Université Bordeaux 1
Licence, module Compilation, 2016/2017

Mini-projet : interprétation et compilation du langage IMP

1 Langage source : syntaxe

Le langage source, IMP [1], est un langage de programmation impératif très simple, où toutes les variables sont de type entier. Voici une grammaire de ce langage (au format de Bison) :

```
E: E Pl T
 | E Mo T
 | T


T: T Mu F
 | F


F: '(' E ')'
 | I
 | V


C : V Af E
  | Sk
  | '(' C ')'
  | If E Th C El C
  | Wh E Do C
  |  C Se C
```

où les symboles `I V Af Sk Se If Th El Wh Do Pl Mo Mu` représentent les lexèmes suivants :
`I` : une suite de chiffres, non-vide, commençant par un chiffre non-nul
`V` : un identificateur de variable

symbole	Af	Sk	Se	If	Th	El	Wh	Do	Pl	Mo	Mu
lexeme	:=	Skip	;	if	then	else	while	do	+	-	*

(i.e. ces lexèmes ne comportent qu'une unité lexicale, qui est donnée dans le tableau). On appelle *commande* un mot engendré par le non-terminal C ; une *commande atomique* est une commande qui n'est pas décomposable sous la forme c_1 `Se` c_2 pour des commandes c_1, c_2 ; une *expression* est un mot e engendré par le non-terminal E.

1. qui est dû à J. Goubault-Larrecq, cours de sémantique et compilation, licence 1, ENS Cachan, 2013.

2 Langage source : sémantique

La sémantique de IMP est définie ci-dessous. Le procédé de définition employé s'appelle une sémantique *opérationnelle à petits pas*. Notons V un ensemble de variables [2]. On appelle *environnement* sur V toute application $\rho : V \to \mathbb{Z}$. Pour tout mot w sur l'alphabet $\{0, 1, \ldots, 9\}$, on note $\nu(w) \in \mathbb{N}$ l'entier dénoté par w en base 10. La *valeur* d'une expression e dans un environnement ρ est définie par :

$$[\![x]\!]\rho = \rho(x) \quad \text{pour toute variable } x \in V,$$

$$[\![w]\!]\rho = \nu(w) \text{ pour tout mot } w \in \{1, \ldots, 9\}\{0, 1, \ldots, 9\}^* \cup \{0\},$$

$$[\![e_1 \texttt{ Pl } e_2]\!]\rho = [\![e_1]\!]\rho + [\![e_2]\!]\rho, \quad [\![e_1 \texttt{ Mo } e_2]\!]\rho = [\![e_1]\!]\rho - [\![e_2]\!]\rho, \quad [\![e_1 \texttt{ Mu } e_2]\!]\rho = ([\![e_1]\!]\rho) * ([\![e_2]\!]\rho), \quad [\![(e)]\!]\rho = [\![e]\!],$$

pour toutes expressions e, e_1, e_2.
Une suite de commandes est un mot sur l'alphabet des commandes. On note $\cdot$ le produit de concaténation des suites de commandes. Pour toute variable $x \in V$, expression e, commande atomique c_0, commande c, suite de commandes C, on pose :

$$\begin{array}{rcll}
((c) \cdot C, \rho) \to & (c \cdot C, \rho) & & \\
(c \texttt{ Se } c_0 \cdot C, \rho) \to & (c \cdot c_0 \cdot C, \rho) & & \\
(\texttt{Sk } \cdot C, \rho) \to & (C, \rho) & & \\
(x \texttt{ Af } e \cdot C, \rho) \to & (C, \rho[x \mapsto [\![e]\!]\rho]) & & \\
(\texttt{If } e \texttt{ Th } c \texttt{ El } c_0 \cdot C, \rho) \to & (c \cdot C, \rho) & & \text{si } [\![e]\!]\rho \neq 0 \\
(\texttt{If } e \texttt{ Th } c \texttt{ El } c_0 \cdot C, \rho) \to & (c_0 \cdot C, \rho) & & \text{si } [\![e]\!]\rho = 0 \\
(\texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) \to & (c_0 \cdot \texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) & & \text{si } [\![e]\!]\rho \neq 0 \\
(\texttt{Wh } e \texttt{ Do } c_0 \cdot C, \rho) \to & (C, \rho) & & \text{si } [\![e]\!]\rho = 0.
\end{array}$$

La sémantique d'une commande c est définie par : $[\![c]\!]\rho = \rho'$ si et seulement si

$$(c, \rho) \to^* (\varepsilon, \rho').$$

3 Langage intermédiaire

Un programme écrit dans le langage C3A (Code à 3 Adresses) consiste en une suite de lignes, chacune de la forme :

$$\textit{Etiquette} : \textit{Operateur} : \textit{Argument} : \textit{Argument} : \textit{Destination}$$

Les champs *Etiquette*, *Destination* sont des identificateurs i.e. des mots qui consistent en une lettre suivie d'un nombre quelconque de lettres ou chiffres. *Operateur* est l'un des symboles :

$$\texttt{Pl}, \texttt{Mo}, \texttt{Mu}, \texttt{Af}, \texttt{Afc}, \texttt{Sk}, \texttt{Jp}, \texttt{Jz}, \texttt{St},$$

et *Argument* peut être soit un identificateur, soit un numéral i.e. une suite de chiffres, précédée éventuellement d'un signe ("+" ou "-").
La sémantique de ce langage est définie (informellement) comme suit :

2. On assimile chaque identificateur, qui est un mot concret, à une seule variable.

- `Pl` affecte à la variable *Destination* la somme du premier et du second argument
- `Mo` affecte à la variable *Destination* la différence du premier et du second argument
- `Mu` affecte à la variable *Destination* le produit du premier et du second argument
- `Af` provoque une affectation de la valeur du second argument au premier argument
- `Afc` provoque une affectation de la valeur du premier argument, qui est un numeral, à la variable *Destination*
- `Sk` ne provoque aucune modification de l'environnement
- `Jp` provoque un saut à l'instruction étiquetée par *Destination*
- `Jz` provoque un saut à l'instruction étiquetée par *Destination*, dans le cas où le premier argument est nul
- `St` arrête l'exécution.

En fait, vue cette sémantique, seul le champ *Operateur* doit impérativement être correctement rempli ; les autres champs peuvent être "vides" c'est à dire consister en une suite de caractères "blanc" ou "tabulation", lorsque l'opération (de la même ligne) ne les utilise pas et ils ne sont pas la destination d'une instruction de saut (depuis une autre ligne).

4 Langage cible

Il s'agit du langage Y86 étudié dans l'UE d'architecture. On trouve sa description à l'adresse : `http://aurelien.esnard.emi.u-bordeaux.fr/teaching/doku.php?id=archi:y86`. On utilisera, par exemple, le simulateur `sim` pour exécuter le code Y86 (disponible à l'adresse `http://dept-info.labri.fr/ENSEIGNEMENT/archi/sim.tgz`).

5 Travail à réaliser

1- Un interpréteur du langage IMP : il s'agit d'une fonction qui prend entrée une commande c et un environnement ρ et retourne l'environnement $[\![c]\!]_{\text{imp}}\ \rho$.
2- Un interpréteur du langage C3A : il s'agit d'une fonction qui prend entrée un programme C3A P et un environnement ρ et retourne l'environnement $[\![P]\!]_{\text{c3a}}\ \rho$.
3- Un traducteur (ou compilateur) de IMP vers C3A : il s'agit d'une fonction qui prend entrée une commande c et retourne un programme C3A P tel que :

$$[\![c]\!]_{imp} = [\![P]\!]_{c3a}.$$

4- Un traducteur (ou compilateur) de C3A vers Y86 : il s'agit d'une fonction qui prend entrée un programme P (écrit en C3A) et retourne un programme Q (écrit en Y86) et un placement des variables $x \in V$ dans des adresses $\text{ad}(x)$ du processeur de façon que : pour tout environnement ρ, si on interprète Q, avec un processeur Y86, où initialement la valeur stockée dans $\text{ad}(x)$ est $\rho(x)$, alors, en fin de calcul, la valeur stockée dans $\text{ad}(x)$ est

$$([\![P]\!]_{c3a}\rho)(x).$$

Autrement dit : Q, avec le placement $\text{ad} : V \to \mathbb{N}$, calcule $[\![P]\!]_{c3a}$.
Les quatre fonctions demandées seront écrites dans le langage C, en utilisant les outils Flex et Bison.

6 Modalités de réalisation

Groupes

Le projet doit être réalisé par groupes d'au plus 2 étudiants. Les projets réalisés par ≥ 3 étudiants *ne* seront *pas* pris en considération. Un responsable par groupe donnera son adresse à son enseignant de TD.

Fichiers

Chaque étudiant placera dans un répertoire nom/compil1 des sources :

— `iimp.l` (flex), `iimp.y` (bison),

— les fichiers de fonctions auxiliaires `*.h`, `*.c` qui lui sembleront utiles ; on peut penser à un module qui traite les arbres "abstraits" et un module qui traite les structures (bilistes de quadruplets, par exemple) utilisées pour stocker le code intermédiaire puis le code Y86.

— un Makefile produisant le programme `iimp` (programme exécutable qui traduit tout programme) à partir de ces sources.

Le fichier `iimp.y` contiendra (en commentaire) :
- les 2 noms des étudiants auteurs du devoir.
- le nom de la fonction C (et du module qui la contient) répondant à chacune des questions 1,2,3,4. de la section 5.

Ces fichiers ne doivent pas avoir été modifiés après le 08/03/17 à 24h.

Notation

La notation sera établie essentiellement en fonction des performances de la commande `iimp` qui traduit tout programme IMP en un programme Y86 équivalent.
Elle sera obtenue en enchaînant les questions 3 et 4. Pour la mettre au point, les interpréteurs des questions 1 et 2 ainsi que le simulateur `sim` seront fort utiles.

Documents

On trouvera sur le site `http://dept-info.labri.fr/ENSEIGNEMENT/compi` :
- la documentation Flex et Bison
- quelques fonctions C utiles (fichiers `environ.c`, `bilquad.c`).

Responsables

Pour tous cas particuliers, adressez-vous à votre enseignant de TD/TP :
H. Derycke, email :`henri.derycke@u-bordeaux.fr` (gr. A5)
L. Clément, email :`lionel.clement@u-bordeaux.fr` (gr. A1,A3)
G. Sénizergues, email :`geraud.senizergues@u-bordeaux.fr` (gr. A2,A4).