puisse être mise en œuvre sur un fragment réaliste du système Coq.

**Langages avec motifs.** Les questions soulevées par la définition récente de plusieurs langages avec motifs sont nombreuses. Le *Pure Pattern Calculus* introduit par Kesner et Jay s’avère plus puissant que d’autres langages avec motifs développés dans la littérature. Une comparaison formelle avec ces autres travaux est en cours de rédaction. D’autre part, l’ajout de types au calcul avec motifs peut se faire sous différentes approches, mais c’est celle inspirée du calcul de séquents de Gentzen qui semble la plus prometteuse, car la plus susceptible de permettre d’explicitement la symétrie entre motifs et termes. Cette idée s’appuie fortement sur la philosophie véhiculée par l’isomorphisme de Curry Howard.

Les types au second ordre pour les langages avec constructeurs sont à l’ordre du jour dans le travail de thèse de Petit. Plusieurs propriétés de ce langage ainsi que plusieurs extensions restent à explorer.

**Langages avec ressources.** Le langage équationnel avec substitutions explicites  $\lambda_{es}$  étudié par Kesner gère la substitution de manière locale car les opérateurs de substitution sont propagés tout au long d’un terme en descendant d’un seul niveau à chaque fois. Un problème intéressant à comprendre est celui de la gestion non-locale de la substitution, comme celui que l’on trouve

dans les systèmes réactifs de Milner, et plus précisément dans le  $\lambda$ -calcul avec substitutions non-locales inspiré de ces systèmes réactifs. Des travaux en cours montrent que la gestion non-locale de la substitution peut s'implémenter via une gestion locale de celle-ci, mettant ainsi en relation le calcul de Milner avec  $\lambda_{es}$ , mais surtout les systèmes réactifs, à la base du calcul de Milner, avec les réseaux de preuve de MELL, à la base du langage  $\lambda_{es}$ .

Plusieurs pistes existent aussi pour penser que le  $\lambda_{es}$ -calcul pourrait donner lieu à un langage encore plus concis et efficace où l'on disposerait de moins de règles de réduction et surtout de moins de conditions contraignant l'application de ces règles. La sémantique opérationnelle de ce calcul ne serait plus inspirée par les réseaux de preuve de MELL. Le défi majeur étant donc la vérification de ses bonnes propriétés pour lesquelles il ne sera plus possible de passer par une simulation du calcul dans les réseaux de preuve. Ce travail mettrait en évidence la relation entre perpétualité et normalisation forte dans les langages avec substitutions explicites et composition forte. Une première manière d'aborder cette étude est de bien comprendre la normalisation forte du système d'élimination des coupures associé aux réseaux de preuve de MELL : ce travail en cours est mené en collaboration par Kesner et René David (Univ. Savoie).

En ce qui concerne les calculs avec ressources, Renaud poursuit son travail sur le cube de ressources (cf. plus haut) en étudiant les propriétés fondamentales de tous les langages du cube d'un point de vue abstrait, où l'on arrive à spécifier le raisonnement formel dans un langage donné par des paramètres représentant les ressources explicites du langage en question. La confluence, la préservation de la normalisation ou la préservation de la  $\beta$ -normalisation forte pourront ainsi être démontrées de manière homogène et concise sans avoir à développer des preuves différentes pour chaque langage. Le travail décrit dans le paragraphe précédent serait très utile dans le cadre du cube avec ressources.

### 3 Concurrency et Modélisation

**Participants.** Samy Abbes (depuis 2007), Roberto Amadio (depuis 2005), Emmanuel Beffara, Pierre-Louis Curien, Frédéric Dabrowski, Vincent Danos, Yuxin Deng, Mehdi Dogguy, Russ Harmer, Samuel Hym, Fabien Tarissan, Daniele Varacca (depuis 2006), Min Zhang.

Le thème Concurrency et Modélisation s'est considérablement développé et diversifié à partir de 2003. En particulier, on remarquera que l'Université Paris-Diderot a contribué au renforcement du thème avec 3 recrutements<sup>1</sup> et que 6 thèses ont été soutenues.<sup>2</sup> Nous articulons la présentation autour de 3 sous-thèmes : modèles de programmation concurrente, systèmes probabilistes, modélisation de la biologie moléculaire.

#### 3.1 Modèles de la programmation concurrente

Dans ce premier sous-thème on s'intéresse à la programmation des systèmes distribués et communicants. Les formalismes utilisés ici sont pour la plupart des variantes ou des extensions du  $\pi$ -calcul, lequel constitue sans doute le modèle à la fois le plus simple et le plus puissant de la programmation concurrente. Du fait de leur simplicité, ces langages, qu'on appelle couramment des calculs de processus, sont particulièrement aptes à la représentation et l'étude des problèmes liés aux calculs décentralisés, comme par exemple la disponibilité et le contrôle de l'usage de

<sup>1</sup> Amadio en 2005, Varacca en 2006 et Abbes en 2007. Nous ne présentons que les résultats obtenus depuis leur arrivée à PPS.

<sup>2</sup> Beffara, Dabrowski, Deng, Hym, Tarissan et Zhang ont soutenu leur thèse et quitté le laboratoire.

ressources. Comme nous allons le voir, cette étude se fait souvent par l'utilisation de typages et logiques ad hoc, mais utilise également d'autres techniques d'analyse statique. Une des particularités de l'approche du laboratoire dans ce type de question est de s'appuyer sur les structures mathématiques et logiques mieux comprises de la programmation fonctionnelle.

### 3.1.1 Sémantique et Typage

Si le  $\pi$ -calcul fait actuellement figure de référence parmi les calculs de processus, l'abondance de ses variantes souligne le fait qu'aucun langage n'a encore atteint une pureté conceptuelle comparable à celle du  $\lambda$ -calcul. Un objectif général est donc de mettre en relation le  $\pi$ -calcul avec d'autres structures fondamentales de la concurrence.

Dans ce contexte, Beffara et Maurel [BM06] ont introduit un calcul de réseaux concurrents, inspiré des réseaux de preuve de la logique linéaire. Il s'agit d'un calcul graphique dans lequel l'ordonnancement entre actions est rendu explicite. Ce modèle de calcul permet d'étudier la notion de préfixage dans le calcul concurrent, et a conduit à montrer qu'il est possible de traduire sous forme de causalité entre communications toute forme d'ordonnancement au sein d'un processus, de façon analogue aux traductions par passage de continuation qui transforment en causalité des stratégies de réduction du  $\lambda$ -calcul.

En continuant l'analogie avec le  $\lambda$ -calcul, il est naturel de se poser la question du *typage* des processus. En adaptant au cas concurrent les principes de la réalisabilité classique de Krivine, Beffara [Bef06] a développé une technique de *réalisabilité concurrente*. À partir d'une notion abstraite d'observation formulée comme une relation d'orthogonalité entre processus (et qui peut représenter les notions usuelles de test *may* et *must*), il construit des opérateurs élémentaires qui décrivent les interactions entre processus. Ces opérateurs permettent de définir une logique qui garantit de bonnes propriétés à l'exécution, telles que l'absence de divergence et de blocage. Cette logique des comportements contient la logique linéaire, ce qui fournit une interprétation des démonstrations linéaires en tant que processus du  $\pi$ -calcul. Cette traduction peut être utilisée pour obtenir une reconstruction systématique de traductions typées de divers  $\lambda$ -calculs dans le  $\pi$ -calcul, et la réalisabilité classique de Krivine peut être reconstituée comme un sous-système de la réalisabilité concurrente.

Une limitation de cette approche est qu'elle ne type que des processus qui sont proches des programmes "fonctionnels". Par ailleurs, dans la pratique du calcul concurrent, on est intéressé à garantir d'autres propriétés que la terminaison. Par exemple, on peut associer des *capacités* aux types de canaux en distinguant les capacités de lire et d'écrire sur un canal. Deng [Den05a] a étudié l'impact de ce système de typage sur la théorie algébrique du  $\pi$ -calcul. En particulier, il a proposé des systèmes de preuve corrects et complets pour les termes (clos) finis.

Un autre système de typage a été proposé dans le cadre du  $D\pi$ -calcul. Il s'agit d'une extension du  $\pi$ -calcul dans laquelle tous les processus sont placés dans des *localités* afin de décrire leur répartition. Dans ce calcul, les processus peuvent *communiquer localement* et *migrer* entre localités. À côté des canaux de communication et des localités, Hym introduit une nouvelle famille d'identifiants, les *passports*, permettant un contrôle fin des migrations de processus : un processus doit disposer d'un passeport adéquat pour entrer dans une localité [HH07, HH06]. Afin de le structurer, le calcul est équipé d'un système de types qui associe un type à chaque identifiant pour vérifier qu'un processus n'utilise que les droits qu'il possède. L'ordre de sous-typage sur les types est étendu aux types de passeports suivant les localités d'origine des processus migrants. Hym a démontré que cet ordre admet des bornes inférieures sous certaines conditions et que les processus se conformant à la politique de typage conservent cette propriété au cours de leurs réductions. Hym a également étudié l'équivalence observationnelle : quand des processus exhibent-ils des comportements indiscernables pour un observateur ? En présence de passeports,

il est indispensable d'imposer à l'observateur d'être loyal, c'est-à-dire d'exiger la possession de passeports pour observer les communications ayant lieu dans les localités correspondantes. Ces contraintes définissent une congruence dite barbuée loyale, dont Hym a montré qu'elle pouvait être vue comme une bisimulation de système de transitions étiquetées.

Parmi les structures fondamentales de la concurrence, on compte aussi les *structures d'événements*. On parle alors de modèle *causal*, car causalité, indépendance et conflit y sont explicitement représentés. Ces structures ont été utilisées pour modéliser des calculs de processus comme CCS, mais on n'avait pas de modèle de structures d'événements pour le  $\pi$ -calcul. Varacca [VY06, CVY07] a récemment proposé un modèle pour le  $\pi$ -calcul à *mobilité interne* (il s'agit d'un fragment significatif du  $\pi$ -calcul où les noms transmis sont toujours nouveaux) (voir aussi les travaux de Faggian et al. décrits au thème 1).

### 3.1.2 Synchronie

Dans les années 80, Milner et ses collaborateurs ont proposé une théorie unifiée du calcul concurrent asynchrone et synchrone en montrant, par exemple, comment voir CCS comme une *désynchronisation* de SCCS. Dans les années qui ont suivi, on a assisté au développement de deux directions de recherche différentes qui se sont focalisées respectivement sur la programmation asynchrone avec le développement du  $\pi$ -calcul et synchrone avec le développement de langages spécialisés comme LUSTRE et ESTEREL. Le passage de CCS à SCCS était assez naturel et l'on peut se demander si un passage similaire existe du  $\pi$ -calcul à un  $\pi$ -calcul *synchrone*. Pour étudier cette question, Amadio a pris comme point de départ le *langage synchrone* SL proposé par Boussinot et De Simone. Il s'agit d'une relaxation du modèle ESTEREL où la réaction à l'*absence* d'un signal dans un instant peut seulement se situer à l'instant suivant. A la différence de SCCS, le modèle SL a graduellement évolué en un langage de programmation concurrent et de nombreuses mises en œuvre du modèle existent dans des environnements basés sur C, JAVA, SCHEME et CAML. Cependant la théorie sémantique de cette famille de langages synchrones n'avait pas été développée. Amadio [ABCB06, Ama07b] a proposé une description plus générale du modèle SL qui comprend un mécanisme de récursion et de génération de threads. Par ailleurs, il a introduit une traduction CPS vers une version récursive terminale du modèle qui comprend un seul opérateur de synchronisation (le **present**) et il a donné une nouvelle notion de bisimulation avec des bonnes propriétés de compositionnalité.

Le modèle SL original suppose que les signaux sont *purs* dans le sens qu'ils ne portent pas de valeurs. Dans ce cas, les calculs sont naturellement déterministes et la bisimulation coïncide avec l'équivalence de traces. Cependant, les langages de programmation qui ont été développés comprennent des types de données et cette extension rend le calcul *non déterministe*. Amadio [Ama07a] a proposé une extension minimale du modèle où les signaux portent des valeurs du premier ordre. La complexité du langage est comparable à celle du  $\pi$ -calcul et pour cette raison le calcul résultant a été appelé  $S\pi$ -calcul (S pour synchrone). Amadio a montré que la notion de bisimulation proposée pour le modèle (déterministe) SL est suffisamment robuste pour s'étendre au  $S\pi$ -calcul. En particulier, la bisimulation a des bonnes propriétés de congruence et peut être caractérisée comme une bisimulation contextuelle.

### 3.1.3 Réactivité

Le contrôle de la *complexité calculatoire* des programmes est un aspect important de la sécurité informatique dans le cadre de systèmes repartis, embarqués, ouverts, mobiles, etc. Une approche *dynamique* à ce problème consiste à contrôler au moment de l'exécution la consommation de ressources et à soulever une exception si une certaine borne est atteinte. Une approche toute différente consiste à analyser *statiquement* le programme pour garantir que pendant l'exécution

il respectera certaines bornes. Les analyses statiques offrent les problèmes les plus intéressants, et peuvent être couplées aux méthodes dynamiques, par exemple en réduisant le nombre de vérifications dynamiques nécessaires.

Lorsque l'on considère le problème du contrôle de ressources, il y a une variété de propriétés d'un programme qu'on peut vérifier. La *termination* est probablement la première à laquelle on pense. Cependant, dans le contexte de *programmes interactifs*, il convient d'étudier une propriété plus forte qu'on appelle *réactivité*. La propriété de réactivité peut être définie (co-inductivement) comme le plus grand ensemble  $R$  de programmes qui terminent et tels que chaque interaction avec l'environnement mène à un programme qui est encore dans l'ensemble  $R$ . Si un programme manipule des données de *taille variable* comme des listes, ou des arbres, l'analyse peut encore préciser la notion de réactivité et, par exemple, établir que le programme réagit en utilisant une quantité *raisonnable* de ressources. On parle alors de *réactivité efficace*.

Le problème de trouver des critères pour assurer la terminaison de processus du  $\pi$ -calcul a été déjà évoqué dans le sous-thème 3.1.1. Comme nous l'avons remarqué, les critères inspirés par la réalisabilité ne couvrent qu'un fragment "fonctionnel" du  $\pi$ -calcul. Le problème de trouver des méthodes de terminaison qui s'appliquent à des programmes concurrents assez généraux reste donc ouvert. Deng [Den05a] a montré comment adapter certaines techniques de terminaison adoptées dans les systèmes de réécriture de termes au problème de la terminaison de processus du  $\pi$ -calcul. Il a montré ensuite l'intérêt pratique de ces systèmes de typage sur des exemples comme la représentation du choix non déterministe et la mise en œuvre d'une table de symboles comme une chaîne de cellules.

On peut remarquer que la propriété de réactivité d'un programme concurrent asynchrone est souvent difficile à obtenir. La situation est différente dans un cadre *synchrone* où la réactivité (efficace) vise juste à assurer la terminaison de chaque instant de calcul (avec des ressources raisonnables). Dans ce contexte, on a pris comme point de départ les travaux sur le contrôle de ressources pour des programmes fonctionnels du premier ordre. En gros, ce contrôle se base sur une combinaison de techniques de terminaison classique (par exemple, l'ordre récursif sur les chemins) et de bornes sur la taille des données. Amadio et Dabrowsky [ADZ06, AD06, AD07] ont développé une méthode pour annoter les programmes ainsi que des techniques d'analyse statique qui permettent de garantir la réactivité efficace de programmes exprimés dans le  $S\pi$ -calcul (cf. section 3.1.2). On notera qu'il n'est pas possible de borner la taille des données et ensuite de démontrer la réactivité du système, car la taille des données va dépendre de la terminaison. La solution proposée passe par une stratification des signaux (la mémoire) en régions.

### 3.1.4 Reversibilité

Dans les systèmes distribués dits *transactionnels*, plusieurs processus se disputent l'accès à certaines ressources, par exemple des droits en écriture sur un fichier ou simplement la connexion à un site donné. Chaque participant a donc une vue assez claire de l'objectif qu'il souhaite atteindre, mais pour ce faire, il doit émerger un consensus correspondant aux règlements des conflits potentiels entre processus, aboutissant à la validation des transactions en cours ou à l'annulation d'une partie d'entre elles quand un consensus général ne peut être trouvé ; on parle alors de verrous (*deadlock*).

La présence de verrous impose donc au programmeur de résoudre des problèmes distincts : d'une part il s'agit de programmer chacun des processus participant à la transaction de sorte que leurs interactions puissent conduire à une solution du problème, d'autre part, il faut aussi s'assurer que le système ne reste pas bloqué dans un état où la requête ne peut plus aboutir. Il faut donc prévoir l'ensemble des verrous possibles induits par le système et programmer une façon d'en sortir, en vérifiant que toute action ayant mené au verrou est correctement annulée

(débit d'argent, pré-réservation...).

On peut donc distinguer deux étapes de programmation différentes : la première correspond à un mode de programmation *vers l'avant* qui est la plupart du temps intuitif car il met en jeu des interactions entre processus dans l'ordonnancement pensé par le programmeur. La deuxième étape est plus compliquée, car les verrous sont précisément causés par les ordonnancements imprévus et correspond à un mode de programmation *vers l'arrière*. Ce dernier est d'autant plus délicat que tout retour en arrière (*backtrack*) d'un calcul distribué se doit de préserver la cohérence des états du système. En d'autres termes, l'annulation d'une communication doit être précédée des annulations de toutes les actions qu'elle aurait pu causer. Une question naturelle est alors de savoir s'il est possible de définir de façon générique, une procédure permettant de revenir en arrière sur une partie d'un calcul distribué, tout en préservant la cohérence globale du système. Il serait ainsi possible de se contenter de codes *vers l'avant* décrivant les étapes nécessaires à l'élaboration d'une transaction concurrente sans avoir à expliciter les procédures d'annulation en cas de verrous.

Selon que l'on regarde une algèbre de processus comme un langage de spécification ou comme la base d'un langage de programmation concurrente, le développement d'un calcul de processus réversible présente divers avantages. Dans le cadre d'un langage de programmation cela revient, nous l'avons dit, à automatiser (et donc fiabiliser) la tâche d'implémentation correspondant aux phases d'annulation d'une transaction vouée à l'échec. Dans le cadre d'un langage de spécification on coupe du même coup une partie des preuves à effectuer pour certifier un programme puisque le code de celui-ci a été réduit au strict nécessaire et que la correction du retour arrière est déjà prouvée.

Danos [DKS07] a défini un CCS réversible (RCCS) qui permet de faire revenir en arrière les calculs sans affaiblir le degré de distribution de ces calculs et en préservant la cohérence des états du système. Le calcul est utilisé comme la base d'une méthode de programmation concurrente déclarative dans le sens esquissé ci-dessus [DK05]. Danos a aussi montré que la définition d'une telle méthode pour un langage donné se réduit à exhiber un système de factorisation sur la catégorie des calculs du langage [DKS07]. Nous verrons plus loin (section 3.3) que les interactions biologiques sont, pour la plupart, réversibles et fondamentalement concurrentes. Au demeurant, ces travaux sur les algèbres de processus réversibles ont été inspirés par une volonté de modéliser de manière élégante les mécanismes de signalisation cellulaire [DK07]. Le travail de modélisation de systèmes biologiques, qui s'appuie à la fois sur ces fondements en théorie de la concurrence et sur leurs aspects probabilistes que nous allons aborder maintenant (3.2), est donc susceptible de tirer profit de ce travail fondamental sur la réversibilité en concurrence.

### 3.2 Systèmes Probabilistes

Le thème concurrence a également vu se développer une série de travaux autour des modèles probabilistes concurrents qui fournissent matière ici à notre second sous-thème. Les probabilités interviennent très naturellement dans les comportements concurrents, soit dans une perspective contrôlée de programmation où celles-ci servent à représenter une incertitude sur le contexte et les calculs afférents, soit dans une perspective de modélisation de systèmes naturels qui sont de manière inhérente incertains. Cette dernière source de probabilités nous rapproche d'ailleurs des aspects quantitatifs de modélisation biologique qui feront l'objet de notre dernier sous-thème.

Danos s'est surtout intéressé à recomprendre la notion d'équivalence probabiliste dans les termes usuels de la théorie de la mesure ; Deng s'est intéressé quant à lui aux théories axiomatiques et au typage ; enfin Varacca s'est intéressé tout particulièrement aux modèles causaux et à la notion de processus stochastique en temps partiel qui en découle. Le laboratoire vient de renforcer son engagement dans cette direction avec le recrutement d'Abbes, dont les travaux,

notamment sur les modèles probabilistes causaux, vont idéalement se situer en relation avec les travaux de Danos et Varacca.

Dans une publication qui conclut une série de travaux sur le sujet publiés par de nombreux intervenants, Danos replace la notion de bisimulation (ou d'équivalence) de systèmes probabilistes dans un cadre plus général fondé sur une utilisation systématique des notions catégorielles de quotient [DDL06]. Cette approche se combine naturellement avec la théorie de la mesure, ce qui est nécessaire pour pouvoir traiter des concepts probabilistes avancés. Elle introduit également une ouverture intéressante vers une dualité de Stone que Danos a commencé de poursuivre plus avant en même temps que la notion de bisimulation “presque sûre”. Cette reformulation a permis également d’approcher la question, importante d’un point de vue pratique, de l’approximation des systèmes probabilistes, au travers de la notion centrale d’espérance conditionnelle [DDP03]. Une autre notion d’approximation peut être construite en empruntant à la théorie des préférences et en manipulant des notions de probabilités faiblement additives [DD03b, DDP04]. Ce dernier développement se prolonge naturellement dans un traitement de type topologique où interviennent des structures de treillis continus familières en théorie des domaines, comme les espaces core-compact (ou exponentiables). Par ailleurs toujours dans ce cadre très général, on peut construire une extension de la logique temporelle probabiliste qui caractérise l’équivalence de processus en introduisant un opérateur de plus grand point fixe [DD03a] et en retrouvant ainsi une partie du  $\mu$ -calcul probabiliste.

Deng, durant son travail de thèse [Den05a] s’est également occupé de modèles probabilistes. En particulier il s’est intéressé à des techniques algébriques pour l’étude du comportement des processus probabilistes. Il a étudié un calcul de processus qui combine probabilités et nondéterminisme [Den07a, Den05b, Den05c], et dont la sémantique est basée sur les automates probabilistes de Segala et Lynch, un des modèles les plus utilisés dans la littérature. Les travaux de Deng ont consisté à donner des axiomatisations finies pour des équivalences (fortes ou faibles) définies en utilisant des notions de bisimulation. Ces travaux sont les premiers à étudier une axiomatisation des équivalences probabilistes en présence de récursion.

Deng a aussi étudié une sémantique de processus probabilistes en termes d’espaces métriques en proposant une notion d’équivalence approchée [Den06]. La mesure de cette approximation est formalisée par une *pseudo-métrique*, où deux processus sont à distance nulle si et seulement si ils sont équivalents.

Finalement, Deng a travaillé à une notion d’anonymat probabiliste pour les protocoles de sécurité [Den07b] formulée en termes de probabilités conditionnelles. Cette idée a été appliquée avec succès à un des exemples les plus populaires dans la théorie de protocoles : le dîner des cryptographes. Et par la suite les collaborateurs de Deng ont continué de raffiner cette notion d’anonymat qui a donné lieu à la publication de plusieurs articles.

Les travaux récents de Varacca en relation avec ce sous-thème ont porté sur un modèle de structures d’événements pour une version probabiliste du  $\pi$ -calcul linéairement typé [VY07]. Varacca [VY06] a donné un modèle du  $\pi$ -calcul linéairement typé basé sur des structures d’événements. Ce travail qui participe du sous-thème de la programmation concurrente, et déjà décrit plus haut, se prête très naturellement à être étendu d’une façon probabiliste. Or, le  $\pi$ -calcul linéairement typé possède la propriété d’être *confluent*, ce qui correspond, dans les structures d’événements à la propriété d’absence de conflit [VY06] ; il est donc naturel de se demander quelle propriété similaire le  $\pi$ -calcul probabiliste linéairement typé possède. Il se trouve que Varacca [VW06] a défini une notion de structure d’événements probabiliste qui permet de bien poser le problème. Une des motivations pour l’étude de ces structures d’événements est le fait qu’elles permettent de poser commodément la notion de *confluence probabiliste* qui est assez difficile à décrire dans les modèles plus communs. Informellement, un processus probabiliste est confluent si tout ordonnancement possible donne lieu à la même distribution de probabilités sur



l'ensemble de ses exécutions. Varacca [VY07] montre que le  $\pi$ -calcul linéairement typé possède cette propriété de confluence probabiliste.

### 3.3 Modélisation en Biologie Moléculaire

Partant du principe que la cellule se présente comme un système massivement distribué et probabiliste, il était tentant de mobiliser les différents types d'expertise en concurrence rassemblés au sein du laboratoire et de tâcher de dégager des fondements sémantiques et syntaxiques clairs pour ce travail de modélisation. Cette démarche semble d'autant plus pertinente que la biologie moléculaire moderne bute sur un problème inattendu et difficile de représentation. Beaucoup des systèmes étudiés manifestent une combinatoire qui semble mettre en échec les approches usuelles. On y a affaire à des systèmes dans lesquels les différents agents peuvent se lier et se modifier d'une infinité de manières non isomorphes durant l'évolution d'un système. C'est particulièrement vrai des voies de signalisation cellulaire qui sont très combinatoires et des interactions dynamiques entre membranes.

#### 3.3.1 Fondements de la modélisation biologique

Une première direction de recherches réfléchit à la portée des mécanismes moléculaires en termes d'auto-organisation, et s'attache à comprendre quels problèmes de ce type sont solubles, et comment raisonner sur ces systèmes - ce sont là les travaux de Tarissan et Danos sur l'auto-assemblage, et le récent résultat de Curien, Danos, Jean Krivine et Zhang sur l'universalité des interactions binaires pour l'assemblage de graphes [CDKZ07] qui complète un travail plus ancien [DL04]. Il ne s'agit pas ici de comprendre des systèmes biologiques réels, ni même de fournir des outils pour ce faire, mais de comprendre le cadre calculatoire dans lequel on peut abstraire ces systèmes. (Un tel effort pourrait d'ailleurs à terme trouver des applications du côté de l'ingénierie directe des systèmes biologiques.) De ce point de vue on est encore proche des problèmes de programmation du premier de nos sous-thèmes (section 3.1.4), quoique les briques de programmation diffèrent sensiblement.

Ainsi la question de l'émergence d'une forme, d'une structure de connexion abstraite, ou plus généralement d'un phénomène collectif à partir de multiples interactions locales entre composants élémentaires est fondamentale. Danos et Tarissan ont montré comment synthétiser à partir d'une spécification d'assemblage exprimée sous forme de réécriture d'arbres [DKT07] ou de graphes [DT05, DT06] un système qui réalise celle-ci avec une granularité naturelle où chaque nœud est un agent. Dans les deux cas, les impasses inhérentes au traitement décentralisé des assemblages sont contournées au niveau de la sémantique du langage de description [DKT07] en s'appuyant sur les idées d'algèbres de processus réversibles exposées plus haut.

Tarissan a par ailleurs développé une variante hiérarchique du langage de modélisation  $\kappa$ , qui introduit une notion limitée d'interactions entre membranes [LT07]. Ce langage décrit une biologie moléculaire simplifiée, qualitative, mêlant les interactions entre protéines de  $\kappa$  (complexations, modifications post-traductionnelles, ...) avec des activités membranaires de fusions et transports. L'outil de bisimulation, traditionnellement associé aux algèbres de processus, est alors adapté dans le but d'approcher une idée d'équivalence observationnelle en biologie. Danos a par ailleurs redéfini une version orientée du calcul de membranes de Cardelli, qui est plus élégante formellement et plus proche du jeu des interactions réelles de membranes biologiques [DP05].

Ces différentes tentatives pour circonscrire un modèle de calcul pertinent pour la modélisation proprement dite permettent une approche méthodique des problèmes concrets de modélisation qui forment notre seconde direction de recherche.

### 3.3.2 Modélisation concurrente de la signalisation cellulaire

Cette autre direction de recherche s’engage donc plus franchement dans la modélisation quantitative de la signalisation et ambitionne de fournir un environnement pour le développement de tels modèles.

Danos, en compagnie de nombreux collaborateurs extérieurs au laboratoire, a développé lors d’une année passée à la Harvard Medical School une implémentation du langage  $\kappa$ , dont il a été fait mention plus haut. Celui-ci est un système de réécriture de graphes à sites dont le but principal est la modélisation des voies de signalisation cellulaire [DL04]. Une partie du travail a consisté en la confection d’un algorithme de simulation stochastique qui permette d’exécuter des systèmes dont le nombre de combinaisons obtenues par liaison et modification est virtuellement infini. Ce travail algorithmique innovant permet de repousser les limites usuelles des systèmes dynamiques, et d’aborder des zones irréprésentables dans les démarches traditionnelles où le nombre total d’agents présents dans le système est négligeable devant le nombre de combinaisons dans lesquelles ils peuvent rentrer. Des outils qui exploitent la causalité inhérente aux systèmes d’agents (ou de processus) sont en cours d’élaboration et devraient permettre aux usagers de poser des questions relatives aux mécanismes causaux qui permettent de déclencher une observable donnée, et auxquelles la simple simulation numérique ne répond pas directement. Dans un cadre simplifié où la dynamique considérée est booléenne, Danos avait d’ailleurs montré comment formuler des requêtes biologiquement intéressantes en logique temporelle sur l’accessibilité absolue et conditionnelle de certains états de réseaux de signalisation [CCD<sup>+</sup>04].

Harmer a pour sa part construit un ensemble de  $\kappa$ -règles pour la voie de signalisation EGF qui inclut ERK, et Akt, et la famille complète des récepteurs EGF (ErbB1–4). Ce modèle a été soumis à une analyse causale pour en déduire des “points critiques” où les taux de réactions devraient modifier le comportement qualitatif du système. Il a été notamment possible d’obtenir des régimes oscillants pour la cascade RAF/MEK/ERK que l’expérience confirme dans certaines lignées cellulaires [DFP<sup>+</sup>07].

## 3.4 Perspectives

Le sous-thème de programmation concurrente comporte de nombreuses similitudes entre, par exemple, les réseaux de preuve de la logique linéaire, les réseaux d’interaction (différentiels), la sémantique des jeux, les structures d’événements et certains fragments du  $\pi$ -calcul. Un objectif général, qui implique plusieurs chercheurs et thèmes du laboratoire, est donc d’arriver à une synthèse de ces points de vue, et d’entretenir dans la mesure du possible les interactions entre la partie logique du laboratoire et la théorie de la programmation concurrente dont les travaux de Beffara mentionnés plus haut donnent un exemple saisissant.

Dans le contexte du projet ANR ParSec, Amadio et Dogguy vont poursuivre l’étude du  $S\pi$ -calcul et d’un certain nombre de problèmes connexes comme la *réactivité efficace*, le *déterminisme* et l’*analyse du flot d’information*. Les analyses statiques développées seront testées dans le cadre d’un langage de programmation inspiré par le  $S\pi$ -calcul qui a été implémenté à l’INRIA Sophia-Antipolis par Boussinot. Remarquons que des propriétés similaires sont étudiées dans le cadre de la programmation concurrente *asynchrone*. Un objectif général est donc d’arriver à une unification des concepts et des méthodes employées dans les modèles asynchrones et synchrones (les résultats évoqués en section 3.1.2 sur la bisimulation constituent un premier pas dans cette direction).

Varacca envisage de fournir une sémantique du calcul réversible RCCS basée sur les structures d’événements qui devraient être particulièrement appropriées puisqu’elles représentent naturellement la causalité. On notera que cette direction de recherche engage directement des enjeux de vérification de programme puisque Jean Krivine a construit une notion de structure d’évène-

ments récursive dans laquelle les critères de convergence relatifs à la technique de programmation réversible développée avec Danos peuvent être calculés effectivement.

Du côté des systèmes probabilistes, Varacca et sans doute Abbes continueront d'étudier les extensions probabilistes des structures d'événements. Une question intéressante à la marge des deux sous-thèmes est de savoir s'il est possible de donner une version probabiliste des sémantiques réversibles de calculs de processus, et si les critères de convergence simplement qualitative ont également un pendant quantitatif.

Danos projette pour sa part de solidifier l'algorithmique de réécriture stochastique mentionnée ci-dessus en la plaçant dans un cadre de grammaire plus générale, et en exprimant la sémantique stochastique en termes d'algèbres d'opérateurs suivant les travaux récents de Mjølness [MY07].

Un autre projet, déjà en place, est celui de donner une théorie des chaînes de Markov calculables en suivant la méthodologie de la théorie des domaines continus à la manière des travaux d'Escardo sur l'intégration sémantique des réels dans les langages fonctionnels.

Quant à la partie de ce thème qui tourne autour de la modélisation biologique, elle est évidemment grosse de perspectives d'applications. Tout d'abord, le travail de modélisation combinatoire réalisé en collaboration avec des biologistes expérimentaux, rapidement présenté ci-dessus, valide l'idée d'une encyclopédie de la signalisation écrite en  $\kappa$ . Harmer entend poursuivre le développement de ce modèle en lui adjoignant d'autres familles de récepteurs (IGF-1, IR) et de voies de signalisation couplées (mTOR, Wnt, ILx, ...). En termes de modélisation, le défi est ici de prédire correctement la variété de comportement observée en fonction des combinaisons de signaux entrants, et de la lignée cellulaire. Certaines sources expérimentales récentes fournissent la quantité de données susceptibles de valider une telle approche de modélisation à grande échelle. Cette tâche encyclopédique a le double intérêt de documenter de manière précise la connaissance biologique dans ce secteur particulier (dans lequel les enjeux biomédicaux sont considérables), et de permettre la construction de modèles sans solution de discontinuité.

Une autre et nécessaire pièce à ajouter à l'édifice est la construction d'un langage de membranes qui aille au-delà du bio- $\kappa$  proposé par Tarissan, en incluant les phénomènes de fission d'une part, et en prenant en charge les aspects stochastiques. L'idée est d'utiliser le langage des bigraphes de Milner comme une matrice syntaxique, dont la théorie sous-jacente est fournie et minutieuse, pour construire cette extension. Une telle inclusion demande également de se réapproprier algorithmiquement l'indépendance du simulateur en la taille du système à simuler (et en le nombre de combinaisons). Par ailleurs l'hyper-combinatorialité de la signalisation est un mystère en soi ; si les combinaisons possibles sont si nombreuses on aimerait comprendre où est l'information, et donc comprendre comment comprimer ces représentations dynamiques.

## 4 Outils Logiciels

**Participants.** [Pietro Abate](#), [Vincent Balat](#), [Jaap Boender](#), [Giuseppe Castagna](#) (depuis 2006), [Emmanuel Chailloux](#) (jusqu'à 2006), [Juliusz Chroboczek](#), [Roberto Di Cosmo](#), [Elena Giachino](#), [Grégoire Henry](#), [Samuel Hym](#), [Yves Legrand Gérard](#), [Pierre Letouzey](#) (depuis 2005), [Zheng Li](#), [Fabio Mancinelli](#), [Alexandre Miquel](#), [Raphaël Montelatici](#), [Ralf Treinen](#) (depuis 2007), [Jérôme Vouillon](#).

Patterson, Snyder et Ullman ont senti la nécessité d'expliquer, dans un court memo sur les critères d'évaluation pour l'évolution des carrières en Informatique [PSU99], ce qui caractérise le travail d'un chercheur en informatique qui écrit du logiciel, par rapport à l'activité d'un développeur dans l'industrie du logiciel, et par rapport à d'autres disciplines qui peuvent être

très grandes consommatrices ou productrices de logiciels, comme les mathématiques appliquées ou la physique.

Pour résumer en quelques lignes leur pensée, le simple fait d'écrire un nouveau logiciel, ou de concevoir un nouvel algorithme, n'est pas en soi une contribution à l'Informatique en tant que science : pour qu'il s'agisse d'une contribution scientifique, il faut que ce logiciel, ou cet algorithme, soit *mieux* que ce qu'on savait faire avant ; il doit y avoir un progrès clairement évaluable par rapport à l'état de l'art, et il faut en plus pouvoir mesurer l'*impact* que cette amélioration a sur la communauté.

Le philosophe napolitain Vico disait il y a plus de quatre siècles que "*connaître, c'est savoir faire*"; pour pouvoir améliorer véritablement l'état de l'art dans le domaine du logiciel, il faut connaître le logiciel, et il n'y a pas de vraie connaissance du logiciel sans pratique du logiciel : dans la réalisation d'un vrai système, il y a beaucoup de détails, importants et essentiels, qui peuvent faire la différence par rapport à l'état de l'art.

C'est là la raison qui explique pourquoi dans un laboratoire aux connotations fortement théoriques comme PPS on retrouve autant de développement logiciel : bon nombre de chercheurs du laboratoire travaillent sur les langages de programmation, leur sémantique, et la preuve de propriétés, et ressentent le besoin de valider leurs contributions et de mieux comprendre les problèmes posés par le développement de certains logiciels.

Si on regarde les contributions logicielles du laboratoire du point de vue de la conception des logiciels, on retrouve des idées fondatrices qui nous viennent des premiers langages issus de la logique : c'est la mise en œuvre de ces idées qui donne à l'ensemble de nos développements sa cohérence :

**Abstraction.** On veut écrire des programmes plus concis ; bien des logiciels développés à PPS naissent de l'identification de notions abstraites qui permettent de fournir à un programmeur des outils de très haut niveau dont la correction est prouvable séparément une fois pour toutes : c'est le cas des threads coopératifs à l'œuvre dans Lwt et CPC, des combinateurs de parallélisme propres à `OcamlP31`, des schémas de programmation Web de `Ocsigen` ; dans certains cas, comme pour CPC, la concision de la programmation est accompagnée de gains très significatifs en performances.

**Sûreté de la programmation.** Nous nous inscrivons dans la tradition ML, et attachons beaucoup d'importance aux garanties statiques sur la sûreté des logiciels (une fois le logiciel compilé et déployé, on a l'assurance formelle qu'à l'exécution certaines erreurs comme les erreurs de type ne se produiront pas). C'est le cas de CDuce, XDuce et des travaux sur la désérialisation typée, et de l'ensemble des travaux sur l'interopérabilité sûre entre modèles de programmation différents, comme `O'Jacare` et `Ocaml`. Dans cette même ligne, on trouve les travaux récents sur les propriétés de cohérence de grandes masses de composants logiciels (comme celles que l'on retrouve dans les dizaines de milliers de paquets d'une distribution Linux moderne), dans le cadre des projets européens EDOS et Mancoosi, et aussi des travaux préliminaires sur le routage.

**Vérification du logiciel.** Ecrire des logiciels qui ne font pas d'erreurs de type est une chose, prouver qu'ils font ce qu'on veut en est une autre. Dans notre laboratoire, on contribue à de nombreux projets qui visent à rendre accessible la preuve formelle : la compilation certifiée, les expériences de preuves de vrais algorithmes utilisés par des larges masses d'utilisateurs (`Unison`), ou alors l'effort d'obtenir une équivalence entre différentes sémantiques d'exécution comme dans le cas d'`OcamlP31`.

Pour la partie de notre travail qui touche au logiciel et à sa programmation, notre ambition est d'avoir un impact durable sur le futur de la programmation, car nous aspirons à poser les bases pour un développement logiciel plus abstrait, plus rapide, plus sûr, voir certifié.

Les membres du laboratoire qui participent à ces activités ont mis en place un *Groupe de travail programmation*, qui se réunit bi-mensuellement après le séminaire du laboratoire. Dans ce cadre, nous avons des échanges fréquents avec des acteurs industriels. Une telle interaction a permis que certains de ces travaux aient déjà en ce moment un impact non négligeable sur le monde industriel.

Le fil conducteur de notre activité est la programmation avancée avec des solides bases théoriques, avec des applications très variées, et nous avons choisi de classer nos contributions selon les trois grands axes suivants : programmation avancée pour le Web, Internet et les réseaux, interopérabilité entre langages et gestion de grandes masses de composants logiciels, certification et preuves.

## 4.1 Programmation avancée pour le Web, Internet et les réseaux

L'explosion de l'usage du Web et la sophistication croissante des applications construites en utilisant le couple navigateur/serveur (et plus généralement les protocoles de base tels HTTP) comme plateforme de programmation ont engendré un foisonnement de nouveaux langages de programmation et de méthodes de construction d'applications distribuées (services Web, AJAX, etc.) qui souffrent d'un manque cruel de bases formelles : on trouve des langages à la sémantique imparfaite (Javascript), des insuffisances dans les langages pour la programmation parallèle et la programmation concurrente (threads et processus), des briques de base de l'infrastructure du Web largement perfectibles (proxy Web, mécanismes de routage dans le réseau), des approches insatisfaisantes à la composition de services, et bien d'autres.

Notre laboratoire a mis son expérience et connaissance des fondements de la programmation à contribution pour relever certains de ces défis, avec un fort degré d'interpénétration entre nos développements : par exemple, le projet *Ocsigen* réutilise en son sein à la fois le travail fait sur les threads dans *Lwt*, et les fonctionnalités avancées de manipulation typée de données XML présent dans *CDuce*, qui sont aussi reprises dans l'outil *ceve* de la suite EDOS, et un des outils d'EDOS est en cours de portage sur *Ocsigen*.

### 4.1.1 Programmation avancée du Web

Dans la programmation sur le Web, comme dans la programmation traditionnelle, la contribution majeure du laboratoire tourne autour du typage et de l'abstraction : ce sont les thèmes porteurs des deux projets logiciels d'envergure décrits dans la suite.

**XDuce, CDuce : manipulation sûre de données XML.** Vouillon s'intéresse à la conception d'un langage spécialisé pour le Web. Le point de départ de cet axe de recherche est le langage XDuce<sup>3</sup> sur lequel il a travaillé durant son postdoctorat. Ce langage utilise des expressions rationnelles d'arbres pour typer les données XML. L'exploration et la décomposition de ces données se fait à l'aide d'une généralisation du filtrage par motifs de ML à des motifs qui sont également essentiellement des expressions rationnelles d'arbres. Le langage reste à l'état de prototype. Afin de le rendre réellement utilisable, il s'avère nécessaire de l'enrichir avec du polymorphisme (la possibilité de définir des fonctions génériques qui opèrent de façon uniforme sur toute une classe de données, par exemple, sur des listes) et des fonctions de première classe (que l'on peut passer en arguments d'autres fonctions). Vouillon a proposé une telle extension [Vou06a, Vou06b], et a commencé à l'implémenter.

---

<sup>3</sup><http://xduce.sourceforge.net/>

Le même langage XDuce est le point de départ d'un autre projet, le langage CDuce<sup>4</sup> [Cas07], dont l'un des responsables (Castagna) a récemment rejoint le PPS. CDuce étend XDuce en plusieurs directions que l'on retrouve dans d'autres composantes de ce thème. Parmi celles-ci on peut citer l'interopérabilité avec d'autres langages (un programme CDuce peut importer toute bibliothèque Objective Caml (OCaml) et être exporté comme une librairie OCaml) et une utilisation poussée des types qui, au delà du simple contrôle statique de correction, sont mis à contribution pour une implantation efficace du filtrage, pour guider le design de nouveaux paradigmes de programmation et pour piloter des interfaces graphiques de programmation. CDuce dépasse le stade de simple prototype qui caractérise XDuce : il compte parmi sa communauté d'utilisateurs plusieurs jeunes pousses européennes et projets collaboratifs open-source ; il est inclus dans les principales distributions Linux (Debian, Ubuntu, Xandros, dans contrib pour Mandriva et "under review" pour Fedora) ; il en existe des versions Mac OSX, et FreeBSD et une version Windows est en cours de préparation.

**Ocsigen : conception évoluée de sites Web.** Balat a lancé fin 2004 un projet visant à expérimenter et populariser de nouvelles techniques de programmation Web basées sur la programmation fonctionnelle et le typage fort.

La principale réalisation du projet est le serveur Web **Ocsigen**. Écrit en OCaml, il repose entièrement sur le modèle de threads coopératifs monadiques **Lwt** de Vouillon. Il implémente toutes les fonctionnalités utiles pour permettre son utilisation en lieu et place d'un autre serveur Web dans de nombreuses configurations (envoi de fichiers statiques, scripts CGI...), avec de très bonnes performances.

Mais la principale nouveauté est un module de création de sites Web dynamiques en OCaml, basé sur le concept de *continuation*, bien connu en programmation fonctionnelle, et qui permet de matérialiser finement le comportement d'une application Web, en particulier l'interaction avec un navigateur [Bal06]. Cette idée, introduite en 1999 par Queinnec, est en train de faire naître un véritable domaine de recherche, comme en témoigne la création récente de plusieurs projets de recherche similaires, comme *Links*, à Edimburgh (Wadler *et al*), *Hop*, développé à l'INRIA Sophia-Antipolis (Serrano *et al*), ou *WebSiCoLa*, à l'université Paris 13 (LIPN, Loddo *et al*).

**Ocsigen** se distingue de ces projets par plusieurs aspects. Notamment, un des choix fondamentaux a été d'utiliser OCaml plutôt qu'un langage spécialement créé, ce qui permet de faire bénéficier les développeurs de la richesse expressive du langage et donc de dépasser rapidement le stade de prototype. L'utilisation de concepts évolués du langage a permis de développer une vision inédite de l'interaction Web, proposant une façon de capturer de manière très fine les principes de la programmation Web (sessions, données persistantes, différents types de paramètres...). De l'utilisation de concepts de haut niveau adaptés au domaine résulte un style de programmation très concis, même pour des sites complexes, et une sécurité accrue grâce à une utilisation poussée du système de types (gestion des sessions automatique, absence de liens cassés,...). Enfin, ce module offre aux développeurs plusieurs façons de valider statiquement les pages xhtml générées, soit grâce au système de types d'OCaml soit en utilisant **OcamlDuce**. Un module pour CDuce est en cours de réalisation (Varoquaux, Abate).

#### 4.1.2 Programmation avancée pour Internet

Nous avons réalisé plusieurs contributions à la programmation concurrente et distribuée qui est devenue incontournable aujourd'hui. Nous avons contribué à permettre de mieux utiliser les threads (**Lwt**, **CPC**), ainsi que la programmation parallèle (**OcamlP31**, **ocf**) et distribuée

---

<sup>4</sup><http://www.cduce.org>

(HirondML, CubeVM), avec à chaque fois des contributions significatives par rapport à l'état de l'art.

**Lwt.** Cette bibliothèque de threads coopératifs pour OCaml est développée par Vouillon. Avec cette bibliothèque, les différents fils d'exécution sont encapsulés en utilisant une monade. L'un de ses principaux avantages est la portabilité, car elle ne dépend pas de l'existence de threads systèmes. Il est également plus facile de programmer avec des threads coopératifs plutôt que préemptifs car les situations de compétition ("race condition") sont limitées.

**CPC.** Les *threads* sont une abstraction commode pour la programmation concurrente. Comme ils sont chers (surtout en mémoire mais aussi en temps d'exécution), beaucoup de programmeurs décident d'utiliser plutôt la technique de plus bas niveau de la *programmation par événements*.

Il est naturel de se demander si un programme concurrent à *threads* peut être traduit automatiquement en un programme en événements équivalent. Il s'avère que c'est le cas, et que la traduction consiste simplement en une conversion en style de passage de continuations (CPS).

CPC est un programme qui effectue une traduction source/source depuis un programme à threads coopératifs écrit en un C augmenté de primitives de concurrence (*yield*, *spawn*, etc.) en C à événements. CPC effectue une suite de 7 transformations, dont la toute dernière est une transformation CPS.

Les *micro-benchmarks* effectués indiquent que les primitives de concurrence implémentées par CPC sont entre 5 et 100 fois plus rapides que les mêmes primitives implémentées par des systèmes de *threads* classiques. Les *macro-benchmarks* indiquent qu'un serveur Web écrit en CPC est 3 fois plus rapide que les serveurs Web libres les plus rapides.

**Programmation parallèle par squelettes.** Le projet OcamlP31 ([www.ocamlp31.org](http://www.ocamlp31.org)) vise la réalisation d'une librairie de squelettes de parallélisation fonctionnels pour l'exécution de programmes sur une machine multiprocesseurs ou sur un ensemble de machines à travers un réseau. Les squelettes sont des "combinateurs de parallélisme" qui permettent au programmeur d'exprimer le parallélisme exploitable dans ses programmes sans avoir à se soucier des aspects de synchronisation qu'implique la programmation du parallélisme.

En changeant simplement les librairies avec lesquelles est lié le code source de l'utilisateur, on peut donner des sémantiques différentes aux combinateurs de parallélisme, de telle sorte qu'il devient possible de déboguer un programme parallèle dans une sémantique séquentielle, sans modifier le code source, et avec une garantie, sous certaines conditions, que le code exécuté en parallèle et le code exécuté en séquentiel réaliseront exactement les mêmes calculs.

On peut ainsi vérifier une fois pour toutes l'adéquation entre les différentes sémantiques en travaillant sur les librairies, et non pas sur les programmes écrits par l'utilisateur.

OcamlP31 a fait preuve d'une grande vitalité : le projet Estime de l'INRIA Rocquencourt l'utilise pour la coordination de solveurs numériques préexistants [CMV<sup>+</sup>06], et cette utilisation a été le point de départ pour l'ARC MOPROSCO entre le laboratoire PPS et l'INRIA, dans le cadre de laquelle l'évolution de OcamlP31 a été poursuivie entre 2004 et 2006, notamment avec la thèse de Li, en cotutelle avec Susanna Pelagatti (Université de Pisa) et en collaboration avec Marco Danelutto (Université de Pisa).

Entre 2003 et 2007, OcamlP31 est passé à travers deux versions majeures, et a été profondément réécrit par Li, qui a réduit considérablement la distance entre les librairies séquentielles et parallèles, ce qui facilite grandement le travail de preuve de leur équivalence.

En même temps, Di Cosmo, Pelagatti et Li ont étudié la compilation efficace d'opérations "data-parallèles" sophistiquées [DCP03, DCLP07].

**Data-parallélisme : ocf.** Une première extension data-parallèle du langage Caml-Light, nommée Caml-Flight, a été réalisée en 1995. Caml-Flight est une extension parallèle de type SPMD du langage Caml-Light. Cette extension a nécessité une modification de la machine abstraite du langage Caml et l'écriture d'un protocole de communication. OCaml a évolué depuis en offrant un mécanisme de processus légers, un modèle de persistance ainsi que l'outil `camlp4` pour les extensions de syntaxe du langage. Ces outils ont facilité la construction d'une nouvelle implantation appelée Objective Caml-Flight ou `ocf`<sup>5</sup> [CF03a]. Celle-ci a été réalisée dans le cadre de l'ACI GRID CARAML<sup>6</sup> (Coordination des Applications Multiprocesseurs en objective camL).

**Migration de calculs : HirondML.** Une des principales difficultés dans la migration de calculs provient de la synchronisation en cas de partage de données dans l'environnement de calcul de la destination. Le système HirondML<sup>7</sup> [CV05] réalisé par Julien Verlaquet et Chailloux utilise des *fair threads* coopératifs à la Boussinot, implantés en OCaml, pour réaliser la synchronisation quand un calcul arrive à destination. L'environnement local est copié lors de la migration, mais l'environnement global (contenant les mêmes déclarations) est spécifique à chaque instance de programme. Ce choix permet d'obtenir des performances raisonnables, tout en étant plus facilement contrôlables par le programmeur.

**La CubeVM.** Peschanski et Hym ont développé la CubeVM [PH06], une machine virtuelle exécutant une variante appliquée du  $\pi$ -calcul et disposant d'une sémantique opérationnelle précise pour les optimisations qu'elle comporte. La principale caractéristique de la CubeVM réside dans son absence de pile qui permet une bonne adéquation à l'exécution de systèmes hautement dynamiques et concurrents. L'absence de pile permet un modèle de gestion des ressources bien plus simple, en particulier pour la gestion mémoire : le ramasse-miettes peut détecter et récupérer la mémoire utilisée par de nombreux cycles de processus morts. La CubeVM s'écarte du  $\pi$ -calcul standard également pour introduire des "réactions en chaîne" au sein desquelles des processus peuvent contourner l'ordonnanceur pour optimiser les passages de contrôle : ces réactions permettent une exécution efficace de processus déterministes.

#### 4.1.3 Infrastructure du réseau Internet

A travers la participation de Legrandgerard à la normalisation de IPV6, notre laboratoire dispose d'une expertise non négligeable en réseaux, qui a été mise à contribution dans deux projets logiciels importants qui visent à améliorer les proxy Web et les algorithmes de routage dans l'Internet sans fil.

**Polipo.** Polipo est un proxy HTTP — un "cache Web" — : il est à la fois le proxy libre le plus rapide (il a été mesuré comme étant entre 10% et deux fois plus rapide que Squid sur des exemples réalistes<sup>8</sup>), et celui ayant le support le plus complet des parties avancées de HTTP/1.1, notamment des algorithmes de cohérence distribuée de cache et des améliorations au niveau du transport (Squid utilise encore HTTP/1.0, avec un petit nombre d'extensions venant de HTTP/1.1). Il est écrit dans le style à événements, avec des fortes analogies avec les concepts mis en œuvre dans le compilateur CPC décrit plus haut dans cette section.

Il s'agit d'un projet influent dans la communauté du logiciel libre, ainsi dans une certaine mesure que dans la communauté scientifique. Il a été inclus dans la plupart des distributions

---

<sup>5</sup><http://www.pps.jussieu.fr/~emmanuel/Public/Dev/>

<sup>6</sup><http://www.caraml.org>

<sup>7</sup><http://www.pps.jussieu.fr/~emmanuel/Public/Dev/HirondML/>

<sup>8</sup><http://www.pps.jussieu.fr/~jch/software/polipo/bench.html>



Linux et Unix BSD (notamment Debian GNU/Linux, Ubuntu/GNU Linux, RedHat Fedora Linux, FreeBSD, NetBSD, OpenBSD). Il est aussi une composante essentielle du *toolkit* de programmation d'applications Web *Dojo Online*, et l'expertise dans les arcanes du protocole HTTP/1.1 qui en découle s'est avérée très utile, notamment pour le projet *Ocsigen*.

**Babel.** La pratique de la programmation réseau mène naturellement à la volonté d'en savoir plus sur les couches basses — de mettre les mains dans le cambouis. Chroboczek s'est intéressé au *routage*, soit la recherche de chemins non pessimaux, qui constitue sans aucun doute la partie la moins satisfaisante à l'heure actuelle de l'architecture Internet.

Les protocoles de la famille *vecteur de distance* ou *Floyd-Warshall*, dont **Babel** fait partie, ont généralement des propriétés prouvables de convergence très fortes — en l'absence de mobilité, la convergence est assurée sous des hypothèses très faibles. Leurs propriétés dynamiques sont moins bonnes : après un événement de mobilité, les nœuds peuvent entrer dans un processus de déception mutuelle qui fait que la convergence peut demander jusqu'à  $n \times M$  itérations, où  $n$  est le nombre de nœuds et  $M$  la métrique maximale autorisée. Cela les rend normalement inapplicables aux réseaux sans fil, où la mobilité est la norme plutôt que l'exception.

**Babel**, développé par Chroboczek, raffine l'algorithme de Floyd-Warshall en ajoutant une *condition de faisabilité*, inspirée par le protocole EIGRP de Cisco, qui garantit l'absence de boucles de routage. Le temps de convergence de **Babel** est prouvablement proportionnel au diamètre du réseau ; le prix à payer pour ces propriétés est la taille des informations passées, qui est environ le double d'un protocole Floyd-Warshall classique tel que RIP.

Sous l'influence de Legrandgérard, **Babel** a été implémenté pour le protocole IPv6 au lieu du IPv4 habituel.

**IPv6.** Depuis 1996, Legrandgérard participe activement au G6 (dont il est trésorier), groupe français d'expérimentation IPv6, et a contribué à la mise en place d'un réseau IPv6 natif dans le cadre de Renater 3 dont la naissance date d'octobre 2002. En particulier, en collaboration avec Tuy, depuis lors en charge du projet IPv6 à Renater, Legrandgérard a procédé à la migration du PIR Ile de France vers ce nouveau réseau IPv6 natif (IPv6/ATM), tout en maintenant en service l'ancien réseau de test. Ceci a demandé d'établir un plan d'adressage IPv6, plan rendu nécessaire par l'allocation par RIPE des premiers préfixes de production IPv6, et dans ce cadre Legrandgérard s'est plus particulièrement intéressé au service DNSv6 (Domain Name Service avec extensions IPv6) et à la mise à jour dynamique et sécurisée du DNS (Dynamic Update) dans le cadre de l'auto-configuration IPv6. Cette implication s'est traduite aussi par une offre de services à la communauté académique et plus généralement à celle des utilisateurs français d'Internet. Parmi ces services, on citera l'hébergement à PPS du site Web du groupe G6, la participation à la rédaction d'un ouvrage consacré à IPv6 (IPv6, Théorie et pratique, Éditions O'Reilly), la fondation du groupe GN6, groupe des néophytes IPv6 animé par le GIP Renater dont l'une des principales activités est d'aider au transfert du savoir-faire des experts du groupe G6 vers les organismes de la communauté Renater, et enfin par la "v6sation" d'un certain nombre d'applications comme le module socket de Python pour la souche IPv6 de l'INRIA développée par Dupont.

## 4.2 Intéropérabilité entre langages et gestion de grandes masses de composants logiciels

Une conséquence de l'essor du Web, couplé avec l'explosion de l'usage des logiciels libres, est l'émergence de nouveaux besoins quand il s'agit de construire des systèmes logiciels complexes

à partir de briques préexistantes qui n'ont pas été forcément conçues pour êtreinteropérables : nous sommes bien rentrés dans l'ère du développement décentralisé et distribué de logiciels.

Les contributions du laboratoire à ces problématiques sont de deux natures : d'un côté, on a depuis longtemps travaillé sur l'interopérabilité entre composants écrits dans des langages de programmation différents ; de l'autre côté, à un autre niveau d'abstraction, nous avons commencé il y a quelques années, dans le cadre du projet européen EDOS, à étudier les problèmes de composabilité des paquetages logiciels dans les grandes distributions GNU/Linux (qui sont similaires aux problèmes qu'on peut rencontrer dans des gros logiciels orientés objets comme Eclipse).

#### 4.2.1 Intéropérabilité entre langages

Nous avons étudié les problèmes d'interopérabilité sous trois angles : l'intégration de données extérieures, l'interopérabilité de composants de différents langages, et le développement d'applications multi-langages.

**Désérialisation sûre.** Une caractéristique intéressante des langages statiquement typés est de ne pas avoir besoin de conserver d'informations de types pour les valeurs créées et manipulées pendant l'exécution d'un programme (celle-ci restant sûre). Pour les valeurs existant à l'extérieur d'une exécution, cette propriété n'est plus garantie et il devient nécessaire de vérifier dynamiquement qu'une valeur désérialisée reste compatible avec le type attendu dans le programme. Ce travail [HMC06] correspond au sujet de la thèse de

Henry<sup>9</sup>, débutée en octobre 2005 et co-encadrée par Chailloux et Michel Mauny (ENSTA). L'originalité de ce travail est de garantir une désérialisation sûre sans sérialiser les types, ce qui permettra de l'appliquer dans le cadre du chargement dynamique de modules dans les langages statiquement typés.

**Ocaml.** La nouvelle plate-forme .NET de la société Microsoft apporte un haut niveau d'interopérabilité entre composants développés dans différents langages, mais autorise aussi grâce aux initiatives "open source" (sscli, mono) d'être multi-plateformes. La compilation d'OCaml vers cette plate-forme a été initiée à INRIA. La thèse de Montelatici [Mon07] partant de ce travail a été soutenue en mars 2007. Elle a permis de vérifier l'intérêt d'utiliser la même bibliothèque d'exécution pour compiler des composants C# et Caml tout en conservant les bonnes propriétés de sûreté sans sacrifier pour autant l'efficacité comme le montrent les articles [MCP04, MCP05].

**O'Jacare.** Dans le but de construire des applications multi-langages, Henry et Chailloux ont défini un IDL (langage de définition d'interface) simple pour la description des classes Java. La communication d'OCaml vers Java est directe. La communication de Java vers OCaml passe par un mécanisme de rappel. L'outil de génération de code, appelé O'Jacare<sup>10</sup>, apporte l'automatisation des codes d'encapsulation [CH04]. Une première application a été d'implanter la visionneuse MLdvi de Miquel sur le patron de conception Modèle-Vue-Contrôleur où la partie algorithmique du Modèle est en OCaml et l'interaction avec l'utilisateur en Java. Ce travail a été adapté à la communication entre Ocaml et C# [CHM04b], qui possèdent une bibliothèque d'exécution commune (en .NET). L'article [CHM04a] montre l'évolution des besoins en interopérabilité des langages en l'illustrant par les réalisations effectuées en OCaml.

---

<sup>9</sup><http://www.pps.jussieu.fr/~henry/marshal/index.fr.html>

<sup>10</sup><http://www.pps.jussieu.fr/~henry/ojacare>

### 4.2.2 Analyse statique de grandes masses de paquetages logiciels

Les logiciels libres, qui sont développés de manière décentralisée, sont normalement fournis sous forme de paquetages (des unités d'installation qui peuvent être gérées de façon automatisée), dans des collections appelées *distributions*, qui visent à simplifier la tâche de l'utilisateur final qui souhaite ajouter ou retirer tel ou tel autre composant de son système.

Chaque paquetage contient des méta-données qui décrivent son contenu ainsi que ses relations avec les autres paquetages, qui prennent la forme de relations de dépendance et de conflit : un paquetage A peut nécessiter la présence d'une série d'autres paquetages dans le système pour être utilisable, et peut être incompatible avec certaines versions d'autres composantes.

Une distribution moderne contient plusieurs dizaines de milliers de paquetages qui forment un réseau d'inter-dépendances avec des centaines de milliers de relations. C'est un objet d'étude extrêmement intéressant, mais de façon surprenante, personne ne s'y était intéressé d'un point de vue strictement scientifique avant les efforts menés à l'initiative de notre laboratoire dans le cadre du projet européen EDOS (Environment for the development and Distribution of Open Source software).

Nous avons effectué une étude de l'état de l'art dans la construction et la gestion des distributions de logiciel libre, mis en évidence la difficulté théorique de la résolution de dépendances (que nous avons montrée NP-complète), fourni des solutions fondées sur des méthodes formelles correctes pour résoudre les problèmes qu'on retrouve dans la gestion des répertoires de paquetages afin de garantir leur cohérence, et développé des outils qui réalisent ces solutions [MBDC<sup>+</sup>06, DCBD<sup>+</sup>06, DCDL<sup>+</sup>06].

L'impact de ce travail est mesurable concrètement par le fait que les distributions Caixa Mágica et Mandriva ainsi que la communauté Debian ont intégré nos outils dans leur processus qualité ; pour Debian, on peut voir <http://edos.debian.net>, réalisé suite à l'intérêt suscité par la présentation faite par l'équipe EDOS à la réunion Debian tenue en Extremadura en décembre 2007.

Nous décrivons brièvement ici quelques uns des outils qui ont été développés pour valider les résultats théoriques et les appliquer en pratique. Ces outils représentent un avancement de l'état de l'art, puisqu'ils introduisent des méthodes d'analyse correctes et complètes par rapport à celles qui sont normalement utilisées dans les outils disponibles actuellement et qui sont basées sur des algorithmes empiriques qui souvent donnent des résultats incorrects.

**debcheck**, **rpmcheck**, écrits par Vouillon, vérifient la propriété d'installabilité pour chaque paquetage présent dans une distribution aux formats DEB ou RPM, via un encodage dans SAT, en utilisant un solveur SAT opportunément adapté à gérer ces types de formules ; l'analyse des plus de vingt-mille paquetages de la distribution Debian se fait en quelques minutes sur un ordinateur standard.

**pkglab**, écrit par Berke Durak et Boender, permet d'analyser et manipuler de façon symbolique les répertoires de paquetages, en utilisant un langage déclaratif qui incorpore comme opération de base le solveur de **debcheck/rpmcheck** et d'autres opérateurs algébriques spécialement adaptés à la manipulation d'ensemble de paquetages. Il est capable de gérer non seulement un répertoire, mais aussi l'évolution dans le temps des répertoires, et permet de répondre avec des programmes de quelques lignes à des questions extrêmement complexes sur l'évolution des répertoires de paquetages.

**ceve**, écrit par Boender, est un parseur capable de lire plusieurs formats de description des méta-données concernant les paquetages et de les transformer en plusieurs formats d'output, directement lisibles par d'autres outils de la suite EDOS. En particulier **ceve** est utilisé pour créer les bases de données nécessaires à **pkglab**, et pour générer toutes les informations

nécessaires à **tart** pour effectuer les calculs d'optimisation. Il utilise **OcamlDuce** pour garantir la bonne formation des sorties XML des données.

**tart**, écrit par Durak, implémente un algorithme de répartition d'un ensemble de paquets sur un ensemble de media de support, en optimisant la solution par rapport à des fonctions d'utilité et de coût associées aux paquets. La répartition des dizaines de milliers de paquets qui composent une distribution moderne (Mandriva ou Debian) se fait en quelques minutes sur un ordinateur standard.

**edoslib**, écrit par Mancinelli, est un ensemble de bibliothèques Java qui peuvent être utilisés pour analyser et manipuler des répertoires de paquets, et qui fournit une API pour construire des applications plus complexes, comme des clients visuels pour représenter le graphe des dépendances et pour y naviguer.

**Juppix.** Le laboratoire dispose d'une longue expérience concernant les aspects légaux du logiciel (libre et propriétaire), et avec le système de paquetage Debian, plusieurs membres du laboratoire étant contributeurs Debian. Cela nous a menés à concevoir notre propre distribution Linux. La distribution *Juppix*, faite par Lebresne et Chroboczek, est un *LiveCD* GNU/Linux basé sur Knoppix destiné aux étudiants de premier cycle d'informatique. Elle est distribuée à nos étudiants de première année depuis Septembre 2005.

### 4.3 Preuves

Le laboratoire a une activité dans le domaine de la certification et preuve de logiciels qui a pris de l'importance ces dernières années à la fois par la contribution au développement d'assistants à la preuve, et par la preuve formelle de logiciels ou d'algorithmes d'une complexité croissante.

Letouzey participe au développement logiciel du système Coq<sup>11</sup>. Ce logiciel est actuellement l'un des plus mûrs pour ce qui est de la formalisation de preuves de programmes et de preuves de résultats mathématiques. Il s'agit d'un logiciel libre, dont l'essentiel du développement est assuré par l'équipe Logical de l'INRIA Futurs.

Par ailleurs, Letouzey participe à l'ANR Compcert, dont le coordinateur est Leroy. L'objectif de ce projet est la réalisation de compilateurs certifiés, en particulier C vers assembleur et ensuite ML vers assembleur (cf. thème 2).

Vouillon a fait des contributions importantes, au cours de son séjour post-doctoral, au logiciel **Unison**<sup>12</sup>, un outil de synchronisation de fichiers entre deux ordinateurs développé principalement à l'université de Pennsylvanie, et dont le nombre d'utilisateurs est estimé à plusieurs dizaines de milliers. Vouillon continue à assurer la maintenance de ce logiciel et à l'enrichir de nouvelles fonctionnalités. Ainsi, il a implémenté la synchronisation des ressources associées aux fichiers sous Mac OS X. Plus récemment, il a complètement spécifié en Coq l'algorithme de synchronisation sous-jacent à **Unison**, et l'a prouvé correct.

Vouillon a aussi écrit une preuve en Coq de différentes propriétés du calcul  $F_{<}$ : (correction, transitivité du sous-typage) dans le cadre du défi **POPLmark**<sup>13</sup>. Ce défi vise à évaluer l'état de l'art en terme d'outils et de techniques de la preuve assistée par ordinateur de métathéories des langages de programmation.

---

<sup>11</sup>Voir <http://coq.inria.fr>

<sup>12</sup><http://www.cis.upenn.edu/~bcpierce/unison>

<sup>13</sup>[www.cis.upenn.edu/~plclub/mmm/](http://www.cis.upenn.edu/~plclub/mmm/)

## 4.4 Perspectives

### 4.4.1 Programmation avancée pour le Web

Nous prévoyons une croissance de l'activité du laboratoire autour des technologies pour le Web, lié à la dynamique créée par le projet **Ocsigen**, ainsi qu'à l'arrivée de Castagna, qui s'intéresse aux contrats pour les services Web.

**Services Web.** Une tendance forte dans l'industrie pour la conception de systèmes logiciels complexes est basée sur la notion de “services Web”, qui suggère des scénarii futurs d'utilisation dans lesquels un “Web-architecte” créera de nouveaux services par assemblage des services pré-existants disponibles sur le Web et qu'il aura découvert par une recherche interactive. Pour qu'une telle recherche et un tel assemblage soient possibles, il faut que le comportement des services soit spécifié de manière formelle par des “contrats”. Nous développons une théorie pour la spécification de services Web, qui vise à définir de manière formelle la compatibilité entre clients et services et la substituabilité et évolution de services [CGP07]. Ceci est un premier pas vers la définition des langages d'assemblage de composants qui permettront non seulement de décrire la coordination des différents services mais aussi de prouver formellement et statiquement des propriétés satisfaites par ces assemblages. Les différentes solutions seront expérimentées sur les plateformes que nous développons, notamment, **Ocsigen**, XDuce et CDuce. Le but à moyen terme est de pouvoir influencer la définition de standards pour la spécification et la coordination de services Web.

**Évolution d'Ocsigen.** La version actuelle d'**Ocsigen** est désormais prête pour le développement d'applications Web dynamiques “côté serveur”. Balat, Vouillon, Fabien Benureau et Till Varoquaux travaillent actuellement en collaboration avec Chailloux à la création de techniques de programmation distribuée entre un serveur Web et un navigateur. Cette question, fortement dépendante des technologies utilisées par les navigateurs, pose des problèmes très intéressants, notamment de compilation, typage, synchronisation et gestion du contrôle (exceptions) et de la mémoire (garbage-collection). L'utilisation de ce travail dans **Ocsigen** permettrait d'avoir pour la première fois une façon de programmer des sites fortement dynamiques (style “Web 2.0”) entièrement en OCaml, en spécifiant les portions de code qui doivent s'exécuter côté client.

### 4.4.2 Interopérabilité, composants logiciels

Le départ de Chailloux, nommé professeur à Paris 6, induira une réduction certaine de l'activité autour de l'interopérabilité entre langages, qui restera cependant présente dans les activités menées dans le cadre du projet **Ocsigen**, en particulier pour ce qui concerne la génération dynamique de Javascript.

En même temps, l'arrivée de Treinen contribuera à la montée en puissance de l'activité sur la gestion de grandes masses de composants logiciels. En particulier, Treinen est responsable de la composante (workpackage) *International competition of constraint solvers for upgradeability problems* du nouveau projet européen Mancoosi, dont Di Cosmo est coordinateur, et qui prend la suite du projet EDOS, terminé en juin 2007.

## 5 Synergies et collaborations

Le thème 1 présente des interactions constantes avec les thèmes 2 et 3. Cette synergie se manifeste par la vivacité des groupes de travail *Sémantique*, qui est commun aux thèmes 1 et 2, et *Concurrence*, attaché au thème 3, mais auquel le thème 1 participe très activement. Cela résulte de l’omniprésence de plus en plus marquée de la concurrence dans le laboratoire (jeux asynchrones, homotopie de la concurrence, réseaux d’interaction différentiels).

Quant au thème 4, le lien avec les trois autres thèmes plus théoriques est tout d’abord “par osmose”. Ainsi, le créateur d’*Ocsigen* a “fait ses classes” en réécriture et  $\lambda$ -calcul. Pour illustrer la manière dont nous comptons appuyer nos développements logiciels sur nos compétences plus théoriques, prenons l’exemple des services Web, caractérisés par la communication entre un client et un serveur dans un environnement hautement distribué. La recherche dans ce secteur est donc largement tributaire de techniques étudiées et de résultats obtenus dans le thème 3. D’un point de vue sémantique, la nature de l’interaction entre client et service est très proche du cadre étudié par la sémantique des jeux et introduit une claire notion d’orthogonalité suggérant l’utilisation de sémantiques par réalisabilité telles que celles étudiées dans le thème 1. Enfin, la définition d’un typage comportemental des services web et l’utilisation de ce dernier pour la recherche semi-automatique dans des répertoires de services (UDDI) fera appel aux techniques de typage, à l’isomorphisme de Curry-Howard et, en perspective, aux isomorphismes de types développés dans le thème 2.

Nos synergies externes sont nombreuses, et sont recensées dans le dossier de contractualisation de PPS. Nous faisons cependant ici un “zoom” sur les synergies externes relatives au thème 4, pour souligner que la partie la plus pratique de nos activités a pris une ampleur cohérente au cours de la période de 4 ans écoulés.

Une collaboration intense existe depuis longtemps avec l’INRIA, et en particulier avec l’équipe Gallium, sur les thématiques touchant à la théorie et la pratique de la programmation ; elle s’est intensifiée avec les actions menées sur la compilation certifiée dans le cadre de l’ANR CompCert.

Le travail sur la gestion de grandes masses logicielles a créé des collaborations avec l’Université de l’Aquila en Italie, l’Université de Tel Aviv en Israël, l’Université de Louvain en Belgique, l’Université de Sophia Antipolis en France, l’INESC au Portugal, mais aussi avec des partenaires industriels dont ILOG, Pixart, Caixa Magica et Edge-IT.

Le travail sur les contrats pour les services Web est développé en collaboration avec l’Università di Bologna et celle d’Urbino, mais il s’insère dans un cadre de recherche européen, incluant de nombreux groupes de recherche tels l’Imperial College, le Queen Mary & Westfield College, l’Université de Southampton en Angleterre, celle de Glasgow en Ecosse, les Universités de Torino et de Venezia en Italie, l’université de Lisboa au Portugal.

Le travail sur les langages et les outils de développement pour le Web et XML est réalisé en collaboration avec l’université Paris 6 (LIP6), l’université Paris 11 (LRI), l’université Paris 13 (LIPN) et l’université de Pisa, ainsi que l’entreprise Net7 (Pisa).

**A propos des références.** Pour des raisons de cohérence avec le dossier de contractualisation rendu à la rentrée universitaire 2007, la bibliographie qui suit reprend l’état de la bibliographie du laboratoire au printemps 2007. Nous signalons ici l’acceptation des articles suivants :

- [CGP07] : accepté à POPL 2008 ;
- [CDKZ07] : accepté, à paraître dans le journal Theoretical Computer Science ;
- [HHM07] : accepté à LICS 2007 ;
- [MM07] : accepté à CONCUR 2007 ;

- [Mét07] : accepté, à paraître dans le journal *Homology, Homotopy and Applications* (avec le titre modifié : Cofibrant objects among higher-dimensional categories).

## Références

- [ABCB06] Roberto Amadio, Gérard Boudol, Ilaria Castellani, and Frédéric Boussinot. Reactive concurrent programming revisited. In *Workshop on Process Algebra*, volume 162 of *Electronic Notes in Theoretical Computer Science*, pages 49–60. Elsevier, September 2006.
- [ACCL91] Martín Abadi, Luca Cardelli, Pierre Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1) :375–416, 1991.
- [AD06] Roberto Amadio and Frédéric Dabrowski. Feasible Reactivity for Synchronous Cooperative Threads. In *Expressiveness in Concurrency*, Electronic Notes in Theoretical Computer Science, 154(3), pages 33–43. Elsevier, July 2006.
- [AD07] Roberto Amadio and Frédéric Dabrowski. Feasible reactivity in a synchronous pi-calculus. In Andreas Podelski, editor, *Proceedings ACM SIGPLAN Principles and Practice of Declarative Programming*, pages 221–231, Wrocław Pologne, July 2007. ACM.
- [ADZ06] Roberto Amadio and Silvano Dal Zilio. Resource Control for Synchronous Cooperative Threads. *Journal of Theoretical Computer Science (TCS)*, 358 :229–254, August 2006.
- [AGMO04] Samson Abramsky, Dan Ghica, Andrej Murawski, and Luke Ong. Applying game semantics to compositional software modeling and verifications. In *Tenth International Conference on Tools and Algorithms for the Construction and Analysis of System (TACAS 2004)*, Barcelona, 2004.
- [AGW04] M. Abadi, G. Gonthier, and B. Werner. Choice in dynamic linking. In *Proceedings of FOSSACS’04 - Foundations of Software Science and Computation Structures 2004*, 2004.
- [AJ92] Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science (LICS’92)*, Santa Cruz, 1992.
- [AJ94] S. Abramsky and R. Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic. *The Journal of Symbolic Logic*, 59(2) :543–574, 1994.
- [AJ03] Samson Abramsky and Radha Jagadeesan. A game semantics for generic polymorphism. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures*, volume 2620 of *LNCS*, pages 1–22. Springer, 2003.
- [AM99] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS’99)*, Trento, 1999.
- [Ama07a] Roberto Amadio. A synchronous pi-calculus. *Journal of Information and Computation*, 205(9) :1470–1490, September 2007.
- [Ama07b] Roberto Amadio. The SL synchronous language, revisited. *Journal of Logic and Algebraic Programming*, 70 :121–150, February 2007.
- [AMR06] Ariel Arbiser, Alexandre Miquel, and Alejandro Ríos. A lambda-calculus with constructors. In *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, pages 181–196, Seattle États-Unis d’Amérique, 2006. Springer. (avec annexes).

- [AMRV07] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 109–122, Nice France, 2007. ACM Press.
- [And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 3 :297–347, 1992.
- [APP91] Martín Abadi, Benjamin Pierce, and Gordon Plotkin. Faithful ideal models for recursive polymorphic types. *International Journal of Foundations of Computer Science*, 2(1) :1–21, March 1991. Summary in Fourth Annual Symposium on Logic in Computer Science, June, 1989.
- [Bai04] Patrick Baillot. Stratified coherent spaces : a denotational semantics for light linear logic. *Theoretical Computer Science*, 318(1–2) :29–55, 2004.
- [Bal06] Vincent Balat. Ocsigen : Typing Web Interaction with Objective Caml. In *ML '06 : Proceedings of the ACM SIGPLAN 2006 workshop on ML*, pages 84–94, Portland États-Unis d'Amérique, September 2006. ACM.
- [Bat98] Michael Batanin. Monoidal globular categories as a natural environment for the theory of weak n-categories. *Adv. Math.*, 136 :39–103, 1998.
- [BB02] S. Berardi and C. Berline. Beta-eta-complete models for System F. *Mathematical Structures in Computer Science*, 12 :823–874, 2002.
- [BBC95] Stefano Berardi, Marc Bezem, and Thierry Coquand. A realization of the negative interpretation of the axiom of choice. In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 1995.
- [BBS02] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114 :3–25, 2002.
- [BCDC83] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48 :931–940, 1983.
- [BCDL07] Patrick Baillot, Paolo Coppola, and Ugo Dal Lago. Light Logics and Optimal Reduction : Completeness and Complexity. In Luke Ong, editor, *Proceedings of Symposium on Logic in Computer Science (LICS '07)*, Wroclaw Pologne, 2007. IEEE Computer Society Press. 10 pages.
- [BDER97] P. Baillot, V. Danos, T. Ehrhard, and L. Regnier. Timeless games. In *Proceedings of the 11th Annual Conference on Computer Science and Logic (CSL'97)*, Aarhus, 1997.
- [Bef06] Emmanuel Beffara. A concurrent model for linear logic. In *Proceedings of the 21st Conference on Mathematical Foundations of Programming Semantics (MFPS'05)*, Electronic Notes in Theoretical Computer Science, pages 147–168. Elsevier, 2006.
- [BEM07] Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. 15 pages. Accepted at the conference Computer Science Logic 2007 (CSL 07), 2007.
- [Ber06] Chantal Berline. Graph models of lambda-calculus at work, and variations. *Mathematical Structures in Computer Science*, 16 :185–221, 2006. preprint Mars 2005.
- [Bie95] Gavin Bierman. What is a categorical model of intuitionistic linear logic ? In *Typed Lambda-Calculus and Applications (TLCA'95)*, Edinburgh, number 902 in LNCS. Springer Verlag, 1995.



- [BKR05a] Eduardo Bonelli, Delia Kesner, and Alejandro Ríos. de Bruijn Indices for Meta-Terms. *Journal of Logic and Computation*, 15 :855–899, 2005.
- [BKR05b] Eduardo Bonelli, Delia Kesner, and Alejandro Ríos. Relating Higher-Order and First-Order Rewriting. *Journal of Logic and Computation*, 15 :901–947, 2005.
- [BL04] Antonio Bucciarelli and Benjamin Leperchey. Hypergraphs and degrees of parallelism, a completeness result. In *Foundations of Software Science and Computation Structures 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004. Proceedings*, LNCS 2987, pages 58–71, Barcelone Espagne, 2004. Springer.
- [BL05] Kai Bruennler and Stéphane Lengrand. On Two Forms of Bureaucracy in Derivations. In Paola Bruscoli, Francois Lamarche, and James Stewart, editors, *Proceedings of the 1st Workshop on Structures and Deductions (SD’05)*, ISSN 1430-211X, pages 75–86, Lisbon Portugal, July 2005. Technische Universitaet Dresden.
- [BLN06] Pascal Berthomé, Sylvain Lebesne, and Kim Nguyen. Computation of Chromatic Polynomials Using Triangulations and Clique Trees. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 362–373, Metz France, December 2006. Springer.
- [BLP03] Antonio Bucciarelli, Benjamin Leperchey, and Vincent Padovani. Relative Definability and Models of Unary PCF. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings*, pages 75–89. Springer, June 2003.
- [BM06] Emmanuel Beffara and François Maurel. Concurrent nets : a study of prefixing in process calculi. *Journal of Theoretical Computer Science (TCS)*, 356(3) :356–373, May 2006.
- [BMS07] Chantal Berline, Giulio Manzonetto, and Antonino Salibra. Lambda theories of effective lambda models. 15 p., accepté à CSL’07, 2007.
- [BS03] Antonio Bucciarelli and Antonino Salibra. The Minimal Graph Model of Lambda Calculus. In *Mathematical Foundations of Computer Science 2003, Proceedings*, LNCS 2747, pages 300–307, Bratislava slovaque, République, 2003. Springer.
- [BS04] Antonio Bucciarelli and Antonino Salibra. The Sensible Graph Theories of Lambda Calculus. In *Logic in Computer Science 2004, Proceedings*, pages 276–285, Turku Finlande, 2004. IEEE.
- [BS06] Chantal Berline and Antonino Salibra. Easiness in graph models. *Theoretical Computer Science*, 354 :4–23, 2006.
- [BS07] Antonio Bucciarelli and Antonino Salibra. Graph Lambda Theories. to appear in MSCS, 2007.
- [Bur93] Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115 :43–62, 1993.
- [Cas07] Giuseppe Castagna. CDuce, an XML Processing Programming Language : From Theory to Practice. In Brazilian Computer Society, editor, *SBLP, Proc. of XI Brazilian Symposium on Programming Languages*, pages 3–4, Brésil, 2007.
- [CCD<sup>+</sup>04] Nathalie Chabrier, Marc Chiaverini, Vincent Danos, François Fages, and Vincent Schachter. Modeling and Querying biological networks. *Theoretical Computer Science*, 325(1) :25–44, September 2004.

- [CDKZ07] Pierre-Louis Curien, Vincent Danos, Jean Krivine, and Min Zhang. Computational self-assembly. 2007.
- [CF03a] Emmanuel Chailloux and Christian Foisy. A portable implementation for Objective Caml flight. *Parallel Processing Letters*, 13(3) :425–436, August 2003. version journal de [CF03b].
- [CF03b] Emmanuel Chailloux and Christian Foisy. A portable Implementation for Objective Caml Flight. In *Second Workshop on High-Level Parallel Programming and Applications*, France, June 2003.
- [CF05] Pierre-Louis Curien and Claudia Faggian. L-nets, strategies and proof-nets. In Springer, editor, *Proc. Computer Science Logic 2005*, Lecture Notes in Computer Science 3634, pages 167–183, Oxford Royaume-Uni, 2005. Springer.
- [CF06] Pierre-Louis Curien and Claudia Faggian. An approach to innocent strategies as graphs. 2006.
- [CGP07] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. In *PLAN-X, ACM-SIGPLAN Workshop on Programming Language Technologies for XML*, France, 2007. Actes électroniques.
- [CH04] Emmanuel Chailloux and Grégoire Henry. O’Jacare : une interface objet entre Objective Caml et Java. *L’Objet, logiciel, base de données, réseaux (RSTI série)*, 10(2-3) :75–88, 2004.
- [CHM04a] Emmanuel Chailloux, Grégoire Henry, and Raphaël Montelatici. Interopérabilité avec Objective Caml. *Technique et Science Informatiques (TSI)*, 24(9) :1055–1080, 2004.
- [CHM04b] Emmanuel Chailloux, Grégoire Henry, and Raphaël Montelatici. Mixing the Objective Caml and C# Programming Models in the .Net Framework. In *Workshop on Multiparadigm Programming with Object-Oriented Languages (MPOOL)*, Oslo Norvège, June 2004.
- [CK04] Serenella Cerrito and Delia Kesner. Pattern matching as cut elimination. *Theoretical Computer Science*, 323(1-3) :71–127, 2004.
- [CMV<sup>+</sup>06] Francois Clément, Vincent Martin, Arnaud Vodicka, Roberto Di Cosmo, and Pierre Weis. Domain decomposition and skeleton programming with OCamlP3l. *Parallel Computing*, 32(7-8) :539–550, September 2006.
- [CV05] Emmanuel Chailloux and Julien Verlaquet. HironML : Fair Threads Migrations for Objective Caml. In *Third Workshop on High-Level Parallel Programming and Applications*, Royaume-Uni, July 2005.
- [CVY07] Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Compositional Event Structure Semantics of the Internal pi-Calculus. Accepté à CONCUR 2007, 2007.
- [DC95] Roberto Di Cosmo. *Isomorphisms of Types*. Progress in Theoretical Computer Science. Birkhäuser, 1995.
- [DCBD<sup>+</sup>06] Roberto Di Cosmo, Jaap Boender, Berke Durak, Xavier Leroy, Fabio Mancinelli, Mario Morgado, David Pinheiro, Ralf Treinen, Paulo Trezentos, and Jérôme Vouillon. News from the EDOS project : improving the maintenance of free software distributions. In Olivier Berger, editor, *Proceedings of the International Workshop on Free Software*, pages 199–207, Porto Alegre Brésil, April 2006. Brazilian Computer Society.
- [DCDL<sup>+</sup>06] Roberto Di Cosmo, Berke Durak, Xavier Leroy, Fabio Mancinelli, and Jérôme Vouillon. Maintaining large software distributions : new challenges from the FOSS

- era. In *Proceedings of the FRCSS 2006 workshop.*, EASST Newsletter, volume 12, pages 7–20, Vienna Autriche, April 2006. EASST.
- [DCKP03] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof Nets and Explicit Substitutions. *Mathematical Structures in Computer Science*, 13(3) :409–450, 2003.
  - [DCLP07] Roberto Di Cosmo, Zheng Li, and Susanna Pelagatti. A calculus for parallel computations over multidimensional dense arrays. *Computer Languages, Systems and Structures*, 33(3-4) :82–110, October 2007.
  - [DCP03] Roberto Di Cosmo and Susanna Pelagatti. A Calculus for Dense Array Distributions. *Parallel Processing Letters*, 13(3) :377–388, August 2003.
  - [DD03a] Vincent Danos and Josée Desharnais. A Fixpoint Logic for Labeled Markov Processes. In *Proceedings of the international Workshop Fixed Points in Computer Science (FICS’03)*, pages 413–422, Pologne, 2003.
  - [DD03b] Vincent Danos and Josée Desharnais. Labeled Markov processes : stronger and faster approximations. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 341–350, Canada, 2003.
  - [DDL06] Vincent Danos, Josée Desharnais, François Laviolette, and Prakash Panangaden. Bisimulation and cocongruence for probabilistic systems. *Information and Computation*, 204(4) :503–523, April 2006.
  - [DDP03] Vincent Danos, Josée Desharnais, and Prakash Panangaden. Conditional expectations and the approximation of labeled Markov processes. In *Proceedings of CONCUR’03*, volume 2761 of *Lecture Notes in Computer Science*, pages 477–491, France, 2003. Springer.
  - [DDP04] Vincent Danos, Josée Desharnais, and Prakash Panangaden. Labeled Markov processes : stronger and faster approximations. *Electronic Notes in Theoretical Computer Science*, 87 :157–203, September 2004.
  - [Den05a] Yuxin Deng. *Axiomatisations et types pour des processus probabilistes et mobiles*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris - ENSMP, July 2005.
  - [Den05b] Yuxin Deng. Axiomatizations for probabilistic finite-state behaviors. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures*, pages 110–124. Springer, February 2005.
  - [Den05c] Yuxin Deng. Compositional Reasoning for Probabilistic Finite-State Behaviors. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles : Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, pages 309–337. Springer-Verlag, 2005.
  - [Den06] Yuxin Deng. Metrics for action-labelled quantitative transition systems. In *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages*, volume 153 of *Electronic Notes in Theoretical Computer Science*, pages 79–96. Elsevier, May 2006.
  - [Den07a] Yuxin Deng. Axiomatizations for probabilistic finite-state behaviors. *Theoretical Computer Science*, 373(1-2) :92–114, March 2007.
  - [Den07b] Yuxin Deng. Weak probabilistic anonymity. In *Proceedings of the 3rd International Workshop on Security Issues in Concurrency*, volume 180 of *Electronic Notes in Theoretical Computer Science*, pages 55–76. Elsevier, June 2007.

- [DFF<sup>+</sup>07] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In *International Conference on Concurrency Theory (CONCUR'07)*, Portugal, September 2007.
- [DG01] René David and Bruno Guillaume. A  $\lambda$ -calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11 :169–206, 2001.
- [DGL06] A. Dimovski, D. Ghica, and R. Lazić. A counterexample-guided refinement tool for open procedural programs. In *13th International SPIN Workshop on Model Checking of Software, Vienna*, 2006.
- [DH01] Vincent Danos and Russell Harmer. The anatomy of innocence. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 01. 10th Annual Conference of the EACSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2001.
- [DHR96] Vincent Danos, Hugo Herbelin, and Laurent Regnier. Games semantics and abstract machines. In *Proceedings of the eleventh annual symposium on Logic in Computer Science (LICS 96)*, pages 394–405. IEEE, IEEE Computer Society Press, July 1996.
- [DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. LKQ and LKT : Sequent calculi for second order logic based upon dual linear decompositions of classical implication. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 211–224. Cambridge University Press, 1995.
- [DK05] Vincent Danos and Jean Krivine. Transactions in RCCS. In *Sixteenth International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Sciences*, pages 398–412, Royaume-Uni, 2005.
- [DK07] Vincent Danos and Jean Krivine. Formal Molecular Biology done in CCS. In *Proceedings of BIO-CONCUR'03*, volume 180 of *Electronic Notes in Theoretical Computer Science*, pages 31–49, France, 2007.
- [DKL06] Roy Dyckhoff, Delia Kesner, and Stéphane Lengrand. Strong cut-elimination systems for Hudelmaier’s depth-bounded sequent calculus for implicational logic. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, pages 347–361. Springer-Verlag, 2006.
- [DKS07] Vincent Danos, Jean Krivine, and Pawel Sobocinski. General Reversibility. In *Proceedings of the 13th International Workshop on Expressiveness in Concurrency (EXPRESS 2006)*, Electronic Notes in Theoretical Computer Science volume 175, pages 75–86. Elsevier, June 2007.
- [DKT07] Vincent Danos, Jean Krivine, and Fabien Tarissan. Self-assembling Trees. In *Proceedings of the Third Workshop on Structural Operational Semantics*, Volume 175, Issue 1 of Electronic Notes in Theoretical Computer Science, pages 19–32. Elsevier, May 2007.
- [DL04] Vincent Danos and Cosimo Laneve. Formal Molecular Biology. *Theoretical Computer Science*, 325(1) :69–110, October 2004.
- [DL06] Roy Dyckhoff and Stéphane Lengrand. LJQ, a strongly Focused Calculus for Intuitionistic Logic. In A. Beckmann, U. Berger, B. Löwe, and J. V Tucker, editors, *Proceedings of the 2nd Conference on Computability in Europe (CiE'06)*, LNCS, volume 3988, pages 173–185, Swansea Royaume-Uni, June 2006. Springer-Verlag.

- [DL07a] Joachim De Lataillade. Curry-style type Isomorphisms and Game Semantics. Accepted to Mathematical Structures in Computer Science, Special Issue on Type Isomorphisms, 2007.
- [DL07b] Joachim De Lataillade. Second-Order Type Isomorphisms Through Game Semantics. accepted by Annals of Pure and Applied Logic, Special Issue on Game Semantics, 2007.
- [DP05] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *International Conference on Computational Methods in Systems Biology (CMSB 2004)*, volume 3082 of *Lecture Notes in Bio-Informatics*, pages 134–148, France, 2005. Springer.
- [DT05] Vincent Danos and Fabien Tarissan. Self-assembling Graphs. In *Proceedings of the International Work-conference on the Interplay between Natural and Artificial Computation*, volume 3561 of *Lecture Notes in Computer Sciences*, pages 498–507. Springer, 2005.
- [DT06] Vincent Danos and Fabien Tarissan. Self-assembling graphs. *Natural Computing*, page 20 p., December 2006.
- [Ehr05] Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4) :615–646, July 2005. 32 pages.
- [EL07] Thomas Ehrhard and Olivier Laurent. Interpreting a Finitary Pi-Calculus in Differential Interaction Nets. Accepted at CONCUR 2007. 32 pages., 2007.
- [ER06] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2) :166–195, November 2006. 30 pages.
- [FDG06] Claudia Faggian and Paolo Di Giamberardino. Jump from Parallel to Sequential Proofs : Multiplicatives. In *CSL 06 - Computer Science Logic*, LNCS, pages 319–333. Springer, 2006.
- [FK03] Julien Forest and Delia Kesner. Expression Reduction Systems with Patterns. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, pages 107–122. Springer-Verlag, 2003.
- [FM05] Claudia Faggian and François Maurel. Ludics Nets, a game Model of Concurrent Interaction. In *LICS 05*, pages 376–385. IEEE Computer Society, 2005.
- [FP07a] Claudia Faggian and Mauro Piccolo. A Graph Abstract Machine Describing Event Structure Composition. In *GT-VC 2006 Graph Transformation for Verification and Concurrency*, pages 60–75. ENTCS, 2007.
- [FP07b] Claudia Faggian and Mauro Piccolo. Ludics is a model for the finitary linear Pi-calculus. In *TLCA 07 - Typed Lambda Calculi and Applications*, *Lecture Notes in Computer Science*, page 15 pages. Springer, 2007.
- [Gau03] Philippe Gaucher. A model category for the homotopy theory of concurrency. *Homology, Homotopy and Applications(HHA)*, 5(1) :549–599, December 2003.
- [Gau05a] Philippe Gaucher. Comparing globular complex and flow. *The New York Journal of Mathematics*, 11 :97–150, April 2005.
- [Gau05b] Philippe Gaucher. Flow does not model flows up to weak dihomotopy. *Applied Categorical Structures*, 13(5-6) :371–388, November 2005.
- [Gau05c] Philippe Gaucher. Homological properties of non-deterministic branchings and mergings in higher dimensional automata. *Homology, Homotopy and Applications(HHA)*, 7(1) :51–76, May 2005.

- [Gau06a] Philippe Gaucher. Inverting weak dihomotopy equivalence using homotopy continuous flow. *Theory and Applications of Categories*, 16(3) :59–83, January 2006.
- [Gau06b] Philippe Gaucher. T-homotopy and refinement of observation (III) : Invariance of the branching and merging homologies. *The New York Journal of Mathematics*, 12 :319–348, September 2006.
- [Gau06c] Philippe Gaucher. T-homotopy and refinement of observation (IV) : Invariance of the underlying homotopy type. *The New York Journal of Mathematics*, 12 :63–95, June 2006.
- [Gau07] Philippe Gaucher. T-Homotopy and Refinement of Observation, Part II : Adding New T-Homotopy Equivalences. *International Journal of Mathematics and Mathematical Sciences*, 2007 :Article ID 87404, 20 pages, May 2007.
- [Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.
- [Gir91] Jean-Yves Girard. A new constructive logic : classical logic. *Mathematical Structures in Computer Science*, 1(3) :255–296, 1991.
- [Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2) :175–204, June 1998.
- [Gir01] Jean-Yves Girard. Locus solum : from the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3) :301–506, 2001.
- [GKK05] John Glauert, Delia Kesner, and Zurab Khasidashvili. Expression Reduction Systems and Extensions : An Overview. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles : Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, pages 496–553. Springer-Verlag, 2005.
- [Gri90] Timothy G. Griffin. A formulae-as-types notion of control. In *17th Annual ACM Symposium on Principles of Programming Languages (POPL’90)*, San Francisco, 1990.
- [Gug07] Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1) :1, 2007.
- [Har06] Russ Harmer. An analysis of innocent interaction. In *2nd International Workshop on Games for Logic and Programming Languages (GALOP)*, Seattle États-Unis d’Amérique, 2006.
- [HH06] Samuel Hym and Matthew Hennessy. Adding recursion to Dpi (Extended abstract). In *Proceedings of the Second Workshop on Structural Operational Semantics (SOS 2005)*, Electronic Notes in Theoretical Computer Science, 156 (1), pages 115–133. Elsevier B.V., May 2006.
- [HH07] Samuel Hym and Matthew Hennessy. Adding recursion to Dpi. *Theoretical Computer Science*, 373(3) :182–212, April 2007.
- [HHM07] Russ Harmer, Martin Hyland, and Paul-André Melliès. Categorical Combinatorics for Innocent Strategies. 2007.
- [HL06] Russ Harmer and Olivier Laurent. The anatomy of innocence revisited. In S. Arun Kumar and Naveen Garg, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 224–235. Springer, November 2006.
- [HL07] Martin Hyland and Olivier Laurent. A categorical analysis of translations of  $\lambda$ -calculi into linear logic. Notes d’un exposé au Workshop Linear Logic, Ludics, Implicit Complexity, Operator Algebras, Siena, May 2007.

- [HMC06] Grégoire Henry, Michel Mauny, and Emmanuel Chailloux. Typing la dé-sérialisation sans sérialiser les types. In *Journée francophone des langages applicatifs (JFLA) 2006*, pages 133–146, Pauillac France, January 2006. INRIA.
- [HO00] Martin Hyland and Luke Ong. On full abstraction for PCF. *Information and Computation*, 163(2) :285–408, December 2000.
- [Hug00] Dominic Hughes. *Hypergame semantics : full completeness for system F*. D.Phil. thesis, Oxford University, 2000.
- [JK06] C. Barry Jay and Delia Kesner. Pure Pattern Calculus. In *Proceedings of the European Symposium on Programming*, pages 100–114. Springer-Verlag, 2006.
- [Joi07] Jean-Baptiste Joinet. Completeness of MLL proof-nets w.r.t. weak distributivity. *Journal of Symbolic Logic*, 72(1) :159–170, March 2007. 11 pages.
- [Joy77] André Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des Sciences Mathématiques du Québec*, 1(2) :46–52, 1977.
- [JSV96] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119 :447–468, 1996.
- [JT91] André Joyal and Myles Tierney. Strong stacks and classifying spaces. In *Category Theory, Proc. Int. Conf., Como/Italy 1990*, number 1488 in Lect. Notes Math., pages 213–236, 1991.
- [Kes07] Delia Kesner. The theory of calculi with explicit substitutions revisited. In *16th EACSL Annual Conference on Computer Science and Logic*, page 15 pages. Springer-Verlag, 2007.
- [Kha90] Zurab Khasidashvili. Expression reduction systems. In *Proceedings of IN Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.
- [KL05] Delia Kesner and Stéphane Lengrand. Extending the Explicit Substitution Paradigm. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, pages 407–422. Springer-Verlag, 2005.
- [KL07] Delia Kesner and Stéphane Lengrand. Resource Operators for lambda-calculus. *Information and Computation*, 205(4) :419–473, 2007.
- [Kri03] J.-L. Krivine. Dependent choice, ‘quote’ and the clock. *Th. Comp. Sc.*, 308 :259–276, 2003.
- [Kri05] J.-L. Krivine. Realizability in classical logic. available on the author’s web page, 2005.
- [Lac02] Stephen Lack. A Quillen model structure for 2-categories. *K-Theory*, 26(2) :171–205, 2002.
- [Lac04] Stephen Lack. A Quillen model structure for bicategories. *K-Theory*, 33(3) :185–197, 2004.
- [Laf03] Yves Lafont. Towards an algebraic theory of boolean circuits. *Journal of Pure and Applied Algebra*, 184(2-3) :257–310, 2003.
- [Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1–2) :163–180, June 2004.
- [Lai01] James Laird. A fully abstract game semantics of local exceptions. In *Proceedings of the sixteenth annual symposium on Logic in Computer Science (LICS 01)*, pages 105–114. IEEE, IEEE Computer Society Press, June 2001.
- [Lau04] Olivier Laurent. A proof of the focalization property of linear logic. Available at <http://www.pps.jussieu.fr/~laurent/11foc.ps.gz>, May 2004.

- [Lau05a] Olivier Laurent. Intersection types with subtyping by means of cut-elimination. Available at <http://www.pps.jussieu.fr/~laurent/intercut.ps.gz>, January 2005.
- [Lau05b] Olivier Laurent. Syntax vs. Semantics : a polarized approach. *Theoretical Computer Science*, 343 :177–206, 2005.
- [LDM06] Stéphane Lengrand, Roy Dyckhoff, and James Mckinna. A Sequent Calculus for Type Theory. In Zoltan Esik, editor, *Proceedings of the 15th Conference on Computer Science Logic (CSL'06)*, LNCS, volume 4207, pages 441–455, Szeged Hongrie, September 2006. Springer-Verlag.
- [Lei04] Tom Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series (No. 298). Cambridge University Press, 2004.
- [Len03] Stéphane Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *Post-proceedings of the 3rd Workshop on Reduction Strategies in Rewriting and Programming (WRS'03)*, ENTCS, volume 86(4), pages 714–730. Elsevier, November 2003.
- [Len04] Stéphane Lengrand. Deriving strong normalisation. In D. Kesner, F. van Raamsdonk, and J. Wells, editors, *Proceedings of the 2nd International Workshop on Higher-Order Rewriting (HOR'04)*, Technical report of the Computer Science Department of RWTH Aachen, pages 84–88, Aachen Allemagne, June 2004. RWTH Aachen.
- [Len05] Stéphane Lengrand. Induction principles as the foundation of the theory of normalisation : Concepts and Techniques. Early 2005. Chapter 0 of my thesis., 2005.
- [Len06] Stéphane Lengrand. *Normalisation & Equivalence en Théorie de la Démonstration & Théorie des Types*. PhD thesis, Université Denis Diderot - Paris VII ; University of St Andrews, December 2006. Commencée en Septembre 2003.
- [LLD<sup>+</sup>04] Stéphane Lengrand, Pierre Lescanne, Dan Dougherty, Mariangiola Dezani, and Steffen Van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1) :17–42, 2004.
- [LM07a] Yves Lafont and François Métayer. Polygraphic resolutions and homology of monoids. soumis, 2007.
- [LM07b] Stéphane Lengrand and Alexandre Miquel. Classical  $F_\omega$ , orthogonality and symmetric candidates. Accepté pour publication dans le journal APAL ; 25 pages, 2007.
- [LT07] Cosimo Laneve and Fabien Tarissan. A simple calculus for proteins and cells. Accepté à MeCBIC'06, 2007.
- [LTDF06] Olivier Laurent and Lorenzo Tortora De Falco. Obsessional cliques : a semantic characterization of bounded time complexity. In Rajeev Alur, editor, *Logic in Computer Science*, pages 179–188. IEEE, 2006.
- [MBDC<sup>+</sup>06] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jérôme Vouillon, Xavier Leroy, and Ralf Treinen. Managing the Complexity of Large Free and Open Source Package-Based Software Distributions. In *Automated Software Engineering*, pages 199–208, Tokyo Japon, September 2006. ACM.
- [MCP04] Raphaël Montelatici, Emmanuel Chailloux, and Bruno Pagano. OCAMiL un compilateur Objective Caml pour .NET. In *Conférence Internationale des chercheurs vietnamiens et Francophones en Informatique (RIVF'04)*, Viet Nam, February 2004.



- [MCP05] Raphaël Montelatici, Emmanuel Chailloux, and Bruno Pagano. Objective Caml on .NET : The OCaml Compiler and Toplevel. In *3rd International Conference on .NET Technologies*, pages 109–120. Vaclav Skala and Piotr Nienaltowski, May 2005.
- [Mel] Paul-André Melliès. Tensorial Logic : a 2-dimensional Algebra of Duality. Habilitation de Recherche en préparation. Soutenance prévue courant 2008.
- [Mel04] Paul-André Melliès. Asynchronous games 3 : An innocent model of linear logic. In Elsevier, editor, *CTCS 2004*, pages 171–192, Copenhagen Denmark, 2004. Elsevier.
- [Mel05] Paul-André Melliès. Asynchronous games 4 : A fully complete model of propositional linear logic. In IEEE, editor, *LiCS 2005 – Logic in Computer Science*, pages 386–395, États-Unis d’Amérique, June 2005. IEEE.
- [Mel06a] Paul-André Melliès. Asynchronous games 2 : the true concurrency of innocence. *Theoretical Computer Science*, 358(2-3) :200–228, August 2006.
- [Mel06b] Paul-André Melliès. Functorial boxes in string diagrams. In Springer Verlag, editor, *Computer Science Logic 2006*, LNCS 4207/2006, pages 1–30, Szeged Hongrie, September 2006.
- [Mel07] Paul-André Melliès. On Hopf algebras and Frobenius algebras. Manuscript, 2007.
- [Mét03] François Métayer. Resolutions by Polygraphs. *Theory and Applications of Categories*, 11(7) :148–184, May 2003.
- [Mét07] François Métayer. Cofibrant complexes are free. 16 pages, soumis, 2007.
- [Mim07] Samuel Mimram. Towards an algebra of innocence. 2007.
- [Miq07] Alexandre Miquel. Classical program extraction in the calculus of constructions. Accepté à CSL’07, 2007.
- [MM07] Paul-André Melliès and Samuel Mimram. Asynchronous games : innocence without alternation. 2007.
- [MO01] Andrzej Murawski and Luke Ong. Evolving games and essential nets for affine polymorphism. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications ’01*, volume 2044 of *LNCS*. Springer, 2001.
- [Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1) :55–92, July 1991.
- [Mon07] Raphaël Montelatici. *Langages fonctionnels, typage et interopérabilité : Objective Caml sur .NET*. PhD thesis, Université Denis Diderot - Paris VII, March 2007.
- [MPS86] David MacQueen, Gordon Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71(1-2) :95–130, 1986.
- [MT07] Paul-André Melliès and Nicolas Tabareau. Resource modalities in game semantics. accepté à LICS’07, 2007.
- [MV05] Paul-André Melliès and Jérôme Vouillon. Recursive Polymorphic Types and Parametricity in an Operational Framework. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS’ 05)*, pages 82–91. IEEE Computer Society, 2005.
- [MY07] Eric Mjølness and Guy Yosiphon. Stochastic process semantics for dynamical grammars. *Annals of Mathematics and Artificial Intelligence*, 2007.
- [Nic94] Hanno Nickau. Hereditarily sequential functionals. In Anil Nerode and Yuri Matiyasevich, editors, *Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994.

- [Par92] Michel Parigot.  $\lambda\mu$ -calculus : an algorithmic interpretation of classical natural deduction. In *Proceedings of International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.
- [PH06] Frédéric Peschanski and Samuel Hym. A Stackless Runtime Environment for a Pi-calculus. In *ACM/Usenix International Conference On Virtual Execution Environments Proceedings of the second international conference on Virtual execution environments*, pages 57–67. ACM Press, 2006.
- [Pol04a] Emmanuel Polonovski. Strong normalization of lambda-bar-mu-mu-tilde-calculus with explicit substitutions. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2004)*, LNCS volume 2987, pages 423–437. Springer-Verlag, 2004.
- [Pol04b] Emmanuel Polonovski. *Substitutions explicites, logique et normalisation*. PhD thesis, Université Denis Diderot - Paris VII, June 2004.
- [PSU99] David Patterson, Lawrence Snyder, and Jeffrey Ullman. Best practices memo. *Computing Research News*, 1999. Computing Research Association.
- [Raf04] Christophe Raffalli. Getting results from programs extracted from classical proofs. *Theoretical Computer Science*, 323 :49–70, 2004.
- [See89] Robert Seely. Linear logic,  $\star$ -autonomous categories and cofree coalgebras. *Contemporary mathematics*, 92, 1989.
- [Squ87] Craig Squier. Word problems and a homological finiteness condition for monoids. *Journal of Pure and Applied Algebra*, 49 :201–217, 1987.
- [Str72] Ross Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2 :149–168, 1972.
- [Tab06] Nicolas Tabareau. De l’opérateur de trace dans les jeux de Conway. 2006.
- [VM04] Jérôme Vouillon and Paul-André Melliès. Semantic Types : A fresh look at the ideal model for types. In *Proceedings of the 31th ACM Conference on Principles of Programming Languages*, pages 52–63, Venezia Italie, 2004. ACM Press.
- [Vou06a] Jérôme Vouillon. Polymorphic Regular Tree Types and Patterns. In *POPL ’06 : Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 103–114, Charleston États-Unis d’Amérique, 2006. ACM Press.
- [Vou06b] Jérôme Vouillon. Polymorphism and XDuce-style patterns. In *PLAN-X’06*, Charleston États-Unis d’Amérique, January 2006.
- [VVK05] Hagen Völzer, Daniele Varacca, and Ekkart Kindler. Defining Fairness. In *Proceedings of CONCUR*, pages 458–472. Springer, 2005.
- [VW06] Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic Event Structures and Domains. *Theoretical Computer Science*, 358(2-3) :173–199, April 2006. Version étendue d’un article du même titre apparu à CONCUR 2004.
- [VY06] Daniele Varacca and Nobuko Yoshida. Typed Event Structures and the pi-calculus. In *Proceedings of MFPS*, pages 373–397. Elsevier, 2006.
- [VY07] Daniele Varacca and Nobuko Yoshida. Probabilistic pi-calculus and Event Structures. Accepté pour le workshop QAPL 2007, associé à ETAPS, 2007.