Projets personnels

Arthur Correnson

18 mai 2020

Introduction

Passionné d'informatique depuis le début de mes années de lycée, j'ai développé un grand nombre de projets dans différents domaines : bio-informatique, algorithmique, intelligence artificielle, compilation. J'ai sélectionné dans ce document les travaux qui me tiennent le plus à cœur et qui me semblent le plus en phase avec mes projets futurs.

Sous la forme d'un portfolio, j'y présente le cheminement qui m'a conduit à une réelle passion pour la recherche. J'exposerai en premier lieu mes premiers pas dans le domaine de la compilation, puis ma découverte de la logique et des méthodes formelles et enfin le lien que je souhaite explorer entre logique, compilation et bio-informatique.

Chip8, un émulateur rétro

Chip8 est à la fois un ordinateur virtuel et un langage d'assemblage. Ce petit système informatique a été conçu dans les années 80 dans le but de servir de plateforme pour l'apprentissage de la programmation des jeux vidéos. Dans ce projet, je propose une machine virtuelle écrite en langage C pour l'exécution de programmes Chip8 ainsi qu'un assembleur pour son langage écrit en OCaml. Ce projet fût l'un des éléments déclencheurs de mon intérêt pour le domaine de la compilation.

La suite de mon parcours en compilation

Lors de ma première année de licence et en parallèle du projet Chip8, je développe un module Python pour le traitement de la logique propositionnelle. J'écris également un interpréteur Lisp en m'inspirant du livre *Structure and interpretation of computer programs* [1]. Lors de ma deuxième année de licence, je me lance dans l'écriture de projets de compilation plus conséquents, tous développés en OCaml. En voici une brève description :

Nacc est un module OCaml pour l'écriture de *parser combinators*. Je le développe avec l'aide d'un camarade pour répondre à notre besoin récurrent d'écrire très rapidement des *parsers*. L'omniprésence de Nacc dans nos projets a motivé notre décision de publier ce module sur le dépôt Opam. Nous continuons aujourd'hui à le maintenir et collaborons parfois avec les développeurs d'outils similaires, comme Opal.

Mfc est un compilateur pour un mini-langage impératif inspiré du langage C. Il produit du code ARM et inclut quelques optimisations comme le court-circuitage des branchements et un allocateur de registres. Le développement de ce compilateur sera largement guidé par la lecture du *Dragon Book* [2].

Lili un vérificateur de preuves. Au delà des langages de programmation, je m'intéresse aussi aux domaines de la logique et des preuves formelles. Je mets à profit les apprentissages tirés de mes précédents projets de compilation pour développer ce mini vérificateur de preuves en logique propositionnelle. Son implémentation est basé sur un lambda-calcul simplement typé ainsi qu'un algorithme d'unification.

Owl un solveur de contraintes. Je découvre la programmation logique en développant Owl, un langage inspiré de Prolog. J'en propose une implémentation très concise et composée essentiellement d'un *parser* et d'un *solver*. Malgré son caractère rudimentaire cet outil est suffisamment complet pour implémenter, par exemple, les opérations arithmétiques sur les entiers naturels ainsi que les listes chaînées.

Vers la logique et les preuves formelles

C'est véritablement durant ma deuxième année de licence et au travers de mes cours de logique, d'algorithmique et de structures de données que je m'intéresse aux preuves formelles. Je me forme en autodidacte à la programmation Coq en suivant les cours de Xavier Leroy [3] au Collège de France ainsi que le cours *Logical Foundations* de Benjamin Pierce [4] diffusé par l'Université de Pennsylvanie. Je mets à l'épreuve ces enseignements dans plusieurs projets dont SATurne, un solveur SAT développé en Coq. J'utilise également ce langage pour résoudre un exercice de mathématiques discrètes soumis par un de mes enseignants. Dans un état d'esprit similaire, je résous un casse-tête proposé par un camarade en utilisant une approche SAT à l'aide de l'outil Touist [5] développé à l'IRIT et auquel nous avons été initié par nos enseignants.

Logique, Compilation et Bio-informatique des séquences

Lors de mon année de L1, j'effectue un stage à l'IRISA au sein de l'équipe Dyliss (*dynamics, logics and inference for biological systems and sequences*) au cours duquel ma tâche consiste à implémenter en `C++` une version améliorée d'un algorithme d'alignement multiple [6] en y intégrant des idées issues du travail de recherche de mon encadrant.

Le travail sur ce projet de stage m'a permis d'identifier plusieurs problématiques liées au développement des logiciels d'alignement. Mon goût pour la compilation et la logique me pousse alors à imaginer des stratégies utiles pour un développement plus efficace des aligneurs.

En particulier, certains types d'alignements sont définis par une combinaison souvent complexe de contraintes. L'implémentation de tels aligneurs est cependant assez longue et fastidieuse et nécessite tout un travail manuel d'optimisation pour conserver des temps d'exécutions raisonnables ainsi qu'une certaine souplesse quant aux propriétés des alignements produits.

Poursuivant l'idée bien répandue d'utiliser la programmation logique et par contraintes en bio-informatique, je souhaite aujourd'hui explorer la possibilité de développer un compilateur qui permettrait de générer un aligneur optimisé à partir de spécifications de haut niveau. Une telle approche déclarative permettrait d'abstraire les détails d'implémentation et donc de favoriser l'exploration de nouvelles techniques d'alignement de séquences.

Diffusion Open Source

Les sources des projets mentionnés sont toutes disponibles sur ma page GitHub personnelle `https://github.com/jdrprod`. Pour un accès rapide, les liens sont cliquables dans le tableau suivant :

Projet	**Description**
`SuperChip8`	Emulateur rétro
`Nacc`	Module d'analyse syntaxique
`Lili`	Vérifieur de preuve
`Owl`	Langage logique
`Mfc`	Compilateur pour un langage impératif
`SATurne`	Solveur Sat en Coq
`Friday Night Mood`	Exercice sur les langages en Coq
`The Incredible Height`	Résolution d'un casse tête logique

Références

[1] Harold Abelson, Gerald Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. McGraw-Hill, Inc., USA, 2 edition, 1996.

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers : Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[3] Xavier Leroy. *Le logiciel, entre l'esprit et la matière*. OpenEdition Books, December 2019.

[4] Pierce B. et al. Logical foundations. https://softwarefoundations.cis.upenn.edu/lf-current/index.html.

[5] Khaled Skander, Ben Slimane, Alexis Comte, Gasquet Olivier, Abdelwahab Heba, Olivier Lezaud, Frédéric Maris, and Maël. Valais. La logique facile avec touist. *Actes Des 9es Journées d'Intelligence Artificielle Fondamentale (IAF 2015)*, 2015.

[6] Goulven Kerbellec. *Apprentissage d'automates modélisant des familles de séquences protéiques*. PhD thesis, 2008. Thèse de doctorat dirigée par Andonov, Rumen Informatique. Bioinformatique Rennes 1 2008.