Projets réalisés

J'ai écrit mes premiers programmes vers mes 14 ans, en commençant par apprendre Java. Mon premier "gros" projet, réalisé en troisième, était un serveur web (écrit en utilisant Java EE) faisant l'interface entre un navigateur et des cartes Arduino. J'avais aussi écrit quelques petits interpréteurs de langages personnels, qui s'apparentaient à des petits langages de script de type shell.

J'ai ainsi appris à programmer en Java, écrit mon premier serveur http, et j'en ai profité pour apprendre à utiliser les API de réflexion, utiles pour écrire des application modulaires en Java, et écrit des petits parseurs.

Durant mes années de lycée, je me suis intéressé à Javascript, et j'ai entamé plusieurs projets de petits jeux vidéos. Cependant aucun n'a abouti car j'étais plus intéressé par l'écriture du moteur et de l'éditeur que par le jeu lui même. J'ai tout de même appris à utiliser Babel (transpilateur permettant d'assurer la compatibilité avec tous les navigateurs), les APIs React et Redux (pour la gestion d'interfaces graphiques) et l'API WebGL (graphismes 3D sur navigateur).

En prépa, j'ai bien sûr dû mettre de côté mes projets personnels, mais je n'ai pas arrêté l'informatique pour autant. Les cours d'info m'ont évidemment permis d'apprendre à programmer en Python et en OCaml, mais aussi de goûter à une informatique plus théorique, à travers les cours mais aussi grâce à mon TIPE et un projet en classe de première année, à la demande de mon professeur d'informatique.

Je détaille ces deux projets ci-dessous.

Projet de première année

Mon professeur d'informatique avait demandé à chaque élève de réaliser un projet. Le sujet était libre, j'ai alors choisi d'écrire un compilateur d'un langage de mon cru, que j'ai nommé OCalme, inspiré principalement de Rust. Le compilateur d'OCalme prend un programme et le compile vers un autre programme écrit en Python. Il est composé d'un parseur, d'un premier compilateur vers une représentation intermédiaire (IR) et d'un deuxième compilateur de l'IR vers Python –

l'intérêt de l'IR est qu'il est facile de modifier le deuxième compilateur pour produire des programmes en un autre langage. Le compilateur est aussi muni d'un vérifieur de type, faisant de OCalme un langage fortement typé, avec un début de système de généricité.

Le compilateur est disponible à cette adresse : `https://github.com/thejohncrafter/OCalme`.
La source du programme exemple est dispinible ici : `https://github.com/thejohncrafter/OCalme/blob/master/example.oklm`.
Pour de plus amples explications : `https://github.com/thejohncrafter/OCalme/blob/master/readme.pdf`.
Si vous téléchargez le programme, vous pouvez essayer :

— `python3 main.py example.oklm` pour lancer le programme d'exemple ;
— `python3 main.py -h` pour obtenir de l'aide.

TIPE

Mon TIPE porte sur la vérification de preuves, mon objectif étant d'écrire un petit vérificateur de preuves. En raison des évènements de cette année, je n'ai pas eu l'occasion de finaliser ce projet, mais la plus grande partie du travail est déjà faite.

Fondements théoriques

Pour la partie théorique, j'ai lu le *cours de logique mathématique* de R.Cori et D.Lascar (les parties pertinentes pour mon sujet, soit le premier tome et une partie du second), et *The formal semantics of computer languages*, de G.Winskel (ici encore les parties pertinentes, soit les premiers chapitres). Avec ces lectures, j'ai écrit un document qui détaille la construction d'un système formel permettant de raisonner sur la logique propositionnelle du premier ordre, étendue au second ordre avec la quantification universelle, et où je démontre sa complétude (*i.e.* une version du théorème de compacité).

Ce document est disponible à cette adresse : `https://github.com/thejohncrafter/thejohncrafter.github.io/blob/master/verifier/theorie.pdf`

Deux points auraient dû être corrigés dans ce document mais n'ont pas pu l'être faute de temps :

— Il manque une démonstration. Cette démonstration n'est pas très compliquée, et son absence ne pose pas de problème.

— Il comporte une erreur : le théorème nommé "petit théorème de complétude" n'est en fait pas un théorème de complétude ! Mais il implique le théorème de complétude, donc l'objectif de la seconde partie du document (montrer la complétude du système) est bien rempli.

Programme

Pour la partie pratique, j'ai écrit un petit vérificateur de preuves. Les preuves sont écrites dans un langage fondé sur des s-expressions que j'aurais souhaité modifier pour le rendre plus lisible, en utilisant ce que j'ai fait avec OCalme, mais les circonstances de cette année m'ont empêché de travailler plus sur ce projet. En l'état actuel, ce programme permet de raisonner dans le calcul des prédicats du second ordre, à titre d'exemple je m'en suis servi pour démontrer l'existence et l'unicité de l'ensemble vide, des singletons et des paires à partir d'un sous-ensemble des axiomes de ZF.

Le coeur du programme est un système qui implémente les fonctions de manipulation des séquents, *i.e.* d'appliquer les règles formelles de dérivation de formules. Le reste du programme permet de faciliter la manipulation des séquents, en particulier en mettant à disposition de l'utilisateur un langage adapté à l'écriture de preuves formelles.

En l'état, ce programme implémente entièrement la déduction naturelle : toute preuve de formule du calcul des prédicats est donc formalisable.

À titre d'exemple, j'ai formalisé les preuves de :

— $(A \wedge B) \Leftrightarrow (B \wedge A)$ (pour A ; B des propositions) ;
— $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$ (pour A ; B ; C des propositions) ;
— $\exists !x, (\exists C, x \in C) \wedge (\forall y, \neg(y \in x))$ (existence et unicité de l'ensemble vide) ;
— $\forall x, \forall y, ((\exists C, x \in C) \wedge (\exists C, y \in C)) \Rightarrow (\exists !a, (\exists C, a \in C) \wedge (\forall z, (z \in a) \Leftrightarrow (z = x \vee z = y))$ (existence et unicité de la paire $x; y$).

Une version interactive de ce programme adaptée aux navigateurs (fonctionne en tout cas sur Firefox) est disponible sur `https://thejohncrafter.github.io/verifier/index.html`. La zone de texte à gauche comporte la preuve formelle à vérifier, la console à droite donne la sortie standard du programme (Ctrl+Entrée pour exécuter).

La source est disponible sur `https://github.com/thejohncrafter/Verifier`. Cependant elle n'est pas particulièrement lisible, encore à cause de l'abandon du projet suite à l'annulation des TIPE. J'espère tout de même qu'une lecture superficielle permet de comprendre le principe du fonctionnement du programme.