Compilation de C−−

Projet Programmation 1: 2ème partie

L'objectif de ce projet est de réaliser un compilateur produisant du code LC3 [1]. Il s'agit d'un fragment relativement petit du langage C. Ce document explique les restrictions de lexique et syntaxe du C−−, le règles de typage, ainsi que quelques éléments de traduction vers LC3.

Attention, une archive contenant du code est distribuée avec ce projet. Vous devez modifier uniquement les fichiers qui sont marqués A MODIFIER.

1 Syntaxe

Le C−− contient des notions déjà vues dans le langage C (variable, fonction, constantes entières, type entier et pointeur) mais beaucoup plus simplifiés. Les principales restrictions sont présentées dans cette section.

1.1 Lexique et analyse lexicale

Le langage C−− ne contient qu'un ensemble restreint de constantes : que les constantes entières. Les constantes entières peuvent être écrites en base 10, 8 ou 16. Il resulte qu'uniquement le type **int** et les types adresse sont utilisables. Les déclarations multiples de variables ne sont pas permises, ni les initialisations à la déclaration.

Les opérateurs arithmétiques et de comparaison sont ceux du C. Les opérateurs logiques sont admis, mais pas les opérateurs bit-à-bit. Les expressions peuvent contenir des appels de fonction. L'expression conditionnelle est aussi autorisée.

Les commentaires et les identificateurs sont ceux du C.

Les instructions sont restreintes à l'affectation, décision (if-then, if-then-else) et aux boucles (for et while).

Le lexique du C−− est formalisé dans le fichier `src/clexer.mll`. Il est compilé avec `ocamllex` pour générer le code OCaml de l'analyseur. Cette façon de compiler est spécifiée dans le fichier `dune` par : `(ocamllex (modules clexer))`.

L'analyseur lexical est appelé dans le fichier `main.ml`, qui contient les appels aux phases principales de la compilation.

1. reference langage

1.2 Syntaxe et analyse syntaxique

La syntaxe de C−− est dont réduite aux constructions du C qui utilisent le lexique restreint ci-dessus.

L'analyseur syntaxique du C−− est codé dans le fichier `src/cparser.mly`, compilé avec `menhir` pour générer le code OCaml de l'analyseur. Cette façon de compiler est spécifiée dans le fichier `dune` par : `(menhir (modules cparser))`.

L'analyseur syntaxique construit un arbre de syntaxe abstraite (AST), valeur de type `file` dans le fichier `src/cast.mli`.

L'analyseur syntaxique est appelé dans le fichier `main.ml`.

Le code distribué permet à présent de transformer des fichiers C−− vers des fichier en format DOT contenant l'arbre de syntaxe abstraite. Pour cela, il faut :

1. nettoyer les fichiers générés avec : `dune build`
2. compiler les sources avec : `dune build`
3. appeler l'exécutable généré avec : `./ccomp --debug test.c`

Ces commandes sont en partie présentes dans le fichier Makefile distribué avec les source.

Affichage de l'arbre syntaxique. Afin de vous aider à lire les AST, nous fournissons un module `src/pretty.ml` qui génère un fichier `.dot` (en format DOT). L'AST en format DOT peut ensuite être visualisée avec l'outil `dot` du paquetage `graphviz` par `dot -v -Tpng -O test_ast.dot`

2 Typage statique

Les règles de typage informelles sont :

1. Les types autorisés sont : entier et pointeur vers un type (c'est à dire, `int*`, `int**`...).
2. Une variable globale est déclarée une seule fois.
3. Une fonction est définie une seule fois.
4. Une variable locale peut être redéfinie dans un bloc imbriqué.
5. Les valeurs affectées doivent être de type de la valeur gauche de l'affectation.
6. La conversion implicite entre type pointeur et type entier n'est pas permise.
7. Entre valeurs de type pointeur vers le même type, les opérations permises sont : la soustraction [2]
8. Entre une valeur de type pointeur et une valeur entière, les opérations permises sont : l'addition et la soustraction.
9. Une fonction doit être appelée avec le type et le nombre des paramètres correspondant à sa définition.

2. Ceci est une simplificatio par rapport à la norme qui demande que les pointeurs soient dans le même bloc mémoire.

10. Le bloc d'une fonction doit contenir une instruction **return** avec une expression du type de retour déclarée pour la fonction.
11. La définition d'une fonction ne doit pas contenir deux paramètres avec le même nom.

Ces règles doivent être codées dans le fichier `ctyping.ml`, dans une fonction `check_file`.

Durant le typage, un arbre syntaxique typé est construit pour les besoins des phases de synthèse. Cet arbre est défini dans le fichier `tast.mli`.

3 Synthèse de code LC3

Cette partie sera faite en deux étapes :

1. Dans un premier temps, votre compilateur devra traiter les expressions et les strucutres de contrôle (if, for, while). Le compilateur traite seulement le contenu du main, où se trouvent toutes les instructions.
2. Pour la deuxième étape, vous ajouterez les fonctions.

À chaque étape, vous devez prévoir une série de test qui permettent de vérifier les différents cas.