

Enseignement de la Programmation Fonctionnelle en M2

*À la recherche d'un bon équilibre
entre théorie et pratique*

Didier Rémy



JFLA 2025, Saint Hilaire, Roiffé

Introduction

Un exposé atypique

- Non scientifique.

L'enseignement de la programmation fonctionnelle en M2

- Un cours sur notre cours. . .

Programmation fonctionnelle et systèmes de types

(alias 2.4 alias **FUN**)

- . . . un cours lui-même atypique à bien des égards
au sein du **MPRI**, lui-même atypique
à plusieurs voix “orchestré” avec mes collègues
- Un périmètre pas très bien défini
qui a évolué au cours du temps,
en équilibre entre théorie et pratique

Un exposé avec des inconnues non résolues. . .

- Quels sont les points qui pourraient vous intéresser ?
- Que puis-je en dire de pertinent pour vous ?
- Beaucoup d'inconnues sur notre cours même. . .

Introduction

Approche retenue

- Vous raconter l'histoire du cours FUN,
- en insistant sur les aspects importants (ou les plus FUN)
- et tirer quelques leçons de FUN,
- *pour vous* et. . .

Avec vous !

- Vous avez sans doute autant de leçons à partager que moi, comme **anciens étudiants, enseignants, ou chercheurs** dans ce domaine.
- ⇒ N'hésitez pas à m'interrompre, poser des questions, ou faire des propositions, pendant, *ou après*, mon exposé.

Je vais vous parler

- un peu du MPRI, mais surtout
- du cours FUN en général, et plus en détail
- de la partie “*Systèmes de types*”, que j’y ai enseignée

En insistant sur

- Le contenu
 - Sujets abordés et leur évolution
 - Équilibres entre théorie et pratique
- Certains aspects particuliers de notre cours,
 - Audience hétérogène
 - Projet de programmation
 - Expériences pédagogiques (et de leur absence)
- L'intérêt de ce cours (et de le maintenir)
- Ce que nous avons bien fait et/ou pourrions mieux faire

Le cours FUN est au coeur de nos thématiques (des JFLA)

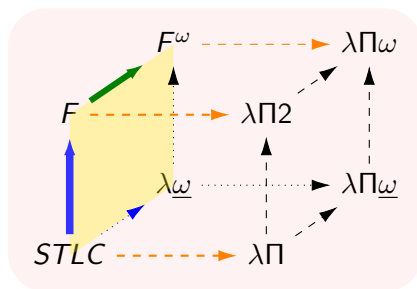
Par plusieurs aspects

- Programmation fonctionnelle (Ses patrons de conception)
- Métathéorie des langages de programmation (Formalisation, Propriétés, Preuves)
- Questions d'implémentation (Typage, Compilation, Optimisation, etc.)

Évidemment, ce n'est pas le seul

- Cours proches au sein du MPRI
 - Fondements des systèmes de preuves
 - Assistants de preuves
 - Preuves de programmes
 - Interprétation abstraite
 - *Modèles des langages de programmation*
- ou ailleurs en IDF :
 - ... Cours de compilation à l'ENS-Paris (Jean-Christophe Filiâtre)
- ... et pleins de cours similaires en province (?)

FUN dans le λ -cube



Au siècle dernier, dans passé très lointain pour beaucoup d'entre vous, il était une fois, en Île de France ...

Deux DEA d'informatique, en IDF :

- DEA Programmation Sémantique Preuves et Langage (porté par R. Dicosmo)
- DEA Algorithmique (porté par[?] P. Gastin)

Qui voulaient unir leurs forces :

- pour être plus visibles...
- offrir un cursus **solide** et **à la carte** aux meilleurs élèves

Dans le DEA d'informatique fondamentale

- *Typage et programmation* l'ancêtre du cours FUN au MPRI
enseigné par **Didier Rémy** (–1997), puis **Xavier Leroy** (1998–)

Table de conversion

Après	L1	L2	L3	M1	M2	Doctorat
Avant	DEUG		Licence	Maîtrise	DEA	Doctorat

MPRI = Master Parisien de la Recherche en Informatique

- Un titre un peu prétentieux...

Un déguisement de qqs étab. pour rentrer dans le costume du LMD

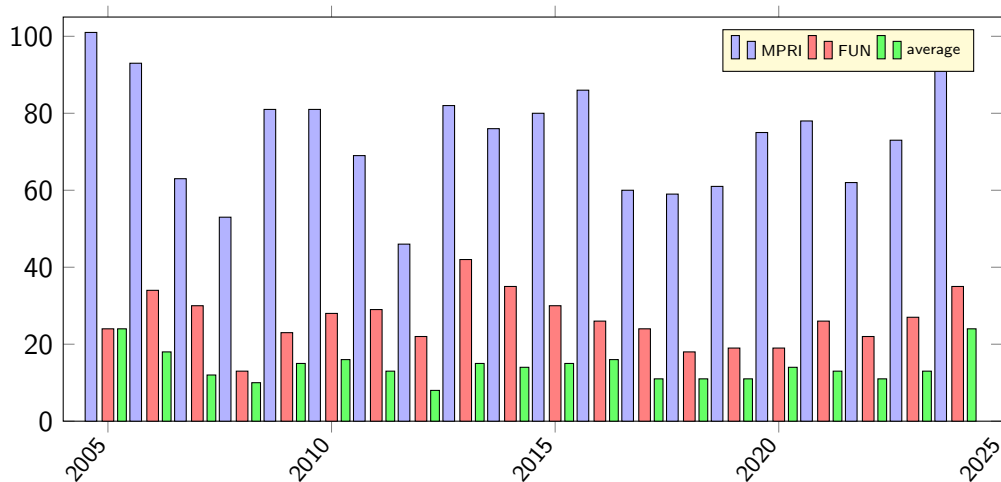
P7	P6	P11	ENS-Paris	ENS-Cachan	X	TP	INRIA
UPC	SU	UPS	PSL	UPS	IPP	IPP	<i>Inria</i>

- Chacun conservant sa maîtrise, renommée en “M1”
- Mais en partageant leurs cours de M2.

Conséquence

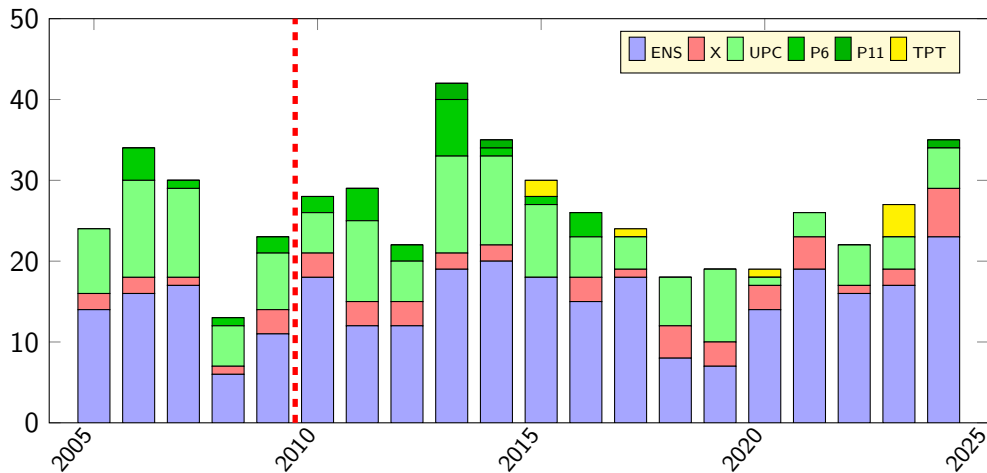
- Des profils hétérogènes à l'arrivée en M2.

(doublement hétérogène)



≈

73 étudiants	26 dans le cours FUN	15 par cours en moyenne
13% de filles	8% de filles	



530 élèves sur 20 ans
26 élèves par an

=

300	50	170
ENS*	X	*

($\approx 8\%$ de filles)

1. Établissement de rattachement, parfois administratif, notamment pour UPC ou ENS-Cachan.

Population variée

- 2/3 sont des étudiants de l'ENS ou de l'X
- 1/3 sont universitaires (dont des étrangers)



Master parfois considéré comme élitiste

(à tord, bien sûr)

- Élèves de niveau varié : une difficulté pour les étudiants (pour les profs aussi)
- Taupins, hyper sélectionnés : profil théorique, matheux, rapide, ...
- Universitaires : profils plus variés, plus informatique, plus pratique

Une ressource rare, à partager

- 1 enseignant $\approx$ 12 heures de cours = $8 * 1h30$ | $4 * 3h$ | $5 * 2h30$
- induits des cours longs et denses
- un master réputé difficile (et notre cours n'y échappe pas)

Mais ce n'est pas un cours de programmation. . .

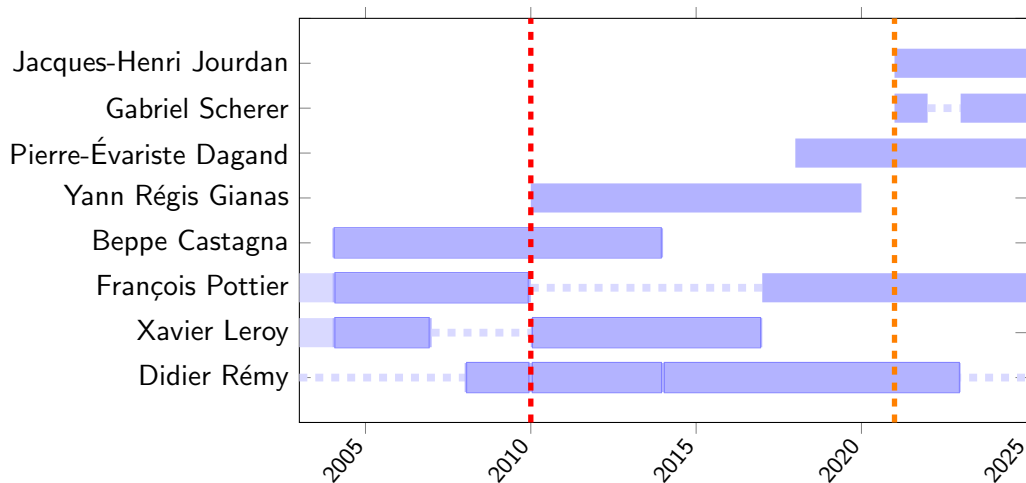
- Métathéorie
 - typage et sémantique (opérationnelle)
 - sûreté, inférence de type, transformations de programmes
 - principes de conception des langages de programmation modernes
- Concepts plus spécifiques à la programmation fonctionnelle (typée ou pas, *compilation*, évaluation paresseuse, effets, monads)
- Comporte un projet de programmation (en OCaml)
 - permet d'approfondir un des aspects vus en cours.
 - d'incorporer un aspect de programmation
 - y compris dans la notation (30%)

Un cours qui a évolué au cours du temps

- Passage de 3 à 6 ECT
- Pour s'adapter
 - à l'équipe enseignante : départs ou arrivée de jeunes chercheurs (anciens élèves :-)
 - aux cours de M1 qui évoluent eux aussi.
 - à l'évolution de la discipline (sujets nouveaux ou démodés)

Évolution de l'équipe enseignante

Un cours partagé $\approx$ 12 heures par personne par an



Année scolaire $n = [\text{Sep } n, \text{Sep } (n+1)]$

Programme du demi-cours (24 ECT = 16 * 1h30) – 2009

Systèmes de types (François Pottier)

8 * 1h30

Sous-typage (Giuseppe Castagna)

4 * 1h30

Modularité (Didier Rémy)

4 * 1h30

Systèmes de types (François Pottier)

8 * 1h30

- Types simples: sûreté du typage; paires, sommes, récursivité
- Types simples, suite: références; inférence de types
- Polymorphisme: le système F; le système de Damas et Milner
- Polymorphisme, suite: sûreté du typage pour ML, puis pour ML avec références
- Inférence de types à base de contraintes pour ML
- Extensions diverses
- Types existentiels
- Closure conversion préservant le typage

Sous-typage (Giuseppe Castagna)

4 * 1h30

Modularité (Didier Rémy)

4 * 1h30

Systèmes de types (François Pottier)

8 * 1h30

Sous-typage (Giuseppe Castagna)

4 * 1h30

- Simple subtyping
- System F subtyping
- Subtyping recursive types
- Semantic Subtyping

(XML et Cduce)

Modularité (Didier Rémy)

4 * 1h30

- Simple modules
- Advanced notions, recursive and mixin modules
- Open existential types
- Overloading

Systèmes de types (François Pottier)

8 * 1h30

Sous-typage (Giuseppe Castagna)

4 * 1h30

Modularité (Didier Rémy)

4 * 1h30

Constat :

- + Un cours qui fonctionnait plutôt bien
 - avec un programme dense
 - très centré autour du typage
- ⇒ que nous voulions élargir
 - à d'autres thématiques
 - avec de nouveaux intervenants

Un programme ambitieux...

(57h–72h)

- ① Les bases du cours (12h-18h)
- ② De la programmation vers la preuve de programme (9h-12h)
- ③ Programmation fonctionnelle avancée et transformations de programmes (12h)
- ④ Programmation à grande échelle (9h-12h)
- ⑤ Sous-typage et données semi-structurées (–2017) (6h-9h)
- ⑥ Typage avancé des effets, Capacités, Types linéaires (9h)

Chaque année = Les bases du cours + 3 modules au choix

Un programme ambitieux...

(57h–72h)

① Les bases du cours

(12h-18h)

- *Sémantiques opérationnelles*
- *Types simples, Polymorphisme, Système F.*
- *Théorèmes de sûreté: auto-réduction et progrès.*
- *Types de données (sommes et produits)*
- *Types rékursifs*
- *Types existentiels*
- *Inférence de types*
- *(Types algébriques généralisés (GADT))*
- *(Surcharge, classes de types)*
- *(Sous-typage syntaxique)*

(2h)

(6h)

(1h30)

- ② De la programmation **vers** la preuve de programme
- ③ Programmation fonctionnelle avancée et transformations de programmes
- ④ Programmation à grande échelle
- ⑤ Sous-typage et données semi-structurées (–2017)
- ⑥ Typage avancé des effets, Capacités, Types linéaires

Un programme ambitieux...

(57h–72h)

- ① Les bases du cours
- ② De la programmation **vers** la preuve de programme (9h-12h)
 - *Types algébriques généralisés (GADT)* (2h)
 - *"Type families"* (2h)
 - *(Types dépendants ?)*
 - *Assertions logiques et vérification déductive* (4h30)
 - *Contrats ?* (1h30)
- ③ Programmation fonctionnelle avancée et transformations de programmes
- ④ Programmation à grande échelle
- ⑤ Sous-typage et données semi-structurées (–2017)
- ⑥ Typage avancé des effets, Capacités, Types linéaires

Un programme ambitieux...

(57h–72h)

- ① Les bases du cours
- ② De la programmation **vers** la preuve de programme
- ③ Programmation fonctionnelle avancée et transformations de programmes (12h)
 - *Continuations et transformation CPS, formes A-normales*
 - *Monades et encapsulation des effets (avec ou sans typage)*
 - *Closure conversion et Défonctionnalisation (non-typées et typées)*
 - *Machines abstraites non-typées*
 - *Machines abstraites typées (ouverture vers l'assembleur typé)*
- ④ Programmation à grande échelle
- ⑤ Sous-typage et données semi-structurées (–2017)
- ⑥ Typage avancé des effets, Capacités, Types linéaires

Un programme ambitieux...

(57h–72h)

- ① Les bases du cours
- ② De la programmation **vers** la preuve de programme
- ③ Programmation fonctionnelle avancée et transformations de programmes
- ④ Programmation à grande échelle (9h-12h)
 - *Surcharge, classes de types* (6h)
 - *(Programmation générique)* (3h)
 - *(Modularité)* (3h-6h)
- ⑤ Sous-typage et données semi-structurées (–2017)
- ⑥ Typage avancé des effets, Capacités, Types linéaires

Un programme ambitieux...

(57h–**72h**)

- ① Les bases du cours
- ② De la programmation **vers** la preuve de programme
- ③ Programmation fonctionnelle avancée et transformations de programmes
- ④ Programmation à grande échelle

⑤ Sous-typage et données semi-structurées (–2017) (6h–9h)

- *Sous-typage syntaxique* (1h30)
- *Sous-typage sémantique* (3h ?)
- *Typage des données semi-structurées (XML) et de leurs transformations.* (3h)

⑥ Typage avancé des effets, Capacités, Types linéaires

Un programme ambitieux...

(57h–**72h**)

- 1 Les bases du cours
- 2 De la programmation **vers** la preuve de programme
- 3 Programmation fonctionnelle avancée et transformations de programmes
- 4 Programmation à grande échelle
- 5 Sous-typage et données semi-structurées (–2017)
- 6 Typage avancé des effets, Capacités, Types linéaires (9h)
($\rightsquigarrow$ Rust : 2021–)

Operational Semantics, Type Systems, and Program Transformations

Semantic Proofs of Type Soundness and Logical Relations

Typed Programming

Programming with Resources in Rust

Operational Semantics, Type Systems, and Program Transformations

- Syntax, semantics, and interpreters
 - From operational semantics to interpreters
 - Machine-checked proofs and de Bruijn syntax in Crocq
- System F and a syntactic proof of its type soundness
- Algebraic data types and existential types
- GADTs
- Closure conversion and defunctionalization (FP).
- System F with mutable state; the value restriction; type soundness (GS)

Semantic Proofs of Type Soundness and Logical Relations

Typed Programming

Programming with Resources in Rust

Operational Semantics, Type Systems, and Program Transformations

Semantic Proofs of Type Soundness and Logical Relations

- Unary logical relations for STLC, normalisation
- Unary logical relations for F, normalization; binary logical relations, parametricity
- CPS transformation.
- Type soundness in Coq for STLC, definition, SR and PG
- Semantic type soundness for F with mutable state in Coq/Iris

Typed Programming

Programming with Resources in Rust

Operational Semantics, Type Systems, and Program Transformations

Semantic Proofs of Type Soundness and Logical Relations

Typed Programming

- Ad-hoc polymorphism and overloading
- Applicative functors and monads
- System F^ω and modules
- Hindley-Milner type inference and elaboration
- Bidirectional type inference and elaboration

Programming with Resources in Rust

Operational Semantics, Type Systems, and Program Transformations

Semantic Proofs of Type Soundness and Logical Relations

Typed Programming

Programming with Resources in Rust

- Introduction to programming in Rust
- Rust: Generics and traits (ex. Binary search trees)
- Practicing Rust.
- Metatheory of Rust's type system

Passage de 24 à 48 ECT

- Facile en doublant le volume ?
- N'a pas diminué la densité des cours
- Plus de sujets enseignés

Révision à volume constant

- Prise en compte de nouveaux sujets
- En préservant un tronc commun
 - Quel est ce tronc commun ?
 - Ex. inférence de types ? sous-typage ?
- En profitant de plus d'acquis en M1
- En diminuant le niveau de détail

Il faudrait un autre cours. . .

- Si l'on voulait tout couvrir avec un bon niveau de détail
- Difficilement envisageable. . .

Le projet de programmation

Un aspect important de notre cours

- Donne la coloration programmation à ce cours
 - Participe probablement au succès de ce cours
 - Les premiers (et un temps les seuls) à l'offrir
- ⇒ attractivité de ce cours

Un calibrage difficile

- Demande beaucoup de temps élève, à partager entre les cours (on n'est plus seuls)
- Un sujet intéressant **est** souvent un peu long
- Un sujet différent tous les ans

Notre approche

- Un projet à trous avec beaucoup de code donné
- Des tests donnés, à compléter par des tests à écrire
- Un minimum à faire et des questions bonus ⇒ souvent quelques super projets!
- Permet de remonter les notes : compte pour 30% ⇒ rassurant, presque toujours

Beaucoup de travail aussi pour les enseignants. . . (plutôt agréable)

Le projet de programmation

Des vrais sujets, toujours intéressants

- Mini-Haskell : élaboration des classes de types (2013)
- Compilation du λ -calcul non typé vers C (2017)
- Typage d'un langage avec "effect handlers" (2020)
- Typeur pour le Système F (2021)
- Différentiation automatique (2022)
- Minirust ("borrow checker") (2024)

Slides

- Conduit à des cours denses, plus de contenu, plus rapide

Retour au tableau, à l'ancienne

(les “plus jeunes” y reviennent)

- ± Plus lent, pas aussi poli que des slides
- + Plus facile à mémoriser pour les étudiants?
- + Dynamique; montre les hésitations, comment se rattraper,
- Peut augmenter l'impression de difficulté
- Lisibilité (écriture, tenue du tableau)

Expériences pédagogiques

- Peu nombreuses (en particulier de ma part)
- + Les enseignants plus jeunes sont plus inventifs. . .
 - Cours inversé
 - Séances avec programmation (en Agda)
 - Études de papier
 - Cours sur/avec portable
 - Séances d'exercices (TD)
- + Retour au tableau pour certains cours

— Yann-Régis Gianas
— Pierre-Évariste Dagand
— Gabriel Scherer

Comment présenter les preuves ?

- Détails en cours avec slides
- ~ Détails en cours au tableau
- + Survol en cours avec des notes de cours détaillées
 - Donner les énoncés, lemmes, et principes
 - Sauter les détails
- ⇒ Nécessité d'avoir les preuves détaillées quelque part
 - exemple de ce qui est attendu (examen ou papier de recherche)
 - savoir justifier des moindre détails, même si on les omets ensuite

Une habitude qui se perd ?

- Beaucoup plus de travail
 - + Appréciables des élèves
 - + Plus auto-contenus et plus pérennes que des slides
 - + Permettent d'alléger (ne dire que le principal à) l'oral
 - Ou de densifier le cours—en renvoyant vers l'écrit.
 - + Peuvent servir de référence / modèle (ex. dans la rédaction des preuves)
- ⇒ Pas aussi utilisées qu'elles pourraient l'être ?
- En faire plus de publicité ?
 - Les polir et les publier ? (encore plus de travail)

Typage et sémantique

(2022)

- Untyped lambda-calculus (rappels, prérequis)
- Simply-typed lambda-calculus
- Polymorphism and System F (ML comme un cas particulier)
- Data-types, existential types, GADTs
- Effects: References, exceptions, and value restriction
- Logical relations and parametricity
- Fomega: higher-kinds and higher-order types
- Type reconstruction (formalisation de l'inférence de types avec contraintes)
- Overloading (formalisation des “type classes” de Haskell)

(*) Réutilisation et extensions des notes de François Pottier.

Un aperçu de mécanisation

- Sémantiques opérationnelles, LCST, since 2017 (Xavier Leroy & François Pottier)
- Un peu d'Agda (Pierre-Évariste Dagand)
- Un peu d'Iris (introduit cette année) (Jacques-Henri Jourdan)

Plus de mécanisation ?

À la **Software foundation** ?

- Pas les prérequis
- Synchronisation difficile avec les cours sur les assistants de preuve
- Pas assez de séances
- Beaucoup de travail pour les enseignants. . .

Choix raisonnable

- Quelques incursions dans la preuve mécanisée (parfois cours en double)

Observations

Un cours qui plaît :

- Avec une audience significative
- Et des retours plutôt favorables

Grâce à (hypothèses)

- Une évolution des thématiques pour le garder attractif
- Un mélange entre théorie et pratique
 - formalisation et implémentation
 - projet de programmation

Parfois choisi comme un cours d'ouverture ?

- plutôt pour apprendre sur les langages de programmation
- que pour faire de la recherche dans ce domaine

Suggestions ou recommandations (en guise de conclusion)

Continuer à enseigner la programmation fonctionnelle en M2 !

- Elle n'est pas toujours/suffisamment enseignée auparavant...
- Surtout une façon d'enseigner la métathéorie des langages de programmation
- Cela donne une bonne base à nos étudiants —et à ceux qui ne le seront pas !
- Cela nous permet de mieux les recruter

Continuer le cours **FUN** au sein du MPRI

- = Maintenir les équilibres actuels et continuer à adapter notre cours
- ⇒ Couper en deux séances hebdomadaires de 1h30 ?
- ⇒ Cours toujours très dense : tenter l'ajout d'un autre demi-cours ? (sujets avancés)
- ≈ Rapprochement avec preuves de programmes programmer $\Leftarrow$ prouver

Au delà du MPRI, mieux partager nos matériaux pédagogiques ?

- + Avec d'autres cours ? Partager, diffuser des notes de cours ?
- ⇒ En faire des notes de cours de référence ?

Le mot de la fin...

Remerciements

- Enseigner (la programmation fonctionnelle en M2) reste un plaisir
(même si cela nous prend du temps)
- Merci à tous ceux qui préparent si bien les étudiants en amont
(licence et M1)

Mes vœux

- + Langages fonctionnels, plus tôt, dans l'enseignement de l'informatique
- + Enseignement d'informatique, plus tôt, en Lycée et Collège
- ⇒ Attirer plus de filles dans nos domaines !
La programmation fonctionnelle peut-elle y aider ?
... en rendant l'informatique moins "geek" ?

<https://cambium.inria.fr/~remy/mpri/jfla-2025.pdf>

Vive la programmation fonctionnelle !