2022

DOSSIER DE CANDIDATURE

APPLICATION

Cochez le concours sur lequel vous candidatez
Check the competition exam for which you are applying

☐ **ISFP - (Inria Starting Faculty Position / Inria Starting Faculty Position)**

☒ **CRCN - (Chargés de recherche de classe normale / Young graduate scientist position)**

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

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

Prénom : Raphaël
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

DEPOT DE VOTRE CANDIDATURE
SUBMITTING YOUR APPLICATION

Le dossier de candidature doit comprendre :

- Formulaire 1 : Parcours professionnel
- Formulaire 2 : Description synthétique de l'activité antérieure
- Formulaire 3 : Contributions majeures
- Formulaire 4 : Programme de recherche
- Formulaire 5 : Liste complète des contributions

CRCN & ISFP :

- Les rapports de thèse ou de doctorat (si disponibles)
- Une copie des derniers titres et diplômes
- Une photographie récente de la candidate / du candidat (facultative)

DR2 :

- Les rapports d'habilitation à diriger des recherches (si applicable)
- Une copie des derniers titres et diplômes
- Une photographie récente de la candidate / du candidat (facultative)

The application file must include:

- *Form 1: Professional history*
- *Form 2: Summary of your past activity*
- *Form 3: Major contributions*
- *Form 4: Research program*
- *Form 5: Complete list of contributions*

CRCN & ISFP:

- *PhD dissertation reports (when available)*
- *A copy of most recent titles and diplomas*
- *A recent photography of the applicant (optional)*

DR2:

- *Habilitation dissertation reports (if applicable)*
- *A copy of most recent titles and diplomas*
- *A recent photography of the applicant (optional)*

SOMMAIRE / ***SUMMARY***

Formulaire 1 — PARCOURS PROFESSIONNEL
Form 1 — PROFESSIONAL HISTORY

1) Parcours Professionnel / *Professional history*

Situation professionnelle actuelle / ***Current professional status***

Statut et fonction[2] / *Position and Status*[2]: Attaché Temporaire d'Enseignement et de Recherche
Etablissement (ville - pays) / *Institution (city - country)*: Sorbonne Université – Paris – France
Date d'entrée en fonction / *Start*: 01/09/2021
[ ] Sans emploi / *Without employment*

Expériences professionnelles antérieures /***Previous professional experiences***

Date début *Start*	Date fin *End*	Etablissement *Institution*	Fonction et statut[2] *Position and status*[2]
01/09/2014	31/08/2018	ENS de Lyon	Fonctionnaire-stagiaire
01/09/2018	31/08/2021	Sorbonne Université	Doctorant contractuel (mission d'enseignement de 64h/an)
01/09/2021	31/08/2022	Sorbonne Université	ATER

Nombre d'années d'exercice des métiers de la recherche après la thèse / *Number of years of professional research experience after the PhD:* 0.5 (thèse soutenue le 22 novembre 2021).

Si vous le jugez nécessaire, vous pouvez apporter des précisions sur votre parcours ou votre situation actuelle **en quelques lignes**. / *If you find it necessary, you can give some details about your professionnal history or your current situation **in a few lines**.*
En mars 2021, j'ai candidaté sur un poste d'ATER pour l'année universitaire courante, afin d'avoir plus de temps pour finir ma thèse (mon contrat de doctorat se terminait le 31 août 2021). À ce moment là, la dernière partie des recherches de ma thèse était en cours d'implémentation et n'avait pas été rédigée dans un article. J'ai donc préféré prendre ce poste d'ATER, afin de pouvoir finir ce travail avant de commencer la rédaction de ma thèse. Au final, l'article décrivant ce dernier travail a été accepté au début de l'été 2021 à SAS, et j'ai pu directement enchaîner sur la rédaction de mon manuscrit, puis soutenir fin novembre 2021.
Ce poste me permet aussi de pouvoir accompagner la direction générale des finances publiques dans le cadre du transfert du compilateur Mlang (dont je suis un des deux développeurs originels) de janvier à août 2022.

2) Interruptions de carrière/*Career breaks*

3) Encadrement d'étudiants et de jeunes chercheurs / *Supervision of students and early-stage researchers*

4) Encadrement de développements technologiques (logiciel, matériel, robotique) / *Supervision of technological development (software, hardware, robotics)*

Supervision du transfert de Mlang à la DGFiP (janvier – août 2022) Encadrement de deux prestataires d'OcamlPro et d'un fonctionnaire inspecteur programmeur système d'exploitation dans le développement de Mlang pour les besoins de la direction générale des finances publiques (DGFiP). Détaillé dans les développements logiciels du Formulaire 5.

5) Responsabilités collectives / *Responsibilities*

Participation à des comités de lecture (revues, conférences), participation à des conseils ou commissions, organisation de conférences, tâches d'intérêt collectif. Fournissez une URL vers tous les comités des conférences et journaux concernés.
Mention any membership in editorial boards or other committees, participation in conference/workshop program or organization committees, tasks of collective interest. Provide a URL to all the relevant editorial boards and program committees.

[2] Indiquez avec précision chaque situation statutaire. Par exemple : pour une situation d'agent titulaire de la fonction publique, précisez 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, précisez 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 in a public institution, indicate the nature of the work contract, etc.*

Conseil du laboratoire du LIP6 (représentant élu des non-permanents, Avril 2021 – Août 2022).

Comité de programme pour l'évaluation d'artefacts logiciels. SPLASH'22, PLDI'22, CAV'22, ECOOP'21, PLDI'21, POPL'21, SAS'20.

Comité de revue externe. SPLASH'22.

Relecteur externe. SAS'21 (via Antoine Miné), ACM TECS (via Albert Cohen) SOAP'21 (via Antoine Miné), LOPSTR'19 (via Caterina Urban).

Étudiant bénévole à POPL'17.

6) Management (si pertinent) / *Management (if relevant)*

7) Mobilité (si pertinent) / *Mobility (if relevant)*

Stages de recherche effectués durant ma scolarité à l'ENS de Lyon.

Extension d'une formalisation en Coq et HOL4 pour un synthétiseur de programmes sur les flottants (MPI-SWS, Sarrebruck, Février à Juin 2017). Stage de recherche de M2 sous la direction d'Eva Darulova. Le groupe de vérification automatique et d'approximation (AVA) du Max Planck Institute for Software Systems, dirigé par Eva Darulova, développe un outil appelé Daisy, permettant de compiler des programmes numériques idéalisés, définis sur les réels, en des programmes calculant sur les flottants, avec une marge d'erreur donnée en sortie par le compilateur. Ce compilateur était capable d'optimiser l'équilibre entre précision et performance des programmes grâce à l'utilisation de plusieurs formats pour les flottants. Un doctorant du groupe, Heiko Becker, avait établit une formalisation (en Coq et HOL) permettant de vérifier via des certificats que la compilation utilisant une analyse d'intervalles et une précision fixée était correcte. J'ai généralisé cette formalisation pour prouver la compilation correcte dans le cas d'une précision mixte. J'ai aussi travaillé à formaliser une analyse basée sur l'arithmétique affine. Une publication de groupe sur ce travail a été acceptée à FMCAD 2018.

Inférence variationnelle pour les langages de programmation probabilistes (Oxford, Mai à Juillet 2016). Stage de recherche de M1 sous la direction de Hongseok Yang. Les modèles probabilistes sont utilisés dans de nombreux domaines pour résoudre différents problèmes, allant de la reconnaissance d'images au diagnostic de maladies. L'utilisation de modèles permet de séparer l'encodage du problème (dans un modèle probabiliste) de sa résolution, et d'avoir des algorithmes d'inférence spécifiques à chaque classe de modèle. Une approche à l'intersection des langages de programmation et de l'apprentissage automatique est la programmation probabiliste. Cette approche permet de généraliser la notion de modèle probabiliste, et a pour but de pouvoir réaliser automatiquement l'inférence à l'exécution des programmes probabilistes. J'ai travaillé avec Hongseok Yang à formaliser les programmes probabilistes sous une forme de système de transition probabiliste. Le problème d'inférence peut être vu comme un problème d'optimisation, que nous avons simplifié pour dériver un nouvel algorithme d'inférence variationnel général pour les programmes probabilistes.

8) Enseignement (si pertinent) / *Teaching (if relevant)*

J'ai effectué mes enseignements à Sorbonne Université, dans l'UFR d'ingénierie, auprès d'étudiants en informatique (à partir de la L2), ou d'étudiants sur un tronc commun mathématiques - physique - informatique (pour les L1). Les charges assurées de septembre 2018 à janvier 2021 sont décrites ci-dessous.
Total d'heures par niveau :

- L1 : 79.5h
- L2 : 84.75h
- L3 : 59.5h
- M1 : 60.5h
- M2 : 42h

Acronymes :

- TME : travaux machines encadrés
- STL : sciences et techniques du logiciel
- MPRI : master parisien de recherche en informatique, spécialisé en informatique fondamentale et à destination des étudiants poursuivant dans la recherche. Formation supervisée par l'université de Paris, ENS Ulm, ENS Paris-Saclay, École Polytechnique, Telecom Paris.

L1 – Éléments de Programmation 1 (Python), chargé de TD/TME (26 étudiant·e·s, 38.5h). Année 2019–2020.
L1 – Éléments de Programmation 1 (Python), chargé de TME (29 étudiant·e·s, 19.25h). Année 2018–2019.
L1 – Éléments de Programmation 1 (Python), remplacement en TD (30 étudiant·e·s, 1.75h). Année 2018–2019.
L1 – Ateliers de Recherche Encadrée, chargé de TD/TME (4 binômes, 20h). Année 2018–2019.

L2 – Mathématiques Discrètes, chargé de TD/TME (33 étudiant·e·s, 44.5h). Année 2021–2022.
L2 – Programmation Fonctionnelle (OCaml), chargé de TD/TME (26 étudiant·e·s, 19.25h). Année 2021–2022.
L2 – Programmation Fonctionnelle (Ocaml), remplacement ponctuel en TME (27 étudiant·e·s, 1.75h). Année 2019–2020.
L2 – Fonctions et Procédures de Calcul (Ocaml), chargé de TME (10 étudiant·e·s, 19.25h). Année 2018–2019.

L3 – Programmation Objet Avancée (Java), chargé de TD/TME (27 étudiant·e·s, 31.5h). Année 2020–2021.
L3 – Compilation, chargé de TD/TME (12 étudiant·e·s, 28h). Année 2018–2019.

M1 – Algorithmique Avancée, chargé de TD (31 étudiant·e·s, 32h). Année 2021–2022.
M1 – Algorithmique Avancée, chargé de TD (31 étudiant·e·s, 20h). Année 2019–2020.
M1 – Projet en filière STL, co-encadrant de binômes sur des projets (8.5h). Année 2019–2020.

M2 – MPRI : Interprétation abstraite, intervenant (15 étudiant·e·s, 2h). Année 2021–2022.
M2 – Typage et Analyse Statique, chargé de TD/TME (31 étudiant·e·s, 28h). Année 2021–2022.
M2 – Spécification et Validation de Programmes (Coq), chargé de TD/TME (10 étudiant·e·s, 10h). Année 2021–2022.
M2 – Groupe de Recherche en Algorithmique et en Programmation, intervenant (12 étudiant·e·s, 2h). Année 2019–2020.

9) Diffusion de l'information scientifique (si pertinent) / *Dissemination of scientific knowledge (if relevant)*

10) Visibilité (si pertinent) / *Visibility (if relevant)*

Orateur invité au Facebook TAV Symposium (décembre 2021). Présentation en 25 minutes de mes travaux sur l'analyse multilangage de programmes Python et C. "Testing and Verification (TAV) Symposium brings together academia and industry in an open environment to exchange ideas and *showcase the top experts from testing and verification scientific research and practice*."

Prix de la meilleur présentation lors du workshop SOAP (juin 2020). Prix attribué suite à ma présentation de mon travail "Value and Allocation Sensitivity in Static Python Analyses".

11) Eléments divers / *Other relevant information*

Formulaire 2 — DESCRIPTION SYNTHÉTIQUE DE L'ACTIVITÉ ANTÉRIEURE

Form 2 — SUMMARY OF YOUR PAST ACTIVITY

Le but de mes recherches est d'améliorer la qualité des logiciels. J'utilise pour cela le domaine des **méthodes formelles**, afin de définir précisément le comportement des programmes. Cela permet de raisonner rigoureusement sur ces comportements (pour déceler par exemple des états dangereux). Dans certains cas, il est possible d'automatiser le raisonnement pour qu'un ordinateur diagnostique les comportements qui nous intéressent, via des **analyses de programme**.

J'aspire à développer et appliquer des méthodes aux systèmes les plus réalistes possibles. D'un point de vue méthodologique, je commence par sélectionner des programmes (le plus souvent, sur GitHub) et des bogues qui seront visés par mon analyse. Cette première étape est partiellement automatisée via des scripts. Je développe ensuite mes approches **à la fois de manière théorique**, pour obtenir une formalisation rigoureuse, **et via une implémentation**, afin de valider l'approche sur les programmes retenus lors de la première étape. Au cours de ces quatre dernières années, j'ai étudié **deux systèmes réalistes, avec des collaborateurs distincts.**
Le premier système est le **langage de programmation Python**, qui est très populaire. C'est en particulier le **deuxième langage le plus utilisé** (après JavaScript) sur GitHub [23]. Ce système est l'objet des recherches effectuées durant ma thèse avec Antoine Miné et Abdelraouf Ouadjaout. J'ai en garde partie **formalisé la sémantique du langage Python**, sur papier. J'ai développé des **analyses statiques de programmes sûres et précises** pour le langage Python. En particulier, j'ai développé une **analyse multilangage**, permettant de détecter des bogues dans les programmes utilisant à la fois Python et C. Cette analyse répond à un besoin réel : les programmes sont souvents écrits avec plusieurs langages. Ce problème d'analyse multilangage a été très peu abordé précédemment. Ces analyses ont été implémentées dans la **plateforme open-source Mopsa**, écrite en **60 000 lignes de code OCaml**, et dont je suis un des développeurs principaux.
Le second système est le **code de calcul de l'impôt sur le revenu des particuliers**. Le code source de ce système est public depuis avril 2016, mais j'ai fait partie des premiers scientifiques du domaine à s'y être interessé en 2019. Ce travail de recherche a été effectué en parallèle de ma thèse, en collaboration avec Denis Merigoux (alors doctorant dans Prosecco, Inria Paris), et illustre ma **capacité à conduire des travaux de recherches indépendamment de ma direction et de mon laboratoire de thèse**. Ce système est critique : il concerne 38 millions de foyers fiscaux français, et rapporte 30% des recettes de l'État chaque année. Nos recherches ont permis de **rendre le calcul** de l'impôt sur le revenu publiquement **reproductible** : le code était disponible en ligne, dans un langage appelé M, propre à l'administration et non documenté. Nous avons pour cela mené un travail de rétro-ingénierie de la sémantique de M, avant de la **formaliser en Coq**. Nous avons aussi développé un **nouveau compilateur open-source (10 000 lignes de code OCaml)** permettant de répliquer le calcul de l'impôt sur le revenu. Je supervise actuellement un **transfert du compilateur** vers la direction générale des finances publiques, en **encadrant le travail de trois développeurs**.
Mon projet de recherche aborde deux thématiques distinctes. J'aimerais développer des analyses statiques de programmes qui sont à la fois **sûres et accessibles** – en terme de précision et d'efficacité – pour être **massivement adoptées** par les développeur·euse·s. Je souhaite aussi appliquer le domaine des méthodes formelles à une classe de programmes assez peu étudiés alors qu'ils ont des impacts sociétaux majeurs : les **implémentations de codes juridiques**.

Une partie de mes recherches utilise le cadre de l'**interprétation abstraite** pour créer des **analyses statiques sûres et automatiques** : toutes les propriétés dérivées par l'analyse sont effectivement valables sur le programme, et l'analyse termine sa tâche de manière autonome, en un temps fini. Ces analyses sont dites statiques car elles n'exécutent pas le programme, afin de pouvoir **raisonner sur toutes les exécutions possibles** (potentiellement en nombre infini). Ces analyses sont cependant incomplètes et peuvent donner lieu à de **fausses alarmes**, lorsque le système ne peut pas prouver qu'un comportement donné est correct. Un objectif est donc d'avoir des systèmes suffisamment précis pour réduire le plus possible le nombre de fausses alarmes. Le cadre de l'interprétation abstraite fournit une méthodologie pour obtenir des analyses statiques. Le point de départ est la **sémantique concrète** du langage étudiée, décrivant un interprète calculant les états accessibles d'un programme donné. Cet interprète ne termine pas en temps fini (par exemple, à cause des boucles). Cette sémantique est donc modifiée pour obtenir une sémantique abstraite, décrivant un interprète calculant une **sur-approximation** des états accessibles en temps fini. Cet interprète va utiliser différents **"domaines abstraits"** pour gérer les traits du langage analysé. Ces approches ont connu un succès particulier pour prouver l'absence d'erreurs à l'exécution dans le contexte des logiciels embarqués critiques. Par exemple, Astrée [12], a été utilisé pour prouver l'absence d'erreurs à l'exécution dans les logiciels de contrôle et de commande des avions Airbus [17].

Science ouverte. En plus du dépôt de mes articles sur HAL, j'attache une importance particulière à rendre les logiciels développés disponibles (sous licence libre), et l'évaluation expérimentale de mes articles reproductible (via par exemple Zenodo). Les artefacts liés à des articles publiés ont aussi fait l'objet d'une évaluation par les pairs [48, 45, 52]. Je préfère soumettre mes travaux à des conférences qui ont des politiques de publication ouverte avec des frais raisonnables pour les auteurs (proches du prix coûtant, ≤ 100 euros), ou qui sont atteignables sans prendre l'avion. ECOOP est la conférence du domaine respectant le plus ces critères, puisqu'elle a lieu en Europe et qu'elle utilise les actes ouverts LIPIcs (les autres conférences du domaine utilisent l'ACM ou Springer comme éditeurs).

Formulaire 3 — CONTRIBUTIONS MAJEURES
Form 3 — MAJOR CONTRIBUTIONS

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

- **CRCN & ISFP – 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 (5 au plus pour les candidatures DR2 — 3 au plus pour les candidatures CRCN & ISFP).

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, matériel, robotique ou autre), d'une action de transfert industriel ou sociétal, d'une responsabilité collective, d'une activité d'animation d'une communauté de recherche, ou tout autre élément relevant des missions d'un chercheur ou d'une chercheuse. Les critères importants sont la créativité, l'originalité et l'impact. Chaque fiche suivra le plan indiqué ci-dessous. Dans l'ensemble du texte, pensez à donner, le cas échéant, les références permettant de consulter sur le Web les documents mentionnés (articles, thèses, logiciels, etc.).

Pour les logiciels, fournissez une autoappréciation selon le canevas disponible dans le document « Criteria for Software Self-Assessment » disponible à l'URL
https://www.inria.fr/sites/default/files/2021-01/Criteria%20software%20self%20assessment.pdf.

Pour les actions de transfert (transfert technologique ou sociétal), fournissez une description selon le canevas disponible dans le document « Evaluation des contributions scientifiques en matière de transfert / Guide méthodologique » disponible à l'URL
https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique_EvaluationTransfert%281%29.pdf.

Fill in one form for each major contribution (at most 5 for the DR2 candidates — 3 for the CRCN & ISFP candidates).

It may be a scientific contribution expressed through a set of publications (or a single major publication) or through a technological development (software, hardware, robotic, or other); it may also be an industrial or a societal 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,...), if available.

For software, please use the « Self-assessment software criteria » guideline, available at the URL
https://www.inria.fr/sites/default/files/2021-01/Criteria%20software%20self%20assessment.pdf.

For transfer actions (technology or society transfer) please describe your achievements following the guidelines « Evaluating scientific contributions in relation to transfer / Methodological guide » available at the URL
https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique-EvaluationTransfert_EN%281%29.pdf.

Fiche 1 : Analyse statique de programmes Python, utilisant des bibliothèques C

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

Analyse de programmes Python. Ma première contribution est la conception d'analyses sûres et précises pour détecter les erreurs à l'exécution des programmes Python réalistes. Dans le cas de Python, ces erreurs sont toutes les exceptions (générées par l'interprète Python ou le programme exécuté) qui ne sont pas rattrapées et qui vont donc interrompre l'exécution du programme. Les analyses supportent les comportements de Python utilisés en pratique par les développeur·euse·s (structure dynamique d'objet, opérateurs d'introspection, mutabilité des objets, générateurs), afin de pouvoir analyser des programmes Python réalistes. J'ai implémenté ces analyses dans la plateforme Mopsa, en cours de développement au LIP6, suite à l'octroi d'un financement "Consolidator Grant" de l'ERC à Antoine Miné. J'ai d'abord défini une analyse capable de détecter précisément les erreurs de type et d'accès à des champs. J'ai ensuite raffiné la précision de l'analyse de type en ajoutant de l'inférence d'informations numériques, pour obtenir ainsi une analyse de valeur. Cette analyse peut utiliser différents domaines numériques, tels que les intervalles, ou des domaines numériques relationnels permettant d'inférer des relations entre les variables tels que les polyèdres. Le support des domaines relationnels exige une attention particulière dans la conception d'un analyseur statique. Intuitivement, l'évaluation d'une expression ne peut plus retourner une valeur abstraite (telle qu'un intervalle) dans un cadre relationnel. Ces domaines relationnels sont au centre de l'architecture de Mopsa et sont supportés par cette analyse. Toutes ces analyses ont été évaluées sur des programmes réels, tels que les benchmarks de performance [60] utilisés par les développeur·euse·s de l'interprète de Python, ou un utilitaire développé par Facebook appelé `PathPicker` [68]. Sur ces benchmarks, l'analyse de type analyse au total plus de 9 000 lignes de code[1] en moins de 4 minutes, celle utilisant des intervalles en 13 minutes [49].

Analyse de programmes combinant Python et C. Les programmes modernes mélangent de plus en plus souvent différents langages. Cela permet aux développeur·euse·s de combiner les atouts de différents langages et de réutiliser des bibliothèques écrites dans d'autres langages. Bien qu'utile, l'interopérabilité est une source additionnelle de bogues : les développeur·euse·s doivent prendre en compte des mécanismes de sécurité et des représentations de la mémoire différents suivant les langages. Ces mécanismes sont par ailleurs rarement supportés de manière sûre par les analyseurs statiques, qui se contentent fréquemment d'ignorer les comportements induits par l'autre langage. J'ai développé une analyse pour des programmes mélangeant l'utilisation de C et de Python. Cette analyse détecte les erreurs d'exécution dans le code C natif (opérations de pointeurs invalides, dépassement d'entiers, ...), dans le code Python (exceptions levées), et à la frontière entre les langages (conversion incorrecte entre les langages, non-concordance de signaux d'erreur, ...). Il est ainsi possible d'analyser directement et de manière entièrement automatique les codes sources Python et C. J'ai implémenté cette analyse de manière modulaire : elle réutilise des analyses C et Python prêtes à l'emploi écrites dans le même analyseur, Mopsa. Cette approche permet le partage entre les domaines abstraits de différents langages, tels que ceux gérant l'allocation dynamique, ou les abstractions numériques (intervalles, polyèdres), ce qui permet notamment d'inférer des relations entre les valeurs des différents langages. Pour cette approche, nous avons utilisé comme benchmarks 6 bibliothèques disponibles sur GitHub, ayant en moyenne 412 étoiles. La plus grosse bibliothèque (5 700 lignes de code Python et C) est analysée en moins de 5 minutes.

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

Abdelraouf Ouadjaout est le développeur principal du cœur de Mopsa, dont les grands principes ont commencé à être établis en collaboration avec Antoine Miné et Matthieu Journault, avant le début de ma thèse et principalement dans une optique d'analyse de programmes C. Ces principes ont été raffinés et éclaircis durant mes travaux sur Mopsa, et l'ajout des analyses Python dans la plateforme. Ils sont décrits dans une publication de groupe [31]. Mopsa est écrit en 60 000 lignes d'OCaml. Je suis le principal développeur de la partie permettant l'analyse de programmes Python (13 000 lignes). J'ai effectué ces recherches en collaboration avec Abdelraouf et Antoine. Je suis le rédacteur principal des articles ECOOP et SOAP. Je me suis occupé des présentations aux deux événements virtuels. J'ai entièrement développé et implémenté l'analyse multilangage (2 700 lignes) lors de ma dernière année de thèse. J'ai ensuite rédigé totalement l'article SAS sur ce sujet, intégré les remarques de mes coauteurs, et je me suis occupé de la présentation lors de la conférence virtuelle. J'ai aussi participé au développement et à la maintenance du reste de Mopsa. En particulier, j'ai mis en place une intégration continue de l'analyseur, et participé au développement d'un mécanisme de "hooks" [51, section 3.4] facilitant le débogage.

3. Originalité et difficulté / *Originality and difficulty*

Analyse de Python. Les langages dynamiques tels que JavaScript et Python comportent plusieurs traits les rendant différents des langages tels que C et Java souvent ciblés par les analyseurs statiques. Ils sont typés dynamiquement : les variables ne sont ni déclarées, ni typées statiquement. Au cours de l'exécution d'un programme, une variable peut pointer vers objets ayant des types différents. Ils ont une structure dynamique d'objet, où il est possible d'ajouter des champs aux objets durant l'exécution. Ces deux traits sont complétés par des opérateurs d'introspection, qui permettent

[1] Les mesures de taille de code présentées ont été faites avec l'outil `cloc`, qui ne prend pas en compte les commentaires et les lignes vides.

de changer le flot de contrôle du programme suite à l'inspection d'un objet. Peu de domaines abstraits ont été développés pour analyser les structures de données offertes par ces langages (tableaux dynamiques, dictionnaires). Le flot de contrôle du programme peut aussi être complexe à analyser lorsque des fonctions asynchrones (ou générateurs) sont utilisés.
Les analyses statiques sûres de langages dynamiques étaient concentrées sur le langage JavaScript [29]. Celles-ci ne font pas d'inférence d'informations numériques (intervalles, polyèdres), qui sont nécessaires pour des analyses précises de Python. Un travail précédent sur l'analyse de Python avait été réalisé par Fromherz et al. [21], mais le prototype utilisé n'était pas public, et ne pouvait pas analyser des programmes faisant plus de 300 lignes de code. Mon but était d'être plus modulaire (en proposant le choix entre une analyse de type et des analyses numériques), d'obtenir un meilleur passage à l'échelle via un support étendu du langage, tout en fournissant une implémentation robuste et maintenable à long terme. Python est un langage extrêmement flexible, avec beaucoup de cas exceptionnels à gérer. Le langage n'est pas standardisé, et l'interprète principal, CPython, sert de référence. J'ai donc beaucoup parcouru le code de CPython (dont le noyau est écrit en 160 000 lignes de C) afin de m'assurer de comprendre la sémantique dans les moindres détails. Cette sémantique extrêmement permissive rend nécessaire la connaissance précise du contexte d'appel pour arriver à analyser des fonctions. Cela est néanmoins très coûteux et entrave le passage à l'échelle de l'analyse. Le développement d'approches compositionnelles d'analyses de fonctions est un des objectifs de mon projet de recherche. L'analyse de programmes réalistes a requis la combinaison de nombreux domaines abstraits, afin de gérer par exemple l'allocation dynamique, les listes, les attributs des objets. Certains domaines existaient déjà dans la littérature ou étaient connus de manière folklorique, mais j'ai dû les adapter, les combiner et les implémenter dans le cadre nouveau de Mopsa.

Analyse multilangage. La plupart des travaux du domaine se cantonnent à l'analyse d'un seul langage. Les recherches précédentes se sont concentrées sur le cas des programmes mélangeant Java et C [22, 34, 61, 63]. De manière générale, ces approches essayaient de réduire le problème de l'analyse multilangage Java/C à une analyse de Java, en traduisant *certains* effets du code C sous forme d'annotations Java. Ces approches ne sont pas sûres : elles ne prennent pas en compte tous les comportements du langage C, ni toutes les interactions possibles entre les deux langages. Par opposition, notre travail est le premier publié à effectivement analyser complètement les deux langages sources utilisés dans les programmes (sauf le ramasse-miettes qui n'est pas supporté actuellement). Ce travail a nécessité d'établir une séparation minutieuse entre les états mémoire des deux langages (qui travaillent sur la même mémoire, mais à des niveaux d'abstraction différents), afin d'éviter d'avoir à effectuer des synchronisation systématiques entre les deux états.

4. Validation et impact / *Validation and impact*

Mopsa. L'implémentation de Mopsa vise à explorer de nouvelles perspectives pour la conception des analyseurs statiques. Mopsa a un triple objectif, qui a nécessité de développer et mettre en œuvre une nouvelle approche lors de la définition et l'implémentation des domaines abstraits. Le premier objectif est de supporter l'analyse de plusieurs langages (C et Python à l'heure actuelle). Les autres analyseurs sûrs tels qu'Astrée [5], Frama-C [15] et TAJS [29] n'analysent qu'un seul langage. Le second objectif est de permettre aux développeur·euse·s de définir des domaines abstraits de manière modulaire – c'est-à-dire aussi indépendamment les uns des autres que possible. Frama-C et TAJS sont implémentés de manière monolithique, où il est difficile de remplacer une abstraction par une autre gérant un même effet. Mopsa permet enfin aux différents domaines abstraits de coopérer et communiquer de manière relationnelle. Mopsa permet de composer plus profondément différents domaines, en autorisant des domaines à partager l'utilisation de domaines sous-jacents [31, pages 9-10], [51, sections 2.5 et 3.3.1]. Cette composition permet schématiquement de passer à un arbre de domaines utilisé par les analyseurs classiques tels qu'Astrée [12, figure 1] à un graphe acyclique dirigé [51, figure 3.2], afin d'améliorer notamment la précision des analyses. Les analyses décrites précédemment sont implémentées dans Mopsa.

5. Diffusion / *Dissemination*

La sémantique concrète du langage Python n'est pour le moment que décrite sur papier, sans être exécutable, dans mon manuscrit de thèse [51, pages 101 à 146]. Un des travaux futurs décrit dans mon projet de recherche est de rendre cette sémantique exécutable afin de la publier. L'analyse de type a été présentée à ECOOP [47]. Un artefact logiciel a été validé par les pairs [48]. Une comparaison de l'analyse de type et l'analyse non-relationnelle de valeurs a été présentée à SOAP (workshop de PLDI) [49], où j'ai reçu le prix de la meilleure présentation. Le passage à des analyses relationnelles, nécessitant un groupement heuristique des variables, est précisé dans mon manuscrit [51, section 8.3]. L'analyse multilangage a été présentée cette année à SAS [50], la principale conférence d'analyse statique. Cette publication est accompagnée d'un artefact évalué par les pairs [52]. J'ai été invité au "Facebook TAV" cette année [67] pour présenter ces travaux. Nous avons présenté Mopsa dans un article invité à une conférence internationale, VSTTE [31]. Nous avons aussi soumis un article court décrivant l'utilisation de Mopsa dans une conférence nationale, les JFLA [38]. Une présentation détaillée de Mopsa est aussi disponible dans le chapitre 3 de ma thèse [51]. Mopsa est disponible publiquement sur Gitlab [32], tout comme les programmes analysés utilisés comme benchmarks que nous avons utilisés [33].

M. Journault, A. Miné, R. Monat, *A. Ouadjaout*, Combinations of Reusable Abstract Domains for a Multilingual Static Analyzer (Invité). VSTTE 2019, 16 p.
R. Monat, *A. Ouadjaout, A. Miné*, Static Type Analysis by Abstract Interpretation of Python Programs. ECOOP 2020, 27 p.
R. Monat, *A. Ouadjaout, A. Miné*, Value and Allocation Sensitivity in Static Python Analyses. SOAP 2020 (PLDI workshop), 6 p.
R. Monat, *A. Ouadjaout, A. Miné*, A Multilanguage Static Analysis of Python Programs with Native C Extensions. SAS 2021, 23 p.

Fiche 2 : Création d'un compilateur moderne pour le calcul de l'impôt sur le revenu

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

Suite à la loi "République Numérique" de 2016 [70], la direction générale des finances publiques (DGFiP) avait rendu public le code permettant de calculer l'impôt sur le revenu des particuliers [69, 19]. La publication consistait seulement en la base de code, écrite dans le langage M propre à la DGFiP, sans aucun moyen d'exécuter le code. Il n'était donc pas possible de reproduire le calcul de l'impôt sur le revenu.
Nos recherches ont permis de rendre le calcul de l'impôt sur le revenu publiquement reproductible. Nous avons pour cela d'abord mené un travail de rétro-ingénierie afin de découvrir la sémantique du langage M, commencé en avril 2019. En septembre 2019, nous avons remarqué qu'une partie du code source était manquante en inspectant certains tests privés transmis par la DGFiP. Suite à la signature d'un accord de confidentialité, nous avons pu accéder au code manquant en juin 2020. La DGFiP avait écrit une partie de son code en C pour modifier l'état global de la base de code M compilée en C, afin de pallier à l'absence de définitions de fonctions dans le langage M. Nous avons donc développé un langage spécifique, appelé M++, afin de pouvoir remplacer cette base de code C. M++ a l'avantage d'être un langage adapté permettant une écriture beaucoup plus compacte : 100 lignes de code M++ permettent d'implémenter 6 000 lignes de code C. M++ permet aussi de compiler le code vers d'autres sources que le C (qui est un besoin réel de certains services). Enfin, la DGFiP ne souhaitait pas publier le code C pour des raisons de sécurité. Par opposition, le code M++ équivalent, que nous avions écrit, est maintenant public. La sémantique du langage M a été formalisée dans l'assitant de preuve Coq [62], afin d'améliorer la confiance en notre formalisation. Nous avons écrit un compilateur, appelé Mlang, permettant de compiler cette base de code M/M++ vers différents langages tels que Python, C et Java. Ce compilateur implémente des optimisations usuelles, et des optimisations spécifiques à la sémantique de M/M++, afin de réduire la taille de la base de code de 80%. Grâce à des outils de fuzzing, nous avons aussi pu générer nos propres cas de tests, ayant une meilleure couverture que les cas confidentiels de la DGFiP. Ces cas de tests générés ont aussi été rendus publics.

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

Denis Merigoux (Prosecco, Inria Paris) et moi avons collaboré à parts égales, comme l'illustre nos positions de co-premiers auteurs dans l'article publié à Compiler Construction [44]. Nous avons effectué ces recherches en parallèle de nos thèses respectives. Jonathan Protzenko, un des directeurs de thèse de Denis, a contribué à la rédaction et partagé son regard critique. Denis a découvert la base de code de la DGFiP, et a initié son travail de rétro-ingénierie fin février 2019 (parsing, inférence de type); je l'ai rejoint début mai 2019. J'ai développé la formalisation Coq du langage M, j'ai conçu le langage M++, ainsi que les optimisations du compilateur spécifiques à M/M++. Je me suis occupé de présenter notre travail à la conférence virtuelle. Cette recherche a été menée avec l'approbation mais sans la direction d'Antoine Miné.

3. Originalité et difficulté / *Originality and difficulty*

Notre travail se concentre sur une base de code réelle, qui calcule l'impôt sur le revenu de 38 millions de foyers fiscaux chaque année, et rapporte 75 milliards d'euros, soit 30% des recettes annuelles de l'État. L'étude et l'implémentation de codes à portée juridique ouvre un nouveau domaine prometteur, que j'aimerais explorer dans mes travaux futurs.
Les interactions avec la DGFiP (par exemple, pour avoir accès à l'entièreté de la base de code) ont nécessité de travailler sur le temps long tout en étant pédagogue avec la hiérarchie, qui a une formation juridique. La création du langage M++ n'a été possible que suite à l'étude approfondie de la base de code C de plusieurs milliers de lignes que nous voulions remplacer. Avoir une sémantique bien définie de M et M++ permettra de créer des outils de vérification formelle sur ce code dans un futur proche.

4. Validation et impact / *Validation and impact*

Ce projet est un succès car la DGFiP a décidé de remplacer leur compilateur actuel par Mlang. Un *transfert technologique* est en cours depuis juin 2021 [66], afin que la DGFiP puisse remplacer son compilateur actuel par Mlang. En parallèle de mon ATER, je suis engagé par la DGFiP depuis janvier 2022 pour faciliter la transition durant une mission ponctuelle de 30 jours, à effectuer d'ici fin août 2022. Les tâches précisées sont définies dans l'article 2 de la lettre d'engagement (cf. page 34). La recherche sur les codes à portée juridique ouvre un nouveau domaine prometteur, illustré par la création d'un workshop spécifique "Programming Languages and the Law" à POPL cette année.

5. Diffusion / *Dissemination*

Ce travail a fait l'objet d'une publication dans une conférence internationale de compilation, Compiler Construction [44] (jointe au dossier), accompagnée d'un artefact logiciel validé par les pairs [45], ainsi que de deux publications [42, 40] dans une conférence nationale de la communauté (les JFLA). Mlang [41] consiste en 10 000 lignes de code OCaml, et 600 lignes de formalisation Coq sur la sémantique de M.

Formulaire 4 — PROGRAMME DE RECHERCHE
Form 4 — RESEARCH PROGRAM

Étant donné l'organisation d'Inria, tout chercheur, ou toute chercheuse, a vocation à être affecté(e) dans une équipe-projet. Un candidat, ou une candidate, indique donc généralement dans son dossier de candidature l'équipe-projet dans laquelle elle ou il souhaite être affecté(e). Il est dans ce cas fortement recommandé de prendre préalablement contact avec la ou le responsable de l'équipe-projet souhaitée.

Il est néanmoins aussi possible de déposer une candidature sans préciser *a priori* une équipe-projet d'accueil. Dans ce cas, si la candidate ou le candidat est déclaré(e) admissible, une ou plusieurs équipes d'accueil pourront lui être proposées entre la phase d'admissibilité et la phase d'admission. Cette proposition d'affectation se fera en prenant en compte les aspirations de la candidate ou du candidat, celles des équipes, et la politique scientifique d'Inria.

Dans le cas où la candidature a lieu dans une équipe-projet, le candidat ou la candidate est invité(e) à expliquer dans son programme de recherche son intégration dans l'équipe-projet souhaitée.

Dans le cas où l'équipe-projet n'est pas choisie au moment de la candidature, le candidat ou la candidate n'est pas tenu(e) de détailler son intégration. Elle ou il peut néanmoins, sans que cela soit une obligation, indiquer des noms de chercheurs ou de chercheuses avec qui elle ou il pourrait collaborer en cas de recrutement.

☒ Je souhaite candidater dans l'équipe-projet, ou les équipes-projets suivante(s) : **SYCOMORES**

☐ **Je ne souhaite pas choisir d'équipe-projet pour l'instant. En cas d'admissibilité, je serai contacté(e) par la présidente ou le président du jury pour discuter de possibles équipes d'accueil.**

Given Inria's organization, any researcher should be assigned to a project-team. A candidate therefore generally indicates in his or her application file the project-team to which he or she wishes to be assigned. In this case, it is strongly recommended to contact the leader of the desired project-team beforehand.

However, it is also possible to submit an application without specifying a priori a host project-team. In this case, if the candidate is declared eligible, one or more host teams will be proposed between the eligibility phase and the admission phase. This assignment proposal will be made taking into account the aspirations of the candidate, those of the teams, and Inria's scientific policy.

In the case of an application in a project-team, the candidate is invited to explain in his or her research program the integration in the desired project-team.

In the case when the project-team is not selected at the time of the application, the candidate is not required to detail his or her integration. However, he or she may, without this being an obligation, indicate the names of researchers with whom he or she could collaborate in the case of recruitment.

☐ *I would like to apply for the following project-team(s) :*

☐ *I prefer not to choose a project-team for the moment. If I am considered eligible, I will be contacted by the chair of the jury to discuss possible host teams.*

Intitulé du programme de recherche : *Analyses statiques précises, sûres et efficaces pour les logiciels généralistes*
Title of research program

Taille maximum de cette partie : 3 pages (CRCN, ISFP et DR2)

Maximum size for this part: 3 pages (CRCN, ISFP and DR2)

Analyses statiques précises, sûres et efficaces pour les logiciels généralistes

Au début des années 2000, des chercheur·euse·s ont développé Astrée, un analyseur statique par interprétation abstraite qui a prouvé que les commandes de vol des Airbus A340 & A380 n'avaient pas d'erreurs à l'exécution [5, 11]. Depuis, des analyseurs de code moins spécialisés, tels que Facebook Infer [18, 20] et Google Tricorder [55], ont été développés, en particulier par les GAFAM. Ces outils fonctionnent sur des codes plus généralistes, passent à l'échelle sur des bases de code gigantesques, et les résultats d'analyse sont compréhensibles et utilisés par des développeur·euse·s. Néanmoins, ces analyseurs se comportent plutôt comme des détecteurs de bogues et ne sont pas sûrs : ils peuvent passer outre certains bogues. Par ailleurs, les propriétés qu'ils essayent d'établir ne sont pas clairement énoncées.
Mon but est de réunifier ces deux approches, afin d'obtenir des analyses statiques de programmes qui sont à la fois sûres et accessibles (en terme de précision et d'efficacité) pour être massivement adoptées par les développeur·euse·s. Dans un premier axe, je souhaite rendre les analyses de programmes plus utilisables, en améliorant leur passage à l'échelle et en les combinant avec d'autres méthodes pour discriminer la présence de bogues potentiels. Ces méthodes seront développées de manière générale, afin de pouvoir les appliquer à différents langages de programmation analysés. Mon second axe de recherche se concentre sur l'amélioration de la sûreté des analyses statiques. Je souhaite pour cela définir des méta-langages adaptés à la formalisation sémantique des langages de programmation, desquels des outils sémantiques (tels que des analyses automatiques et sûres) peuvent être dérivés automatiquement.
Un troisième axe de recherche, orthogonal aux précédents, se concentre sur l'application des méthodes formelles à une classe de programmes assez peu étudiés jusqu'ici, alors qu'ils ont des impacts sociétaux critiques : les implémentations de codes juridiques. Par exemple, l'impôt sur le revenu des particuliers rapporte autour de 75 milliards d'euros par an, soit 30% des recettes de l'État. S'assurer que ces implémentations sont maintenables et correctes est un enjeu majeur.

1. Rendre les analyses statiques plus utilisables

Je compte développer des techniques permettant d'améliorer le passage à l'échelle des analyses statiques basées sur l'interprétation abstraite, en réutilisant des invariants inférés précédemment, ou en inférant des contrats pour les fonctions. J'aimerais aussi développer des techniques de génération de contre-exemples lorsqu'un bug potentiel est détecté. Ces techniques permettront d'améliorer l'attractivité des analyses statiques auprès des développeur·euse·s. Ces techniques ne seront pas spécifiques à un langage de programmation, afin de pouvoir les intégrer dans différentes plateformes d'analyse statique, et de les évaluer simultanément sur plusieurs langages si Mopsa est utilisé.

Inférence de contrats pour les analyses de fonctions. La plupart des analyseurs précis actuels (Astrée [5], Frama-C [15]) agissent par inlining et analysent les fonctions à chacun des sites d'appel, ce qui est coûteux. Différentes approches infèrent des contrats pour les fonctions, qui peuvent donc être réutilisés. Plusieurs travaux se concentrent sur les fonctions numériques [6, 24, 26], et un autre est dédié à l'analyse de fonctions avec des pointeurs partagés dans la pile d'appel [59]. Ces analyses sont néanmoins trop spécialisées pour analyser des programmes généraux. La seule exception est une approche récente mêlant logique de séparation et analyse statique [27], capable d'analyser 3 000 lignes du code C de l'éditeur de texte Emacs. Cette approche nécessite néanmoins une intervention manuelle d'un utilisateur expert. Les analyses précédentes visent toutes des langages impératifs simplifiés proches de C. Dans le cas des langages de programmation dynamiques, le fort polymorphisme des fonctions rend leur analyse compositionnelle, sans aucun contexte d'appel, extrêmement difficile. J'envisage de développer des approches partiellement compositionnelles (comme Illous et al. [27]), basées sur de l'inlining, mais capables de généraliser les résultats d'analyses obtenus sous la forme de contrats pouvant être réutilisés a posteriori. Ce travail s'inscrit pleinement dans le développement d'approches de vérification modulaires de l'équipe SyCo**Mo**RES. Il pourrait être intégré dans l'analyse relationnelle de code binaire développée par Giuseppe Lipari, Julien Forget et Clément Ballabriga [3]. Ces approches partiellement compositionnelles nécessiteront de commencer par développer de nouvelles abstractions génériques et symboliques de l'état mémoire (les abstractions actuelles sont basées sur les sites d'allocation [2]). Un point de départ pourrait être le travail de Cox et al. [14], actuellement spécialisé pour le langage JavaScript. Ma connaissance précise d'analyse de langages différents (C et Python) sera un atout pour le développement d'une solution générique. Ces abstractions de l'état mémoire pourraient faire l'objet de collaboration avec l'équipe. En effet, Vlad Rusu, Julien Forget et Clément Ballabriga ont proposé un sujet de thèse intitulé "Analyse d'allocation mémoire dynamique pour la sécurité par interprétation abstraite de code binaire" [54]. Je me suis intéressé à l'adaptation au cas de l'analyse de Python d'une abstraction de l'allocation mémoire dynamique [2] dans mes travaux [49] [51, section 4.2]. De manière générale, mon intégration dans l'équipe consoliderait la connaissance en interprétation abstraite de l'équipe. Même si l'analyse de binaire peut sembler éloignée de celle de Python, certaines problématiques, comme l'absence d'information de typage [7] sont communes. La création d'analyses compositionnelles de fonctions est un but ambitieux, nécessitant un travail à long terme.

Analyses incrémentales. Les analyseurs statiques actuels combinent une partie faisant de l'inférence (par exemple d'invariants de boucles), avec la propagation de ces invariants dans la suite de l'analyse. Un même programme peut être analysé de manière répétée (dans le cas de l'intégration continue) : il serait intéressant de découpler inférence et

propagation. Cette propagation reviendrait à faire de la vérification de programmes, qui est un problème strictement plus simple en termes de calculabilité que de l'analyse [13]. La sauvegarde des invariants, par exemple grâce à des formules logiques, permettrait de les appliquer aux analyses suivantes et de réduire significativement le temps d'analyse des programmes. Additionnellement, l'utilisation de formules logiques simplifierait la communication avec des solveurs SMT. Cette approche s'inspire de la vérification par certificats, sur lequel j'ai travaillé en stage de M2 à Sarrebruck [4].
Les analyses statiques précises ont recours à des domaines relationnels, tels que les polyèdres [10] ou les octogones [46], qui permettent d'exprimer des relations linéaires entre des variables. Ces domaines sont néanmoins coûteux, avec une complexité au moins cubique en le nombre de variables utilisées. Une technique standard pour réduire leur coût est de remplacer un domaine relationnel par une union de domaines relationnels travaillant sur des groupements restreints de variables [5, section 3.9.6]. Ces groupements sont définis de manière heuristique et propres à des classes de programmes analysés. Je propose d'évaluer à court terme des techniques de raffinement itératif du groupement de ces variables sur des analyses successives, afin d'obtenir une approche générale. La première itération consisterait en une analyse non relationnelle. À la fin de chaque itération, une phase de diagnostic permettrait de détecter des relations nécessaires à l'amélioration de la précision, et de créer de nouveaux groupements. L'analyse de binaire développée par l'équipe est relationnelle et pourrait ainsi bénéficier des améliorations proposées ici. Cette analyse utilise une bibliothèque spécialisée pour le domaine des polyèdres [1]. Je pourrais apporter ma connaissance de l'utilisation de la bibliothèque de domaines numériques Apron [28] (que j'ai déjà manipulée lors des développement logiciels de Mopsa et Batman) afin de pouvoir instancier l'analyse avec d'autres domaines relationnels tels que les octogones [46]. Cela permettrait en outre de profiter des améliorations de performance récentes du domaine dans le cadre des domaines numériques relationnels [57, 58].

Génération de contre-exemples. Certaines analyses statiques – telles que celles développées durant mes travaux précédents – font des sur-approximations des états de programme accessibles afin de pouvoir terminer. Cela peut se traduire par la présence de fausses alarmes : l'analyse imprécise détecte une erreur potentielle à un certain point, alors que le programme est correct à cet endroit. Trop de fausses alarmes peuvent mener les développeur·euse·s à abandonner l'utilisation d'outils d'analyse statique [9]. Je souhaite explorer l'utilisation de techniques sous-approximantes (telles que l'exécution symbolique ou le fuzzing), guidées par les résultats des analyses statiques, afin de chercher des contre-exemples réels. La combinaison de fuzzing et d'analyse statique a déjà été utilisée dans des cas restreints, pour guider le fuzzing de parseurs [56] ou de smart contrats [65]. Dans le cas d'une cible binaire, les contre-exemples correspondraient à trouver des failles permettant d'exploiter directement le binaire. Cette approche pourrait compléter les développements récents de l'équipe, qui cherche à étendre l'analyse de binaire actuelle pour détecter des vulnérabilités logicielles.

2. Formalisation et analyse sûre de sémantiques complexes

L'interprétation abstraite fournit une méthodologie pour obtenir des analyses statiques, en les dérivant depuis la sémantique du langage cible, appelée sémantique concrète. Les analyseurs de programmes peuvent être vus comme des interprètes sur des "domaines abstraits" spécifiques, calculant de manière efficace une abstraction des valeurs accessibles pour chaque variable. Un analyseur est sûr lorsque la sémantique concrète du langage analysé est sur-approximée par l'interprète abstrait. Hormis Verasco [30], qui est un analyseur statique prouvé correct de bout en bout en Coq, les preuves de sûreté sont effectuées sur papier, ce qui réduit leur garantie de sûreté.

Sémantique formelle de Python. Le langage Python, qui est le deuxième langage le plus utilisé sur GitHub, n'est pas standardisé et ne possède actuellement pas de formalisation sémantique réaliste, qui se conforme aux tests de l'interprète de référence, CPython. En tirant profit de mon expertise sur la sémantique de Python acquise durant ma thèse, je souhaiterais développer à court terme une sémantique formelle de Python, en utilisant un formalisme adapté, capable d'extraire automatiquement un interprète de cette sémantique. Ce formalisme pourrait être $\mathbb{K}$, utilisé régulièrement par Vlad Rusu dans ses recherches. Une ébauche de formalisation avait été créée pour Python en 2012 par un étudiant de l'université d'Illinois [25], mais elle n'est pas assez complète pour supporter les tests de CPython et n'a pas été maintenue.
Une fois le comportement de Python modélisé par la sémantique, il faut encore vérifier que cette modélisation est correcte en la comparant à l'interprète de référence sur des tests. Je souhaite passer outre la limitation des travaux précédents sur Python (le seul accepté par des pairs étant [53]), qui n'avaient pas une sémantique suffisamment complète pour l'évaluer sur les tests de CPython. De manière duale, je souhaite pouvoir générer automatiquement des jeux de tests pour chacun des cas définis dans le formalisme sémantique. Cela permettrait d'obtenir des tests spécialisés, et une couverture systématique de la modélisation. Pour cela, il faudrait arriver à générer des cas de tests satisfaisant des contraintes sur l'exécution de la sémantique, ce qui est un problème difficile. Je compte d'abord tester une approche basée sur le fuzzing (plus efficace dans certains cas [35]) de la sémantique interprétée, ou utiliser de l'exécution symbolique [36] sinon.

Synthèse d'analyses à partir de sémantique concrète. Il serait intéressant d'arriver à synthétiser partiellement des analyses à partir de la sémantique concrète, pour avoir une méthode générale, indépendante du langage, et une garantie de sûreté par construction. Si les différents effets du langage (exceptions, allocation mémoire) ont bien été séparés dans la formalisation de la sémantique concrète, il sera possible de synthétiser seulement certains domaines abstraits et de

fournir manuellement des abstractions pour gérer ces effets de manière plus efficace. Cette séparation des effets est un des éléments clés de Mopsa que j'ai l'habitude de pratiquer suite à mes travaux. La preuve de sûreté d'une analyse composée de domaines générés automatiquement et de ceux ajoutés manuellement pourra bénéficier des recherches présentées dans le prochain paragraphe. À plus court terme, la synthèse d'outils de slicing [64] – basés sur des analyses de dépendances plus simples que les analyses numériques effectuées durant mes travaux antérieurs – pourrait être un premier projet dans cette direction. Mon but est d'arriver à générer automatiquement des parties d'analyses sûres en se basant sur un modèle sémantique du langage, afin d'obtenir une technique générale, indépendante du langage. Vlad Rusu a travaillé sur la dérivation d'exécutions symboliques depuis $\mathbb{K}$, avec une méthode indépendante du langage [37]. Le but de cet axe de recherche serait de développer des analyses qui sont sur-approximantes et qui terminent en temps fini.

Définitions modulaires de domaines abstraits. Les analyses statiques se composent de différents domaines abstraits complémentaires qui gèrent les différents effets du langage. La sûreté des analyses est exprimée via l'utilisation d'une fonction de concrétisation, qui permet de définir un état de l'analyse comme l'ensemble concret des états de programme correspondants. Les fonctions de concrétisation ainsi que les preuves de sûreté sont souvent faites sur papier, de manière monolithique, sur une combinaison spécifique de domaines abstraits. Il serait intéressant de rendre ces preuves plus modulaires, en considérant un domaine abstrait à la fois, afin de réduire les efforts de preuve lorsqu'un domaine change. La nouvelle forme de composition permise par Mopsa, où des domaines peuvent partager l'utilisation de domaines sous-jacents, nécessite de définir les concrétisations de manière modulaire afin que celles-ci soient correctes [51, section 2.4.6]. Cette composition est présente dans l'implémentation de Mopsa depuis 2018, mais l'écriture des concrétisations a longtemps été une question récurrente dans notre équipe de développement de Mopsa. J'ai enfin pu obtenir une forme satisfaisante pour la rédaction de mon manuscrit de thèse en 2021 [51]. Les preuves modulaires de sûreté ne sont pas encore établies et sont un objectif de recherche à moyen terme. Elles nécessiteront un travail fondamental consistant à développer de nouvelles techniques de preuves, qui pourra faire l'objet de collaborations avec Patrick Baillot et Vlad Rusu. En particulier, Vlad Rusu a déjà travaillé sur une formalisation d'analyseur via l'utilisation d'un système de contraintes [8].

3. Méthodes formelles pour les codes juridiques

Les implémentations liées aux codes juridiques – tels que le code calculant l'impôt sur le revenu en France, existant depuis 1990 – ont longtemps été privées, et donc non étudiées. Les administrations françaises sont dans l'obligation de publier les codes sources qu'elles utilisent depuis 2016 [70]. Le code de calcul de l'impôt sur le revenu des particuliers a été le premier publié, et l'objet de mes travaux précédents dans ce domaine [44, 42, 38]. L'accès à des bases de code réelles rend ce champ assez nouveau, mais l'impact potentiel de ces recherches est très large et propice à être compris par un grand nombre. Mon expérience actuelle de travaux en commun avec la Direction Générale des Finances Publiques (DGFiP) me permettra de dialoguer efficacement avec d'autres administrations, dans le but de créer de nouveaux partenariats.

De la loi vers le code, avec Catala. La structure spécifique des textes de loi rend difficile leur implémentation [43, section 2]. Additionnellement, la surcharge fréquente de certains textes par des nouveaux au fil des années complexifie la maintenance des implémentations, d'autant plus lorsqu'il n'y a pas de correspondance structurelle entre le code et les textes. Dans le cas de l'impôt sur le revenu, la structure n'est pas préservée dans le code M/M++, et 30% des 90 000 lignes de code ont été modifiées pour mettre à jour l'implémentation de 2019 à 2020. Des langages de programmation adaptés à l'encodage de la loi permettraient de faciliter ces implémentations, tout en augmentant la confiance qu'elles implémentent bien la spécification (la loi). Mes collaborateurs des travaux précédents – Denis Merigoux et Jonathan Protzenko – ont ainsi développé un nouveau langage prometteur, appelé Catala [43]. Je fais partie depuis le début de l'année 2022 de l'équipe de développement de Catala [39]. Cette équipe est internationale et multidisciplinaire ; elle est dirigée par Denis Merigoux (SRP dans l'équipe Prosecco, Inria Paris). Nous souhaitons vérifier que Catala peut passer à l'échelle sur des vraies bases de code tout en simplifiant la maintenance des codes juridiques. En particulier, la DGFiP souhaiterait porter le calcul de l'impôt sur le revenu, actuellement écrit en M et M++, en utilisant Catala. Ce projet est ambitieux et nécessitera un travail au long cours. Le portage sera fait de manière progressive, afin de pouvoir vérifier que l'implémentation est correcte au fil de l'eau, au coût du développement d'une solution interopérable avec la base de code actuelle.

Vérification de la loi encodée avec Catala. De manière générale, les textes de loi consistent en une clause générale, qui peut être surchargée par des cas exceptionnels. J'aimerais implémenter des techniques permettant de vérifier que les différents cas traités par un texte de loi ne créent pas d'ambigüités entre ceux-ci. Ces techniques pourraient s'intégrer dans la plateforme de preuve pour Catala proposée par Delaët et al. [16] lors d'un workshop. Posséder des techniques de décision automatiques sur ces cas permettra aussi de simplifier la compilation des programmes Catala. La structure des programmes Catala est différente des langages usuels, car chaque article de loi peut réutiliser d'autres articles en modifiant partiellement leur contexte. Catala est actuellement défini formellement par des traductions successives dans différents langages intermédiaires, jusqu'à un langage fonctionnel très simple. Une première étape de ce travail est de définir une formalisation sémantique de Catala afin de disposer de fondations solides pour développer des analyses de programmes. Cette formalisation sémantique pourrait être faite avec le framework $\mathbb{K}$ sur lequel Vlad Rusu a déjà travaillé.

Formulaire 5 — LISTE COMPLÈTE DES CONTRIBUTIONS[2]

Form 5 — COMPLETE LIST OF CONTRIBUTIONS[2]

Ordre des auteurs. Dans le domaine des méthodes formelles dont je fais partie, les articles sont majoritairement publiés dans des conférences internationales avec comité de sélection. Les articles dans les journaux avec comités de sélection sont plus rares. Les auteurs sont usuellement ordonnées par contribution décroissante. Pour les publications parlant de travaux de groupe sur Mopsa (VSTTE 2019 et JFLA 2021), les auteurs sont donnés par ordre alphabétique. Les trois premières publications (SAS 2021, CC 2021, ECOOP 2020) sont jointes au dossier de candidature. SAS est la conférence spécialisée dans le domaine de l'analyse statique par interprétation abstraite.

Légende.

- Artefacts logiciels : (ACM, artefacts validés, disponibles, extensibles), (autres).
- article double colonne (sinon).
- article démonstration d'un outil.

Les titres sont cliquables et renvoient aux informations de publication sur mon site, où tous les articles sont disponibles. Les acronymes des conférences pointent vers le programme de l'événement correspondant (prouvant ainsi l'authenticité de la publication), et les badges d'artefacts logiciels vers l'artefact validé par les pairs.

1. Publications caractéristiques/*Representative publications*

Dans cette sous-section, les publications sont ordonnées par année de publication croissante.

- Raphaël Monat, *Abdelraouf Ouadjaout, Antoine Miné*, Static Type Analysis by Abstract Interpretation of Python Programs. ECOOP 2020, 27 pages .

  Cette publication commence par exposer en détail quels traits de Python rendent une analyse statique difficile. Nous décrivons la sémantique concrète de certaines parties de Python. Nous formalisons ensuite les différents domaines utilisés pour définir l'analyse en elle-même. Un exemple est repris tout au long des différentes sections pour illustrer les différentes formalisations. Nous montrons, grâce à une implémentation de l'analyse dans Mopsa, que notre approche peut analyser des programmes de quelques milliers de lignes, ainsi qu'un petit utilitaire réel appelé PathPicker. Comparé aux cinq travaux précédents, notre approche est sûre, tout en gardant une efficacité et une précision similaires.

- Raphaël Monat, *Abdelraouf Ouadjaout, Antoine Miné*, A Multilanguage Static Analysis of Python Programs with Native C Extensions. SAS 2021, 23 pages .

  Cette publication décrit l'analyse multilangage entre les programmes Python et les bibliothèques C. L'analyse multilangage est nécessaire pour pouvoir analyser des programmes réels, qui utilisent souvent plusieurs langages de programmation. Cette analyse est la première capable de détecter les erreurs à l'exécution dans les différents langages composant un programme. L'approche est détaillée tout au long de l'article sur un exemple jouet autocontenu. Une formalisation de la sémantique multilangage est fournie. L'analyse a été implemntée dans Mopsa; elle est évaluée sur des bibliothèques populaires et open-source, comprenant quelques milliers de ligne de code Python et C.

- *Denis Merigoux,* Raphaël Monat, *Jonathan Protzenko*, A Modern Compiler for the French Tax Code. CC 2021, 11 pages .

  Cette publication présente Mlang, un compilateur open source permettant de répliquer le calcul de l'impôt sur le revenu. Mlang est basé sur une rétro-ingénierie du système de la DGFiP, et a été validé sur leur base de tests. Nous avons défini une sémantique formelle pour Mlang en Coq et éliminé du code palliatif qui avait été écrit en C via l'introduction d'un nouveau langage spécifique. Cela permet notamment de compiler la base de code vers différents langages (Python, C), et d'instrumenter le code généré, notamment pour générer nos propres tests.

[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 be, as much as possible, available for consultation via the web page of the applicant.

2. Publications

2.1 Revues internationales/*International journals*

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

- Raphaël Monat, *Abdelraouf Ouadjaout*, *Antoine Miné*, A Multilanguage Static Analysis of Python Programs with Native C Extensions. SAS 2021, 23 pages.
- *Denis Merigoux,* Raphaël Monat, *Jonathan Protzenko*, A Modern Compiler for the French Tax Code. CC 2021, 11 pages.
- Raphaël Monat, *Abdelraouf Ouadjaout, Antoine Miné*, Static Type Analysis by Abstract Interpretation of Python Programs. ECOOP 2020, 27 pages.
- *Matthieu Journault, Antoine Miné,* Raphaël Monat, *Abdelraouf Ouadjaout*, Combinations of Reusable Abstract Domains for a Multilingual Static Analyzer (Invité). VSTTE 2019, 16 pages.
- *Heiko Becker, Nikita Zyuzin,* Raphaël Monat, *Eva Darulova, Magnus O. Myreen, Anthony Fox*, A Verified Certificate Checker for Finite-Precision Error Bounds in Coq and HOL4. FMCAD 2018, 9 pages.
- Raphaël Monat, *Antoine Miné*, Precise Thread-Modular Abstract Interpretation of Concurrent Programs using Relational Interference Abstractions. VMCAI 2017, 17 pages.

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

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

Workshop international avec comité de lecture

- Raphaël Monat, *Abdelraouf Ouadjaout, Antoine Miné*, Value and Allocation Sensitivity in Static Python Analyses. SOAP 2020, 6 pages.

2.5 Revues nationales/*National journals*

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

- *Denis Merigoux,* Raphaël Monat, Mlang: an Open-Source Toolchain for the Income Tax Computation. JFLA 2021, 2 pages.
- *Matthieu Journault, Antoine Miné,* Raphaël Monat, *Antoine Miné*, Démonstration de la plateforme Mopsa d'analyse statique de programmes par interprétation abstraite. JFLA 2021, 2 pages.
- *Denis Merigoux,* Raphaël Monat, *Christophe Gaie*, Étude formelle de l'implémentation du code des impôts. JFLA 2020, 16 pages.

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

- Raphaël Monat, Static Type and Value Analysis by Abstract Interpretation of Python Programs with Native C Libraries. Manuscrit de thèse, 275 pages.
- Raphaël Monat, Static Analysis by Abstract Interpretation Collecting Types of Python Programs. Rapport de stage (M2 2018), 20 pages.
- Raphaël Monat, Certificate Checking in Coq and HOL4 for Static Analyses of Mixed-Precision Floating-Point Arithmetic. Rapport de stage (M2 2017), 24 pages.
- Raphaël Monat, Variational Inference in Probabilistic Programs: Formal Derivation of a Black-box Approach. Rapport de stage (M1), 26 pages.
- Raphaël Monat, Thread-Modular Analysis Designing Relational Abstractions of Interferences. Rapport de stage (L3), 21 pages.

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

Mopsa, plateforme open-source d'analyse statique de programmes. Mopsa est une plateforme d'analyse statique développée au LIP6 depuis 2016, suite à l'octroi d'un financement "Consolidator Grant" de l'ERC à Antoine Miné. Le développement est open-source (sous licence LGPL v3) et une version publique est disponible depuis Octobre 2020.
J'ai participé au développement de Mopsa, et intégré mes analyses dans cette plateforme. Mopsa est écrit en 60 000 lignes d'OCaml. Je suis le mainteneur des analyses Python (13 000 lignes), et seul développeur de l'analyse multilangage (2 700 lignes).
J'ai accordé une importance particulière à maintenir mes implémentations, ainsi qu'à vérifier que nos benchmarks passaient toujours au fil des modifications. J'ai mis en place une intégration continue de l'analyseur.
J'ai aussi créé deux artefacts logiciels [48, 52] validés par les pairs lors de nos publications ECOOP et SAS [47, 50].
`Family=research; Audience=partners; Evolution=lts; Duration=3.5; Contribution=devel,softcont;`
`Url=https://gitlab.com/mopsa/mopsa-analyzer`

Mlang, compilateur open-source pour le code des impôts. Mlang permet de répliquer le calcul de l'impôt sur le revenu. Je suis avec Denis Merigoux un des deux contributeurs principaux. Mlang est écrit en 10 000 lignes d'OCaml, et distribué sous licence GPL v3. Je participe au développement depuis mai 2019. J'ai créé un artefact logiciel [45] validé par les pairs lors de notre publication CC [44]. **Un transfert technologique est en cours depuis juin 2021.** Ce transfert vise tout d'abord à rendre Mlang interopérable avec les autres outils et la DGFiP, l'évaluer en condition réelles, et finaliser le développement de Mlang dans le cadre du calcul "correctif" (lors de contentieux entre l'administration et un particulier). Ce développement est fait de manière publique et open-source sur GitHub [41]. Nous assurons aussi la formation de l'équipe de la DGFiP (l'administration en charge des finances publiques) à la prise en main de Mlang. Depuis janvier 2022, je m'occupe du transfert (cf. lettre d'engagement page 34, en particulier l'article 2, et lien vers communiqué). J'encadre ainsi le développement de trois personnes (deux ingénieurs de recherche prestataires, issus d'OCamlPro, et un inspecteur programmeur système d'exploitation, fonctionnaire de la DGFiP), en dirigeant les choix techniques à mettre en œuvre et en passant en revue le code qu'ils écrivent. Le temps consacré à cette mission est estimé à 30 jours entre janvier et août 2022. Le calcul de l'impôt sur le revenu devrait utiliser du code compilé par Mlang en 2023.
`Family=transfer; Audience=community; Evolution=lts; Duration=2.75; Contribution=leader;`
`Url=https://github.com/MLanguage/mlang`

Daisy, compilateur de programmes numériques sur les réels vers des flottants. J'ai participé au développement de Daisy de février à juin 2017, lors de mon stage au MPI-SWS à Sarrebruck. Daisy est écrit en 14kLoc Scala.
`Family=research; Audience=partners; Evolution=basic; Duration=.4; Contribution=devel;`
`Url=https://github.com/malyzajko/daisy/`

FloVer, vérification mécanisée de certificats de correction générés par Daisy. J'ai généralisé la formalisation de FloVer (qui contient actuellement 10kLoc HOL et 25kLoc Coq), lors de mon stage au MPI-SWS à Sarrebruck. Je faisais partie des deux développeurs principaux (avec Heiko Becker, en thèse) durant ma période de stage. La généralisation effectuée avait augmenté la taille des formalisations de 30% pour Coq et 18% pour HOL. Daisy et FloVer ont été l'objet d'une publication de groupe à FMCAD'18.
`Family=research; Audience=partners; Evolution=basic; Duration=.4; Contribution=devel;`
`Url=https://gitlab.mpi-sws.org/AVA/FloVer`

Batman, preuve de concept d'un analyseur statique capable d'analyser différents threads de manière modulaire. Je suis le seul contributeur de cette preuve de concept. J'ai commencé le développement durant mon stage de L3, et je l'ai poursuivi de manière intermittente jusqu'en janvier 2017. Batman est écrit en 5,5kLoc OCaml, et distribué sous licence GPL v3. Ce prototype a été utilisé pour l'évaluation expérimentale de notre papier à VMCAI'17.
`Family=research; Audience=personal; Evolution=nofuture; Duration=1.5; Contribution=leader;`
`Url=https://github.com/rmonat/batman`

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

Cette partie vous permet de décrire vos activités ayant eu un impact socio-économique (une fiche par activité). Elle ne se limite pas au transfert industriel proprement dit, et inclut également le transfert vers des partenaires publics ou associatifs, les actions de standardisation...

Please describe your activities that have had a socio-economic impact. Note that this is not limited to industrial transfer, and also includes transfer to public bodies or associations, standardization initiatives...

Pour chaque action de transfert (technologique ou sociétal), fournissez une description selon le canevas disponible dans le document « Evaluation des contributions scientifiques en matière de transfert / Guide méthodologique » disponible à l'URL https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique_EvaluationTransfert%281%29.pdf.

For each transfer action (technology or society transfer) please describe your achievements following the guidelines « Evaluating scientific contributions in relation to transfer / Methodological guide » available at the URL https://www.inria.fr/sites/default/files/2020-01/2018-06-GuideMethodologique-EvaluationTransfert_EN%281%29.pdf.

Transfert du compilateur Mlang auprès de la direction générale des finances publiques (DGFiP). Décrit dans la section précédente.

Bibliographie / *Bibliography*

[1] R. Bagnara, P. M. Hill, and E. Zaffanella. The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.*, 72(1-2):3–21, 2008.

[2] G. Balakrishnan and T. W. Reps. Recency-abstraction for heap-allocated storage. In *Static Analysis Symposium (SAS)*, pp. 221–239, 2006.

[3] C. Ballabriga, J. Forget, L. Gonnord, G. Lipari, and J. Ruiz. Static analysis of binary code with memory indirections using polyhedra. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pp. 114–135, 2019.

[4] H. Becker, N. Zyuzin, R. Monat, E. Darulova, M. O. Myreen, and A. C. J. Fox. A verified certificate checker for finite-precision error bounds in coq and HOL4. In *Formal Methods in Computer-Aided Design*, pp. 1–10, 2018.

[5] J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. *Foundations and Trends in Programming Languages*, pp. 71–190, 2015.

[6] R. Boutonnet and N. Halbwachs. Disjunctive relational abstract interpretation for interprocedural program analysis. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pp. 136–159, 2019.

[7] J. Caballero and Z. Lin. Type inference on executables. *ACM Comput. Surv.*, 48(4):65:1–65:35, 2016.

[8] D. Cachera, T. P. Jensen, D. Pichardie, and V. Rusu. Extracting a data flow analyser in constructive logic. In D. A. Schmidt, editor, *European Symposium on Programming ESOP*, 2004.

[9] M. Christakis and C. Bird. What developers want and need from program analysis: an empirical study. In *International Conference on Automated Software Engineering*, 2016.

[10] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Principles of Programming Languages (POPL)*, pp. 84–96, 1978.

[11] P. Cousot, R. Cousot, J. Feret, A. Miné, X. Rival, B. Blanchet, D. Monniaux, and L. Mauborgne. The astrée static analyzer, 2004. URL https://www.astree.ens.fr/. Accessed: 2021-12.

[12] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the Astrée static analyzer. In *Annual Asian Computing Science Conference (ASIAN)*, pp. 272–300, 2006.

[13] P. Cousot, R. Giacobazzi, and F. Ranzato. Program analysis is harder than verification: A computability perspective. In *International Conference in Computer-Aided Verification*, pp. 75–95, 2018.

[14] A. Cox, B. E. Chang, and X. Rival. Automatic analysis of open objects in dynamic language programs. In *Static Analysis Symposium (SAS)*, pp. 134–150, 2014.

[15] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C - A software analysis perspective. In *International Conference on Software Engineering and Formal Methods*, pp. 233–247, 2012.

[16] A. Delaët, D. Merigoux, and A. Fromherz. Turning catala into a proof platform for the law. In *Programming Languages and the Law (POPL workshop)*, 2022.

[17] D. Delmas and J. Souyris. Astrée: From research to industry. In *Static Analysis Symposium (SAS)*, pp. 437–451, 2007.

[18] T. I. development team. Infer, a static analysis tool for Java, C++, Objective-C, and C., 2021. URL https://github.com/facebook/infer. Accessed: 2021-07.

[19] Direction Générale des Finances Publiques (DGFiP). Les règles du moteur de calcul de l'impôt sur le revenu, 2019. URL https://gitlab.adullact.net/dgfip/ir-calcul.

[20] D. Distefano, M. Fähndrich, F. Logozzo, and P. W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, pp. 62–70, 2019.

[21] A. Fromherz, A. Ouadjaout, and A. Miné. Static value analysis of python programs by abstract interpretation. In *Nasa Formal Methods (NFM)*, pp. 185–202, 2018.

[22] M. Furr and J. S. Foster. Checking type safety of foreign function calls. *ACM Trans. Program. Lang. Syst.*, pp. 18:1–18:63, 2008.

[23] GitHub. State of the github octoverse. https://octoverse.github.com/#top-languages-over-the-years, 2021. Accessed: 2021-09.

[24] E. Goubault, S. Putot, and F. Védrine. Modular static analysis with zonotopes. In *Static Analysis Symposium (SAS)*, pp. 24–40, 2012.

[25] D. Guth. A formal semantics of Python 3.3. Master's thesis, University of Illinois, 2013. URL https://www.ideals.illinois.edu/bitstream/handle/2142/45275/Dwight_Guth.pdf?sequence=1&isAllowed=y.

[26] J. Henry. *Static Analysis by Abstract Interpretation and Decision Procedures. (Analyse statique de programme par interprétation abstraite et procédures de décision)*. PhD thesis, University of Grenoble, France, 2014.

[27] H. Illous, M. Lemerre, and X. Rival. Interprocedural shape analysis using separation logic-based transformer summaries. In *Static Analysis Symposium (SAS)*, 2020.

[28] B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *Computer Aided Verification*, pp. 661–667. Springer, 2009.

[29] S. H. Jensen, A. Møller, and P. Thiemann. Type analysis for JavaScript. In *Static Analysis Symposium (SAS)*, pp. 238–255, 2009.

[30] J. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. A formally-verified C static analyzer. In *Principles of Programming Languages*, 2015.

[31] M. Journault, A. Miné, R. Monat, and A. Ouadjaout. Combinations of reusable abstract domains for a multilingual static analyzer. In *Verified Software: Theories, Tools, Experiments (VSTTE)*, pp. 1–18, 2019.

[32] M. Journault, A. Miné, R. Monat, and A. Ouadjaout. MOPSA: modular open platform for static analysis. https://gitlab.com/mopsa/mopsa-analyzer, 2021. Accessed: 2021-04.

[33] M. Journault, A. Miné, R. Monat, and A. Ouadjaout. Benchmarks used by MOPSA. https://gitlab.com/mopsa/benchmarks, 2021. Accessed: 2021-09.

[34] S. Lee, H. Lee, and S. Ryu. Broadening horizons of multilingual static analysis: Semantic summary extraction from C code for JNI program analysis. In *Automated Software Engineering (ASE)*, pp. 127–137, 2020.

[35] D. Liew, C. Cadar, A. F. Donaldson, and J. R. Stinnett. Just fuzz it: solving floating-point constraints using coverage-guided fuzzing. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE)*, 2019.

[36] B. Loring and J. Kinder. Systematic generation of conformance tests for javascript. *CoRR*, abs/2108.07075, 2021.

[37] D. Lucanu, V. Rusu, and A. Arusoaie. A generic framework for symbolic execution: A coinductive approach. *J. Symb. Comput.*, 80:125–163, 2017.

[38] R. M. e. A. O. Matthieu Journault, Antoine Miné. Démonstration de la plateforme mopsa d'analyse statique de programmes par interprétation abstraite. In *32ème Journées Francophones des Langages Applicatifs*, pp. 45–47, 2021.

[39] D. Merigoux. Catala contributors. https://catala-lang.org/en/about, 2022. Accessed: 2022-02.

[40] D. Merigoux and R. Monat. Mlang: an open-source toolchain for the income tax computation. In *32ème Journées Francophones des Langages Applicatifs*, pp. 155–156, 2021.

[41] D. Merigoux and R. Monat. Mlang compiler. https://github.com/MLanguage/mlang, 2021. Accessed: 2021-12.

[42] D. Merigoux, R. Monat, and C. Gaie. Étude formelle de l'implémentation du code des impôts. In *31ème Journées Francophones des Langages Applicatifs*, pp. 31–46, 2020.

[43] D. Merigoux, N. Chataing, and J. Protzenko. Catala: a programming language for the law. *Proc. ACM Program. Lang.*, (International Conference on Functional Programming), 2021.

[44] D. Merigoux, R. Monat, and J. Protzenko. A Modern Compiler for the French Tax Code. In *Compiler Construction (CC)*, pp. 71–82, 2021.

[45] D. Merigoux, R. Monat, and J. Protzenko. A Modern Compiler for the French Tax Code - Artefact, January 2021. URL https://doi.org/10.5281/zenodo.4456774.

[46] A. Miné. The octagon abstract domain. *High. Order Symb. Comput.*, pp. 31–100, 2006.

[47] R. Monat, A. Ouadjaout, and A. Miné. Static type analysis by abstract interpretation of python programs. In *European Conference on Object-Oriented Programming (ECOOP)*, pp. 1–29, 2020.

[48] R. Monat, A. Ouadjaout, and A. Miné. Static type analysis by abstract interpretation of python programs (artefact). *Dagstuhl Artifacts Ser.*, pp. 11:1–11:6, 2020.

[49] R. Monat, A. Ouadjaout, and A. Miné. Value and allocation sensitivity in static Python analyses. In *ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis (SOAP)*, pp. 8–13, 2020.

[50] R. Monat, A. Ouadjaout, and A. Miné. A multilanguage static analysis of python programs with native C extensions. In *Static Analysis Symposium (SAS)*, pp. 323–345, 2021.

[51] R. Monat. *Static Type and Value Analysis by Abstract Interpretation of Python Programs with Native C Libraries*. PhD thesis, Sorbonne Université, France, 2021. 275 pages.

[52] R. Monat, A. Ouadjaout, and A. Miné. A Multi-Language Static Analysis of Python Programs with Native C Extensions - Artefact, July 2021. URL https://doi.org/10.5281/zenodo.5141314.

[53] J. G. Politz, A. Martinez, M. Milano, S. Warren, D. Patterson, J. Li, A. Chitipothu, and S. Krishnamurthi. Python: The full monty. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, pp. 217–232, 2013.

[54] V. Rusu, J. Forget, and C. Ballabriga. Proposition de thèse : Analyse d'allocation mémoire dynamique pour la sécurité par interprétation abstraite de code binaire. Annonce : https://cristal.univ-lille.fr/sujets-these/details.html?id=ce67f9c14ec24c969bf2df6bf3b99cbb, détails : https://cristal.univ-lille.fr/sujets-these/index.cgi/ce67f9c14ec24c969bf2df6bf3b99cbb/pdf. Accessed: 2022-01.

[55] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter. Tricorder: Building a program analysis ecosystem. In *International Conference on Software Engineering (ICSE)*, pp. 598–608, 2015.

[56] B. Shastry, M. Leutner, T. Fiebig, K. Thimmaraju, F. Yamaguchi, K. Rieck, S. Schmid, J. Seifert, and A. Feldmann. Static program analysis as a fuzzing aid. In *Research in Attacks, Intrusions and Defenses (RAID)*, pp. 26–47, 2017.

[57] G. Singh, M. Püschel, and M. T. Vechev. Making numerical program analysis fast. In *Programming Language Design and Implementation (PLDI)*, pp. 303–313, 2015.

[58] G. Singh, M. Püschel, and M. T. Vechev. Fast polyhedra abstract domain. In *Principles of Programming Languages*, pp. 46–59, 2017.

[59] P. Sotin and B. Jeannet. Precise interprocedural analysis in the presence of pointers to the stack. In *European Symposium on Programming (ESOP)*, pp. 459–479, 2011.

[60] V. Stinner and the Python Benchmark Suite team. Performance benchmarks from Python's reference interpreter. https://github.com/python/pyperformance/. Accessed: 2021-08.

[61] G. Tan and G. Morrisett. Ilea: inter-language analysis across Java and C. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, pp. 39–56, 2007.

[62] The Coq Development Team. The coq proof assistant, October 2017. URL http://coq.inria.fr.

[63] F. Wei, X. Lin, X. Ou, T. Chen, and X. Zhang. JN-SAF: precise and efficient NDK/JNI-aware inter-language static analysis framework for security vetting of Android applications with native code. In *Computer and Communications Security (CCS)*, pp. 1137–1150, 2018.

[64] M. Weiser. Program slicing. In *International Conference on Software Engineering (ICSE)*, pp. 439–449, 1981.

[65] V. Wüstholz and M. Christakis. Targeted greybox fuzzing with static lookahead analysis. In *International Conference on Software Engineering (ICSE)*, pp. 789–800, 2020.

[66] Avec Mlang, Inria participe à la modernisation du calcul de l'impôt sur le revenu. https://www.inria.fr/fr/mlang-modernisation-calcul-impot-revenu.

[67] Facebook's 2021 testing and verification symposium. https://fbresearchevents.bevylabs.com/events/details/facebook-research-facebook-research-events-presents-2021-testing-and-verification-symposium/.

[68] Facebook PathPicker. https://github.com/facebook/PathPicker. Accessed: 2021-12.

[69] Un hackathon pour l'ouverture du code source du calculateur de l'impôt sur le revenu. https://www.economie.gouv.fr/hackathon-calculateur-impots.

[70] Loi république numérique du 7 octobre 2016. https://www.vie-publique.fr/eclairage/20301-loi-republique-numerique-7-octobre-2016-loi-lemaire-quels-changements. Accessed: 2021-11.