
# Journée de séminaire en ligne du groupe de travail LVP du GdR GPL du CNRS, vendredi 12 mars 2021

https://groupes.renater.fr/wiki/lvp/public/2103journee

## Programme

### Session 1 : Coq

09:00 - Pierre-Évariste Dagand (CNRS). Intermittent Computing with Peripherals, Formally Verified

09:30 - Diane Gallois-Wong (LMF). Formalisation en Coq des algorithmes de filtre numérique calculés en précision finie

**10:00 - Pause**

### Session 2 : Langages et spécifications

10:30 - Denis Merigoux (Inria Paris). Catala, un langage fonctionnel dédié pour formaliser le droit fiscal

11:00 - Jean-Christophe Filliâtre (CNRS). Gospel, un langage de spécification pour OCaml

11:30 - Virgile Robles (CEA LIST). Méthodologie de spécification de besoins haut niveau avec MetAcsl

**12:00 - Pause**

### Session 3 : Vérification de programmes

13:30 - Xavier Denis (LMF). Vérification déductive de Rust

14:00 - Raphaël Monat (Sorbonne Université, LIP6). Présentation de la plateforme Mopsa d'analyse statique de programmes par interprétation abstraite

14:30 - Khaoula Boukir (CEA LIST). Vérification des implémentations d'ordonnanceurs temps réel par model-checking

**15:00 - Pause**

## Exposé invité

15:30 - Louis Mandel (IBM Research). Compiling Stan to generative probabilistic languages and extension to deep probabilistic programming

## Session 4 : Compilation

16:15 - Nicolas Nalpon (ENAC). Vers la vérification de Smala un langage réactif interactif

16:45 - David Monniaux (CNRS, Verimag). Optimisation dans CompCert

**17:15 - Fin de la journée**

# Résumés des exposés

**Exposé invité : Compiling Stan to generative probabilistic languages and extension to deep probabilistic programming**, Louis Mandel (IBM Research)

Stan is a probabilistic programming language that is popular in the statistics community, with a high-level syntax for expressing probabilistic models. Stan differs by nature from generative probabilistic programming languages like Church, Anglican, or Pyro. In this talk, I will present a comprehensive compilation scheme to compile any Stan model to a generative language and proves its correctness. We use our compilation scheme to build two new backends for the Stanc3 compiler targeting Pyro and NumPyro. Building on Pyro we extend Stan with support for explicit variational inference guides and deep probabilistic models. That way, users familiar with Stan get access to new features without having to learn a fundamentally new language.

**Intermittent Computing with Peripherals, Formally Verified**, Pierre-Évariste Dagand (CNRS)

Transiently-powered systems featuring non-volatile memory as well as external peripherals enable the development of new low-power sensor applications. However, as programmers, we are ill-equipped to reason about systems where power failures are the norm rather than the exception. A first challenge consists in being able to capture all the volatile state of the application -- external peripherals included -- to ensure progress. A second, more fundamental, challenge consists in specifying how power failures may interact with peripheral operations. We will present a formal specification of intermittent computing with peripherals, an axiomatic model of interrupt-based checkpointing as well as its proof of correctness, machine-checked in the Coq proof assistant.

**Catala, un langage fonctionnel dédié pour formaliser le droit fiscal**, Denis Merigoux (Inria Paris)

Un certain nombre d'administrations et d'entreprises maintiennent des programmes informatiques afin de calculer des montants d'impôt, allocations, cotisations, pensions, etc. à partir de données de leurs usagers. Transformer les textes de loi qui régissent ces calculs en code est une tâche très difficile,

car elle requiert une double expertise juridique et informatique. De plus, la complexité inhérente des textes de loi et leur style de rédaction empêche activement la constitution d'une base de code saine et maintenable. En s'inspirant des travaux de Sarah Lawsky, professeure de droit fiscal et titulaire d'un doctorat en logique, le langage dédié Catala propose un nouveau processus de transformation de la loi en code, basé sur la programmation littéraire et en binôme. Le compilateur de Catala, basé sur une sémantique formelle du langage, permet également l'intégration du code dans virtuellement n'importe quelle architecture legacy grâce à la compilation. Le projet est soutenu par une équipe interdisciplinaire comprenant notamment Denis Merigoux, doctorant en informatique à l'Inria et Liane Huttner, doctorante en droit à Paris I.

**Vers la vérification de Smala un langage réactif interactif**, Nicolas Nalpon (Equipe Informatique Interactive-ENAC)

Smala est un langage réactif dédié à la programmation de systèmes à forte composante interactive. Pour pouvoir utiliser ce langage dans un contexte critique, il est nécessaire de pouvoir apporter des garanties. Ainsi, nous nous intéressons à la vérification formelle du compilateur de Smala, avec pour objectif de garantir la préservation de la sémantique du programme source à la compilation. Dans un premier temps, nous avons limité notre étude à un sous-ensemble de Smala. Dans cette présentation, nous allons présenter les motivations à représenter la sémantique du langage à l’aide des bigraphes.

**Gospel, un langage de spécification pour OCaml**, Jean-Christophe Filliâtre (CNRS)

Dans cet exposé, nous présenterons Gospel, un langage de spécification pour OCaml en cours de développement, et plusieurs projets bâtis sur ce langage (vérification déductive, vérification dynamique, bibliothèque algorithmique vérifiée).

**Présentation de la plateforme Mopsa d'analyse statique de programmes par interprétation abstraite**, Raphaël Monat (LIP6, Sorbonne Université & CNRS)

Mopsa est un analyseur statique dont la conception est modulaire et extensible. Il comporte une plate-forme indépendante des choix de langage et d’abstraction, dans laquelle peuvent être ajoutées et combinées des abstractions arbitraires (numériques, de pointeurs, de mémoire, etc.) et des itérateurs de syntaxe pour de nouveaux langages. Mopsa encourage le développement d’abstractions indépendantes et leur intégration dans une analyse grâce à l’utilisation de signatures uniformes de domaines et de mécanismes de communication génériques. Mopsa est un logiciel libre écrit en majorité en OCaml. Mopsa supporte actuellement des sous-ensembles significatifs des langages C99 (détection des comportements indéfinis et des utilisations invalides de la bibliothèque standard) et Python 3 (détection des exceptions non rattrapées, avec un domaine de type ou de valeurs).

**Vérification des implémentations d'ordonnanceurs temps réel par model-checking**, Khaoula Boukir (Université Paris-Saclay, CEA, LIST)

La mise en oeuvre d’une nouvelle politique d’ordonnancement au sein d’un système d’exploitation temps réel n’est pas une tâche facile. Le passage d'une spécification littéraire abstraite de la politique à son implémentation sur une

plateforme réelle exige la prise en compte de contraintes de diverses natures inhérentes à cette dernière. Par conséquent, une telle implémentation doit impérativement être accompagnée d'un travail de vérification permettant d'apporter un niveau de confiance en validant la conformité de son comportement par rapport à la spécification d'origine de la politique.

Dans cette optique, je présente une approche de vérification par model-checking visant à identifier les erreurs subtiles dans les implémentations de politiques d'ordonnancement temps réel. Cette approche est menée sur deux implémentations d'ordonnanceurs globaux : G-EDF et EDF-US, au sein d'un système d'exploitation temps réel conforme aux standards OSEK/VDX et AUTOSAR appelé Trampoline. L'approche a permis ainsi la vérification de la correction fonctionnelle du comportement des deux implémentations. Toutefois, son caractère modulaire et générique permet d'en envisager l'usage pour d'autres politiques et dans d'autres systèmes d'exploitation.

**Formalisation en Coq des algorithmes de filtre numérique calculés en précision finie**, Diane Gallois-Wong (LMF, Université Paris-Saclay)

Les filtres numériques sont utilisés dans de nombreux domaines, des télécommunications à l'aérospatiale. En pratique, ces filtres sont calculés sur machine en précision finie (virgule flottante ou souvent virgule fixe). Les erreurs d'arrondi résultantes peuvent être particulièrement problématiques dans les systèmes embarqués. En effet, de fortes contraintes énergétiques et spatiales peuvent amener à privilégier l'efficacité des calculs, souvent au détriment de leur précision. De plus, les algorithmes de filtres enchaînent de nombreux calculs, au cours desquels les erreurs d'arrondi se propagent et risquent de s'accumuler.

Comme certains domaines d'application sont critiques, j'analyse les erreurs d'arrondi dans les algorithmes de filtre en utilisant l'assistant de preuve Coq. Il s'agit d'un logiciel qui garantit formellement que cette analyse est correcte. Un premier objectif est d'obtenir des bornes certifiées sur la différence entre les valeurs produites par un filtre implémenté (calculé en précision finie) et par le filtre modèle initial (défini par des calculs théoriques exacts). Un second objectif est de garantir l'absence de comportement catastrophique comme un dépassement de capacité supérieur imprévu.

Je définis en Coq les filtres numériques linéaires invariants dans le temps (LTI), considérés dans le domaine temporel. Je formalise une forme universelle appelée la SIF, à laquelle on peut ramener n'importe quel algorithme de filtre LTI sans modifier leurs propriétés numériques. Je prouve ensuite deux théorèmes essentiels à l'analyse numérique d'un filtre sous forme de SIF. Cette analyse dépend aussi de l'algorithme de somme de produits utilisé dans l'implémentation. Je formalise donc plusieurs algorithmes de somme de produits offrant différents compromis entre précision du résultat et vitesse de calcul, dont un nouvel algorithme correctement arrondi au plus proche. Je définis également en Coq les dépassements de capacité supérieurs modulaires, afin de prouver la correction d'un de ces algorithmes même en présence de tels dépassements de capacité.

**Vérification déductive de Rust**, Xavier Denis (LMF)

Rust a démontré comment un système de types avancé peut réduire la complexité de la programmation système. Le "borrow checker" de Rust permet l'utilisation de pointeurs sans compromettre la sûreté de mémoire ou le

parallélisme. Le succès de Rust mène à un désir de l'utiliser pour des rôles critiques avec des obligations de vérification plus conséquentes.

Je vous présenterai Creusot, un outil expérimental pour la vérification déductive de code Rust en développement au LMF. On verra comment en s'appuyant sur la discipline de typage de Rust il est possible de simplifier le traitement des pointeurs mutables. Nous couvrirons aussi les spécifications surprenantes qu'on obtient en conséquence de ce traitement.

**Méthodologie de spécification de besoins haut niveau avec MetAcsl**, Virgile Robles (Université Paris-Saclay, CEA, LIST)

La spécification ainsi que la vérification formelle de propriétés de haut niveau (par exemple de sécurité, comme l'intégrité ou la confidentialité des données) sur des logiciels de taille importante est toujours un défi au sein de l'industrie. Nous avons introduit MetAcsl, un greffon de la plateforme de vérification Frama-C, qui permet à l'utilisateur de spécifier des propriétés haut-niveau sur des programmes C et de les transformer en assertions pouvant être vérifiées par vérification déductive. Cet exposé donne les grandes lignes d'une méthodologie de spécification d'un grand ensemble de besoins haut niveau avec MetAcsl et l'illustre sur plusieurs exemples. Le but est de donner un aperçu des propriétés revenant souvent et la manière de les spécifier avec MetAcsl en s'aidant de deux cas d'études. L'un d'entre eux (le bootloader de Wookey, un périphérique USB sécurisé développé par l'ANSSI) a été entièrement vérifié grâce à cette approche. L'autre (le microkernel d'un OS simple et imaginaire) sert à illustrer d'autres types de propriétés.

**Optimisations dans CompCert**, David Monniaux (CNRS, VERIMAG)

CompCert est un compilateur formellement prouvé correct : il y a une preuve, vérifiée à l'aide de l'assistant de preuve Coq, que chaque exécution correcte au niveau source C donne une exécution assembleur correspondante.

CompCert est modérément optimisant, et il y a beaucoup à faire. Je vous présenterai les travaux de Verimag sur CompCert : diverses optimisations (ordonnancement…) ainsi qu'une nouvelle cible (processeur Kalray KVX).

# Organisation

Organisé par Alain Giorgetti et Julien Signoles, avec le soutien du CEA, de l'Université de Franche-Comté et de l'institut FEMTO-ST.