Ali Baydoun
Romain Bossy
Elie Carrot
Jorane Guidetti
Seryozha Hakobyan
Noémie Pellegrin

Université Grenoble Alpes

Projet — Master 1 Informatique

Compilateur pour le language MinCaml
Documentation utilisateur

LesPerdus – Janvier 2024

Table des matières

1 Introduction 1
2 Présentation générale du compilateur 1
3 Installation des logiciels/librairies nécessaires 2
4 Présentation des options du compilateur 2
5 Utilisation et éxécution 3
6 Lancement de tests prédéfinis 4

1 Introduction

Ce document présente de manière succincte les fonctionnalités et options du compilateur MinCaml conçu par le groupe LesPerdus. Vous trouverez dans ce manuel les librairies et logiciels à installer pour pouvoir exécuter le compilateur, les commandes pour lancer les tests automatiques sur les programmes MinCaml prédéfinis et la procédure à suivre pour lancer le compilateur sur de nouveaux programmes MinCaml.

2 Présentation générale du compilateur

Le présent compilateur est un programme permettant de lire des fichiers écrits dans le langage MinCaml (extension de fichier `.ml`, sous-ensemble du langage OCaml), de vérifier leur syntaxe ainsi que leur sémantique pour ensuite les compiler en fichiers assembleurs. Il sert à « traduire » un fichier dans un langage donné (ici MinCaml) en fichier ayant des commandes compréhensibles pour la machine. Il est l'un des premiers outils de la chaîne de compilation-exécution d'un fichier.

Les fonctionnalités qu'assure ce compilateur en prenant un fichier MinCaml en entrée sont :

- Découpage du programme fourni en entrée en symboles et vérification de la syntaxe du programme (analyse syntaxique) ;
- Vérification et inférence de types du programme fourni en entrée (typage correct) ;
- Génération du code intermédiaire ASML correspondant au fichier d'entrée ;
- Génération du code assembleur correspondant au fichier d'entrée ;
- Affichage du résultat des différentes étapes du compilateur appliquées au fichier d'entrée (α-conversion, `let`-reduction, ...) ;
- Réalisation de tests automatiques sur des fichiers Mincaml prédéfinis.

N.B. Cette version du compilateur ne gère pas les programmes mélangeant des flottants et des entiers (c'est soit l'un, soit l'autre). En particulier, il ne gère pas les tableaux de flottants. Il persiste aussi quelques erreurs sur les fonctions retournant une fonction.

Le dossier mincaml-compiler est composé de plusieurs sous-dossiers dont chacun a une fonction parmi :

ARM contient la librairie standard de mincaml pour la compilation de fichiers assembleurs (ARM) ;
asml présente des exemples de fichiers écrits en code intermédiaire asml ;
mincaml possède plusieurs exemples de programmes écrits en langage mincaml ;
ocaml regroupe tous les fichiers nécessaire au fonctionnement du compilateur construit ;
scripts contient tous les scripts de tests automatisés ;
tests possède tous les programmes mincaml sur lesquels les tests sont exécutés ;
tools détient l'interpréteur asml permettant d'exécuter des fichiers asml.

3 Installation des logiciels/librairies nécessaires

Afin de lancer le compilateur, plusieurs librairies et programmes sont nécessaires et doivent être installés. Ces derniers sont :

- Le langage **OCaml** (version 4.13 minimum requise – `https://v2.ocaml.org/docs/install.fr.html`) afin de pouvoir le compiler. Il est conseillé d'utiliser le gestionnaire de paquets **Opam** (`https://opam.ocaml.org/`) pour installer facilement Ocaml et les autres packages nécessaires au projet ;
- Le package **ocamlbuild** (`https://opam.ocaml.org/packages/ocamlbuild/`) qui contient un ensemble de librairies utilisées pour la compilation du compilateur ;
- Le programme **qemu-user** (`https://www.qemu.org/download/`) et l'ensemble d'outils GNU ARM **gcc-arm-linux-gnueabi** (`https ://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads/11-2-2022-02`) afin de transformer un fichier assembleur en un fichier exécutable. Ce dernier est lancé par QEMU et donne le résultat de l'exécution du programme MinCaml écrit initialement.

4 Présentation des options du compilateur

L'exécution du compilateur s'effectue avec la syntaxe de commande :

```
./mincamlc nom_du_fichier_mincaml_entree -options [nom_du_fichier_sortie]
```

Toutes les commandes suivent cette syntaxe à quelques différences près. Le détail des commandes existantes et leur écriture est présenté ci-dessous (cf. README).

Création du fichier assembleur

Pour créer le fichier assembleur à partir d'un fichier mincaml, écrivez simplement :

```
./mincamlc chemin_relatif/nom_du_fichier_mincaml -o nom_du_fichier_sortie.s
```

Affichage du parsing d'un fichier mincaml

Si vous souhaitez seulement tester le résultat de l'analyseur syntaxique (parser) d'un fichier mincaml, écrivez la commande :

```
./mincamlc chemin_relatif/nom_du_fichier_mincaml -p
```

Remarque : Cette commande n'affiche rien, elle retourne 0 en cas de succès et 1 sinon.

Affichage du résultat du typechecking

Pour effectuer seulement l'analyse de type d'un programme mincaml, notez la commande :

```
./mincamlc chemin_relatif/nom_du_fichier_mincaml -t
```

Dans le cas où l'analyse échoue, un message d'erreur est affiché (`...not unifiable...`).

Test du fonctionnement de chaque étape du compilateur

Le compilateur est aussi doté d'options afin de visualiser chaque transformation effectuée par le compilateur sur un programme mincaml. Cela permet de tester le bon fonctionnement de chaque étape.

Pour lancer chaque étape, la syntaxe de la commande est la suivante :

```
./mincamlc chemin_relatif/nom_du_fichier_mincaml -test-nom_etape
```

La liste des `nom_etape` possible est :

- `knorm` pour la k-normalisation ;
- `alpha` pour l'α-conversion ;
- `let` pour la réduction des expressions imbriquées ;
- `closure` pour la conversion en closures ;
- `optim` pour toutes les optimisations front-end optionnelles ;
- `back` pour afficher la structure intermédaire du back-end.

Par exemple, pour afficher la transformation du programme mincaml `5-change_value_in_array.ml` situé dans `tests/gen-code/` jusqu'à l'étape de closure sans optimisations supplémentaires, la commande est :

```
./mincamlc ../tests/gen-code/5-change_value_in_array.ml -test-closure
```

Paramétrage supplémentaires

L'option `-n_iter` permet de contrôler le nombre de fois que les optimisations secondaires sont appliquées pour simplifier le programme d'entrée au maximum. Ce nombre est fixé par défaut à 50 itérations. La valeur 0 permet de les désactiver.

De plus, il est possible de choisir le nombre maximal d'opérations qu'une fonction peut avoir pour qu'elle soit considérée par l'« extension inline » (remplacement d'une fonction par son code si cette dernière est suffisamment courte et non récursive). Par défaut ce nombre est de 10, mais il est possible de le modifier avec l'option : `-inline_depth <n>` .

Par exemple, si l'on souhaite afficher la transformation du programme `source.ml` par les optimisations secondaires en acceptant l'extension inline des fonctions ayant 20 opérations maximum, on écrit la commande :

```
./mincamlc source.ml -test-optim -inline_depth 20
```

Enfin, si vous souhaitez afficher le type des variables et des fonctions définies dans un programme mincaml (pour toutes les étapes du compilateur avant la closure), vous pouvez rajouter l'option : `-show_type`.

4.1 Affichage du fichier de code intermédiaire ASML

Si vous souhaitez obtenir le code intermédiaire ASML plutôt que le fichier résultat assembleur, d'un programme mincaml déjà existant, vous devez vous placez dans le dossier `ocaml` et écrire la commande :

```
./mincamlc chemin_relatif/nom_du_fichier_mincaml -asml
```

Cela vous affichera sur la sortie standard le code ASML généré avec toutes les optimisations activées côté front-end (l'ensemble des étapes sont : k-normalisation, α-conversion, β-réduction, réduction des expressions imbriquées, extension inline, propagation de constantes, élimination des définitions inutiles et conversion en closures).

Si vous souhaitez obtenir un fichier asml contenant le code ASML généré, écrivez la commande :

```
./mincamlc source.ml -asml -o resultat.asml
# ou ./mincamlc source.ml -o resultat.asml -asml
```

5 Utilisation et éxécution

Pour compiler le compilateur, vous devez ouvrir un terminal de commandes, vous placez dans le dossier `mincaml-compiler/ocaml/` et entrer la commande `make` ou `make all` afin de compiler les fichiers exécutables permettant de lancer le compilateur. A l'inverse, pour nettoyer l'ensemble des dossiers des fichiers temporaires créés et des exécutables, écrivez la commande `make clean`.

5.1 Code intermédiaire ASML

Vous pouvez exécuter le code ASML généré à l'aide de l'exécutable fourni en amont en suivant la procédure suivante :

1. Créez le code ASML avec le compilateur : `mincamlc -asml -o code.asml source.ml`
2. Exécutez-le avec l'utilitaire fourni dans le dossier `tools` : `asml code.asml`

5.2 Code d'assemblage

Un fichier compilé en assembleur peut ensuite être exécuté par qemu en plaçant les fichiers assembleur dans le dossier `ARM` et en exécutant `make test`.

Si vous souhaitez directement compiler et exécuter avec qemu un fichier mincaml déjà existant, écrivez (depuis le dossier `ocaml`) :

`make chemin_relatif_depuis_la_racine_du_projet/nom_du_fichier_mincaml.ml`

Par exemple, pour compiler et exécuter le fichier `2-back-imbricate_external_function.ml` localisé dans le dossier `tests/gen-code/` il faut écrire :

`make ./tests/gen-code/2-back-imbricate_external_function.ml`

Toutefois, si vous souhaitez tester la compilation et l'exécution d'un fichier mincaml que vous créez vous-mêmes, suivez les étapes suivantes :

1. `./mincamlc prog.ml -o prog.s`
2. `arm-linux-gnueabi-gcc prog.s libmincaml.S -o prog.arm -mfpu=fpv5-d16 -lm`
3. `qemu-arm -L /usr/arm-linux-gnueabi prog.arm`

Cette procédure est automatisée dans le script `mincaml_compile.sh` qui prend en paramètre un chemin vers un fichier à compiler et exécuter.

6 Lancement de tests prédéfinis

6.1 Présentation de la suite de tests

Le compilateur dispose d'un ensemble de tests prédéfinis pour le parser, le typechecker, les étapes du côté front-end et l'ensemble du compilateur (allocation des registres et génération du code ARM). Pour les exécuter, vous devez écrire la commande : `make test` (depuis le dossier `ocaml` !).

Le résultat des tests pour chaque élément du compilateur testé sera écrit sous la forme :

```
1 ------------ TESTING nom_de_la_fonctionnalité_testée ------------
2 Iteration numero_de_l_iteration : nom_de_l_iteration -
3 Test on : nom_du_fichier_testé ... Résultat du test (OK si valide, KO si
     invalide + message d'erreur correspondant)
4 ...
5 -------- END TESTING ------
6 Tests passed : nombre_de_tests_correspondants / nombre_de_tests_total
7 Tests failed : nombre_de_tests_correspondants / nombre_de_tests_total
8 --------------------------------
```

Code 1 – Aperçu d'écran des résultats des tests

A la fin, un résumé global des tests est présenté avec le nombre de tests réussis et le nombre de tests ayant échoués.

6.2 Définition et lancement des tests sur de nouveaux fichiers

Les tests automatiques précédents sont lancés uniquement sur les fichiers mincaml présents dans le dossier `tests/`. Si vous souhaitez tester d'autres programmes mincaml. Il vous faut :

1. Créer le fichier mincaml et le sauvegarder dans le dossier correspondant aux tests à effectuer : `tests/parser/(invalid ou valid)` pour le parser, `tests/typechecker/(invalid ou valid) /0-general_tests/` pour l'analyse de types ou `tests/gen-code/` sous la forme `8-fichier.ml`.
2. Ecrire le fichier résultat correspondant à l'exécution du programme mincaml sous la forme (uniquement pour le cas de fichiers dans `tests/gen-code/`) : `nom_du_fichier_mincaml.expected` ;
3. Relancez les tests dans le dossier `ocaml/` avec la commande `make test`.

Attention, il est impératif de laisser une ligne vide à la fin des fichiers finissant par `.expected` !