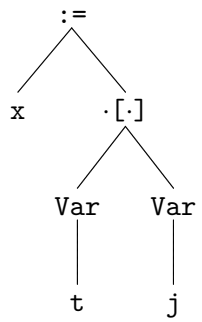
Corrigé de l'examen final de compilation

ÉNSIIE, semestre 3

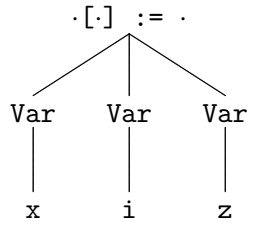
jeudi 8 janvier 2015

Exercice 1 : Syntaxe (2,5 points)

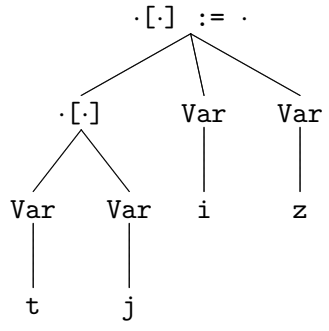
1.



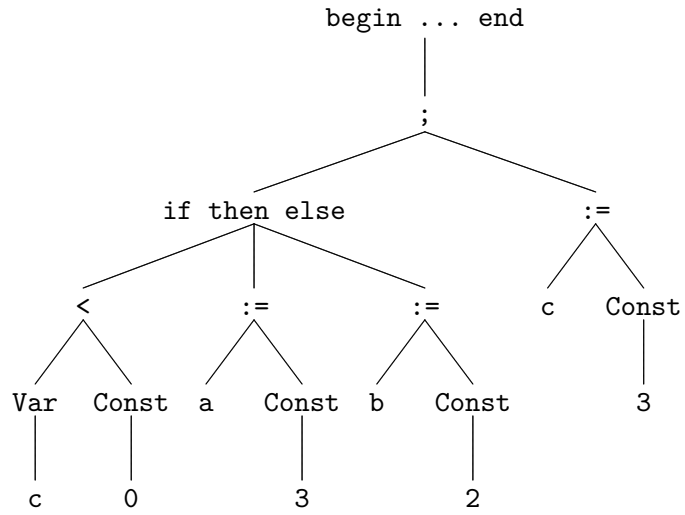
2.



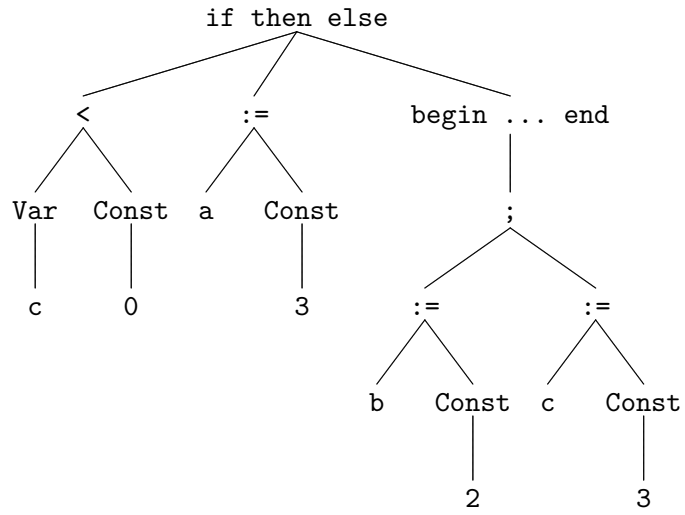
3.



4.

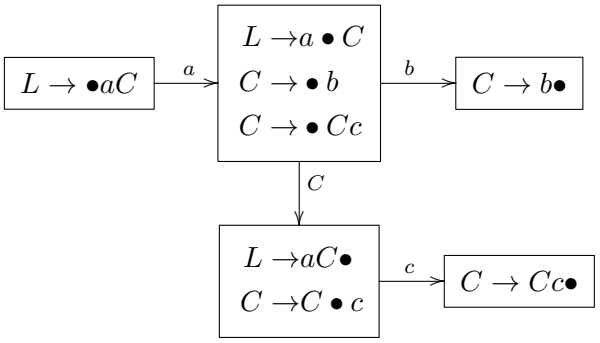


5.

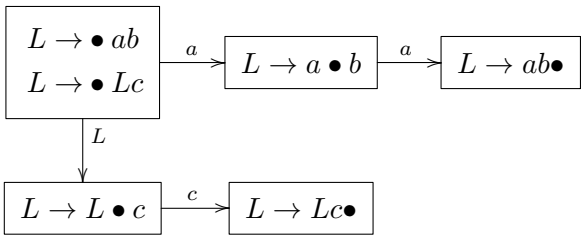


Exercice 2 : Analyse syntaxique (3 points)

1. $\{abc^n : n \in \mathbb{N}\}$
2.



3. Il y a un conflit décaler réduire dans l'état $\boxed{\begin{array}{l} L \to aC\bullet \\ C \to C \bullet c \end{array}}$: réduction par $L \to aC$ et décalage vers $\boxed{C \to Cc\bullet}$.
4. Par exemple $L \to ab \mid Lc$, on obtient l'automate LR(0) déterministe



Il n'y a pas de conflit.

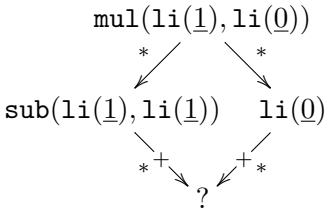
Exercice 3 : Sélection d'instructions (4,5 points)

1. $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{add}(x, y))$
2. $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{add}(x, y))$ se réécrit avec la règle (3) en $\texttt{mul}(\texttt{add}(x, y), \texttt{li}(\underline{1}))$, qui se réécrit en $\texttt{add}(x, y)$, qui ne peut plus se réécrire. $\texttt{add}(x, y)$ est donc une forme normale de $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{add}(x, y))$.
3. Non, en particulier on a la suite infinie d'étapes de réécriture

   $\texttt{mul}(\texttt{li}(\underline{2}), \texttt{li}(\underline{3})) \longrightarrow \texttt{mul}(\texttt{li}(\underline{3}), \texttt{li}(\underline{2})) \longrightarrow \texttt{mul}(\texttt{li}(\underline{2}), \texttt{li}(\underline{3})) \longrightarrow \texttt{mul}(\texttt{li}(\underline{3}), \texttt{li}(\underline{2})) \longrightarrow \ldots$

4. – (1) et (1) : pas de paire critique
   – (1) et (2) : pas de paire critique
   – (1) et (3) : à partir de $\texttt{mul}(\texttt{li}(\underline{n}), \texttt{li}(\underline{1}))$ on obtient $\texttt{li}(\underline{n})$ avec (1) et $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{li}(\underline{n}))$ avec (3). La paire critique est donc $\texttt{li}(\underline{n}), \texttt{mul}(\texttt{li}(\underline{1}), \texttt{li}(\underline{n}))$.
   – (2) et (2) : pas de paire critique
   – (2) et (3) : à partir de $\texttt{mul}(\texttt{li}(\underline{n}), \texttt{li}(\underline{0}))$ on obtient $\texttt{sub}(\texttt{li}(\underline{n}), \texttt{li}(\underline{n}))$ avec (2) et $\texttt{mul}(\texttt{li}(\underline{0}), \texttt{li}(\underline{n}))$ avec (3). La paire critique est donc $\texttt{sub}(\texttt{li}(\underline{n}), \texttt{li}(\underline{n})), \texttt{mul}(\texttt{li}(\underline{1}), \texttt{li}(\underline{n}))$.
   – (3) et (3) : pas de paire critique

   Les deux paires critiques trouvées sont joignables : en effet, $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{li}(\underline{n}))$ se réécrit par (3) en $\texttt{mul}(\texttt{li}(\underline{n}), \texttt{li}(\underline{1}))$ qui se réécrit par (1) en $\texttt{li}(\underline{n})$; et $\texttt{mul}(\texttt{li}(\underline{0}), \texttt{li}(\underline{n}))$ se réécrit par (3) en $\texttt{mul}(\texttt{li}(\underline{n}), \texttt{li}(\underline{0}))$ qui se réécrit par (2) en $\texttt{sub}(\texttt{li}(\underline{n}), \texttt{li}(\underline{n}))$.
5. Le système de réécriture n'est pas confluent : à partir de $\texttt{mul}(\texttt{li}(\underline{1}), \texttt{li}(\underline{0}))$ on peut obtenir d'une part $\texttt{sub}(\texttt{li}(\underline{1}), \texttt{li}(\underline{1}))$ avec (2) ; et d'autre part $\texttt{mul}(\texttt{li}(\underline{0}), \texttt{li}(\underline{1}))$ avec (3) puis $\texttt{li}(\underline{0})$ avec (1). Or ni $\texttt{sub}(\texttt{li}(\underline{1}), \texttt{li}(\underline{1}))$ ni $\texttt{li}(\underline{0})$ ne peuvent se réécrire, il n'y a donc pas confluence.

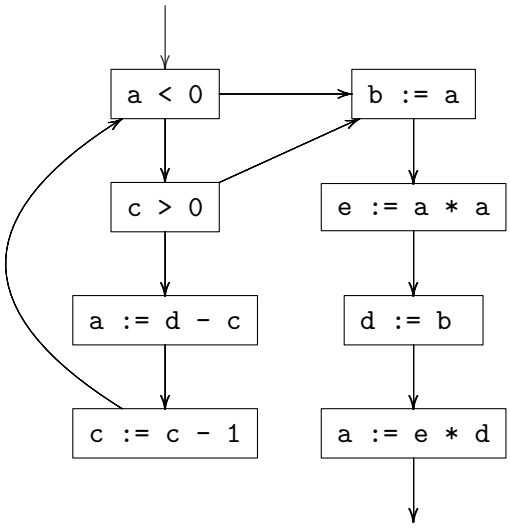


À noter : il n'était pas possible d'utiliser le théorème du cours qui dit que si les paires critiques sont joignables alors le système de réécriture est confluent, parce que le système n'est pas fortement normalisant.

6. La règle (2) duplique l'argument E, elle ne peut donc être appliquée que si l'expression qui remplacera E est pure, c'est-à-dire qu'elle ne fait pas d'effet de bord.

Exercice 4 : Allocation de registres (6 points)

1.

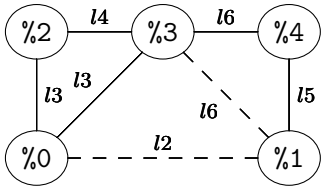


2.
```
l0: bltz %0 -> l1, l2
l1: bgtz %2 -> l3, l2
l3: sub %0, %3, %2 -> l4
l4: addi %2, %2, -1 -> l0
l2: move %1, %0 -> l5
l5: mul %4, %0, %0 -> l6
l6: move %3, %1 -> l7
l7: mul %0, %4, %3 -> l8
```

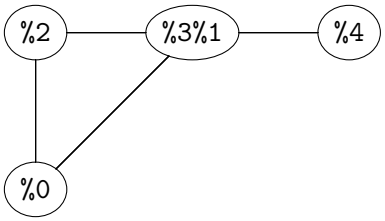
3.

instruction	vivantes après	vivantes avant
l0	%0 %2 %3	%0 %2 %3
l1	%0 %2 %3	%0 %2 %3
l2	%0 %1	%0
l3	%0 %2 %3	%2 %3
l4	%0 %2 %3	%0 %2 %3
l5	%1 %4	%0 %1
l6	%3 %4	%1 %4
l7		%3 %4

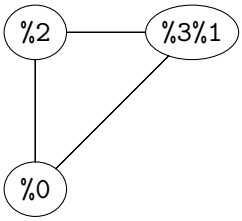
4.



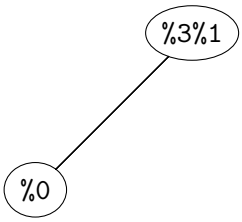
5. On ne peut pas simplifier : le nœud trivialement colorable `%1` a des arêtes de préférence.
   On ne peut pas fusionner `%0` et `%1` : en effet `%2` est un voisin de `%0` qui n'est pas voisin de `%1`, et `%4` est un voisin de `%1` qui n'est pas voisin de `%0`.
   On peut fusionner `%3` et `%1` : en effet le seul voisin `%4` de `%1` est voisin de `%3`.



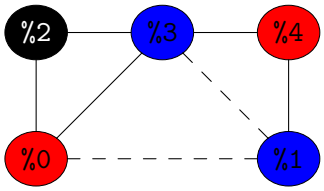
   On peut maintenant simplifier `%4`.



   On ne peut plus simplifier, ni fusionner, ni geler, il faut essayer de spiller. On spille par exemple `%2` qui est moins utilisé que `%0` et qui n'est pas la fusion de deux pseudo-registres.



   On peut simplifier `%0` et `%3%1`.
   En remontant, on attribue une couleur à `%0`, la deuxième à `%3%1`, on regarde s'il reste une couleur pour `%2` mais ce n'est pas le cas. On donne la première couleur à `%4`, les registres `%3` et `%1` ont la deuxième couleur.



6.
```
l0:  bltz $s0 -> l1', l2
l1': lw $t0, 0($sp) -> l1
l1:  bgtz $t0 -> l3', l2
l3': lw $t0, 0($sp) -> l3
l3:  sub $s0, $s1, $t0 -> l4'
l4': lw $t0, 0($sp) -> l3
l4:  addi $t0, $t0, -1 -> l4"
l4": sw $t0, 0($sp) -> l0
l2:  move $s1, $s0 -> l5
l5:  mul $s0, $s0, $s0 -> l7
l7:  mul $s0, $s1, $s0 -> l8
```
   Une optimisation ultérieure permettrait de supprimer les lignes `l3'` et `l4'`.

Exercice 5 : Convention d'appel (4 points)

1. `f` doit sauvegarder le registre callee-saved `$s0` dans lequel est stockée la valeur de `z`, c'est le seul qu'elle modifie.

   Ne faisant pas d'appel, `f` n'a pas à sauvegarder de registres caller-saved.

   La fonction `g` modifie le registre callee-saved `$ra` quand elle appelle `f`, elle doit donc le sauvegarder. Elle ne modifie pas d'autres registres callee-saved.

   `g` doit sauvegarder le registres `$a0`, puisqu'elle a besoin de la valeur de `u` après l'appel à `f`. Par contre, il n'est pas nécessaire de sauvegarder le registre `$a1`, `v` n'étant pas utilisée ensuite. Aucun autre registres caller-saved n'est utilisé.

2. Trame de `f` :

sauvegarde de `$s0`

   Trame de `g` :

sauvegarde de `$ra`
sauvegarde de `$a0`

3.
   -
     ```
     addi $sp, $sp, -4
     sw $s0, 0($sp)
     ```
   -
     ```
     lw $s0, 0($sp)
     addi $sp, $sp, 4
     ```
   -
     ```
     addi $sp, $sp, -8
     sw $ra, 4($sp)
     ```
   - avant
     ```
     sw $a0, 0($sp)
     ```
     après
     ```
     lw $a0, 0($sp)
     ```
   -
     ```
     lw $ra, 4($sp)
     addi $sp, $sp, 8
     ```