Logique et Fondements de l'Informatique
Bibliographie et conseils de lecture

David A. Madore

7 novembre 2025

CSC-3TC34-TP / INF110

Git: `0a7ceb7 Fri Nov 7 13:46:34 2025 +0100`

Ce document rassemble quelques conseils de lecture pour *prolonger* ou bien *compléter* le cours de « Logique et Fondements de l'Informatique », ou simplement en rapport avec des thématiques évoquées dans ce cours.

— Commençons par un livre de vulgarisation, qui est aussi un grand classique (et une référence de toute une génération de geeks): *Gödel, Escher, Bach* de Douglas Hofstadter (1979).

  C'est un livre très difficile à classer, ou même à décrire, puisqu'il intercale des chapitres sérieux avec des dialogues humoristiques, et qu'il parle à la fois de sciences et d'art (ça va des maths à la biologie en passant par la physique, la linguistique, la philosophie zen, la linguistique... et bien sûr les dessins d'Escher et la musique de Bach, ainsi que des réflexions sur l'intelligence artificielle qui restent étonnamment pertinentes pas loin de 50 ans après). Le thème général est celui de la récursion et de l'auto-référence (et en particulier, de variations sur le théorème de Gödel et l'astuce de Quine). On trouve aussi dedans une formalisation précise de l'arithmétique du premier ordre (mais présentée à un niveau grand public) ainsi qu'une description d'un langage de programmation (« BlooP ») qui calcule précisément les fonctions primitives récursives.

  En tout cas, je recommande très vivement ce livre (à la lecture duquel je dois beaucoup d'être moi-même devenu mathématicien) à tous ceux qui trouvent intéressants les thèmes abordés par ce cours.

  L'original est en anglais, mais il y a une excellente traduction française, à laquelle l'auteur lui-même a contribué.

— Sur la **calculabilité**, il existe de nombreux ouvrages, parmi lesquels on peut par exemple recommander :
  — *Computability Theory* de Barry Cooper (2003) : assez court et pas trop indigeste
  — *Calculabilité* de Benoît Monin & Ludovic Patey (2022) : beaucoup plus complet, mais très bien présenté
  — *Turing Computability: Theory and Applications* de Robert I. Soare (2016) : concis et complet

  - *Computability Theory: An Introduction to Recursion Theory* de Herbert B. Enderton (2011) : beaucoup plus élémentaire et moins complet que les précédents

— Sur le **typage**, le livre *Types and Programming Languages* de Benjamin Pierce (2002) aborde les aspects à la fois théoriques et pratiques du typage dans différents langages de programmation de manière bien plus complète que ce que j'ai pu faire en cours.

  Beaucoup plus costaud : *Proofs and Types* de Jean-Yves Girard (disponible à l'adresse `http://www.paultaylor.eu/stable/prot.pdf` ; il s'agit de la traduction anglaise d'un original français, mais je crois que l'original est devenu introuvable). Contient des définitions précises de divers systèmes logiques et de typage, la preuve de leur normalisation, et les explications mathématiquement précises sur la correspondance de Curry-Howard.

— Sur le **théorème de Gödel** : *Gödel's Theorem (An Incomplete Guide to its Use and Abuse)* de Torkel Franzén (2005) est un ouvrage de semi-vulgarisation très bien écrit, et il prend notamment le temps d'expliquer plein de malentendus sur l'usage fait de ce théorème (notamment par des non-mathématiciens).

  Je peux aussi recommander *An Introduction to Gödel's Theorems* de Peter Smith (Cambridge University Press, 2013), qui est moins de la vulgarisation et plus précis et complet, et qui est également très bien écrit, mais il est difficile à trouver.

— Sur la **programmation fonctionnelle** : si ce style de programmation vous intéresse, je vous recommande fortement d'apprendre au moins un des langages de programmation Scheme (non typé, proche du λ-calcul) et/ou Haskell (fortement typé, mais diffère du OCaml en ce qu'il est « paresseux » et « purement fonctionnel », c'est-à-dire sans effets de bord, et c'est très instructif de comprendre ce que cela permet).

— Concernant la programmation fonctionnelle **non typée**, le livre *Structure and Interpretation of Computer Programs* d'Abelson et Sussman (1996 ; aussi appelé le « Wizard Book », aussi bien en référence à l'illustration de couverture que la compétence qu'il vous fera acquérir) est un grand classique, qui a servi pendant des années de support de cours au MIT. Il est épuisé au format papier, mais il est librement disponible en ligne à l'adresse `https://web.mit.edu/6.001/6.037/sicp.pdf` sur le site du MIT. Il est basé sur le Scheme (mais ne présuppose pas la connaissance du Scheme), mais il ne s'agit pas non plus d'un cours de Scheme : il s'agit d'un cours de programmation fonctionnelle non typée et des concepts fondamentaux qui vont avec. (Il y a aussi une "édition JavaScript" du livre, plus récente, si vous préférez ce langage ; mais je ne sais pas ce qu'elle vaut.)

  Notamment, ce livre explique comment écrire un interpréteur Scheme en Scheme (et ensuite comment le modifier), autrement dit, un programme universel, et ce que ça nous apprend sur la programmation et la sémantique des languages de programmation. (L'exposé *The Most Beautiful Program Ever Written* de William Byrd, disponible sur YouTube à l'adresse `https://www.youtube.com/watch?v=OyfBQmvr2Hc`, peut également être intéressant comme une sorte de version très abrégée de ce point.)

  Le livre de Christian Queinnec *Lisp in Small Pieces* (2007 ; traduction d'un original français *Les langages Lisp* devenu introuvable) est également très instructif.

— Concernant la programmation fonctionnelle **typée**, outre le OCaml, le Haskell est un langage très intéressant (y compris mathématiquement). À son sujet, le livre de Christopher Allen et

Julie Moronuki *Haskell Programming from first principles* me semble bien (il n'existe pas au format papier, mais le PDF est achetable en ligne à l'adresse `https://haskellbook.com/`).

J'aurai certainement des ajouts à faire à cette liste ultérieurement.