Compte-rendu de fin de projet ANR-15-CE25-0008 "VOCaL"

Jean-Christophe Filliâtre

Compte-rendu de fin de projet

Project ANR-15-CE25-0008

VOCaL

Programme CE25 2015

A. Identification

Acronyme du projet	VOCaL
Titre du projet	Bibliothèque OCaml vérifiée
Coordinateur du projet (société/organisme)	Jean-Christophe Filliâtre (CNRS)
Période du projet (date de début — date de fin)	01/10/2015 – 31/3/2021
Site web du projet, le cas échéant	https://vocal.lri.fr/

Rédacteur de ce rapport	
Civilité, prénom, nom	M. Jean-Christophe Filliâtre
Téléphone	01 69 15 70 48
Adresse électronique	Jean-Christophe.Filliatre@lri.fr
Date de rédaction	juin 2021

Si différent du rédacteur, indiquer un contact pour le projet	
Civilité, prénom, nom	
Téléphone	
Adresse électronique	

Liste des partenaires présents à la fin du projet (société/organisme et responsable scientifique)	Laboratoire Méthodes Formelles (Jean-Christophe Filliâtre) Inria Paris (François Pottier) Verimag (Jean-François Monin) TrustInSoft (David Maison) OCamlPro (Fabrice Le Fessant)

B. Résumé consolidé public

B.1. Résumé consolidé public en français

VOCaL — vérification déductive de bibliothèques OCaml

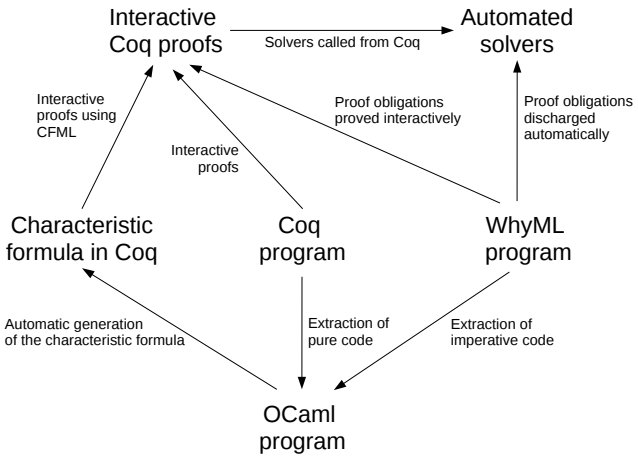
Des bibliothèques de confiance pour des logiciels plus sûrs. Aussi surprenant que cela puisse paraître, très peu de bibliothèques de langages de programmation offrent des garanties de fiabilité. Le projet VOCaL y remédie, en proposant des bibliothèques de structures de données où la correction fonctionnelle est établie à l'aide d'outils de vérification déductive. Le projet VOCaL se focalise sur le langage OCaml, un langage de choix pour des applications où la sûreté et la correction sont essentielles. Le premier objectif du projet VOCaL est de munir le langage OCaml d'un langage de spécification formelle, à l'instar de ce qui existe déjà pour les langages C et Java, dont on peut espérer qu'il sera réutilisé dans d'autres contextes (documentation, vérification dynamique, *model checking*, etc.). Au-delà, le projet VOCaL entend contribuer à la vérification déductive sur plusieurs plans : la proposition de nouvelles techniques de preuve, et notamment de preuve de complexité ; l'amélioration d'outils de vérification déductive et de démonstration automatique ; l'exploration d'études de cas originales et non triviales.

Une approche originale centrée sur la collaboration de plusieurs outils de vérification. Le projet VOCaL se distingue par une approche impliquant plusieurs outils de vérification, à savoir l'assistant de preuve Coq, et notamment plusieurs instances de la logique de séparation à l'intérieur de Coq (à savoir CFML et Iris), et l'outil Why3. Une telle approche tire les bénéfices maximaux de ces différents outils : une vérification la plus automatique possible avec Why3 lorsque cela est possible, en utilisant notamment des démonstrateurs tels que Alt-Ergo, et une vérification interactive avec Coq lorsque la nature du code ou des spécifications ne se prête plus à une vérification automatique. Le projet VOCaL se distingue également par une contribution significative à la preuve de complexité d'un programme, au travers de la notion de crédits-temps, et à la preuve d'absence de débordement arithmétique, au travers de la notion de reçus-temps. Enfin, le projet VOCaL contribue à un important développement logiciel, qu'il s'agisse d'outils de vérification et de programmes formellement vérifiés.

Résultats majeurs. Le résultat principal du projet VOCaL est la conception, l'implémentation et la distribution de **Gospel**, un langage de spécification formelle pour le langage OCaml. Une autre contribution logicielle du projet VOCaL est la réalisation et la distribution d'une bibliothèque OCaml de structures de données formellement vérifiée. Une contribution scientifique majeure du projet VOCaL est une extension de la logique de séparation pour raisonner sur la complexité des programmes, ainsi que son outillage. À ces contributions s'ajoutent l'amélioration d'outils tels que CFML, Alt-Ergo et Why3, des techniques de preuve nouvelles et de très nombreuses études de cas.

Production. La production du projet VOCaL inclut 35 articles scientifiques (revues, conférences internationales et nationales), 5 rapports techniques et 3 logiciels. Ces derniers sont des logiciels libres, sous licence MIT, dont une implémentation du langage Gospel, potentiellement réutilisable dans d'autres projets, et d'une bibliothèque OCaml de structures de données formellement vérifiée d'ores et déjà disponible pour la communauté OCaml.

Illustration. La figure suivante illustre les interactions entre les différents outils et langages utilisés dans le projet VOCaL :



Informations factuelles. Le projet VOCaL est un projet de recherche fondamentale et appliquée coordonné par Jean-Christophe Filliâtre (Laboratoire Méthodes Formelles). Il associe les laboratoires LMF (ex-LRI), Inria Paris et Verimag et les entreprises OCamlPro et TrustInSoft. Le projet a commencé en octobre 2015 et a duré 66 mois. Il a bénéficié d'une aide ANR de 764 k€ pour un coût global de l'ordre de 2,5M€.

B.2. Résumé consolidé public en anglais

VOCaL — deductive verification of OCaml libraries

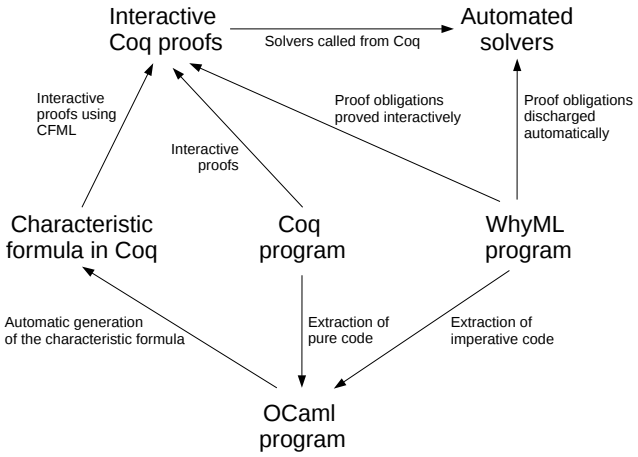
Trustworthy libraries for better software. Surprisingly, few programming libraries provide soundness guarantees. The VOCaL project intends to offer a solution, by providing data structure libraries where soundness is established using deductive verification tools. The VOCaL project focuses on the OCaml programming language, a language of choice for software where safety and soundness are of utmost importance. The first goal of VOCaL is to equip OCaml with a behavioral specification language, similarly to what already exists for C and Java, with the aim that it will also be used in other contexts (documentation, runtime verification, model checking, etc.). Beyond, the VOCaL project intends to contribute to deductive verification on many levels : new proof techniques, and notably to verify complexity ; improvement of deductive verification and automated theorem proving tools ; exploration of original and non trivial case studies.

An original approach based on the collaboration of several verification tools. The VOCaL project distinguishes itself from other approaches by simultaneous use of several verification tools, namely the Coq proof assistant, and notable several instances of separation logic within Coq (namely CFML and Iris), and the Why3 platform. Such an approach maximizes the benefits of these tools : a verification process as automated as possible with Why3 when possible, involving SMT solvers such as Alt-Ergo, and an interactive verification process with Coq when the code or the specification is no more amenable to an automatic verification. The VOCaL project also distinguishes itself by a significant contribution to the proof of complexity, through the notion of time credits, and to the absence of arithmetic overflow, through the notion of time receipts. Last, the VOCaL project contributes to a large body of free software, including tools and verified programs.

Main results. The main result of the VOCaL project is the design and implementation of **Gospel**, a behavioral specification language for OCaml. Another software contribution is the elaboration of a formally verified library of data structures for OCaml. A major scientific contribution of the project is an extension of separation logic to reason about programs complexity, and its implementation into existing separation logic frameworks. Other contributions include the improvement of several tools such as CFML, Alt-Ergo, and Why3, novel proof techniques and many case studies.

Production. The scientific production of the VOCaL project includes 35 scientific papers (journals, international and national conferences), 5 technical reports and 3 software. The latter are free software, under MIT license, including an implementation of the Gospel language, which is expected to be reused in other projects, and a formally verified library of data structures for OCaml, which is already available for the OCaml community.

Illustration. The following figures illustrates the interactions between the various tools and languages used in the VOCaL project :



Facts. The VOCaL project is a fundamental and applied research project coordinated by Jean-Christophe Filliâtre (Laboratoire Méthodes Formelles). It involves the research labs LMF (ex-LRI), Inria Paris and Verimag, and the companies OCamlPro and TrustInSoft. The project started on October 2015 and lasted 66 months. It was supported by a 764 k€ grant from ANR, for a total cost of 2,5M€.

C. Mémoire scientifique

Mémoire scientifique confidentiel : non

C.1. Résumé du mémoire

(repris du résumé consolidé public)

Aussi surprenant que cela puisse paraître, très peu de bibliothèques de langages de programmation offrent des garanties de fiabilité. Le projet VOCaL y remédie, en proposant des bibliothèques de structures de données où la correction fonctionnelle est établie à l'aide d'outils de vérification déductive. Le projet VOCaL se focalise sur le langage OCaml, un langage de choix pour des applications où la sûreté et la correction sont essentielles. Le premier objectif du projet VOCaL est de munir le langage OCaml d'un langage de spécification formelle, à l'instar de ce qui existe déjà pour les langages C et Java, dont on peut espérer qu'il sera réutilisé dans d'autres contextes (documentation, vérification dynamique, *model checking*, etc.). Au-delà, le projet VOCaL entend contribuer à la vérification déductive sur plusieurs plans : la proposition de nouvelles techniques de preuve, et notamment de preuve de complexité ; l'amélioration d'outils de vérification déductive et de démonstration automatique ; l'exploration d'études de cas originales et non triviales.

Le projet VOCaL se distingue par une approche impliquant plusieurs outils de vérification, à savoir l'assistant de preuve Coq, et notamment plusieurs instances de la logique de séparation à l'intérieur de Coq (à savoir CFML et Iris), et l'outil Why3. Une telle approche tire les bénéfices maximaux de ces différents outils : une vérification la plus automatique possible avec Why3 lorsque cela est possible, en utilisant notamment des démonstrateurs tels que Alt-Ergo, et une vérification interactive avec Coq lorsque la nature du code ou des spécifications ne se prête plus à une vérification automatique. Le projet VOCaL se distingue également par une contribution significative à la preuve de complexité d'un programme, au travers de la notion de crédits-temps, et à la preuve d'absence de débordement arithmétique, au travers de la notion de reçus-temps. Enfin, le projet VOCaL contribue à un important développement logiciel, qu'il s'agisse d'outils de vérification et de programmes formellement vérifiés.

Le résultat principal du projet VOCaL est la conception, l'implémentation et la distribution de **Gospel**, un langage de spécification formelle pour le langage OCaml. Une autre contribution logicielle du projet VOCaL est la réalisation et la distribution d'une bibliothèque OCaml de structures de données formellement vérifiée. Une contribution scientifique

majeure du projet VOCaL est une extension de la logique de séparation pour raisonner sur la complexité des programmes, ainsi que son outillage. À ces contributions s'ajoutent l'amélioration d'outils tels que CFML, Alt-Ergo et Why3, des techniques de preuve nouvelles et de très nombreuses études de cas.

C.2. Enjeux et problématique, état de l'art

Les enjeux initiaux du projet VOCaL étaient formulés de la façon suivante :

— Concevoir et distribuer la première bibliothèque de structures de données et algorithmes formellement prouvée pour le langage OCaml.
— Démontrer la possible collaboration de plusieurs outils de vérification dans une telle entreprise.
— Améliorer l'état de l'art des outils et techniques de vérification déductive.

Le projet VOCaL s'est appuyé sur une expertise en méthodes formelles des cinq partenaires, notamment en conception de langages de spécification, en vérification déductive et en démonstration automatique. Le projet VOCaL s'est également appuyé sur la maîtrise et le développement de plusieurs outils au sein des cinq partenaires, à savoir Coq, CFML, Alt-Ergo et Why3, ainsi que sur une expertise commune du langage OCaml.

Le projet VOCaL a pu se reposer sur un état de l'art déjà conséquent en matière de vérification déductive et de vérification de bibliothèques, incluant notamment

— de la vérification déductive reposant sur un générateur d'obligations de preuves, tel que Dafny [12], VCC [4], VeriFast [9], KeY [1] ou encore Why3 [6] ;
— de la vérification déductive au sein d'un assistant de preuve, tel que Coq [20], PVS [13], Isabelle [15], ou encore ACL2 [11] ;
— des exemples de bibliothèques vérifiées, telle que EiffelBase2 [17] ou encore le tri fusion de GHC [18], y compris en OCaml, comme les ensembles et dictionnaires de la bibliothèque standard d'OCaml [5] ou encore des structures de données issues du livre d'Okasaki [2].

C.3. Approche scientifique et technique

VALS

Note : L'équipe VALS était une équipe du LRI (Laboratoire de Recherche en Informatique), laboratoire qui n'existe plus depuis le 1er janvier 2021. Ses membres ont intégré le LMF (Laboratoire Méthodes Formelles), créé au 1er janvier 2021. Dans ce rapport, on continue d'utiliser VALS pour désigner ce partenaire.

La recherche effectuée dans l'équipe VALS est centrée sur les méthodes formelles, dans un sens large : test, *model checking*, démonstration automatique, vérification déductive. Son activité est caractérisée par une approche orientée solutions, avec une part importante de développement logiciel. En particulier, l'équipe VALS est à l'origine du démonstrateur SMT Alt-Ergo, aujourd'hui développé par OCamlPro. L'équipe VALS développe également l'outil Why3 [19], une plateforme logicielle pour la vérification déductive largement utilisée par le projet VOCaL.

Note : L'un des membres de l'équipe VALS au démarrage du projet, Arthur Charguéraud, a quitté l'équipe en cours de projet pour rejoindre l'équipe Inria Camus à Strasbourg, mais a continué à contribuer au projet.

Gallium & Camus

L'équipe Gallium de l'Inria Paris, renommée Cambium en 2019, étudie (entre autres) la méta-théorie et les applications de la Logique de Séparation d'ordre supérieur. Cette puissante logique de programmes permet de vérifier des logiciels (ou des composants logiciels) qui utilisent l'état modifiable, l'allocation dynamique de mémoire, les fonctions d'ordre supérieur, et l'abstraction de données.

Pendant la durée du projet VOCaL, François Pottier (Gallium) et Arthur Charguéraud (Camus) ont étendu la Logique de Séparation en lui ajoutant de nouveaux traits, dont les **crédits-temps** et les **permissions temporaires en lecture seule**. Ils ont implémenté une partie de ces extensions dans l'outil CFML [2, 3], un système composé d'un générateur de « formules caractéristiques » et d'une bibliothèque Coq. Ils ont appliqué cet outil à la vérification de structures de données et d'algorithmes évolués, dont **une table de hachage et ses itérateurs** (correction fonctionnelle), **une structure de données union-find** (correction fonctionnelle et complexité en temps), et **une structure de données pour la détection incrémentale de cycles** (correction fonctionnelle et complexité en temps). Dans cette approche, le code est d'abord écrit à la main en OCaml ; puis il est automatiquement traduit par l'outil CFML en une forme qui permet le raisonnement en Coq.

François Pottier (Gallium) et Jacques-Henri Jourdan (VALS) ont également travaillé dans le cadre d'Iris [10], une autre implémentation de la Logique de Séparation. Ils ont proposé le concept de **reçus-temps**, implémenté les crédits-temps et les reçus-temps en Iris, et utilisé Iris pour vérifier **un algorithme incrémental de calcul de plus petit point fixe** (correction fonctionnelle). Dans ce cas, il n'y a (pour le moment) pas de traduction automatique d'OCaml vers Coq. Le code OCaml doit être traduit manuellement en une forme qui permet le raisonnement en Coq.

Notre travail sur les crédits-temps a inspiré d'autres chercheurs. Reproduisant notre démarche, Zhan et Haslbeck implémentent la Logique de Séparation avec crédits-temps dans Isabelle/HOL [21]. Haslbeck et Nipkow proposent une étude

comparative de plusieurs logiques de programmes capables d'exprimer des bornes sur la complexité en temps [8]. Haslbeck et Lammich [7] vérifient un algorithme de tri récent en Logique de Séparation avec crédits-temps dans Isabelle/HOL et le compilent (de façon prouvablement correcte) vers LLVM.

Verimag

Des approches complémentaires à la vérification de programmes OCaml ont été également développées. L'une d'entre elles (FVDP, pour *Formally-Verified Defensive Programming*) consiste à embarquer dans Coq des fonctions OCaml externes "impures" vues come oracles non déterministes non fiables, en considérant seulement leur type ML, en combinant l'idée des *théorèmes gratuits* de Wadler pour les types polymorphes avec de la programmation défensive vérifiée exprimée en Coq. Une autre, appelée *méthode de Braga*, vise à faciliter considérablement l'extraction de programmes OCaml récursifs corrects par construction à partir d'un développement Coq – un mécanisme notoirement utilisé dans CompCert – particulièrement dans le cas où la terminaison des programmes OCaml visés ne provient pas d'une simple récursion structurelle sur une donnée en entrée, mais d'un raisonnement arbitrairement complexe pouvant nécessiter des lemmes préalables de correction partielle. Il a fallu pour cela examiner en détail le langage fonctionnel "frère" de OCaml que contient Coq ainsi que ses mécanismes de preuve, tant pour isoler les aspects impurs (pour FVDP) que pour gérer convenablement des fonctions non nécessairement totales (méthode de Braga).

C.4. Résultats obtenus

Les contributions principales, et multipartenaires, du projet sont
— la conception de Gospel [M2], un langage de spécification formelle pour OCaml ;
— son implémentation et sa distribution comme un logiciel libre [L1] ;
— son application à une bibliothèque d'algorithmes et structures de données formellement vérifiée, VOCaL [M1], également distribuée comme un logiciel libre [L3].

VALS

L'équipe VALS a contribué à une large part de l'implémentation de Gospel [L1], principalement réalisée par Cláudio Lourenço en post-doctorat dans l'équipe VALS sur la période 01/09/2019–31/08/2020. Par ailleurs, un greffon Why3 a été développé [L2], qui traduit les spécifications Gospel en spécifications Why3. Il est alors possible de réaliser une implémentation en WhyML (le langage de programmation de Why3), de faire la preuve de raffinement côté Why3, puis d'obtenir du code OCaml automatiquement à partir du code WhyML. C'est avec cette technologie qu'ont été prouvés une majorité des modules OCaml de la bibliothèque VOCaL [L3]. La preuve par raffinement dans l'outil Why3 a fait l'objet d'une publication spécifique [C12].

La thèse de doctorat de Mário Pereira a été effectuée dans l'équipe VALS sur la période du projet (sur un financement de la fondation des sciences du Portugal). Elle a abordé principalement la problématique de l'itération et de sa spécification [C13, N1] mais aussi proposé la défonctionnalisation comme une technique de preuve [N2].

La preuve de structures récursives mutables, telles que des structures utilisant des listes chaînées par exemple, ne peut se faire directement dans l'outil Why3 sans la conception d'un modèle mémoire. Ceci a été également abordé dans le projet, au travers de plusieurs études de cas [N3].

De nombreuses études de cas ont été développées dans l'équipe VALS tout au long du projet VOCaL [C8, J2, J3, C14]. Enfin, l'équipe VALS a contribué à la dissémination des résultats du projet VOCal au travers d'exposés invités dans des conférences internationales [I1, I2].

Gallium & Camus

Pottier utilise CFML pour vérifier les tables de hachage de la bibliothèque standard OCaml [C21]. Parce que ce composant logiciel joue un rôle central dans la bibliothèque standard et est utilisé par de nombreux projets logiciels, il semblait hautement souhaitable de le vérifier. Bien sûr, comme ce composant a déjà été éprouvé au cours du temps, cet effort de vérification n'a pas permis de découvrir de nouveaux défauts. Les aspects les plus délicats de ce travail sont liés à la spécification des itérateurs pour une structure de données modifiable : la spécification formelle doit indiquer que les itérateurs sont invalidés quand la structure de données est modifiée. Les idiomes de spécification développés pour les tables de hachage peuvent être ré-utilisés pour d'autres structures de données.

Charguéraud et Pottier proposent une extension de la Logique de Séparation qui permet un accès temporaire partagé en lecture seule à une structure de données, tout en restant très simple [C6]. Ils démontrent la sûreté de cette extension. Celle-ci n'a pas encore été implémentée dans l'outil CFML.

Un aspect important de notre travail est l'extension de la Logique de Séparation pour permettre le raisonnement à propos du temps. En particulier, un « crédit-temps » est une permission à dépenser une unité de temps, tandis qu'un « reçus-temps » est une preuve qu'une unité de temps s'est écoulée. Ces concepts permettent de raisonner à propos de la complexité en temps d'un programme. Ils permettent également d'exploiter certains arguments basés sur le temps pour démontrer la sûreté d'un programme.

Charguéraud et Pottier [C5, J1] présentent une preuve vérifiée par la machine de la correction et de la complexité en temps d'une structure de données Union-Find, implémentée en OCaml et présentée comme un type de données abstrait. Ceci démontre que cette technique peut être appliquée à des analyses de complexité difficiles et à un véritable code source (par opposition à un pseudo-code). Dans ce travail, les spécifications contiennent des constantes explicites : par exemple, la complexité de *find* est $2\alpha(n)+4$.

Dans des travaux plus récents, Guéneau, Charguéraud, et Pottier [C15] développent un ensemble de définitions et de tactiques Coq qui permettent d'établir des bornes de complexité asymptotiques : la complexité affichée de *find* est alors $O(\alpha(n))$. Guéneau, Charguéraud, Jourdan, et Pottier exploitent ces techniques pour mener à bien une étude de cas ambitieuse, à savoir la vérification de la correction et de la complexité amortie d'un algorithme incrémental de détection de cycles, qui représente l'état de l'art du domaine [C16].

Mével, Jourdan, et Pottier [C19] étendent Iris, une évolution moderne de la Logique de Séparation Concurrente, pour y introduire des moyens de raisonner à propos du temps. Iris est définie dans Coq, et sa sûreté est établie dans Coq. C'est également le cas de notre extension : nous nous appuyons sur la preuve de sûreté existante, sans devoir la modifier. Nous ajoutons à Iris non seulement les crédits-temps, mais aussi les reçus-temps, un concept nouveau. Les reçus-temps permettent de montrer que certains événements indésirables, comme les dépassements de capacité entiers, ne peuvent pas avoir lieu avant qu'un temps très long ne se soit écoulé. En guise d'application, nous mettons à jour notre analyse précédente d'Union-Find et démontrons que les rangs entiers employés de façon interne par cette structure de données ne peuvent pas déborder, même s'ils sont représentés à l'aide de seulement $\log W$ bits, où W est la taille d'un mot machine.

De Vilhena, Pottier et Jourdan [C10] vérifient un morceau de code court mais très subtil : un algorithme de calcul de plus petit point fixe incrémental, qui fonctionne à la demande et exploite la mémoisation. Il s'agit d'une version légèrement simplifiée de `Fix`, une bibliothèque OCaml publiée par Pottier en 2009. La spécification de cet algorithme est simple : le solveur calcule le plus petit point fixe optimal d'un système d'équations monotones. Bien que le solveur s'appuie sur un état interne modifiable à des fins de mémoisation et « d'espionnage », une forme de découverte dynamique des dépendances, aucun effet de bord n'est mentionné dans la spécification. Le défi est précisément de justifier formellement pourquoi on peut se permettre de ne pas divulguer l'existence de ces effets. La preuve est effectuée dans le cadre d'Iris et utilise de façon cruciale les variables de prophétie, une caractéristique très récemment ajoutée à Iris.

TrustInSoft

TrustInSoft a pour but de déployer et d'utiliser la bibliothèque prouvée de VOCaL dans son analyseur statique TrustInSoft Analyzer. Des tests de performance (en vitesse d'exécution et de mémoire) ont aussi effectués, avec l'utilisation de la bibliothèque VOCaL, sur des cas d'utilisation réels, complexes et industriels de l'analyseur.

L'évaluation de la bibliothèque VOCaL se concentre sur l'utilisation, la correction et les performances :

— La bibliothèque VOCaL s'est facilement intégrée au sein des 410k lignes de code de TrustInSoft Analyzer. La facilité d'utilisation de VOCaL n'est donc aucunement un frein pour son adoption auprès de la communauté OCaml.

— L'utilisation de VOCaL a permis de déceler des erreurs subtiles au sein du code de TrustInSoft Analyzer, renforçant la confiance à utiliser cette bibliothèque.

— Les performances avec VOCaL sont très similaires à celle des performances sans cette bibliothèque, y compris sur des exécutions de TrustInSoft Analyzer de plusieurs dizaines de minutes et une consommation de plusieurs Gigaoctets de mémoire vive.

Les détails de l'évaluation sont disponibles dans le livrable 4.2.

TrustInSoft a aussi participé à la rédaction d'un tutoriel pour étendre la bibliothèque VOCaL, et donc au développement et à la preuve d'une petite partie de la bibliothèque. Ce tutoriel explique comment le module Queue a été complété afin de pouvoir être complètement substitué au même module de la bibliothèque standard.

En plus de détailler comment étendre la bibliothèque, le tutoriel inclut un retour d'expérience sur la difficulté à entreprendre une telle opération. Ce retour d'expérience explique que le développement d'une extension de la bibliothèque reste accessible à des ingénieurs formés aux méthodes formelles, mais devient plus ardu pour ceux qui n'ont pas eu une telle formation. Néanmoins, la quantité d'exemples et de spécifications permet de réduire ce frein petit à petit.

L'utilisation du langage de spécification Gospel (pour OCaml) et la preuve associée pour la bibliothèque VOCaL sont des problématiques très similaires à celle que TrustInSoft rencontre à son quotidien avec son analyseur statique et le langage de spécification ACSL (pour le C et le C++). Ainsi l'évolution de ce langage de spécification et son adoption auprès de la communauté OCaml intéresse fortement TrustInSoft, qui suivra cette évolution de près à l'avenir.

Le tutoriel d'extension de la bibliothèque VOCaL est disponible dans le livrable 5.2.

OCamlPro

Résultats sur le prouveur Alt-Ergo OCamlPro maintient et développe le prouveur automatique de théorèmes SMT Alt-Ergo, développé initialement au LRI, et utilisé pour la vérification formelle dans l'atelier Why3, en particulier dans le cadre du projet ANR VOCaL. Les travaux d'OCamlPro sur Alt-Ergo dans le cadre du projet ont portés sur deux axes principaux :

— l'ajout à Alt-Ergo d'une nouvelle théorie afin de raisonner sur les types algébriques. Cette nouvelle théorie est nécessaire afin qu'Alt-Ergo puisse raisonner directement [1] sur des problèmes générés pour vérifier du code OCaml, puisque les types algébriques sont extrêmement répandus dans le code OCaml. La langage natif d'Alt-Ergo a aussi naturellement été étendu pour supporter les types algébriques.
— la fusion des représentations internes pour les termes et les formules. Jusqu'à récemment, Alt-Ergo faisait une distinction entre termes (tels que '1 + x') et formules (telles que 'forall x, 1 + x = y'), avec l'invariant qu'une formule ne peut pas apparaître comme sous-terme d'un terme. Cette distinction et cet invariant empêchaient Alt-Ergo de correctement supporter certaines expressions telles que des définitions de variables locales (de la forme 'let x = e in body') qui définissaient des formules dans des termes, ou bien des structures conditionnelles (if-then-else) qui mélangeaient aussi termes et formules. L'équipe Alt-Ergo a donc effacé cette distinction dans Alt-Ergo, autorisant des formules à apparaître dans des termes, ce qui a ensuite permis d'étendre Alt-Ergo pour mieux supporter les formes de définitions locales de variables et les branchements conditionnels.

Ces résultats sont disponibles dans la release open-source 2.2 d'Alt-Ergo (livrable D3.2).

Résultats sur l'optimiseur Flambda du compilateur OCaml OCamlPro développe l'optimiseur Flambda, intégré dans le compilateur OCaml, qui permet d'accroître les performances de la plupart des outils OCaml utilisés dans le projet VOCaL (Alt-Ergo encore, mais aussi Coq par exemple).

Après la publication initiale du framework d'optimisation Flambda avec la version 4.03.0 d'OCaml, l'équipe compilation d'OCamlPro s'est attaquée à dépasser certaines des limitations de la version initiale.

Deux axes principaux ont été explorés :

— Tout d'abord, la représentation sous forme ANF des termes manipulés avait déjà commencé à montrer ses limites. Nous avons donc fait des prototypes utilisant une représentation CPS. En intégrant également des techniques pour représenter efficacement les variables liées et l'alpha-renommage, cela nous a permis de baisser significativement le coût de l'inlining, ce qui s'avère particulièrement important avec des algorithmes d'inlining spéculatifs.
— Ensuite, pour des raisons techniques, la version publiée initialement de Flambda ne pouvait pas prendre en compte certains types d'optimisation lors de son estimation de bénéfice. En particulier les optimisations d'unboxing étaient faites par une passe de compilation après Flambda, ce qui conduisait parfois à des décisions d'inlining sous-optimales. Après quelques expériences pour estimer l'impact de ces optimisations en avance, nous nous sommes finalement décidés à intégrer complètement l'unboxing dans le framework Flambda.

Associés ensemble, ces changements ont fait qu'il n'était plus vraiment bénéfique d'améliorer incrémentalement la version existante de Flambda, donc nous avons lancé un nouveau projet indépendant appelé Flambda 2. Nous en avons profité pour retravailler aussi le système d'approximations, le transformant en un système de type plus complet, et en réutilisant des techniques de l'interprétation abstraite pour la formalisation et la propagation de ces types ou approximations.

Nous avons également choisi de consolider les multiples passes de Flambda 1, chacune responsable d'un nombre limité d'optimisations, en une seule passe. Cela nous donne un avantage en terme de temps de compilation (moins de passages à travers les termes, moins d'allocations de termes) et rend les interactions entre différentes optimisations plus efficaces, au prix d'un coût de maintenance plus élevé du code, plus complexe.

Verimag

Résultats sur FVDP Un *design pattern* appelé *Polymorphic LCF Style* a été conçu pour des oracles producteurs de certificats, et utilisé dans de grandes preuves Coq concernant : la compilation optimisante (ordonnancement d'instructions, pour une version de CompCert développée pour l'architecture multicoeurs Kalray), l'analyse statique (dans le domaine des polyèdres convexes), la déduction automatique (SAT solving, arithmétique linéaire rationnelle [T1, C2, C1, C22], où FVDP a allégé à la fois les temps de développement et les temps de calcul. Au passage, le compilateur optimisant mentionné ici contient un mécanisme pour effectuer du hash-consing certifié [C22], d'intérêt indépendant.

Résultats sur la méthode de Braga Cette partie a été développée en collaboration avec Dominique Larchey-Wendling (LORIA). Cette approche permet d'écrire en Coq des fonctions s'extrayant en OCaml exactement comme souhaité, et d'en étudier la correction partielle sans avoir à définir préalablement un certificat de terminaison. Plusieurs variantes sont disponibles (forme inductive du domaine de définition, graphe relationnel ou simulation d'un schéma inductif-récursif). Il a fallu également améliorer les techniques d'inversion disponibles en Coq. La méthode de Braga a montré son applicabilité à toute une gamme de programmes récursifs généraux, y compris ceux qui font appel à de la récursion emboîtée, dont un exemple non trivial est l'algorithme d'unification. Elle a été exposée succinctement à Braga pour TYPES'18 [C18] puis avec beaucoup plus de détails dans un chapitre d'ouvrage [B1]. Les échanges récents à l'intérieur du projet, notamment avec Arthur Charguéraud, ont établi une connexion qui devra être explorée avec ses travaux antérieurs sur le combinateur de point fixe optimal.

1. Sans cette théorie, il est nécessaire d'encoder et d'axiomatiser les types algébriques avant d'utiliser Alt-Ergo, et cet encodage a un effet néfaste sur les performances d'Alt-Ergo.

C.5. Exploitation des résultats

Le langage Gospel, et son implémentation [L1], est actuellement utilisé dans au moins deux projets.

— Au laboratoire NOVA-LINCS (Lisbonne), Mário Pereira développe `cameleer` [16], un outil de vérification déductive pour OCaml. En particulier, Mário Pereira a étendu le langage Gospel aux implémentations OCaml. Mário Pereira avait fait sa thèse dans le cadre du projet VOCaL (équipe VALS), avec un financement de la fondation des sciences du Portugal.

— Dans le cadre d'une thèse CIFRE entre la société Tarides et le Laboratoire Méthodes Formelles, Clément Pascutto développe `ortac` [14], un outil de vérification dynamique de code OCaml spécifié avec Gospel. Clément Pascutto est encadré par Jean-Christophe Filliâtre (LMF) et Thomas Gazagnaire (Tarides).

Les techniques FVDP sont exploitées dans le cadre d'une thèse CIFRE encadrée entre la société Kalray et Verimag, dans laquelle Cyril Six réalise une version de CompCert optimisante pour un processeur VLIW. Suite à la soutenance de cette thèse (13 juillet 2021), Cyril est recruté à Kalray, ce qui permettra à la coopération de continuer. Verimag encadre déjà un étudiant ingénieur qui est motivé pour poursuivre en thèse.

Par ailleurs, TrustInSoft continue de déployer et utiliser une version de son analyseur statique TrustInSoft Analyzer en production avec la bibliothèque VOCaL, afin d'améliorer la fiabilité de l'analyseur.

C.6. Discussion

Si les résultats du projet VOCaL sont très positifs, il reste à prouver que le langage Gospel saura s'imposer dans la communauté OCaml comme un langage de spécification universel. Plus modestement, il reste également à développer une traduction entre CFML et Gospel.

Les résultats sur la méthode de Braga et les techniques d'inversion ont été présentés récemment à l'équipe qui gère l'assistant à la preuve Coq. Dans un premier temps, une contribution dans un fichier de la bibliothèque standard Coq devrait être intégrée très prochainement. Il est ensuite prévu de discuter dès septembre de points plus profonds afin que l'inversion standard de Coq bénéficie de nos développements récents.

En ce qui concerne nos travaux dans le cadre de la Logique de Séparation, nos résultats démontrent que vérifier la complexité en temps d'un programme n'est pas nécessairement beaucoup plus difficile que de vérifier simplement sa correction fonctionnelle. Nous proposons une approche où ces tâches sont effectuées simultanément et peuvent interagir l'une avec l'autre de façon utile. Dans une direction, l'analyse de complexité peut s'appuyer sur un invariant établi par la preuve de correction ; dans la direction inverse, la preuve de correction peut s'appuyer sur le fait que certains événements indésirables, comme les débordements entiers, ne peuvent pas avoir lieu dans un temps raisonnable. Plusieurs études de cas montrent que cette approche est viable. Toutefois, parce que nous avons choisi d'utiliser une logique puissante pour effectuer des preuves et des analyses de complexité difficiles, par opposition à une logique plus restreinte dont l'utilisation aurait pu être plus automatisée, notre approche requiert une connaissance de la Logique de Séparation et une familiarité avec les assistants de preuve comme Coq. De plus, même pour un expert, le « raisonnement en grands-O » se révèle beaucoup plus difficile en Coq que sur papier, car il exige des jeux subtils sur les quantificateurs et les méta-variables. À l'avenir, il serait souhaitable de rendre cette activité plus accessible. Une autre piste intéressante consisterait à marier l'utilisation d'outils interactifs et automatiques pour l'analyse de complexité dans un cadre vérifié.

C.7. Conclusions

Les contributions attendues du projet VOCaL, à savoir un langage de spécification formelle pour OCaml, son implémentation et son application à une bibliothèque formellement vérifiée, ont été réalisées, ont été distribuées comme logiciel libre et ont fait l'objet de publications scientifiques et de présentations.

Au-delà de ces résultats, le projet VOCaL a été le contexte de très nombreuses contributions à la preuve de programmes : améliorations des outils Alt-Ergo, Why3 et CFML ; techniques de preuves Coq ; crédits et reçus temps ; études de cas variées.

Le langage Gospel, et son implémentation [L1], est actuellement utilisé dans au moins deux projets, à savoir `cameleer` [16], un outil de vérification déductive pour OCaml et `ortac` [14], un outil de vérification dynamique de code OCaml. Au-delà, on peut espérer que le langage Gospel soit repris par d'autres outils encore et s'installe *de facto* comme un standard dans la communauté OCaml.

Le projet VOCaL a permis d'établir un lien assez fort entre les participants, qui espèrent poursuivre leur collaboration dans le futur au travers d'autres projets.

C.8. Références

[1] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software : The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*. Springer, 2007.

[2] Arthur Charguéraud. Program verification through characteristic formulae. In *International Conference on Functional Programming (ICFP)*, pages 321–332, September 2010.

[3] Arthur Charguéraud. Characteristic formulae for the verification of imperative programs. In *International Conference on Functional Programming (ICFP)*, pages 418–430, September 2011.

[4] Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC : A practical system for verifying concurrent C. In *Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *Lecture Notes in Computer Science*. Springer, 2009.

[5] Jean-Christophe Filliâtre and Pierre Letouzey. Functors for Proofs and Programs. In *Proceedings of The European Symposium on Programming*, volume 2986 of *Lecture Notes in Computer Science*, pages 370–384, Barcelona, Spain, April 2004.

[6] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, March 2013.

[7] Maximilian P. L. Haslbeck and Peter Lammich. For a few dollars more - verified fine-grained algorithm analysis down to LLVM. In *European Symposium on Programming (ESOP)*, volume 12648 of *Lecture Notes in Computer Science*, pages 292–319. Springer, March 2021.

[8] Maximilian P. L. Haslbeck and Tobias Nipkow. Hoare logics for time bounds : A study in meta theory. volume 10805 of *Lecture Notes in Computer Science*, pages 155–171, April 2018.

[9] Bart Jacobs, Jan Smans, and Frank Piessens. A quick tour of the VeriFast program verifier. In Kazunori Ueda, editor, *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, volume 6461 of *Lecture Notes in Computer Science*, pages 304–311. Springer, 2010.

[10] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up : A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28 :e20, 2018.

[11] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning : An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[12] K. Rustan M. Leino. Dafny : An automatic program verifier for functional correctness. In *LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010.

[13] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[14] Clément Pascutto. Ortac, OCaml runtime assertion checking. Open Source Software, with MIT license. https://github.com/ocaml-gospel/ortac.

[15] Lawrence C. Paulson. Isabelle : the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.

[16] Mário Pereira. Cameleer, a deductive verification tool for OCaml programs. Open Source Software, with MIT license. https://github.com/ocaml-gospel/cameleer.

[17] Nadia Polikarpova, Julian Tschannen, and Carlo A. Furia. A fully verified container library. In *FM'15*, 2015. To appear.

[18] Christian Sternagel. Proof pearl—a mechanized proof of GHC's mergesort. *Journal of Automated Reasoning*, 51(4) :357–370, 2013.

[19] The Why3 team. The Why3 platform. Open Source Software, with LGPL license. http://why3.lri.fr/.

[20] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.4*, 2014. http://coq.inria.fr.

[21] Bohua Zhan and Maximilian P. L. Haslbeck. Verifying asymptotic time complexity of imperative programs in Isabelle. July 2018.

D. Liste des livrables

date de livraison	N°	Titre	Nature*	Partenaires	Commentaires
T0		réunion de démarrage		tous	
T0+3	D1.1	preuve de concept	logiciel	Gallium	
T0+6		site web		VALS	
T0+12	D2.1	preuve formelle	logiciel	Gallium	
T0+24	D2.2a	distribution des preuves	logiciel	Gallium, VALS	
T0+24	D3.1	nouvelle distribution de Why3 (1.3)	logiciel	VALS	
T0+24	D3.3	nouvelle distribution de CFML	logiciel	Gallium + Camus	
T0+30	D3.2	nouvelle distribution de Alt-Ergo (v. 2.2)	logiciel	OCamlPro	
T0+30		rapport mi-parcours	rapport	tous	
T0+36	D0.1	accord de consortium		tous	
T0+36		greffon Gospel pour Why3 [L2]	logiciel	VALS	
T0+36	D1.2a	distribution de la bibliothèque [L3]	logiciel	tous	
T0+48	D4.2	étude de cas [T3]	rapport	TrustInSoft	
T0+54		implémentation de Gospel [L1]	logiciel	VALS	
T0+60	D5.2	étude de cas [T5]	rapport	TrustInSoft	
T0+60	D3.1	nouvelle distribution de Why3 (1.4)	logiciel	VALS	
T0+60	D1.2b	distribution de la bibliothèque [L3]	logiciel	tous	
T0+60	D2.2b	distribution des preuves	logiciel	tous	
T0+60		rapport final	rapport	tous	

* rapport, logiciel, prototype, données, ...

E. Impact du projet

E.1. Indicateurs d'impact

Nombre de publications et de communications (à détailler en E.2)

		Publications multipartenaires	Publications monopartenaires
International	Revues à comité de lecture		[J1, J2, J3]
	Ouvrages ou chapitres d'ouvrage		[B1]
	Communications (conférences)	[M1, M2]	[C9, C13, C14, C8, C5, C3, C6, C21, C7, C15, C17, C19, C10, C16, C18, C12, C20, C4, C11, I1, I2, C23, C1, C2, C22]
France	Revues à comité de lecture		
	Ouvrages ou chapitres d'ouvrage		
	Communications (conférences)		[N1, N2, N3, N4]
Actions de diffusion	Articles vulgarisation		
	Conférences vulgarisation		
	Autres	[L1, L3]	[L2, T1, T2, T3, T4, T5]

Autres valorisations scientifiques (à détailler en E.3)

	Nombre, années et commentaires (valorisations avérées ou probables)
Brevets internationaux obtenus	
Brevets internationaux en cours d'obtention	
Brevets nationaux obtenus	
Brevets nationaux en cours d'obtention	
Licences d'exploitation (obtention / cession)	
Créations d'entreprises ou essaimage	
Nouveaux projets collaboratifs	
Colloques scientifiques	
Autres (préciser)	

E.2. Liste des publications et communications

Revues internationales

[J1] Arthur Charguéraud and François Pottier. Verifying the correctness and amortized complexity of a union-find implementation in separation logic with time credits. *Journal of Automated Reasoning*, September 2017.

[J2] Martin Clochard, Léon Gondelman, and Mário Pereira. The Matrix reproved. 60(3) :365–383, *Journal of Automated Reasoning*, 2018.

[J3] Jean-Christophe Filliâtre. Simpler proofs with decentralized invariants. volume 121, *Journal of Logical and Algebraic Methods in Programming*, January 2021. See http://why3.lri.fr/spdi/.

Chapitres d'ouvrages

[B1] Dominique Larchey-Wendling and Jean-François Monin. *The Braga Method : Extracting Certified Algorithms from Complex Recursive Schemes in Coq*, chapter 8, pages 305–386. In Klaus Mainzer, Peter Schuster, and Helmut Schwichtenberg, editors. *Proof and Computation II : From Proof Theory and Univalent Mathematics to Program Extraction and Verification*. World Scientific, September 2021.

Conférences internationales invitées

[I1] Jean-Christophe Filliâtre. Deductive verification of OCaml libraries. In *15th International Conference on integrated Formal Methods*, Bergen, Norway, December 2019. Invited talk.

[I2] Jean-Christophe Filliâtre. The Why3 tool for deductive verification and verified OCaml libraries. In *Frama-C & SPARK Day 2019*, Paris, France, June 2019. Invited talk.

Conférences internationales (multipartenaires)

[M1] Arthur Charguéraud, Jean-Christophe Filliâtre, Mário Pereira, and François Pottier. VOCaL – A Verified OCaml Library. ML Family Workshop, September 2017.

[M2] Arthur Charguéraud, Jean-Christophe Filliâtre, Cláudio Lourenço, and Mário Pereira. GOSPEL — providing OCaml with a formal specification language. Annabelle McIver and Maurice ter Beek, editors. *FM 2019 23rd International Symposium on Formal Methods*, October 2019.

Conférences internationales (monopartenaires)

[C1] Sylvain Boulmé and Alexandre Maréchal. Refinement to Certify Abstract Interpretations, Illustrated on Linearization for Polyhedra. *Journal of Automated Reasoning*, November 2018.

[C2] Sylvain Boulmé, Alexandre Maréchal, David Monniaux, Michaël Périn, and Hang Yu. The Verified Polyhedron Library : an overview. In Erika Ábrahám, Viorel Negru, Dana Petcu, Daniela Zaharie, Tetsuo Ida, Tudor Jebelean, and Stephen Watt, editors, *20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pages 9–17, Timişoara, Romania, September 2018. Universitatea de Vest din Timişoara, IEEE Computer Society.

[C3] Arthur Charguéraud. Higher-order representation predicates in separation logic. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, page 3–14, New York, NY, USA, 2016. Association for Computing Machinery.

[C4] Arthur Charguéraud. Separation logic for sequential programs (functional pearl). *Proc. ACM Program. Lang.*, 4(ICFP), August 2020.

[C5] Arthur Charguéraud and François Pottier. Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In *Proceedings of the 6th Conference on Interactive Theorem Proving (ITP 2015)*, volume 9236 of *Lecture Notes in Computer Science*, pages 137–153. Springer, August 2015.

[C6] Arthur Charguéraud and François Pottier. Temporary read-only permissions for separation logic. In Hongseok Yang, editor, *Proceedings of the European Symposium on Programming (ESOP 2017)*, Lecture Notes in Computer Science. Springer, April 2017.

[C7] Arthur Charguéraud and Mike Rainey. Efficient representations for large dynamic sequences in ML. In *ML Family Workshop*, 2017.

[C8] Martin Clochard, Léon Gondelman, and Mário Pereira. The matrix reproved. Blazy and Marsha Chechik, editors. *VSTTE 2016*, Lecture Notes in Computer Science, Toronto, Canada, July 2016. Springer.

[C9] Sylvain Conchon, Albin Coquereau, Mohamed Iguernlala, and Alain Mebsout. Alt-Ergo 2.2. In *SMT Workshop : International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, July 2018.

[C10] Paulo Emílio de Vilhena, François Pottier, and Jacques-Henri Jourdan. Spy game : Verifying a local generic solver in Iris. *Proceedings of the ACM on Programming Languages*, 4(POPL), January 2020.

[C11] Paulo Emílio de Vilhena and François Pottier. A separation logic for effect handlers. *Proceedings of the ACM on Programming Languages*, 5(POPL), January 2021.

[C12] Jean-Christophe Filliâtre and Andrei Paskevich. Abstraction and genericity in why3. In Tiziana Margaria and Bernhard Steffen, editors, *9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, volume 12476 of *Lecture Notes in Computer Science*, pages 122–142, Rhodes, Greece, October 2020. Springer. http://why3.lri.fr/isola-2020/.

[C13] Jean-Christophe Filliâtre and Mário Pereira. A modular way to reason about iteration. In *8th NASA Formal Methods Symposium* volume 9690 of *Lecture Notes in Computer Science*, Minneapolis, MN, USA, June 2016. Springer.

[C14] Jean-Christophe Filliâtre and Mário Pereira. Producing all ideals of a forest, formally (verification pearl). Blazy and Marsha Chechik, editors. *VSTTE 2016*, Lecture Notes in Computer Science, Toronto, Canada, July 2016. Springer.

[C15] Armaël Guéneau, Arthur Charguéraud, and François Pottier. A fistful of dollars : Formalizing asymptotic complexity claims via deductive program verification. In Amal Ahmed, editor, *Proceedings of the European Symposium on Programming (ESOP 2018)*, volume 10801 of *Lecture Notes in Computer Science*, pages 533–560. Springer, April 2018.

[C16] Armaël Guéneau, Jacques-Henri Jourdan, Arthur Charguéraud, and François Pottier. Formal Proof and Analysis of an Incremental Cycle Detection Algorithm. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18 :1–18 :20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[C17] Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. Mosel : A general, extensible modal framework for interactive proofs in separation logic. *Proc. ACM Program. Lang.*, 2(ICFP), July 2018.

[C18] Dominique Larchey-Wendling and Jean-François Monin. Simulating induction-recursion for partial algorithms. In Josè Espìrito Santo and Luìs Pinto, editors, *TYPES 2018 Abstracts*, Braga, June 2018.

[C19] Glen Mével, Jacques-Henri Jourdan, and François Pottier. Time credits and time receipts in Iris. In *European Symposium on Programming (ESOP)*, volume 11423 of *Lecture Notes in Computer Science*, pages 1–27. Springer, April 2019.

[C20] Glen Mével, Jacques-Henri Jourdan, and François Pottier. Cosmo : A concurrent separation logic for Multicore OCaml. *Proceedings of the ACM on Programming Languages*, 4(ICFP), June 2020.

[C21] François Pottier. Verifying a hash table and its iterators in higher-order separation logic. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*, pages 3–16, January 2017.

[C22] Cyril Six, Sylvain Boulmé, and David Monniaux. Certified and efficient instruction scheduling. Application to interlocked VLIW processors. *Proceedings of the ACM on Programming Languages, OOPSLA 2020*, page 129–158, November 2020.

[C23] Yuxin Deng and Jean-François Monin. Formalisation of probabilistic testing semantics in coq. In Mário S. Alvim, Kostas Chatzikokolakis, Carlos Olarte, and Frank Valencia, editors, *The Art of Modelling Computational Systems : A Journey from Logic and Concurrency to Security and Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*, volume 11760 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 2019.

Conférences nationales (monopartenaires)

[N1] Jean-Christophe Filliâtre and Mário Pereira. Itérer avec confiance. In *Vingt-septièmes Journées Francophones des Langages Applicatifs*, Saint-Malo, France, January 2016.

[N2] Mário Pereira. Défonctionnaliser pour prouver. In Boldo and Signoles *Journées Francophones des Langages Applicatifs*, Gourette, France, January 2017.

[N3] Jean-Christophe Filliâtre, Mário Pereira, and Simão Melo de Sousa. Vérification de programmes fortement impératifs avec Why3. In Sylvie Boldo and Nicolas Magaud, editors. *Vingt-neuvièmes Journées Francophones des Langages Applicatifs*, Banyuls-sur-mer, France, January 2018.

[N4] François Pottier. Strong automated testing of OCaml libraries. In *Journées Francophones des Langages Applicatifs (JFLA)*, February 2021.

Rapports techniques

[T1] Sylvain Boulmé and Alexandre Maréchal. Toward Certification for Free ! working paper or preprint, July 2017.

[T2] Jean-Christophe Filliâtre, Léon Gondelman, Andrei Paskevich, Mário Pereira, and Simão Melo de Sousa. A toolchain to Produce Correct-by-Construction OCaml Programs. Technical report. https://hal.inria.fr/hal-01783851.

[T3] TrustInSoft. Report on the deployment of the VOCaL library in TrustInSoft kernel, Sep 2020.

[T4] Jean-Christophe Filliâtre and Clément Pascutto. Ortac : Runtime assertion checking for OCaml. Technical report, Université Paris-Saclay, May 2021.

[T5] TrustInSoft. Extending VOCaL's Queue module. Technical report, March 2021

Logiciels

[L1] The VOCaL project. Gospel, a tool-agnostic formal specification language for OCaml. Open Source Software, with MIT license, 2018. https://github.com/ocaml-gospel/gospel.

[L2] The VOCaL project. Why3Gospel, a Why3 plugin to read Gospel specifications. Open Source Software, with MIT license, 2018. https://github.com/ocaml-gospel/why3gospel.

[L3] The VOCaL project. The VOCaL library. Open Source Software, with MIT license, 2018. https://github.com/ocaml-gospel/vocal.

E.3. Liste des éléments de valorisation

Logiciels créés dans le cadre du projet VOCaL.

- `gospel`, une implémentation du langage Gospel [L1].
- `why3gospel`, un greffon Why3 pour lire des spécifications Gospel [L2].
- `vocal`, une bibliothèque OCaml formellement vérifiée [L3].

Logiciels améliorés dans le cadre du projet VOCaL.

- Alt-Ergo (OCamlPro)
- CFML (Gallium + Camus)
- Why3 (VALS)

Création du Club Alt-Ergo par OCamlPro. L'ANR VOCaL a permis à OCamlPro de mieux connaître et interagir avec les utilisateurs d'Alt-Ergo. Cela s'est traduit par la création du Club Alt-Ergo, regroupant les utilisateurs académiques et industriels d'Alt-Argo, créé fin 2018, et dont la première réunion annuelle a eu lieu le 14 février 2019. Les membres du Club Alt-Ergo financent la maintenance et l'évolution d'Alt-Ergo au travers d'une souscription annuelle ou de contrats de R&D.