Travaux de recherche jusqu'à l'été 2022

Charles Grellois

http://www.grellois.fr

charles.grellois@univ-amu.fr

Chapitre 1

Recherche : travaux effectués

1.1 Vérification des programmes fonctionnels *déterministes*

Programmation fonctionnelle et types simples. La programmation fonctionnelle considère que les fonctions sont des citoyens de première classe du langage de programmation. Cela signifie qu'une fonction peut être argument ou valeur de retour d'une autre fonction. Un exemple classique est celui de la fonction *map* : soient $L = [n_1, \ldots, n_k]$ une liste d'entiers et f une fonction des entiers dans les entiers. Alors *map* f L renvoie la liste d'entiers $[f(n_1), \ldots, f(n_k)]$. Ceci n'est pas spécifique aux entiers : on peut rendre *map polymorphe*, c'est-à-dire paramétrée par un type qui sera celui de la liste fournie en entrée (et également celui manipulé par f), de sorte que map n'est pas restreinte aux entiers. Nous ne considèrerons le polymorphisme que dans le projet de recherche.

Ce style de programmation vient du lambda-calcul, un langage théorique minimaliste introduit par Church dans les années 1930, qui permet de former des fonctions, de les appliquer, et d'introduire des variables. Ces trois constructions suffisent à donner un système muni d'une relation de réécriture qui le rend Turing-complet, de sorte que la réécriture de certains termes ne termine jamais. Un exemple classique est celui du terme $\Omega = \Delta\,\Delta$ (c'est-à-dire qu'Ω est défini comme l'application de Δ à lui-même).

Une telle application d'un terme à lui-même peut générer des problèmes de terminaison. L'introduction des types simples peut être vue comme une façon de contraindre la formation des termes pour que cela n'arrive pas : si Δ a un type A, il ne peut pas avoir aussi le type $A \to A$ dans un système de types simples, terme qu'il devrait avoir dans Ω pour son instance de gauche, alors que l'instance de droite devrait être typée A.

Les types simples sont alors une façon d'étiqueter les termes, que l'on peut voir comme une description de propriétés. Un terme simplement typable est fortement normalisant : si, lors du calcul du résultat, on a plusieurs choix de réécriture, alors tout choix amènera à la terminaison, et le résultat sera toujours le même.

Comme le fragment simplement typé du calcul donne lieu à des calculs terminants uniquement, il n'est plus Turing-complet. On peut alors formuler un certain nombre d'extensions, notamment en introduisant des constructeurs pour former des types de base (construire les entiers, les arbres, les listes...), et également introduire la récursion, au coeur du travail que je vais présenter dans cette section et la suivante. Avec suffisamment d'extensions, on peut former un langage tel que PCF, qui est Turing-complet tout en étant typable ; dans ce cadre, le typage n'implique donc plus la terminaison. PCF contient les booléens (avec les tests conditionnels), les entiers positifs (avec successeur, prédécesseur et test à zéro) et la récursion.

Model-checking des programmes fonctionnels. Une façon de vérifier les programmes est de faire du *model-checking* ([CES86]). On veut décider si un programme a une propriété donnée, par exemple la terminaison, mais cela est impossible en général lorsque le langage de programmation est Turing-complet. On procède donc à une approximation du programme, sur laquelle on vérifiera une logique décidable.

Pour les langages fonctionnels, le choix s'est historiquement porté sur une modélisation par arbres infinis (certaines branches, ou toutes, peuvent rester finies). Un aspect crucial de ces arbres est qu'ils ne sont pas réguliers : on ne peut pas les représenter comme des graphes finis dont ils seraient le déploiement. Il faut en fait une représentation donnée par exemple par un schéma de récursion d'ordre supérieur (HORS). Il s'avère que ces schémas sont équivalents au lambda-calcul simplement typé avec récursion et constantes ; ils diffèrent de PCF par le fait que le prédécesseur n'est pas disponible si on encode les entiers,

mais également par le fait qu'ils permettent de décrire plus de structures, les arbres infinis par exemple. Ceci contraint suffisamment l'expressivité des schémas, et nous permettra de faire de la vérification.

En un sens, on va modéliser les programmes fonctionnels en les plongeant dans des programmes fonctionnels moins expressifs, qui génèrent des arbres potentiellement infinis. Les propriétés à vérifier sont traditionnellement spécifiées en logique monadique du second ordre (MSO) sur les arbres infinis générés par les schémas. Cette logique est équivalente sur les arbres infinis au mu-calcul modal, et contient notamment les logiques LTL et CTL que l'on rencontre souvent en vérification. Un modèle d'automate équivalent au mu-calcul modal, et donc à MSO sur les arbres infinis engendrés par les schémas, est celui des *automates alternants à parité*.

Théorème de décidabilité du model-checking d'ordre supérieur. Soient un schéma de récursion d'ordre supérieur et une formule MSO. On peut décider (en n-EXPTIME, où n est une quantité appelée l'ordre du schéma) si l'arbre infini engendré par le schéma satisfait la formule MSO. Initialement prouvé par Ong en 2006, le théorème a été démontré à nouveau plusieurs fois depuis, avec des outils divers : sémantique des jeux, jeux et automates à piles de piles...effondrants, systèmes de types, machine de Krivine et jeux, modèles sémantiques...[Ong06 ; Hag+08 ; KO09 ; Bro+10 ; TO14 ; CS12 ; SW15a ; GM15b ; Wal19].

Le système de types de Kobayashi et Ong. Kobayashi et Ong [KO09] ont proposé une façon de raffiner les types simples du lambda-calcul avec des *types intersection* basés sur les états de l'automate codant la formule à vérifier. Par exemple, un schéma qui produit un arbre infini accepté depuis les états q_0 et q_1 de l'automate sera typé $q_0 \wedge q_1$. Une fonction qui prend un tel arbre en argument pour produire un nouvel arbre accepté depuis q_2 sera elle typée $q_0 \wedge q_1 \to q_2$. L'incorporation de la condition de parité demande l'ajout d'une coloration des types par un entier, ceux-ci ayant alors par exemple la forme $\Box_2\, q_0 \wedge \Box_1\, q_1 \to q_2$. Kobayashi et Ong donnent ensuite un jeu de parité pour modéliser la récursion du schéma à vérifier, dans lequel les couleurs jouées par les joueurs sont guidées par celles étiquetant les types.

Travaux effectués I : logique linéaire, modèles dénotationnels, et model-checking d'ordre supérieur. Avec Paul-André Melliès, au cours de ma thèse, nous avons tout d'abord relié les types intersection du système de types de Kobayashi et Ong à la logique linéaire indexée (workshop ITRS 2014, [GM15c]), une version de la logique linéaire particulièrement adaptée à la modélisation des types intersection. La logique linéaire a été introduite par Girard dans les années 80 [Gir87] pour refléter une analyse mathématique de la programmation fonctionnelle. Cette logique est particulièrement connue pour sa décomposition de la flèche intuitionniste : $A \Rightarrow B \quad = \quad !A \multimap B$. Ceci signifie qu'un programme qui prend A pour fournir B est, dans la logique linéaire et dans ses modèles mathématiques, interprété en deux phases. Tout d'abord, on duplique A autant de fois que nécessaire (ou on l'oublie s'il n'est pas utilisé). Si A sert trois fois dans la preuve ou le programme, alors on en fait trois copies. Puis la phase $\multimap$ utilise *linéairement* chaque copie de A, c'est-à-dire sans duplication ni effacement. On voit là apparaître un lien clair et naturel avec les types intersection : avoir un programme de type $q_0 \wedge q_1 \to q_2$, c'est avoir un programme qui fait des copies de son argument, acceptées soit par q_0, soit par q_1, pour ensuite produire linéairement un résultat de type q_2. Cette observation est à la source de notre travail, qui commence donc avec l'article [GM15c] reliant types intersections et logique linéaire (dans sa variante indexée).

La sémantique la plus simple d'utilisation, et peut-être la plus naturelle de la logique linéaire, est sa sémantique relationnelle. Mais elle est traditionnellement adaptée à des modèles finitaires, considérant qu'un programme manipule des arguments en nombre fini. Ce n'est pas le cas des schémas, dans lesquels une variable peut être utilisée une infinité de fois lors de la production de l'arbre infini représenté par le schéma. Nous avons donc commencé par introduire une variante infinitaire de la sémantique relationnelle de la logique linéaire, munie de plus d'une opération de coloriage reflétant celle des types de Kobayashi et Ong, dans (FOSSACS 2015, [GM15a]), et d'un opérateur de point fixe modélisant le mécanisme du jeu de parité de Kobayashi et Ong à l'intérieur de la sémantique. Ce modèle permet ainsi l'interprétation des schémas vis-à-vis d'un automate alternant à parité.

Il restait à explorer la dualité automate/schéma du point de vue de la logique linéaire, et à donner le théorème suivant : l'interprétation d'un schéma vis-à-vis d'un automate alternant à parité dans le modèle relationnel infinitaire coloré contient q_0 si et seulement si l'arbre infini engendré par le schéma est accepté depuis q_0 par l'automate (CSL 2015, [GM15d]). En d'autres termes, il suffit de calculer l'interprétation du schéma dans le modèle pour savoir si l'arbre qu'il engendre est accepté par l'automate paramétrant le modèle.

Cependant, la sémantique relationnelle infinitaire interprète les schémas à l'aide de structures infinies, et donc inadaptées à la décidabilité. Nous avons donc répliqué nos résultats dans une autre sémantique de la logique linéaire, plus délicate à manipuler mais finitaire : celle des modèles de Scott. Elle est intrinsèquement liée au modèle relationnel usuel (finitaire, sans couleurs) par un résultat d'effondrement extensionnel [Ehr12b ; Ehr12a] qui en faisait un choix naturel. Nous avons donc étendu cette sémantique avec une opération de coloration, et un opérateur de point fixe adapté. Nous obtenons alors dans (MFCS 2015, [GM15b]) un théorème similaire à celui de [GM15d] : l'interprétation d'un schéma dit si l'arbre infini qu'il engendre est reconnu par un automate. La sémantique étant finitaire, ceci fournit une preuve de décidabilité du théorème d'Ong obtenue via une approche de sémantique dénotationnelle.

Ces résultats ont été obtenus en parallèle de ceux de Salvati et Walukiewicz [SW13a ; SW13b ; SW14 ; SW15a ; SW15b], qui donnent, sans passer par la logique linéaire, des modèles finitaires donnant lieu au même théorème.

Les preuves de Salvati et Walukiewicz et de Grellois et Melliès sont les deux premières à reposer sur des modèles dénotationnels. Dans le même esprit, on remarquera les travaux ultérieurs de Hofmann et Ledent [HL17].

Travaux effectués II : logique linéaire et typage, une analyse de complexité. On peut pousser plus loin encore l'analyse du model-checking des schémas d'ordre supérieur, en utilisant plus avant la logique linéaire. Nous avons donc introduit dans [CGM18], avec Clairambault et Murawski, la notion de schéma linéaire (LHORS), une généralisation des schémas traditionnels, dans laquelle nous ajoutons de la linéarité aux schémas. Les HORS sont alors un cas particulier des LHORS. Celle-ci se ressent particulièrement lorsque l'on formule la généralisation des automates alternants à parité qui correspond au typage linéaire-non linéaire des LHORS : en certains noeuds de l'arbre infini généré par un LHORS, l'automate doit faire un choix linéaire. Au lieu de pouvoir explorer plusieurs successeurs, et ce éventuellement plusieurs fois, l'automate doit choisir une unique direction à visiter et ne peut faire de copies du sous-arbre associé.

Comme dans le cas des HORS, les automates alternants considérés se ramènent à un jeu de typage, mais ici sur un typage mêlant linéarité et non-linéarité. Cela importe particulièrement du point de vue de la complexité : sur la base de ce typage, nous définissons l'*ordre linéaire* d'un LHORS, qui raffine la notion usuelle d'ordre, et montrons que la complexité du model-checking d'ordre supérieur est bien n-EXPTIME, mais pour n l'ordre *linéaire* du schéma, qui peut être bien inférieur à l'ordre usuellement défini. Nous pensons que cela explique pourquoi, en pratique, la vérification d'un certain nombre de programmes est faisable en temps réaliste alors que la complexité du problème est très élevée.

Un autre apport de ce cadre linéaire est la simplicité avec laquelle il permet de reformuler des preuves existant dans la littérature. Deux modèles étendent les HORS avec du pattern-matching [KTU10 ; NRO12] ; on peut prouver à nouveau les résultats de décidabilité, avec une complexité optimale, par une simple traduction dans les LHORS préservant l'ordre linéaire. De même pour un problème de vérification de ressources en appel par valeur [TK14], considérablement simplifié par le passage au cadre linéaire : une traduction CPS linéaire suffit, là où la preuve originale était bien plus complexe.

En cours. J'ai recommencé à travailler de façon très régulière avec Pierre Clairambault depuis son arrivée au LIS début septembre 2021. Nous avançons bien sur une nouvelle preuve de décidabilité du model-checking d'ordre supérieur, basée sur les outils les plus contemporains de la sémantique des jeux. L'intérêt d'une telle démarche est double. Tout d'abord, nous pensons que le théorème que nous cherchons à démontrer à nouveau n'est pas encore établi avec toute la clarté et la précision que l'on souhaiterait. Les preuves sont extrêmement lourdes, et comportent en général des passages ad hoc. Le deuxième intérêt réside en ce que ce théorème difficile pousse la sémantique dans ses retranchements : cadre infinitaire, points fixes reflétant les jeux de parité...et amène donc à étendre les outils existants, voire à en développer de nouveaux. C'est par exemple le cas de l'effondrement extensionnel [Ehr12b ; Ehr12a], qui relie sémantique infinitaire et finitaire, et que nous cherchons à étendre au cadre nécessaire pour le model-checking d'ordre supérieur. Nous soumettrons un premier article dans les mois à venir ; la phase exploratoire étant terminée, nous en sommes aux débuts de la rédaction.

1.2 Vérification des programmes fonctionnels *probabilistes*

L'étude du calcul probabiliste remonte aux débuts de l'informatique fondamentale [Lee+56 ; Rab63 ; San69]. Dans un calcul probabiliste, les algorithmes s'émancipent du déterminisme en utilisant des constructions stochastiques : on peut par exemple lancer une pièce et faire une action ou une autre

selon que le résultat est pile ou face. Le calcul probabiliste a été utilisé au cours du demi-siècle dernier pour donner des algorithmes plus efficaces [MR95], des primitives cryptographiques sécurisées [GM84]...il est également utilisé en linguistique [MS01], en robotique[Thr02], pour la description de modèles probabilistes [Goo+08 ; KF09 ; Tol+16]...

L'incorporation de l'aléa aux langages de programmation remonte à la fin des années 70 [SD78 ; Koz81]. Cependant, cette ligne de recherche s'est pleinement développée avec l'arrivée des langages de programmation bayésiens tels que Church ou Anglican [Goo+08 ; Tol+16], qui permettent en plus du choix probabiliste une opération de conditionnement (conditioning). A ce jour, l'étude mathématique de ces programmes s'est essentiellement cantonnée au cadre fonctionnel avec opérateur probabiliste, et sans conditionnement, ce qui pose déjà d'intéressants challenges. Par exemple, il a été extrêmement difficile de donner une sémantique dénotationnelle satisfaisante aux langages fonctionnels avec un simple choix probabiliste binaire (cf [JP89 ; JT98] et [ETP14 ; GL15] pour une résolution).

On conçoit alors que les questions de vérification de ces programmes fonctionnels avec choix probabiliste binaire soient particulièrement délicates. Une première question, naturelle et préliminaire, est celle de la terminaison quasi-sûre (AST) de ces programmes. En effet, la terminaison de toutes les exécutions probabilistes d'un programme de cette nature est extrêmement contraignante, et la terminaison quasi-sûre consiste à vérifier que les exécutions divergentes forment un ensemble de probabilité 0 : la divergence, si elle n'est pas impossible, est improbable.

Typage pour la terminaison quasi-sûre. Avec Ugo Dal Lago, nous avons étudié un langage fonctionnel défini sous forme d'un lambda-calcul avec choix binaire probabiliste, récursion, et les entiers comme type de données de base (avec pattern-matching), dans un travail publié à ESOP en 2017 [DG17] puis dans ACM TOPLAS en 2019 [LG19]. Nous avons donné un système de types pour ce langage, qui force les termes à être *affines* : une fois effectués les choix probabilistes, le programme ne peut pas appeler la fonction récursivement plusieurs fois. Ceci exclut tout particulièrement la composée $f(f(x)))$, qui est très délicate à modéliser.

Nous montrons alors qu'un terme typable est presque sûrement terminant, par une preuve de réalisabilité adaptée au cadre probabiliste. L'idée maîtresse est de modéliser les appels récursifs par des processus probabilistes, dans notre cas les OC-MDPs (one counter Markov decision processes). On généralise les *sized types*, qui permettent de garantir la terminaison dans le cadre déterministe : ceux-ci s'assurent, dans le cadre déterministe donc, que la fonction récursive termine sur un argument de taille 0, et est toujours appelée récursivement sur un argument plus petit. Dans notre cadre, les tailles des arguments appelés récursivement, avec la probabilité de ces appels, forment un processus probabiliste que nous décrivons avec les OC-MDPs. Ceux-ci, couplés au typage et à l'argument de réalisabilité, permettent d'obtenir ce système de typage garantissant la terminaison quasi-sûre pour les programmes probabilistes affines.

Depuis notre travail, et toujours pour la terminaison quasi-sûre des langages fonctionnels, on remarquera notamment [KRO21 ; DLFR21].

Schémas de récursion d'ordre supérieur *probabilistes*. Avec Kobayashi et Dal Lago, nous avons présenté à LICS 2019 [KLG19] et dans LMCS 2020 [KLG20] les schémas de récursion d'ordre supérieur *probabilistes* (PHORS). On rajoute aux HORS une opération de choix binaire probabiliste, en définissant une réécriture appropriée. Le but de ce travail préliminaire est de faire de la vérification de terminaison et non de la vérification plus générale : on ne génère donc plus d'arbres, mais simplement un symbole de terminaison ou de divergence, si la branche probabiliste que la réduction suit termine.

Le but est alors de déterminer automatiquement si la réduction probabiliste donnée par un PHORS termine avec probabilité 1 ou, plus généralement, de calculer aussi précisément que possible cette probabilité de convergence.

La terminaison avec probabilité 1 (AST) est indécidable dès qu'une mesure de la complexité du PHORS, appelée son ordre, est supérieure ou égale à 2 (ce qui est le cas dans la grande majorité des exemples intéressants). Nous donnons cependant une façon d'approximer les probabilités de terminaison qui, bien que probablement incomplète, nous a permis de développer un outil pratique calculant rapidement des probabilités de terminaison approximées pour des schémas d'ordre 2, malgré le résultat d'indécidabilité ci-dessus.

Cette approche diffère de la précédente : ici, on considère un langage plus restreint, celui des PHORS ; mais on vise à donner des résultats sur tous ces schémas probabilistes. Dans l'autre approche, le langage est plus expressif, mais on a simplement un résultat disant que les programmes typables sont AST : on ne considère que le fragment typable des termes, et on n'a pas cherché de résultat de décidabilité pour AST.

1.3 Autour des logiques modales

Les modalités permettent de nuancer l'expression de la vérité dans une logique de base. Nous nous intéresserons ici à des extensions modales de la logique propositionnelle, typiquement avec deux modalités $\Box$ et $\diamond$. On peut ainsi exprimer de nombreuses notions : $\Box\,\phi$ peut signifier que la formule ϕ est très probablement vraie ou, sur un système de transitions, qu'elle est vraie sur tous les successeurs de l'état courant. On peut également, dans les logiques dites déontiques, spécifier l'obligation qu'a un agent de satisfaire une formule : $\Box_A\,\phi$ signifie alors que l'agent A doit, au sens de l'obligation morale, satisfaire la formule ϕ. On voit sur cet exemple que la logique modale est fortement reliée à la philosophie ; c'est notamment pour cette raison que certaines de nos publications sont en philosophie [DGO20] ou dans des conférences mêlant mathématiques, informatique et philosophie [DGO21a].

Trois traditions coexistent : la plus courante est celle de la logique modale classique, qui se base sur la logique propositionnelle classique. La logique modale intuitionniste définit un analogue intuitionniste des logiques modales classiques, à la suite de Simpson [Sim94]. La troisième version est celle des logiques modales constructives, motivées par le lien avec l'informatique théorique, et notamment étudiées dans l'optique d'obtenir des correspondances à la Curry-Howard ; elles sont aussi étudiées pour leur sémantique, pour la vérification et pour la représentation de connaissances. Notre travail s'appuie sur ces trois approches.

Un cadre général pour les logiques intuitionnistes non-normales. Dans la tradition de la logique modale, la grande majorité des travaux suppose que plusieurs axiomes de base sont satisfaits. Nous étudions ici les logiques dites non-normales, qui sont plus minimales en ce qu'elles ne requièrent pas tous axiomes. Il en existe un certain nombre : nous identifions 8 logiques modales classiques non-normales, et 24 logiques modales intuitionnistes non-normales. Nous donnons un cadre unifié pour ces logiques : axiomatisation à la Hilbert, sémantique, calcul des séquents avec élimination des coupures, et décidabilité des logiques. Il s'avère que ce cadre capture aussi deux logiques constructives déjà connues et étudiées, dont Constructive K (CK) et le fragment propositionnel de la logique de Wijesekera appelée *Constructive Concurrent Dynamic Logic (CCDL)* (qui s'applique à la vérification). Ce travail unifiant les logiques modales non-normales a été publié dans le Journal of Philosophical Logic en 2020 [DGO20].

Calculs terminants et extraction de contre-modèles pour les logiques modales constructives. Nous reprenons les logiques constructives CK et CCDL introduites à la fin du paragraphe précédent. Nous donnons des calculs terminants pour ces logiques, possédant une procédure d'élimination des coupures, et donnant lieu à une procédure de décision pour ces logiques. Nous présentons ensuite des calculs de *réfutation* pour ces deux logiques : une formule est fausse si et seulement si on peut la prouver dans ces calculs. Ces réfutations permettent l'extraction automatique de contre-modèles : si une formule est fausse, on peut calculer automatiquement une sémantique finie qui en témoigne. Ce travail a été publié à la conférence TABLEAUX 2021 [DGO21b].

Systèmes de preuve pour des logiques "faire en sorte que". Elgesem a introduit une logique dans laquelle les modalités expriment deux types de capacité : $\Box_A\,\phi$ signifie que A peut réaliser ϕ alors $\diamond_A\,\phi$ signifie que A peut faire en sorte que ϕ soit vraie. On peut ainsi exprimer, par exemple, la délégation : A peut faire en sorte que B réalise une action. Nous considérons aussi une extension, due à Troquard, à des groupes d'agents : par exemple, A, B et C peuvent se réunir pour réaliser ϕ ou déléguer ψ à D. Nous donnons une nouvelle sémantique pour cette logique, qui est plus adaptée à l'extraction automatique de contre-modèles que celles qui préexistaient dans la littérature, couplée à un système d'hyperséquents. La combinaison de ces outils donne une procédure effective de construction de contre-modèle pour les logiques d'Elgesem et de Troquard. Ce travail a été publié à DEON 2020/2021 [DGO21a].

En cours. La logique déontique exprime quatre notions via des modalités : l'obligation, l'interdiction, la permission et le facultatif. Elle est à ce titre liée au raisonnement philosophique et juridique. Nous avons commencé à en étudier la version intuitionniste, dans l'esprit de nos travaux récents. Il s'agira là encore d'étudier axiomatisation, calcul des séquents, sémantique, et preuve : procédures de décision donnant une preuve d'une formule vraie, ou un contre-modèle d'une formule fausse.

1.4 Autres travaux et aspects administratifs

Communication scientifique. J'ai donné 65 exposés depuis mon premier séminaire en stage de L3, en 2009. J'ai notamment été invité à parler à NII Shonan et au mois thématique Automata, Logic and Games à Singapour (deux exposés). Le détail de ces interventions est donné au Chapitre **??**.

Dimension internationale. J'ai effectué des stages en Finlande (Turku, avec Juhani Karhumäki) et à Oxford (avec Luke Ong), où j'ai également fait plusieurs visites par la suite. J'ai été invité à Aarhus, Salerne, Turku (quelques années après mon stage), Bologne (après mon post-doctorat). Ces visites ont duré une à deux semaines. J'ai été invité à une rencontre NII Shonan sur la vérification des programmes fonctionnels, et au mois thématique Automata, Logic and Games à Singapour (cinq semaines avec un financement pour post-doctorant). J'étais invité à Tokyo en avril 2019 pour une semaine dans le groupe d'Ichiro Hasuo, mais mon fils aîné est né très prématurément 8h avant le départ, d'où une annulation de dernière minute. Avec le covid, ma mobilité a été réduite, mais je compte reprendre sous peu les déplacements afin de disséminer ma rechercher et de mener des collaborations fructueuses.

Organisation de séminaires. J'ai commencé à organiser des séminaires lors de mes années d'étude à Paris. Le groupe de travail des étudiants en logique se réunissait chaque semaine de 2010 à 2012 à l'ENS Ulm. Je faisais partie des organisateurs, et y ai donné sept exposés (non comptés dans les 65 exposés plus institutionnels dont il est fait état précédemment).

J'ai ensuite organisé en 2013 le groupe de lecture Schémas, Automates, Sémantique à PPS (aujourd'hui IRIF ; audience de 5 à 10 personnes) qui est ensuite devenu le séminaire joint "Sémantique et Vérification" au LIAFA-PPS (qui ont fusionné depuis pour donner l'IRIF), et qui réunissait une vingtaine de personnes de 2014 à 2015.

Du printemps 2020 à l'automne 2021, j'ai co-organisé, notamment avec Nathanaël Fijalkow (LaBRI), le séminaire en ligne YR-OWLS qui proposait deux fois par mois, sur invitation, à un jeune chercheur de présenter ses travaux à une audience internationale (en général 50 à 60 connectés).

Administration. Il me tient à coeur de participer à la vie institutionnelle de la tutelle d'enseignement, mais également du laboratoire. J'ai ainsi été représentant élu des doctorants au conseil de mon école doctorale (2014-2016), et suis membre élu du conseil de mon laboratoire depuis mars 2018.

Conférences et journaux. J'ai été membre des comités de programme des workshops internationaux DICE-FOPARA 2017 (8th Workshop on Developments in Implicit Computational complExity and 5th Workshop on Foundational and Practical Aspects of Resource Analysis) et ITRS (Intersection Types and Related Systems) 2018.

J'ai participé au comité d'organisation de STACS 2022.
J'ai été reviewer de 31 articles, pour :
- les conférences TCS (2014), ICFP (2015), MFPS (2015), CSL (2015, 2016), FoSSaCS (2016, 2017, 2018), FSTTCS (2017), LICS (2016, 2017, 2020, 2021), POPL (2018), FSCD (2016, 2017, 2018), MFCS (2017, 2018), KR (2018), CONCUR (2021), ICFP (2022)
- les workshops LCC (2016), DICE-FOPARA (2017), ITRS (2018, 2019)
- les journaux LMCS (2017), ACM TOPLAS (2019), Bulletin of Symbolic Logic (2020), MSCS (2020)

ANRs. Je suis membre de cinq projets ANR en cours :
- PPS : sémantique des programmes probabilistes
- ReciProg : autour des preuves circulaires
- LambdaComb : questions d'énumération autour du lambda-calcul
- TICAMORE : autour des logiques modales et de leurs calculs
- THEME : ANR de pédagogie : création de contenu hybride pour les formations MIAGE de France.

Citations personnelles

[CGM18] Pierre Clairambault, Charles Grellois et Andrzej S. Murawski. “Linearity in higher-order recursion schemes”. In : *Proc. ACM Program. Lang.* 2.POPL (2018), 39:1-39:29 (cf. p. 5).

[DG17] Ugo Dal Lago et Charles Grellois. “Probabilistic Termination by Monadic Affine Sized Typing”. In : *ESOP 2017*. 2017 (cf. p. 6).

[DGO20] Tiziano Dalmonte, Charles Grellois et Nicola Olivetti. “Intuitionistic Non-normal Modal Logics: A General Framework”. In : *J. Philos. Log.* 49.5 (2020), p. 833-882 (cf. p. 7).

[DGO21a] Tiziano Dalmonte, Charles Grellois et Nicola Olivetti. “Proof Systems for the Logics of Bringing-It-About”. In : *Deontic Logic and Normative Systems - 15th International Conference, DEON 2020/21, Munich, Germany [virtual], July 21-24, 2021*. Sous la dir. de Fenrong Liu, Alessandra Marra, Paul Portner et Frederik Van De Putte. College publications, 2021, p. 114-132 (cf. p. 7).

[DGO21b] Tiziano Dalmonte, Charles Grellois et Nicola Olivetti. “Terminating Calculi and Countermodels for Constructive Modal Logics”. In : *Automated Reasoning with Analytic Tableaux and Related Methods - 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6-9, 2021, Proceedings*. Sous la dir. d’Anupam Das et Sara Negri. T. 12842. Lecture Notes in Computer Science. Springer, 2021, p. 391-408 (cf. p. 7).

[GM15a] Charles Grellois et Paul-André Melliès. “An Infinitary Model of Linear Logic”. In : *FoSSaCS ’15*. Sous la dir. d’Andrew M. Pitts. T. 9034. LNCS. Springer, 2015, p. 41-55 (cf. p. 4).

[GM15b] Charles Grellois et Paul-André Melliès. “Finitary Semantics of Linear Logic and Higher-Order Model-Checking”. In : *MFCS ’15*. Sous la dir. de Giuseppe F. Italiano, Giovanni Pighizzini et Donald Sannella. T. 9234. LNCS. Springer, 2015, p. 256-268 (cf. p. 4, 5).

[GM15c] Charles Grellois et Paul-André Melliès. “Indexed linear logic and higher-order model checking”. In : *ITRS ’14*. Sous la dir. de Jakob Rehof. T. 177. EPTCS. 2015, p. 43-52 (cf. p. 4).

[GM15d] Charles Grellois et Paul-André Melliès. “Relational Semantics of Linear Logic and Higher-order Model Checking”. In : *CSL ’15*. Sous la dir. de Stephan Kreutzer. T. 41. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, p. 260-276 (cf. p. 4, 5).

[KLG19] Naoki Kobayashi, Ugo Dal Lago et Charles Grellois. “On the Termination Problem for Probabilistic Higher-Order Recursive Programs”. In : (2019), p. 1-14 (cf. p. 6).

[KLG20] Naoki Kobayashi, Ugo Dal Lago et Charles Grellois. “On the Termination Problem for Probabilistic Higher-Order Recursive Programs”. In : *Log. Methods Comput. Sci.* 16.4 (2020) (cf. p. 6).

[LG19] Ugo Dal Lago et Charles Grellois. “Probabilistic Termination by Monadic Affine Sized Typing”. In : *ACM Trans. Program. Lang. Syst.* 41.2 (2019), 10:1-10:65 (cf. p. 6).

Références

[Bro+10] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong et Olivier Serre. "Recursion Schemes and Logical Reflection". In : *LICS '10*. IEEE Computer Society, 2010, p. 120-129 (cf. p. 4).

[CES86] Edmund M. Clarke, E. Allen Emerson et A. Prasad Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". In : *ACM Trans. Program. Lang. Syst.* 8.2 (1986), p. 244-263 (cf. p. 3).

[CS12] Arnaud Carayol et Olivier Serre. "Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection". In : *LICS '12*. IEEE Computer Society, 2012, p. 165-174 (cf. p. 4).

[DLFR21] Ugo Dal Lago, Claudia Faggian et Simona Ronchi Della Rocca. "Intersection Types and (Positive) Almost-Sure Termination". In : *Proc. ACM Program. Lang.* 5.POPL (2021) (cf. p. 6).

[Ehr12a] Thomas Ehrhard. "Collapsing non-idempotent intersection types". In : *CSL '12*. Sous la dir. de Patrick Cégielski et Arnaud Durand. T. 16. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, p. 259-273 (cf. p. 5).

[Ehr12b] Thomas Ehrhard. "The Scott model of linear logic is the extensional collapse of its relational model". In : *Theor. Comput. Sci.* 424 (2012), p. 20-45 (cf. p. 5).

[ETP14] Thomas Ehrhard, Christine Tasson et Michele Pagani. "Probabilistic Coherence Spaces Are Fully Abstract for Probabilistic PCF". In : *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA : Association for Computing Machinery, 2014, 309–320 (cf. p. 6).

[Gir87] Jean-Yves Girard. "Linear Logic". In : *Theor. Comput. Sci.* 50 (1987), p. 1-102 (cf. p. 4).

[GL15] Jean Goubault-Larrecq. "Full abstraction for non-deterministic and probabilistic extensions of PCF I: The angelic cases". In : *Journal of Logical and Algebraic Methods in Programming* 84.1 (2015). Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011, p. 155-184 (cf. p. 6).

[GM84] Shafi Goldwasser et Silvio Micali. "Probabilistic Encryption". In : *J. Comput. Syst. Sci.* 28.2 (1984), p. 270-299 (cf. p. 6).

[Goo+08] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz et Joshua B. Tenenbaum. "Church: a language for generative models". In : *UAI 2008*. Sous la dir. de David A. McAllester et Petri Myllymäki. AUAI Press, 2008, p. 220-229 (cf. p. 6).

[Hag+08] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong et Olivier Serre. "Collapsible Pushdown Automata and Recursion Schemes". In : *LICS '08*. IEEE Computer Society, 2008, p. 452-461 (cf. p. 4).

[HL17] Martin Hofmann et Jérémy Ledent. "A cartesian-closed category for higher-order model checking". In : *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, p. 1-12 (cf. p. 5).

[JP89] C. Jones et G.D. Plotkin. "A probabilistic powerdomain of evaluations". In : *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. 1989, p. 186-195 (cf. p. 6).

[JT98] Achim Jung et Regina Tix. "The Troublesome Probabilistic Powerdomain". In : *Electronic Notes in Theoretical Computer Science* 13 (1998). Comprox III, Third Workshop on Computation and Approximation, p. 70-91 (cf. p. 6).

[KF09] Daphne Koller et Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009 (cf. p. 6).

[KO09] Naoki Kobayashi et C.-H. Luke Ong. "A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes". In : *LICS '09*. 2009 (cf. p. 4).

[Koz81] Dexter Kozen. "Semantics of probabilistic programs". In : *Journal of Computer and System Sciences* 22.3 (1981), p. 328-350 (cf. p. 6).

[Kre15] Stephan Kreutzer, éd. *CSL '15*. T. 41. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[KRO21] Andrew Kenyon-Roberts et C.-H. Luke Ong. "Supermartingales, Ranking Functions and Probabilistic Lambda Calculus". In : *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA : Association for Computing Machinery, 2021 (cf. p. 6).

[KTU10] Naoki Kobayashi, Naoshi Tabuchi et Hiroshi Unno. "Higher-order multi-parameter tree transducers and recursion schemes for program verification". In : *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*. Sous la dir. de Manuel V. Hermenegildo et Jens Palsberg. ACM, 2010, p. 495-508 (cf. p. 5).

[Lee+56] Karel de Leeuw, Edward F. Moore, Claude E. Shannon et Norman Shapiro. "Computability by probabilistic machines". In : *Automata Studies* 34 (1956), p. 183-212 (cf. p. 5).

[MR95] Rajeev Motwani et Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995 (cf. p. 6).

[MS01] Christopher D. Manning et Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001 (cf. p. 6).

[NRO12] Robin P. Neatherway, Steven J. Ramsay et C.-H. Luke Ong. "A traversal-based algorithm for higher-order model checking". In : *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*. Sous la dir. de Peter Thiemann et Robby Bruce Findler. ACM, 2012, p. 353-364 (cf. p. 5).

[Ong06] C.-H. Luke Ong. "On Model-Checking Trees Generated by Higher-Order Recursion Schemes". In : *LICS '06*. IEEE Computer Society, 2006, p. 81-90 (cf. p. 4).

[Pit15] Andrew M. Pitts, éd. *FoSSaCS '15*. T. 9034. LNCS. Springer, 2015.

[Rab63] Michael O. Rabin. "Probabilistic automata". In : *Information and Control* 6.3 (1963), p. 230-245 (cf. p. 5).

[San69] Eugene S. Santos. "Probabilistic Turing Machines and Computability". In : *Proc. of the AMS* 22.3 (1969), p. 704-710 (cf. p. 5).

[SD78] N. Saheb-Djahromi. "Probabilistic LCF". In : *Mathematical Foundations of Computer Science 1978*. Sous la dir. de J. Winkowski. Berlin, Heidelberg : Springer Berlin Heidelberg, 1978, p. 442-451 (cf. p. 6).

[Sim94] Alex K. Simpson. "The proof theory and semantics of intuitionistic modal logic". Thèse de doct. University of Edinburgh, UK, 1994 (cf. p. 7).

[SW13a] Sylvain Salvati et Igor Walukiewicz. "Evaluation is MSOL-compatible". In : *FSTTCS '13*. Sous la dir. d'Anil Seth et Nisheeth K. Vishnoi. T. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, p. 103-114 (cf. p. 5).

[SW13b] Sylvain Salvati et Igor Walukiewicz. "Using Models to Model-Check Recursive Schemes". In : *TLCA '13*. Sous la dir. de Masahito Hasegawa. T. 7941. LNCS. Springer, 2013, p. 189-204 (cf. p. 5).

[SW14] Sylvain Salvati et Igor Walukiewicz. "Krivine machines and higher-order schemes". In : *Inf. Comput.* 239 (2014), p. 340-355 (cf. p. 5).

[SW15a] Sylvain Salvati et Igor Walukiewicz. "A Model for Behavioural Properties of Higher-order Programs". In : *CSL '15*. Sous la dir. de Stephan Kreutzer. T. 41. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, p. 229-243 (cf. p. 4, 5).

[SW15b] Sylvain Salvati et Igor Walukiewicz. “Typing Weak MSOL Properties”. In : *FoSSaCS ’15*. Sous la dir. d’Andrew M. Pitts. T. 9034. LNCS. Springer, 2015, p. 343-357 (cf. p. 5).

[Thr02] Sebastian Thrun. “Robotic Mapping: A Survey”. In : *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002 (cf. p. 6).

[TK14] Takeshi Tsukada et Naoki Kobayashi. “Complexity of Model-Checking Call-by-Value Programs”. In : *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. Sous la dir. d’Anca Muscholl. T. 8412. Lecture Notes in Computer Science. Springer, 2014, p. 180-194 (cf. p. 5).

[TO14] Takeshi Tsukada et C.-H. Luke Ong. “Compositional higher-order model checking via ω-regular games over Böhm trees”. In : *CSL-LICS ’14*. Sous la dir. de Thomas A. Henzinger et Dale Miller. ACM, 2014, 78:1-78:10 (cf. p. 4).

[Tol+16] David Tolpin, Jan-Willem van de Meent, Hongseok Yang et Frank D. Wood. “Design and Implementation of Probabilistic Programming Language Anglican”. In : *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages, IFL 2016, Leuven, Belgium, August 31 - September 2, 2016*. Sous la dir. de Tom Schrijvers. ACM, 2016, 6:1-6:12 (cf. p. 6).

[Wal19] Igor Walukiewicz. “Lambda Y-Calculus With Priorities”. In : *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, p. 1-13 (cf. p. 4).