Master Info - 2017-2018
MIF08 Compilation

Contrôle continu 1 - Durée 42 min (tiers-temps 56 min)

Éléments de correction

On rappelle en annexe les règles de génération de code pour les affectations, le test, et certaines règles de typage. On considère le programme suivant :

```
var x2: int;
x2 = 13;
while (x2 > 8) do
   x2 = x2 - 1;
done
```

Question 1 (8 points)

Montrer que ce programme est bien typé (déclarations, instructions).

Solution:

La déclaration de variable fournit l'environnement de typage $\Gamma : x_2 \rightarrow int$ (en toute rigueur il faudrait utiliser les règles pour construire Γ je ne vous en ai pas tenu rigueur lors de la correction). Ensuite, il s'agit de construire un arbre de typage clos pour ce programme : En ascii art ici :

```
Gamma(x2)=int
---------------   ---------------
Gamma |- x2:int   Gamma |- 13:int
---------------------                        ---- ** ----
Gamma |- x2=13 : void         Gamma |- while (x2 >8) do ... done : void
------------------------------------------------------------------------ [;]
Gamma |- x2=13 ; while ... done : void
```

où les doubles étoiles sont le sous-arbre suivant :

```
Gamma(x2)=int                            Gamma(x2)=int         ***
---------------   -------------          ------------------------------[-]
Gamma |- x2:int   Gamma |- 8:int         Gamma |- x2:int   Gamma |- x2-1:int
-------------------------------            ----------------------------[:=]
Gamma |- x2>8 : bool                          Gamma |- x2=x2-1 : void
-----------------------------------------------------------------------
```

et les triples étoiles celui-là :

```
Gamma (x2)=int
--------------    --------------
Gamma |- x2:int   Gamma |- 1: int
```

Cet arbre de typage étant cohérent et clos, le programme est bien typé. Attention à bien utiliser Γ sur les feuilles qui concernent des variables du programme.

Question 2 (12 points)

Générer le code 3 adresses LEIA pour ce programme, en suivant à la lettre les règles de

génération de code données en annexe. Les calculs intermédiaires seront documentés. **Vous mettrez la feuille courante en format paysage, et vous utiliserez trois colonnes : une pour décrire les appels récursifs, une pour le code généré et une pour les temporaires.**

Solution:

Voici le début de la rédaction, le reste est laissé au lecteur (ainsi que la mise en page). Dans la colonne de droite, l'effet de la déclaration de la variable x_2 est la création d'un temporaire $temp_0$. Ensuite, dans la colonne de gauche :

```
genCodeSmt(x2=13;reste)=
   genCodeSmt(x2=13)=
     dr <- genCodeSmt(13)
           dr2 = temp_1 <- newtmp()
           code.add(InstructionLETL(temp_1,13)
     (donc dr = dr2 = temp1)
     loc (x2) = temp0
     code.add(InstructionCopy(temp_0,temp_1)

   genCodeSmt(reste) ...
```

en regard dans la colonne du milieu, on obtient donc :

```
        .let temp_1 13    #ou letl temp_1 13
        copy temp_0 temp_1
```

Le code généré est finalement le suivant (obtenu avec la version prof du TP4), dans la deuxième colonne).

```
        ;; (stat (assignment x2 = (expr (atom 13)) ;))
        .let temp_1 13
        copy temp_0 temp_1
        ;; (stat (while_stat while (expr (atom ( (expr (expr (atom x2)) > (expr (atom
    8))) ))) (stat_block { (block (stat (assignment x2 = (expr (expr (atom x2)) - (
    expr (atom 1))) ;))) }))))
lbl_l_while_begin_0 :
        .let temp_2 8
        .let temp_3 0
        CONDJUMP lbl_end_relational_1 temp_0 "<=" temp_2
        .let temp_3 1
lbl_end_relational_1 :
        CONDJUMP lbl_l_while_end_0 temp 3 "!=" 1
        ;; (stat (assignment x2 = (expr (expr (atom x2)) - (expr (atom 1))) ;))
        .let temp_4 1
        sub temp_5 temp_0 temp_4
        copy temp_0 temp_5
        jump lbl_l_while_begin_0
lbl_l_while_end_0 :
```

Question 3 (Bonus points)

Remplacez le saut conditionnel du code généré par les instructions `snif` adéquates.

Solution:

Le premier cond jump est remplacé par :

SNIF temp_0 **gt** temp_2
JUMP lbl_end_relational_1
end{lstlising}
et **le** deuxième par:
\begin{ lstlisting }[style = target]
SNIF temp_3 **eq** 1
JUMP lbl_l_while_end_0
