DOSSIER DE CANDIDATURE
APPLICATION

Cochez le concours sur lequel vous candidatez
Check the position on which you are applying

☒ **CR2 - (Chargés de recherche de deuxième classe / *Young graduate scientist position*)**

☐ **CR1 - (Chargés de recherche de première classe / *Young experienced scientist position*)**

☐ **DR2 - (Directeurs de recherche de deuxième classe / *Senior researcher position*)**

Nom[1] **:** Scherer
Last name

Prénom : Gabriel
First name

Sexe : ☐**F** ☒**M**
Sex

Nom utilisé pour vos publications (facultatif) :
Name used for your publications (optional):

[1] Il s'agit du nom usuel figurant sur vos pièces d'identité
It is the name appearing on your identity cards

SOMMAIRE / ***SUMMARY***

Formulaire 1 — PARCOURS PROFESSIONNEL
Form 1 — PROFESSIONAL HISTORY

1) Parcours Professionnel / *Professionnal history*

1. Situation professionnelle actuelle / *Current professionnal status*

Statut et fonction[2] / *Position and statute*[2]: Postdoctoral Research Associate
Établissement (ville - pays) / *Institution (city -country)*: Northeastern University (Boston, USA)
Date d'entrée en fonction / *Start*: January 11th, 2016
[ ] Sans emploi / *Without employment*

2. Expériences professionnelles antérieures / *Previous professionnal experiences*

Date début *Start*	Date fin *End*	Établissement *Institutions*	Fonction et statut[2] *Positions and status*[2]
Septembre 2008	Juin 2010	Lycée Louis-le-Grand	colleur en informatique
Septembre 2011	Juin 2012	idem	idem
Septembre 2013	Février 2015	moniteur	Université Paris 7

Nombre d'années d'exercice des métiers de la recherche après la thèse / *Number of years of professionnal research experience after thesis*: 1.5

2) Interruptions de carrière/*Career breaks*

Date début *Start*	Date fin *End*	Motif de l'interruption/*Reason for interruption*
.....	.....	.....
.....	.....	.....
.....	.....	.....

3) Prix et distinctions / *Prizes and awards*

ICFP Programming Contest 2014: Jury's prize, in the Gagallium team with François Bobot, Pierre Boutillier, Thomas Braibant and Damien Doligez.

4) Encadrement d'activités de recherche / *Supervision of research activities*

.....

5) Responsabilités collectives / *Responsibilities*

Organisateur (avec Max New) de NEPLS (New England Programming Language Seminar) d'automne 2016 – événement régional d'une journée.
Membre du comité de programme du workshop OCaml 2016.
PC Chair pour le workshop OCaml 2017 – workshop international hébergé par ICFP.
Membre du **comité de programme** de ICFP 2017.
Relecture d'articles de conférence (17) : ESOP 2012, POPL 2014, postML 2014, POPL 2015 (2), JFLA 2015, ICFP 2015 (2), ICTACT 2015, LICS 2016 (3), OOPSLA 2016 (4), Fossacs 2017 (1).
Relecture d'articles de journal (2): MSCS (2).

6) Management / *Management*

.....

[2] Indiquer avec précision chaque situation statutaire. Par exemple : pour une situation d'agent titulaire de la fonction publique, préciser le corps et le grade de rattachement, pour une situation de salarié.e du secteur privé ou d'agent non titulaire d'un établissement public, préciser la nature du contrat salarial, etc.

[2] *For each position, indicate grade or rank. For example, for a tenured civil servant position, indicate the branch and rank, for a private sector position or non-tenured position, indicate the nature of the work contract, etc.*

7) Collaborations, mobilité / *Collaborations, visits*

2011: Stage de recherche de 5 mois avec Andreas Abel à Munich (LLudwig-Maximilian Universität), “Universe subtyping in Martin-Löf Type Theory”, co-rédaction d’un article journal (LMCS), “On Irrelevance and Algo- rithmic Equality in Predicative Type Theory”.

2011+: Collaboration à distance avec l’équipe de développement de l’implémentation du langage de programmation OCaml. Collaborations spécifiques avec Alain Frisch en 2011 (compilation des tuples), Wojciech Meyer en 2013 (ocaml-build), Benoît Vaugon en 2014 (formats et GADTs), Florian Angeletti (manuel et ocamldoc) et Bernhard Schommer (arguments de ligne de commande passés dans un fichier) en 2016.

2013: Collaboration à distance avec Jan Hoffman à la suite d’une visite courte au laboratoire FLINT dirigé par Zhong Shao à Yale. Co-rédaction d’un article de conférence (LPAR), “Tracking Data-Flow with Open Closure Types”.

2014+: Collaboration régulière avec Guillaume Munch-Maccagnoni sur les liens entre nos travaux. Co-rédaction d’un article de conférence (LICS) en 2015, “Polarised Intermediate Representation of Lambda Calculus with Sums”.

2016+: Post-doc à Northeastern University, Boston, sous la supervision d’Amal Ahmed. Collaboration régulière avec Amal et ses étudiants, ainsi que Jan Vitek et ses étudiants.

8) Enseignement / *Teaching*

2008-2009, 2009-2010, 2012: colles de Caml Light en classe préparatoire (MPSI), Lycée Louis-le-grand. Rédaction de sujets destinés à enseigner la programmation aux élèves, en complément d’un cours d’informatique plus théorique. 4 heures par semaine pendant toute l’année scolaire.

2013-2014, 2014-2015: moniteur à l’université Paris 7 – deux services de 72 heures. Enseignement de Java en L1 (deux ans). Enseignement C en L3 et OCaml en M1 (Programmation Fonctionnelle Avancée), un an.

9) Diffusion de l’information scientifique / *Dissemination of scientific knowledge*

2007+: Rédaction d’articles de vulgarisation sur des sites d’informatique francophones grand public (Site du Zéro et LinuxFR): John Backus et programmation fonctionnelle (2007), Robin Milner (2010) (article cité dans la revue de presse du site Images des Mathématiques du CNRS), le langage Scala (2011), le langage OCaml (2016), son extension MetaOCaml (2016) et enfin LLVM (2016).

2012-2015: Lancement, co-rédaction et édition du blog Gagallium de l’EPI Gallium. 31 articles rédigés sur les 85 articles publiés, 500 visites par jour en moyenne.

2016: Article (rémunéré) OCaml 4.03 sur le magazine technique en ligne Linux Weekly News (LWN.net).

2016-2017: Lancement et co-rédaction du blog PRL blog du Programming Research Lab de Northeastern. 8 articles rédigés sur les 20 publiés.

10) Eléments divers / *Other relevant information*

. . . . . .

Formulaire 2 — DESCRIPTION SYNTHÉTIQUE DE L'ACTIVITÉ ANTÉRIEURE
Form 2 — SUMMARY OF YOUR PAST ACTIVITY

Nom / *Last name*: Scherer
Prénom/*First name*: Gabriel

Travaux de thèse J'ai étudié pendant ma thèse, encadrée par Didier Rémy dans l'EPI Gallium, une question théorique motivée par d'inférence ou synthèse de code dans les langages typés : quels sont les types qui ont un habitant unique ? Pour cela, j'ai dû combiner des techniques de la théorie des langages de programmation avec des outils de théorie de la démonstration. J'ai démontré des liens jusque-là inconnus entre ces deux domaines dans un article à TLCA 2015, et proposé un algorithme qui décide la question de l'unicité pour les types simples à ICFP 2015 (version journale soumise à l'été 2016).
Au début de mon post-doc, j'ai étendu cette approche pour obtenir un résultat imprévu : un algorithme de décision de l'équivalence de programmes purs en présence des types simples (fonctions, paires, sommes) et du *type vide*, représentant les cas impossibles ou branches mortes d'un programme. Le type vide rend l'équivalence difficile (dans une branche morte, tous les sous-programmes sont équivalents). La question de la décidabilité de cette équivalence était ouverte depuis au moins 1995[1]. Ce travail a été présenté à POPL 2017.
En parallèle, j'ai travaillé sur des problèmes théoriques de conception de langages de programmation : sur les types dépendants avec Andreas Abel (publication dans le journal LMCS en 2012), sur les machines abstraites, en collaboration avec Guillaume Munch-Maccagnoni (publication à LICS 2015), sur la réalisabilité classique avec Pierre-Évariste Dagand (publication aux JFLA 2015, version journal en préparation), sur les analyses de resources en programmation fonctionnelle avec Jan Hoffmann (LPAR 2013), et sur les abstractions de coercions et la cohérence avec Didier Rémy (ESOP 2015), à partir d'un travail de Julien Crétien.

Travail sur le langage OCaml Pendant la période de ma thèse, j'ai aussi eu l'occasion de travailler sur la théorie et l'implémentation du langage de programmation **OCaml**, logiciel libre développé en grande partie au sein de l'EPI Gallium. J'ai essayé en particulier de rendre le processus de développement plus ouvert, en encourageant les contributions externes et en accompagnant les contributeurs par des relectures de code, conseils d'améliorations, etc. Avant le début de ce travail, 11 développeurs différents avaient contribué à l'implémentation; il y en a eu 93 sur la période 2012-2016. Sur cette période j'étais le deuxième contributeur le plus actif à l'implémentation du langage (en nombre de changements effectués), et je suis le mainteneur de logiciels utilisés par une partie importante de la comunauté, la bibliothèque `Batteries` et l'outil de compilation `ocamlbuild`.
Mon travail sur le langage OCaml a aussi soulevé des questions de recherche fondamentale variées, dont une étude formelle d'une extension du système de typage (ESOP 2013), et une collaboration avec Luc Maranget sur un nouvel avertissement dans le compilateur de filtrage de motifs (ML workshop 2016, version longue en préparation).

Travaux en cours Je suis depuis Janvier 2016 en post-doc à Northeastern University, Boston, sous la direction d'Amal Ahmed. Je travaille principalement sur deux projets.
Le premier projet, avec Amal Ahmed, consiste à étudier la conception de systèmes de programmation multi-langages, où au lieu de concevoir un langage riche qui répond à tous nos besoins on conçoit plusieurs langages plus simples, complémentaires, que l'on fait interagir entre eux. Nous proposons une nouvelle façon de formaliser la propriété de "bonnes interactions" entre ces langages, qui capture l'absence de "fuites d'abstraction" d'un langage à un autre, une personne qui ne connaîtrait qu'un des langages du système peut raisonner sur les programmes qu'elle écrit avec seulement ses connaissances, et nous prouvons que tout raisonnement de cette forme reste valide quand on rajoute les autres langages.
Le second projet, en collaboration avec Amal Ahmed mais aussi Jan Vitek, consiste à étudier les optmisations spéculatives des compilateurs Just In Time. Une optimisation spéculative consiste à faire une hypothèse incertaine en optimisant un programme, en se réservant le droit d'annuler ce choix et de faire une nouvelle compilation moins optimiste par la suite. Cette approche est populaire aujourd'hui (utilisée dans les implémentations performantes de Smalltalk, Java, Lua, Javascript) mais les praticiens qui les implémentent ne savent pas toujours quelles transformations de programme sont valides dans ce contexte, en présence de ces hypothèses invalidables. Nous cherchons à concevoir un langage simplifié, compromis entre simplicité et réalisme/représentativité, pour étudier la correction de ces transformations et guider ainsi les implémentations existantes.

[1] TLCA list of Open Problems, Problem 15 (Alex Simpson, 1995)

Formulaire 3 — CONTRIBUTIONS MAJEURES
Form 3 — MAJOR CONTRIBUTIONS

Nom / *Last name*: Scherer
Prénom/*First name*: Gabriel

Taille maximum de cette partie/*Maximum size for this part:*

- **CR2 : 3 fiches**
  Taille maximum de cette partie : 3 pages
  Maximum size for this part: 3 pages

- **CR1 : 3 fiches**
  Taille maximum de cette partie : 3 pages
  Maximum size for this part: 3 pages

- **DR2 : 5 fiches**
  Taille maximum de cette partie : 5 pages
  Maximum size for this part: 5 pages

Remplir une fiche par contribution majeure.

Il peut s'agir d'une contribution scientifique donnant lieu à un ensemble de publications (voire une publication majeure), ou à un développement technologique (logiciel ou autre), d'une action de transfert industriel, d'une responsabilité collective, d'une activité d'animation d'une communauté de recherche, ou tout autre élément relevant des missions d'une chercheuse ou d'un chercheur. Les critères importants sont la créativité, l'originalité et l'impact. Chaque fiche suivra le plan indiqué ci-dessous. Dans l'ensemble du texte, penser à donner, le cas échéant, les références permettant de consulter sur le Web les documents mentionnés (articles, thèses, logiciels, etc.).

Fill in one form for each major contribution.

It can be a scientific contribution expressed through a set of publications (or a single major publication) or through a technology development (software or hardware or other); it can also be an industrial transfer, a participation to the management of research or to the animation of a scientific community, or any other element. The main criteria are creativity, originality and impact. Each form should follow the guidelines given below. In the body of the text, give the Web references for quoted documents (articles, dissertations, software, etc.), if available.

Fiche 1 : Équivalence et unicité dans les langages de programmation à type sommes

1. Description de la contribution / *Description of the contribution*

Pour mieux comprendre les mécansimes d'inférence de code implicite dans les langages de programmation typés (surcharge, paramètres implicites, types classes), j'ai étudié la question suivante : quels sont les types qui ont un habitant unique, modulo l'équivalence de programme ? J'ai pu donner (ICFP 2016) une procédure de décision de cette question dans le cas des types "simples", qui représentent les types de données de base (fonctions, paires, sommes disjointes). J'ai ensuite étendu ce travail en ajoutant un traitement formel du type à un élément (1) et du type vide (0), ce qui a fourni un résultat inattendu, la décidabilité de l'équivalence en présence du type vide (POPL 2017), qui était un problème ouvert depuis au moins 1995.
Cette travail m'a aussi donné l'occasion de contribuer à la compréhension théorique des types sommes, du point de vue logique avec le multi-focusing (TLCA 2015) et, en collaboration avec Guillaume Munch-Maccagnoni, en utilisant les machines abstraites polarisées (LICS 2015).

2. Contribution personnelle de la candidate ou du candidat / *Personal contribution of the applicant*

C'est en grande majorité un travail personnel. L'étude de l'unicité pour les types simples a été faite pendant ma thèse, en collaboration avec mon directeur Didier Rémy.
J'ai fait seul mon travail sur le multi-focusing maximal, qui montre des liens très forts entre des techniques de preuve en théorie de la démonstration (l'étude du multi-focusing maximal en logique linéaire) et des algorithmes utilisés pour décider l'équivalence de programmes fonctionnels avec sommes. Cela m'a ensuite donné l'occasion de collaborer avec Guillaume Munch-Maccagnoni pour une présentation de ce travail sur dans le cadre des machines abstraites polarisées qu'il a développé pendant sa thèse. Notre article à LICS 2015 est partagé en deux moitiés : dans la première, Guillaume fait une exposition générale des machines abstraites présentées dans sa thèse, dans la seconde j'utilise cette présentation pour proposer un nouvel algorithme, plus simple et plus régulier, pour décider l'équivalence avec sommes, et je discute des liens avec le multi-focusing maximal.
Le résultat sur les types vides a été obtenu seul, au début de mon post-doc.

3. Originalité et difficulté / *Originality and difficulty*

La question de "quels sont les types qui ont un habitant unique" a été étudiée de façon parcellaire depuis 1981 (Mints); les résultats connus à ce jour étaient des conditions suffisantes (si un type est de cette forme, alors ses habitants sont uniques), uniquement sur les types de fonctions (pas de paires et de sommes), obtenus en 1982 (Babev et Soloviev), 1994 (Aoto et Ono), 1999 (Aoto) et 2005 (Broda et Damas). Le système de types que j'étudie est plus riche (paires et sommes, nécessaires pour envisager des applications pratiques aux langages de programmation), et l'approche est différente : c'est la première fois qu'une procédure de décision est proposée (et implémentée).
Ce qui distingue clairement mon travail, c'est l'idée utiliser le *focusing*, outil de la théorie de la démonstration. On peut voir cela comme une nouvelle façon d'appliquer la correspondance de Curry-Howard entre preuves et programmes: les travaux sur la *recherche de preuve* sont ré-interprétés comme fournissant des représentations *canoniques* des programmes.
L'équivalence de programmes purs en présence de types sommes est connue comme un problème difficile, résolu (preuve de décidabilité) pour la première fois en 1995, mais encore étudié dans les années 2000, principalement sous l'angle de deux familles d'approches, la *réécriture* et de la *normalisation par évaluation*. En présence du type vide, le problème de décidabilité était resté ouvert. La preuve de décidabilité est un résultat majeur qui intéresse la communauté du lambda-calcul typé; pour moi, c'est un signe que les nouveaux outils proposés (le *focusing*) sont prometteurs et permettront une meilleure compréhension d'autres problèmes du domaine, complémentant les deux approches existantes.
Maintenant que l'équivalence sur les types de données est mieux comprise, j'espère pouvoir étudier des applications pratiques de l'équivalence de programme aux langages de programmation. Aujourd'hui il y a beaucoup de travaux qui reposent sur le fait de décider automatiquement si un programme respecte une spécification, mais très peu sur l'équivalence entre programmes.

4. Validation et impact / *Validation and impact*

Mon projet pour l'étude des types à habitant unique était de développer une compréhension théorique du problème de l'inférence de code dirigée par les types, qui pourrait complémenter et dialoguer avec les utilisations en pratiques de cette méthode. Par exemple, mon travail propose des "représentations canoniques" des programmes; quand on cherche à parcourir l'espace des programmes possibles, au lieu d'énumérer les expressions syntaxiques possibles, on peut énumérer leurs formes canoniques. Cette forme d'impact apparaît doucement : la communauté qui travaille sur ce problème prend conscience de l'intérêt du *focusing*, et mon travail a été cité en particulier dans les travaux d'un groupe à Penn et Princeton (Peter Michael-Osera, Steve Zdancewic, David Walker).

Mon travail avec Guillaume Munch-Maccagnoni a apporté des améliorations à la présentation des machines abstraites polarisées qu'il a pu réutiliser dans ses travaux ultérieurs, par exemple son travail à POPL 2016 avec Marcelo Fiore et Pierre-Louis Curien.
Pour l'importance de la décidabilité de l'équivalence de programme en présence de types vides, on pourra contacter Alex Simpson <Alex.Simpson@fmf.uni-lj.si>, qui a étudié le problème, et l'a signalé comme un problème ouvert intéressant à la communauté, en 1995.

5. Diffusion / *Dissemination*

Publications: TLCA 2015, ICFP 2015, LICS 2015, POPL 2017. Implémentation: https://gitlab.com/gasche/unique-inhabitant
Exposés publics sur le sujet: 4 exposés dans des conférences internationales, 2 exposés dans des workshops internationaux, 6 exposés dans des équipes de recherche, un exposé aux journées des GdT Lac-GeoCal 2015.
Trois billets de blog (destinés à la communauté scientifique): un, deux, trois.

Fiche 2 : Développement du langage OCaml

1. Description de la contribution / *Description of the contribution*

Je fais partie du groupe qui développe le langage OCaml.
Sur le plan pratique, je contribue au développement de l'implémentation du langage et d'outils de l'écosystème qui l'entoure – je suis le troisième développeur de l'implémentation le plus actif depuis 2012, le mainteneur de l'outil `ocamlbuild` utilisé par des centaines de programmeurs, et je participe activement aux discussions sur son évolution et en particulier l'intégration de contributions externes.
Sur le plan théorique, j'ai poursuivi des recherches motivées par les besoins du langage ou les questions théoriques soulevées par son évolution, dont certains résultats ont été présentés dans des workshops et conférences internationales de bon niveau (OCaml Workshop, ESOP). Je participe aux discussions scientifiques plus larges sur la conception de langages fonctionnels typés; en particulier, je suis en contact régulier avec les concepteurs de Haskell et Rust pour discuter des questions de conception s'appliquant à plusieurs langages.

2. Contribution personnelle de la candidate / du candidat / *Personal contribution of the candidate*

Le langage OCaml, né à l'INRIA, a aujourd'hui un développement fortement collaboratif où Xavier Leroy et Damien Doligez (EPI Gallium) occupent une place centrale. Je fais partie des 16 développeurs ayant les droits de modification sur le répertoire central; sur la période où j'ai participé au développement du langage, je suis le troisième développeur le plus actif en terme de nombre de changements. Je participe régulièrement aux discussions collectives sur l'évolution du langage, en ligne ou, pendant ma thèse, dans des réunions physiques organisées par Gallium.
Mon axe de contribution principal a été de faire évoluer le processus de développement pour le rendre plus ouvert aux contributions extérieures. OCaml avait il y a quelques années la réputation d'être un projet au développement assez fermé, accueillant tièdement les contributions externes pour améliorer le langage ou son implémentation. J'ai proposé l'utilisation de la plateforme de développement Github pour faciliter les contributions extérieures, et fourni une partie significative du travail de revue, de retours et d'intégration sur ces contributions, pour les encourager.
Ce travail de développement est constitué de beaucoup de petis ajouts ou correctifs indépendants les uns des autres. J'ai aussi travaillé sur des contributions de plus grandes ampleur, souvent en collaboration avec d'autres contributeurs, tels que Alain Frisch en 2011 (compilation des tuples), Wojciech Meyer en 2013 (ocamlbuild), Benoît Vaugon en 2014 (formats et GADTs), Florian Angeletti (manuel et ocamldoc) et Bernhard Schommer (arguments de ligne de commande passés dans un fichier) en 2016. Je suis depuis devenu le mainteneur principal de l'outil `ocamlbuild` (utilisé par 1400 des 2600 projets OCaml disponibles publiquement). Dans les autres cas ma participation était surtout un accompagnement, peut-être un tiers du travail total – le plus gros était le travail sur les formats avec Benoît Vaugon, sur lequel j'ai passé près d'un mois équivalent-temps-plein. Dans le cas de Benoît Vaugon, Florian Angeletti et Bernhard Schommer, je pense pouvoir dire que leur travail ne serait pas intégré à l'implémentation aujourd'hui si je n'avais pas été présent pour le soutenir et l'accompagner.
Ma participation au développement du langage OCaml a soulevé des problématiques de recherche variées. J'ai travaillé avec Nicolas Pouillard à une proposition d'espaces de noms (namespaces) pour le langage, qui n'a pas été retenue, mais a permis de mieux comprendre les choix de conception. Mon étude avec Didier Rémy de la variance des types algébriques gardés a été présentée au ML Workshop en 2012 et publiée à ESOP 2013. Mon travail avec Luc Maranget sur les variables ambiguës dans un motif, présenté au ML workshop 2016, a une version longue en cours de rédaction. Je discute actuellement de ce travail avec les membres de la communauté Haskell, en particulier Ömer Sinan Ağacan, qui cherchent à ajouter à Haskell les fonctionnalités qui le rendent applicable.
Enfin, ma participation à la communauté du langage se manifeste aussi par une participation régulière à la `mailing-list` Caml, réponse aux questions sur des forums tels que `StackOverflow` ou `reddit`. J'ai fait plusieurs interventions au groupe d'utilisateurs OCaml de Paris (OUPS). En 2014, j'ai parlé du processus de développement du langage au OCaml Workshop, un workshop international affilié à ICFP. En 2016, j'ai été invité comme membre du comité de programme de workshop, et en 2017 je suis le *chair* en charge de son organisation.

3. Originalité et difficulté / *Originality and difficulty*

Faire évoluer un langage de programmation, dans sa conception, son implémentation et ses méthodes de contribution, demande à la fois des compétences techniques et sociales. Il faut comprendre les aspects techniques de l'implémentation, respecter et promouvoir des bonnes pratiques de développement logiciel (tests, intégration continue, traçabilité des changements...), être patient dans l'accompagnement des contributeurs, savoir quand laisser la décision à une personne plus experte, et surtout trouver l'équilibre délicat entre une évolution efficace et fructueuse et le conservatisme nécessaire pour garder un langage élégant, stable et cohérent.
Le résultat principal auquel j'ai contribué, c'est l'ouverture du développement à un plus grand nombre de contributeurs et contributrices externes. Entre 2006 et 2012, 11 développeurs différents, dont un seul ne faisait partie ni de l'équipe de développement ni de Gallium, ont apporté des changements et améliorations à l'implémentation principale. Sur la période

2012-2016, 93 personnes ont contribué. La perception du langage OCaml comme un projet fermé a majoritairement disparu aujourd'hui.

4. Validation et impact / *Validation and impact*

Le langage OCaml est utilisé comme langage d'implémentation de choix par une partie importante de la communauté de recherche en langages de programmation, en France comme à l'étranger. Le manuel d'utilisation du langage, utilisé comme référence, est cité dans 677 publications – bien que la plupart des travaux effectués en OCaml omettent cette citation.
Au cours de mon travail personnel j'ai été en contact régulier avec des membres de la communauté académique, par exemple Alan Schmitt, François Bobot ou Anil Madhavapeddy, tout particulièrement avec les développeurs du projet Coq (par exemple Hugo Herbelin et Matthieu Sozeau), et des utilisateurs industriels comme Jane Street (par exemple Leo White et Yaron Minsky), et enfin avec les mainteneurs du langage, en particulier Damien Doligez et Xavier Leroy. Chacun d'entre eux pourra confirmer l'impact de ce travail.
Enfin, j'ai été en contact régulier avec Niko Matsakis, un des concepteurs du langage Rust développé par Mozilla, sur des questions de conception de langages de programmation qui s'appliquent aux deux langages, ou plus généralement du transfert de connaissances académiques vers un projet industriel. Il pourra confirmer la variété de ces discussions.

5. Diffusion / *Dissemination*

Mes publications liées au langage OCaml concernent les GADTs (ESOP 2013, ESOP 2015) et les variables ambiguës dans un motif (ML Workshop 2016, version longue en cours de rédaction).

Formulaire 4 — PROGRAMME DE RECHERCHE
Form 4 — RESEARCH PROGRAM

Nom / *Last name*: Scherer
Prénom/*First name*: Gabriel

Equipe(s)-Projet(s) d'affectation souhaitée(s)//*Project(s)-team(s) assignation wishes:* PARSIFAL

Intitulé du programme de recherche/*Title of research program :* Outils logiques pour l'aide à la programmation.

Une partie importante de la recherche en sciences du logiciel se concentre sur la question de la vérification : comment prouver qu'un programme correspond à sa spécification ?
J'aimerais me consacrer à un problème complémentaire : comment utiliser les outils formels développés dans la communauté des langages de programmation et ses voisines (théorie de la démonstration, logique, et démonstration automatique) pour aider l'écriture de programmes, en développant des outils et langages qui rendent la programmation plus aisée, plus concise, et réduisent les risques d'erreurs ?

Saturer les types riches
J'ai transposé pendant ma thèse la notion de *focusing* de la théorie de la démonstration aux langages de programmation, et ai obtenu, en ajoutant la notion de *saturation*, une représentation canonique des programmes utilisant des types dits "simples", représentants les types de données usuels en programmation : fonctions, paires, sommes disjointes. J'ai pu montrer ainsi que des types simples dont les habitants sont uniques apparaissent dans des programmes typés existants.
Mais les types *simples* que mon algorithme peut gérer ne font que décrire les structures de données; il existe d'autres types dans les langages typés modernes, en particulier les types *polymorphes* qui expriment la généricité et les types *dépendants* qui peuvent exprimer des spécifications mathématiques.
Une notion de forme canonique qui prendrait aussi en compte ces types riches serait nécessaire à une application à plus grande échelle de cette technique pour la synthèse de programmes – et permettrait aussi d'appliquer cette technique à des types plus fins, et donc plus souvent à habitants uniques.
C'est un projet ambitieux qui comporte des obstacles scientifiques à surmonter. La décidabilité de l'équivalence de types en présence du type vide, résultat obtenu pendant mon post-doc en utilisant le *focusing* et la *saturation*, était resté un problème ouvert connu depuis au moins 1995. Nous espérons que cette nouvelle approche permettra aussi des avancées importantes pour les types riches.
Une difficulté à surmonter est l'indécidabilité. Dans un système avec polymorphisme, trouver un habitant correspond à la recherche de preuve dans des logiques d'ordre supérieur: l'existence d'un habitant, et d'autant plus son unicité, deviennent indécidables. D'un algorithme qui termine sur tous les types simples, on passerait à une procédure partielle répondant parfois "je ne sais pas" – ce qui ne l'empêcherait pas d'être très utile aux programmeurs dans les cas où elle sait, tout comme un solveur SAT ou SMT peut exploser sur certaines requêtes mais reste un outil précieux dans les cas courants.

Équivalence de programmes
Les outils de preuve automatique ont fait d'énormes progrès et sont maintenant utilisés dans de nombreux domaines, dont la vérification de programmes et les sciences du logiciel en général – des techniques de recherche de preuve puissantes sont aussi utilisées, par exemple, pour les bases de données ou les ontologies. On a donc des approches fructueuses, et beaucoup d'implémentations utilisées en pratique, pour vérifier automatiquement qu'un programme répond à une spécification. Comparativement, il y a eu peu de travail sur la vérification automatique que deux programmes sont équivalents : les types sommes rendent la théorie délicate et sont rencontrés très vite en pratique. Les progrès accomplis pendant ma thèse et mon post-doc devraient permettre de passer ces difficultés. Je propose par exemple les deux applications suivant:
Vérifier le *refactoring*. La restructuration de programmes est une partie importante de l'activité de programmation. Elle est souvent une étape préalable à des changements plus ambitieux, faite séparément pour être plus facile à relire et vérifier par les collègues ou collaborateurs: un bon *refactoring* transforme le programme pour le rendre plus maintenable mais ne modifie pas son comportement.
Vérifier la compatibilité ou non-ambiguïté. Dans le système de modules dits "applicatifs" des langages fonctionnels de la famille ML, il est possible de construire un module à partir d'un module paramétré, et d'un choix de paramètre (un autre module). Si le même module paramétré est appliqué à deux modules "identiques" passés en paramètre, les modules résultats sont compatibles – ils peuvent échanger des valeurs. C'est important pour faire communiquer deux bibliothèques ayant fait indépendamment le même choix de paramètre, sans devoir les modifier pour dépendre d'un parent commun.
Aujourd'hui, la notion d'identité des paramètres est définie par une approximation syntaxique qui repose sur les noms. Deux bibliothèques peuvent faire le même choix de paramètre indépendamment et coopérer ensuite, mais elles ne peuvent pas

utiliser deux paramètres définis de la même façon, le paramètre doit venir d'une dépendance commune. Un algorithme robuste d'équivalence entre programmes (et modules) permettrait de lever cette restriction, en testant que deux paramètres (non syntaxiquements égaux) sont équivalents, pour rendre plus de modules compatibles et donc la programmation modulaire plus aisée.
Un autre cas d'usage proche correspond à des cas d'héritage en diamant: que faire quand un objet hérite les opérations de deux sous-objets qui définissent une opération de même nom ? On peut interdire ce cas, choisir un ordre arbitraire, produire deux opérations séparées, forcer l'utilisateur à renommer... Souvent les deux opérations étaient en fait définies de façon identique : un test d'équivalence de programme (même incomplet) permettrait de respecter l'intention du programmeur dans ce cas. Ce problème se présente aussi naturellement dans les formalisations de hiérarchies de structures algébriques, par exemple avec les espèces de Focalize ou les coercions en Coq: un monoïde fini a la structure d'un ensemble fini et d'un monoïde, et ces deux sous-structures ont le même ensemble de support.

Systèmes multi-langages

L'évolution des langages généralistes tend vers l'accrétion de fonctionnalités pour mieux servir des domaines spécialisés divers, qui peuvent poser des problèmes de cohérence ou de complexité pour le langage dans son ensemble. OCaml, GHC Haskell, Scala ou encore C++ ont acquis au fil des années des fonctionnalités qui réjouissent leurs utilisateurs experts mais intimident leurs débutants, rendent leur enseignement plus difficile, rendent les compilateurs plus fragiles, et augmentent le coût et la difficulté de la construction d'outils gérant tout le langage. La recherche en langages de programmation intensifie ces difficultés en proposant des fonctionnalités puissantes mais souvent plus complexes et difficile à combiner entre elles : types dépendants, systèmes d'effets, types linéaires, sous-ensembles normalisants, opérateurs de contrôle, etc. Il est aujourd'hui nécessaire de faire diminuer cette complexité d'utilisation pour que les résultats de nos recherches puissent continuer à toucher un public le plus large possible.
Je propose d'étudier la conception de systèmes *multi-langages*, dans lesquels plusieurs langages séparés co-opèrent pour couvrir les mêmes domaines applicatifs, en en gérant chacun seulement une petite partie. Idéalement, à expressivité égale ces systèmes seraient moins complexes à utiliser, en permettant en particulier aux utilisateurs qui le souhaitent d'apprendre et d'utiliser seulement une partie des langages du système.
Pour cela, il faut mieux comprendre les problèmes d'interactions entre langages : comment concevoir ces systèmes de façon à éviter les "fuites d'abstraction", où l'usage interne des fonctionnalités d'un langage peut casser les propriétés attendues par les utilisateurs d'un autre langage ? Une étude précise de cette question, telle que je l'ai commencée pendant mon post-doc à Northeastern, devrait nous donner des outils pour pallier aux intéractions problématiques entre langages existants, et pour concevoir de nouveaux systèmes multi-langages plus harmonieux.
Notre approche pour formaliser l'absence de fuites d'abstractions (*abstraction leaks*) est de prouver la propriété de *pleine abstraction*: si deux fragments de programmes sont indistinguables par tout usage dans le même langage, alors ils restent indistinguables dans le système multi-langage. Pendant mon post-doc j'ai formalisé un cas d'étude, un système formé d'un langage fonctionnel simple et d'un langage à types linéaires – permettant des manipulations sûres de protocoles complexes ou de plus bas niveau – et prouvé que le plongement du langage fonctionnel dans le multi-langage est pleinement abstrait.
À moyen terme, on imagine un langage fonctionnel généraliste, comme OCaml, accompagné d'un "langage avancé" complémentaire, qui permette d'écrire les implémentations internes avec un contrôle fin sur les performances, avec une garantie d'utilisabilité : tant qu'une personne n'utilise que des interfaces exprimées en OCaml, elle n'a pas besoin de connaître le langage avancé.
Pour concevoir un "langage du futur" qui ne soit pas un monstre, il faut chercher le *multi-langage* du futur.

Intégration à Parsifal

L'équipe Parsifal est aujourd'hui le meilleur endroit au monde pour travailler sur le *focusing*. Elle a par exemple accompli un travail remarquable sur l'étude des liens fins entre le focusing et les stratégies existantes de recherche de preuve et de programmation logique, et son travail sur l'adaptation du focusing à la logique intuitioniste et sur les formes canoniques de preuves multi-focalisées m'a beaucoup inspiré.
Stéphane Lengrand et Kaustuv Chaudhuri ont une expertise riche en preuve automatique dont j'aimerais profiter. Stéphane connaît les approches SMT, grâce en particulier à des collaborations fréquentes avec le SRI à Stanford, et travaille à les intégrer dans un assistant de preuve. Kaustuv est un expert de la "méthode inverse", une approche de la recherche de preuve par saturation qui est performante sur des problèmes difficiles en logique linéaire ou intuitionniste.
J'apporterais à cette équipe une expertise en conception de langages de programmation fonctionnels. L'étude de la programmation logique est un point fort de Parsifal, et mes travaux ont montré que les deux points de vue incarnés par ces deux approches (les programmes comme preuves et comme résultat d'une recherche de preuve) peuvent produire des résultats nouveaux.
Enfin, je contribuerais aux développements logiciels du projet Parsifal, notamment l'assistant de preuve Abella, particulièrement adapté à l'étude des langages de programmation et qui commence à être utilisé en-dehors de l'équipe, et le projet plus expérimental Psyche, qui utilise la recherche sur le *focusing* pour trouver un équilibre harmonieux entre preuve automatique et preuve interactive.

Formulaire 5 — LISTE COMPLÈTE DES CONTRIBUTIONS[2]
Form 5 — COMPLETE LIST OF CONTRIBUTIONS[2]

Nom / *Last name*: Scherer
Prénom/*First name*: Gabriel

1. Publications caractéristiques/*Representative publications*

Gabriel Scherer. "Deciding equivalence with sums and the empty type". ***POPL***. 2017.
Gabriel Scherer and Didier Rémy. "Which simple types have a unique inhabitant?" ***ICFP***. 2015.
Gabriel Scherer and Didier Rémy. "Full reduction in the face of absurdity". *ESOP*. 2015.

2. Publications

2.1 Revues internationales/*International journals*

Andreas Abel and Gabriel Scherer. "On Irrelevance and Algorithmic Equality in Predicative Type Theory". *Logical Methods in Computer Science* (2012).

2.2 Conférence internationales avec comité de lecture/*Reviewed international conferences*

Gabriel Scherer. "Deciding equivalence with sums and the empty type". ***POPL***. 2017.
Gabriel Scherer and Didier Rémy. "Which simple types have a unique inhabitant?" ***ICFP***. 2015.
Guillaume Munch-Maccagnoni and Gabriel Scherer. "Polarised Intermediate Representation of Lambda Calculus with Sums". ***LICS***. 2015.
Gabriel Scherer. "Multi-focusing on extensional rewriting with sums". *TLCA*. 2015.
Gabriel Scherer and Didier Rémy. "Full reduction in the face of absurdity". *ESOP*. 2015.
Gabriel Scherer and Jan Hoffmann. "Tracking Data-Flow with Open Closure Types". *LPAR*. 2013.
Gabriel Scherer and Didier Rémy. "GADTs meet subtyping". *ESOP*. 2013.

2.3 Livres et chapitres de livre/*Books and book chapters*

2.4 Autres publications internationales (posters, short papers)/*Other international publications (posters, short papers)*

2.5 Revues nationales/*National journals*

2.6 Conférence nationales avec comité de lecture/*Reviewed national conferences*

Pierre-Évariste Dagand and Gabriel Scherer. "Normalization by realizability also evaluates". *JFLA*. 2015.
Gabriel Scherer and Jérôme Vouillon. "Macaque: Interrogation sûre et flexible de bases de données depuis OCaml". *JFLA*. 2010.

2.7 Rapports de recherche et articles soumis/*Research reports and publications under review*

Gabriel Scherer, Amal Ahmed, and Max New. "Multi-language programming systems: a linear experiment". In submission. submitted in November 2016.
Gabriel Scherer and Didier Rémy. "Which simple types have a unique inhabitant?" *Journal of functional programming* (In submission). submitted in August 2016.

[2]Les publications et réalisations les plus significatives devront, dans la mesure du possible, être consultables sur la page web de la candidate ou du candidat.
Most relevant contributions (publications, software) should, as much as possible, be available for consultation via the web page of the applicant.

3. Développements technologiques : logiciel ou autre réalisation / *Technology development : software or other realization*

Implémentation du langage OCaml Software Assessment:

- Audience A5: wide-audience software
- Originality O4: fair number of original ideas
- Maturity SM4: major software project
- Evolution EM4: possible for users to use the software for important projects
- Distribution SDL5: external packaging and distribution

Contribution personnelle:

- Design and Architecture DA2: occasional contributor
- Coding and Debugging CD2: occasional contributor
- Maintenance and support MS3: regular contributor
- project management TPM3: regular contributor

4. Impact socio-économique et transfert / *Socio-economic impact and transfer*

Cf. la contribution 2, développement du langage OCaml.