Algorithmes, Types, Preuves: Présentation du projet

Martin Strecker

Univ. J.-F. Champollion

Année 2021/2022

Plan

Plan

Survol du projet

Le développement à faire :

- Formalisation de la syntaxe d'un langage impératif en Caml ($\rightsquigarrow$ Langages et automates)
- Vérification de types de ce langage
- Conception d'un assembleur simple
- ... avec sémantique de cet assembleur
- Traduction langage impératif vers assembleur
- Génération et affichage d'un graphe de flot de contrôle

Survol du projet

Motivation et but d'apprentissage :

- Savoir transférer les concepts vus pour un langage fonctionnel vers langage impératif (surtout : typage)
- Développer une intuition pour différents niveaux de langage (de haut niveau, assembleur)
- S'initier à des concepts de compilation
- Transférer et appliquer des concepts de théorie des graphes

Plan

Inspiration

Langage inspiré du langage C, mais :

- Moins sauvage :
  - Distinction nette entre expressions et instructions.
    Par exemple : impossible d'écrire `(x = 3) + (x = 2);`
  - Permet la définition d'une sémantique précise et claire
- Pourtant compatible avec C : Le même programme peut être
  - compilé avec `gcc` et exécuté
  - compilé avec *votre* compilateur et exécuté

Exemple : Programme source

```
int fac (int n) {
  int res;
  res = 1;

  while (n > 1) {
    res = res * n;
    n = n - 1;
  }
  return(res) ;
}
```

Nous manipulons :

- des fonctions simples
- avec paramètres et variables locales
- contenant des instructions impératives
- sans appel d'autres fonctions
- qui renvoient une valeur (`return`)

Expressions : Type de données

Sémantique informelle : Une expression a une *valeur*

```
type 'a expr =
    Const of 'a * value   (* constante *)
  | VarE of 'a * vname      (* variable *)
  | BinOp of 'a * binop * ('a expr) * ('a expr)
                          (* operation binaire *)
  | IfThenElse of 'a *
        ('a expr) * ('a expr) * ('a expr)
                          (* if - then - else *)
```

Note : `IfThenElse` est une expression analogue à `(.. ? .. : ..)` en C

Instructions : Type de données

Sémantique informelle : Une instruction (*statement*) induit un *changement d'état*

```
type 'a stmt =
    Skip                                    (* ne fait rien *)
  | Assign of 'a * vname * ('a expr)  (* affectation *)
  | Seq of ('a stmt) * ('a stmt)  (* sequence *)}
  | Cond of ('a expr) * ('a stmt) * ('a stmt)
                                  (* if .. then .. else .. *)
  | While of ('a expr) * ('a stmt)
  | Return of ('a expr)   (* Renvoi d'une valeur *)
```

Variables et valeurs

Variables et fonctions :

```
(* variable names *)
type vname = string

(* function names *)
type fname = string
```

Valeurs :

```
type value =
    BoolV of bool
  | IntV of int
```

Typage : bases

Juste deux types :

```
type tp = BoolT | IntT
```

Déclaration de variable :

```
type vardecl =  Vardecl of tp * vname
```

Fonctions :

```
(* function declaration:
   return type; parameter declarations *)
type fundecl =
    Fundecl of tp * fname * (vardecl list)

(* function definition:
   function decl; local var decls; function body *)
type 'a fundefn =
    Fundefn of fundecl * (vardecl list) * ('a stmt)
```

Exemple

```
int fac (int n) {
  int res;
  res = 1;

  while (n > 1) {
    res = res * n;
    n = n - 1;
  }
  return(res) ;
}
```

```
# parse "Tests/fac.c" ;;
- : int Lang.fundefn =
Fundefn (
  Fundecl(IntT, "fac",
          [Vardecl (IntT, "n")]),
 [Vardecl (IntT, "res")],
 Seq
  (Seq (Assign (0, "res",
                Const (0, IntV 1)),
        ....),
  Return (VarE (0, "res"))))
```

Typage : Environnement

Le typage doit

- prendre en compte les déclarations des variables
- le type de résultat de la fonction

```
type environment =
    {localvar: (vname * tp) list;
     returntp: tp}
```

Typage

Le typage annote l'arbre syntaxique (expressions) avec le type des sous-expressions.
Résultat du parser :

```
Seq (Assign (0, "res", Const (0, IntV 1)),
  While (BinOp (0, BCompar BCgt,
          VarE (0, "n"), Const (0, IntV 1)), ...))
```

Après typage :

```
Seq (Assign (IntT, "res", Const (IntT, IntV 1)),
  While (BinOp (BoolT, BCompar BCgt,
          VarE (IntT, "n"), Const (IntT, IntV 1)),...))
```

A faire :

- Écrire typage d'expressions, instructions et fonctions
- Analyses de bone formation :
  noms des variables disjoints ; fonctions terminent avec `return`

Plan

Plan

Plan