# Sébastien Michelland

*Docteur en informatique*

sebastien.michelland@inria.fr • Nov. 1999
Site personnel : https://silent-tower.net


## Postes et études

**Centre Inria de l'Université de Rennes** — Post-doc Rennes, France • *Nov. 2025–en cours*

**Laboratoire de Conception et d'Intégration des Systèmes** — Ph.D Valence, France • *2022–Oct. 2025*
Thèse intitulée « Compilation au-delà de la sémantique pour la sécurité matérielle ».

**École Normale Supérieure de Lyon** — Master en Informatique Lyon, France • *2017–2022*
Obtenu ma license en 2018 et mon master en 2020. La maquette couvre l'informatique théorique en général ; je me suis concentré sur les langages, la sémantique et la compilation. Cette période couvre plusieurs stages :

- **University of Edinburgh** — Sémantique compositionnelle pour MLIR Edinburgh, R.U. • *2022*
  J'ai délimité une sémantique compositionnelle pour MLIR, un framework modulaire pour créer des représentations intermédiaires type LLVM. La sémantique est basée sur des interpréteurs monadiques et a été poussée jusqu'à l'étude de réécritures peephole.
- **Verimag** — Extensions de l'algorithme de clôture par congruence Grenoble, France • *2020*
  J'ai développé des extensions à l'algorithme de clôture par congruence et implémenté la procédure de décision qui en résulte en OCaml, destinée à servir à la tactique congruence en Coq [5, 4].
- **UQÀM** — Étude des interactions entre les passes de LLVM Montréal, Canada • *Été 2019*
  J'ai collecté une base de données des passes de LLVM et de leurs dépendances, et esquissé des outils logiciels pour étudier leurs interactions pour guider le problème d'ordonnancement de phase [6].
- **Inria** — Coq formalization of the dancing links algorithm Paris, France • *Été 2018*
  J'ai prouvé l'algorithme des liens dansants en Coq et partiellement prouvé une implémentation OCaml [7].

## Projets de recherche

Mon travail de recherche rentre dans la catégorie générale de la spécification et compilation de programmes, dont j'aime étudier le comportement au passage des frontières d'abstraction. J'ai du coup travaillé à la fois tout en bas de la pile logicielle, avec ma thèse qui touche à la sécurité contre des vulnérabilités matérielles ; et tout en haut, mon deuxième plus gros projet portant sur la certification d'interpréteurs abstraits spécifiés par des sémantiques monadiques.

### Thèse : Compilation au-delà de la sémantique pour la sécurité matérielle

Ma thèse aborde des problèmes méthodologiques liés aux contremesures contre les attaques par injection de faulte et par canaux auxiliaires. Ces attaques matérielles ont des effets logiciels complexes dont la caractérisation *(modélisation de fautes et de canaux auxiliaires)* est un art à part entière. Crucialement, ces effets ne peuvent pas être transportés dans la chaîne d'abstraction sans perdre en précision, ce qui signifie que

1. les contremesures doivent idéalement cibler les modèles précis, donc bas-niveau ;
2. les propriétés de sécurité métier des utilisateurs, qui proviennent du code source et des contremesures appliquées à différentes étapes de la compilation, doivent être connectées au code cible et donc préservées jusqu'à ce qu'il soit généré.

Ces points sont tous les deux subtils. Les modèles de fautes bas-niveau sont généralement analysés de façon empirique parce que les efforts de vérifiation formelle commencent plutôt aux langages haut-niveau, qui s'y prêtent mieux (et ces communautés s'intersectent peu). Il n'est du coup pas clair si on peut capturer

méthodiquement des détails architecturals dans l'étude d'un programme protégé. J'ai démontré dans un papier [2] qu'il est possible de modéliser sémantiquement une attaque de saut sur l'étape de fetch d'un processeur RISC-V. Ce travail a permis la conception d'une contremesure logicielle/matérielle fine dont la correction est prouvée d'abord et testée ensuite. Ce papier est significatif dans le sens où il connecte deux types de communautés ; d'un côté la compilation avec la sémantique et l'analyse formelle, de l'autre la sécurité matérielle avec les modèles de fautes précis tenant compte de la micro-architecture.

L'étape de compilation est tout aussi subtile ; pour générer du code sécurisé, le compilateur doit soit prendre du code sécurisé en entrée, soit insérer une contremesure ; puis préserver les propriétés qui rendent ce code « sécurisé » jusqu'au langage cible. Ces propriétés ne peuvent pas être exprimées uniformément le long de la chaîne d'abstraction, ce qui nécessite une *suite* de propriétés de sécurité à chaque niveau intermédiaire, quelque chose que je défends dans ma thèse mais qui n'est sinon quasiment jamais exploré.

L'outil central produit par ce projet est *Tracing LLVM*, un mod LLVM léger qui fournit des outils pour la production de code sécurisé [1]. Son principe est de *tracer* des éléments spécifiques du programme source (accès aux variables, calculs, flot de données...) durant leur compilation, tout en garantissant leur intégrité vis-à-vis des transformations du compilateur. Cette méthode définit un périmètre dans lequel une ingénieure sécurité peut manuellement contrôler la compilation, facilitant l'implémentation de contremesures.

**Sémantique abstraite pour des interpréteurs monadiques** (2021–2024).

J'ai conçu en Rocq un framework basique pour écrire et certifier des interpréteurs abstraits pour des langages minimaux, dans le cadre du projet Vellvm [8]. Vellvm définit une sémantique dénotationnelle de LLVM grâce à un modèle sémantique appelé *arbres d'interactions*, qui se trouve être exécutable. Ça ouvre la perspective de définir une *sémantique abstraite* dont l'exécution constituerait un interpréteur abstrait.

J'ai montré qu'il était possible de certifier un interpréteur abstrait ainsi défini par « interprétation monadique ». Cela permet d'unifier en partie les sémantiques concrète et abstraite d'un langage et de dériver une grande partie de la preuve de correction de l'interpréteur abstrait. L'implémentation open-source de ce framework contient nombre de composants réutilisables qui sont applicables à plus d'un langage [2].

Ce travail commencé par un stage a continué pendant environ deux ans avant d'être publié à ICFP'24 [1].

**Autres projets de recherche.**

Dans un papier de 2018 à COMPAS [3] découlant d'une expérience en cours d'architecture, j'ai testé le design d'une ISA que minimise la bande passante mémoire. J'ai écrit un émulateur/debugger interactif [3] pour le processeur fictif, qui a ensuite été utilisé plusieurs années dans un cours de compilation à l'UCBL.

## Outils

**Développement noyau embarqué sur calculatrices** (2015–ongoing).

Depuis 2015 j'ai développé et maintenu un uni-noyau et un SDK pour programmer nativement sur une douzaine de modèles de calculatrices graphiques, ce qui m'a apporté une grande expérience du développement bas-niveau. Ces machines ont déjà un OS, mais mon noyau peut prendre et rendre dynamiquement le contrôle du matériel via un changement de contexte similaire à un hyperviseur. Le noyau contient à date des pilotes de modules, un allocateur mémoire, des I/O asynchrones par interruptions, un driver USB 2.0, du debuggage à distance, et une libc C99 entre autres. Ce projet a requis pas mal de reverse-engineering de binaires bruts d'OS pour reconstruire des documentations fermées. Plus de détails en ligne [4].

1. https://gricad-gitlab.univ-grenoble-alpes.fr/tracing-llvm
2. https://gitlab.inria.fr/sebmiche/itree-ai
3. https://github.com/lephe/memory-light-isa
4. https://silent-tower.net/projects/gint

## Enseignement

J'ai enseigné ~180 heures à Grenoble INP–Esisar durant ma thèse et créé pas mal de ressources :

**Algorithmique et programmation C** dans deux cours différents (CS221–L2, IN330–L3).
- — Dont 3 cours magistraux avec mes ressources originales sur la gestion de la mémoire en C.
- — J'ai aussi monté un projet d'émulateur RISC-V complet, dont exercices de préparation, implémentation de référence, tests cachés, processus de notation, etc.

**Programmation fonctionnelle en Haskell** (CS222–L2).
- — Ici j'ai réécrit les 7 sujets de TP et construit un système de test semi-automatique robuste aux erreurs.

**Langages et compilation** (CS444–M1).

**Analyse de la sécurité logicielle et matérielle de systèmes** (OS430–M1).

## Certification/Compétences

**Français** : Langue maternelle
**Anglais** : Niveau C2 CEFR *CAE 203/210 (2018), TOEFL 116/120 (2021)*
**Programmation C/embarquée** : Avancé *Noyau/userspace, LLVM en détail, Édition des liens, RE basique*
**Programmation fonctionnelle et théorie des types** : À l'aise *Coq, Lean 4, Haskell*
Et de l'expérience en : C++, administration Linux, LaTeX...

## Bibliographie

### Articles de journaux


[1] Sébastien **Michelland**, Yannick Zakowski et Laure Gonnord. "Abstract Interpreters : A Monadic Approach to Modular Verification". In : *Proceedings of the ACM on Programming Languages* 8.ICFP (août 2024), p. 1-28. URL : https://hal.science/hal-04628727.


### Articles de conférences


[2] Sébastien **Michelland**, Christophe Deleuze et Laure Gonnord. "From low-level fault modeling (of a pipeline attack) to a proven hardening scheme". In : *Compiler Construction*. Edinburgh (Scotland), United Kingdom, mars 2024. URL : https://hal.science/hal-04438994.

[3] Florent de Dinechin, Maxime Darrin, Antonin Dudermel, Sébastien **Michelland** et Alban Reynaud. "Une architecture minimisant les échanges entre processeur et mémoire". In : *ComPAS 2018 - Conférence d'informatique en Parallélisme, Architecture et Système*. Toulouse, France, juill. 2018, p. 1-8. URL : https://inria.hal.science/hal-01959855.


### Workshops, rapports, etc


[4] Sébastien **Michelland**. *Rapport de stage : Une procédure de décision pour relations d'équivalence*. https://silent-tower.net/static/internship_congruence_closure.pdf. Juin 2020.

[5] Sébastien **Michelland**, Pierre Corbineau, Lionel Rieg et Karine Altisen. "A Decision Procedure for Equivalence Relations". In : *Coq Workshop 2020*. Aubervilliers, France, juill. 2020. URL : https://hal.science/hal-04880486.

[6] Sébastien **Michelland**. *Rapport de stage : Exploration et cartographie des passes de LLVM*. https://silent-tower.net/static/internship_llvm_passes.pdf. Juill. 2019.

[7] Sébastien **Michelland**. *Rapport de stage : Permutations et liens dansants vérifiés en CFML*. https://silent-tower.net/static/internship_dancing_links.pdf. Juill. 2018.

Autres publications citées pour le contexte

[8] Yannick Zakowski, Calvin Beck, Irene Yoon, Ilia Zaichuk, Vadim Zaliva et Steve Zdancewic. “Modular, Compositional, and Executable Formal Semantics for LLVM IR”. In : *Proc. ACM Program. Lang.* 5.ICFP (août 2021). URL : https://doi.org/10.1145/3473572.

*Compilé le 6 novembre 2025*