Compilation

S1 2021–2022, Master 1 Informatique

https://gaufre.informatique.univ-paris-diderot.fr/aguatto/compilation-m1-2021

Adrien Guatto
Bureau 4027, bâtiment Sophie Germain
guatto@irif.fr

Contenu Implémentation des langages de programmation. Analyse lexicale et syntaxique. Interprétation. Inférence de types. Génération de code. Rudiments d'assembleur x86-64. Conventions d'appel.

Objectifs Ce module offre une introduction à l'implémentation des langages de programmation ainsi qu'à la conception et réalisation de compilateurs. Les concepts sont illustrés de façon concrète par la réalisation guidée d'un compilateur de langage fonctionnel ciblant l'architecture x86-64.

À la fin du module, vous :

- connaîtrez l'organisation standard des compilateurs,
- saurez écrire un analyseur syntaxique convertissant une chaîne de caractères en un format structuré,
- saurez traduire la spécification formelle d'un langage en un interprète de référence,
- saurez implémenter des règles de typage en termes de contraintes entre types,
- saurez ramener les structure de contrôle de haut niveau à l'assembleur,
- comprendrez les conventions qui assurent l'interopérabilité des langages au niveau système.

Prérequis Le module suppose une familiarité avec la programmation en général et la programmation fonctionnelle en particulier, ainsi qu'avec le développement sous environnement UNIX. Le compilateur servant de support au projet est écrit en OCaml, dont on supposera les bases connues.

Matériel pédagogique Le contenu du module est résumé dans le journal disponible sur son dépôt git. Il ne s'appuie pas sur un manuel particulier, mais le livre d'Appel [2012] est écrit dans un esprit proche.

Organisation Chaque semaine comprend 2h de cours magistral et 2h de travaux pratiques.

Séance	Enseignant	Horaire	Lieu
Cours magistral	Adrien Guatto	mardi 16h15–18h15	salle 2035, Sophie Germain
Travaux pratiques	Peter Habermehl	lundi 8h30–10h30	salle 2001, Sophie Germain

Les séances de travaux pratiques seront consacrés soit à la réalisation d'un TP guidé, soit au soutien à la réalisation du projet. Le projet consiste en la réalisation d'un compilateur allant d'une variante très simplifiée d'OCaml à l'assembleur x86-64. Il est structuré en jalons indépendants, chacun consistant en la réalisation d'une passe de compilation distincte.

Planning des séances de cours

1. Présentation du module. Un petit panorama de la compilation via le micro-langage Marthe.
   - Lecture conseillée : Appel [2012, §1].
2. Présentation du langage Hopix et du compilateur flap. Analyse lexicale et syntaxique appliquée.
   - Lecture obligatoire : les manuels du générateur d'analyseurs lexicaux ocamllex [Leroy et al., 2020, §13] et du générateur d'analyseurs syntaxiques Menhir [Pottier and Régis-Gianas, 2020].
   - Lecture conseillée : Appel [2012, §2 et §3].
   - **Jalon 1** : analyse lexicale et syntaxique d'Hopix.
3. Sémantique opérationnelle et interprètes.
   - Lecture obligatoire : les notes disponibles sur la page du module.

— **Jalon 2** : interprète Hopix.

4. Typage et vérification des types.
   — Lecture conseillée : les notes disponibles sur la page du module.
   — **Jalon 3** : vérification de types pour Hopix.
5. Assembleur x86-64 et ABI POSIX System V. Traduction de Retrolix vers x86-64.
   — Lecture obligatoire : notes sur la programmation x86-64 de Tolmach [2012].
   — **Jalon 4** : traduction de Retrolix vers x86-64.
6. De Fopix à Retrolix : linéarisation du contrôle ; matérialisation de la convention d'appel.
   — Lecture conseillée : Appel [2012, §8].
   — **Jalon 5** : traduction de Fopix vers Retrolix.
7. De Hobix à Fopix : explicitation des fermetures.
   — Lecture conseillée : Appel [2012, §15.2].
   — **Jalon 6** : traduction de Hobix vers Fopix.
8. De Hopix à Hobix : élimination du filtrage, des valeurs structurées, des boucles dénombrées.
   — **Jalon 7** : traduction de Hopix vers Hobix.

9+. Perspectives sur la génération de code optimisé. Soutien au projet.

Évaluation La note finale est formée pour moitié de la note du projet, évalué via une soutenance, et pour moitié de la note à l'examen.

Les projets sont réalisés en binôme mais la note est individuelle. Chaque binôme doit livrer un travail autonome. Vous êtes autorisés à discuter des problèmes que vous rencontrez avec les autres binômes, mais chacun doit fournir sa propre solution. En particulier, *l'échange de code est interdit* et entraînera l'attribution d'un 0 aux projets des n binômes concernés.

Informations pratiques

— Le cours reposera sur un projet qui utilise la chaîne standard de développement en OCaml, et notamment les outils OPAM, Dune, Menhir et utop. Je vous conseille de lire leurs manuels respectifs. Si vous n'êtes pas familiers du langage, vous devez vous mettre à niveau, par exemple via son site.
— Vous devez vous inscrire sur la liste de diffusion du cours, qui centralise et archive nos échanges.
https://listes.u-paris.fr/wws/info/m1.2021.compilation.info
— Vous devez avoir un compte sur le serveur GitLab de l'UFR, et forker le dépôt du module.
https://gaufre.informatique.univ-paris-diderot.fr/aguatto/compilation-m1-2021

Références

Andrew W. Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, 2012.

Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system release 4.10, 2020. URL https://ocaml.org/releases/4.10/htmlman/index.html.

François Pottier and Yann Régis-Gianas. Menhir Reference Manual, 2020. URL http://gallium.inria.fr/fpottier/menhir/manual.html. version 20200624.

Andrew Tolmach. Notes on x86-64 programming, 2012. URL http://web.cecs.pdx.edu/apt/cs491/x86-64.pdf.