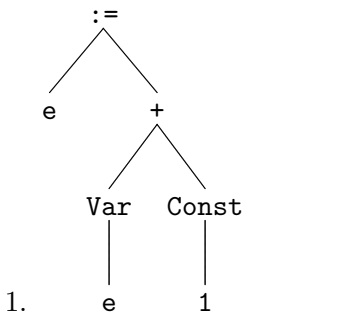
Corrigé de l'examen final de compilation

ÉNSIIE, semestre 3

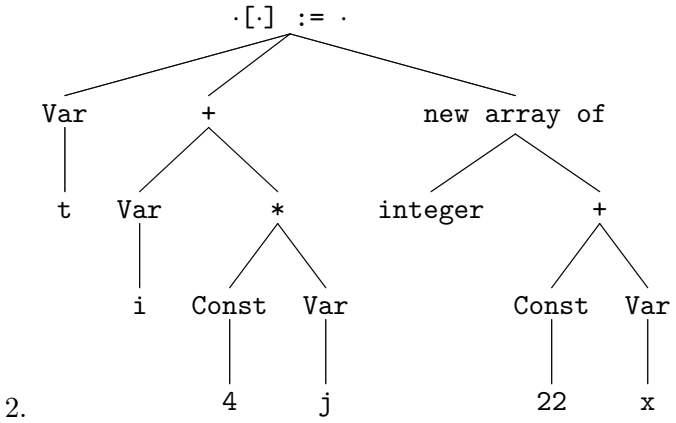
jeudi 10 janvier 2013

Exercice 1 : Syntaxe (2 points)

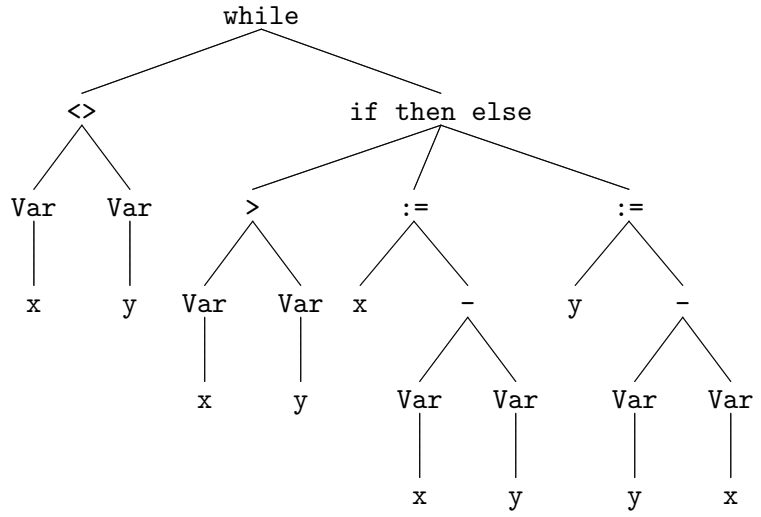
1.



2.

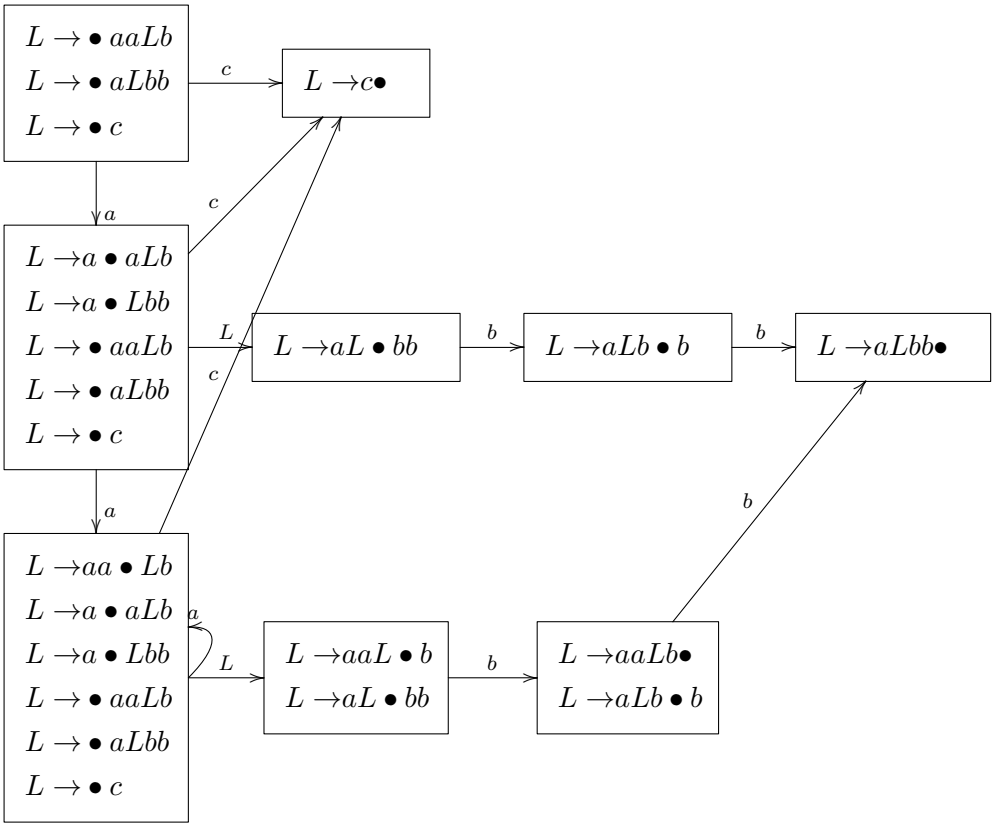


3.



Exercice 2 : Analyse syntaxique (4 points)

1.



2. Non, il y a un conflit réduire/décaler dans l'état en bas à droite.

3. On peut montrer qu'un langage reconnu par une grammaire LR(0) ne contient pas de un mot et un préfixe strict de ce mot, or dans le langage reconnu par la grammaire donnée on a *aacb* et *aacbbbb*.

Exercice 3 : Sélection d'instruction (3 points)

1. `mul(li(3), mul(li(4), x))`
2. $$\begin{aligned}\texttt{mul}(\texttt{li}(3), \texttt{mul}(\texttt{li}(4), x)) &\longrightarrow \texttt{mul}(\texttt{li}(3), \texttt{sll}(x, 2)) \\ &\longrightarrow \texttt{mul}(\texttt{li}(12), x)\end{aligned}$$

   La forme normale est donc `mul(li(12),x)`.
3. Oui, le nombre de symboles diminuent et le système est linéaire à droite (pas de duplication des variables).
4. Il faut calculer les paires critiques, qui sont
   - $(\texttt{sll}(e, n), \texttt{mul}(\texttt{li}(2^n), e))$ entre la première et la dernière règle.
   - $(\texttt{li}(0), \texttt{mul}(\texttt{li}(0), e))$ entre la deuxième et la dernière règle.
   - $(\texttt{sll}(\texttt{sll}(e, n_1), n_2), \texttt{mul}(\texttt{li}(2^{n_1+n_2}), e))$ entre la troisième et la dernière règle.

   Les deux premières paires critiques sont joignables : $\texttt{mul}(\texttt{li}(2^n), e) \longrightarrow \texttt{sll}(e, n)$ et $\texttt{mul}(\texttt{li}(0), e) \longrightarrow \texttt{li}(0)$. La dernière n'est pas joignable : la partie gauche ne peut pas se réécrire, la partie droite peut se réécrire en $\texttt{sll}(e, \underline{n_1 + n_2})$. Le système n'est donc pas confluent.

   Pour le rendre confluent, on peut rajouter la règle $\texttt{sll}(\texttt{sll}(e, \underline{n_1}), \underline{n_2}) \to \texttt{sll}(e, \underline{n_1 + n_2})$. Le système est alors toujours fortement terminant. Il y a deux nouvelles paires critiques :
   - $(\texttt{mul}(\texttt{li}(k * 2^{n_2}), \texttt{sll}(e, n_1)), \texttt{mul}(\texttt{li}(k), \texttt{sll}(e, n_1 + n_2)))$ entre la quatrième et la nouvelle ;
   - $(\texttt{sll}(\texttt{sll}(e, \underline{n_1 + n_2}), \underline{n_3}), \texttt{sll}(\texttt{sll}(e, \underline{n_1}), \underline{n_2 + n_3})$ entre la nouvelle et elle-même.

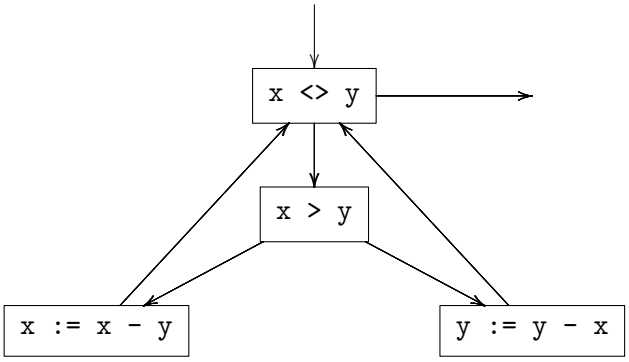
   Les deux sont joignables : les deux côtés de la première se réduisent sur $\texttt{mul}(\texttt{li}(k * 2^{n_1+n_2}), e)$, les deux côtés de la seconde sur $\texttt{sll}(e, n_1 + n_2 + n_3)$.

   Le système avec la nouvelle règle est donc confluent.

Exercice 4 : Graphe de flot de contrôle (2 points)

-

1.



2.
```
l1: sne %2, %0, %1 -> l2
l2: bgtz %2 -> l7, l3
l3: slt %3, %1, %0 -> l4
l4: bgtz %3 -> l5, l6
l5: sub %0, %0, %1 -> l1
l6: sub %1, %1, %0 -> l1
```

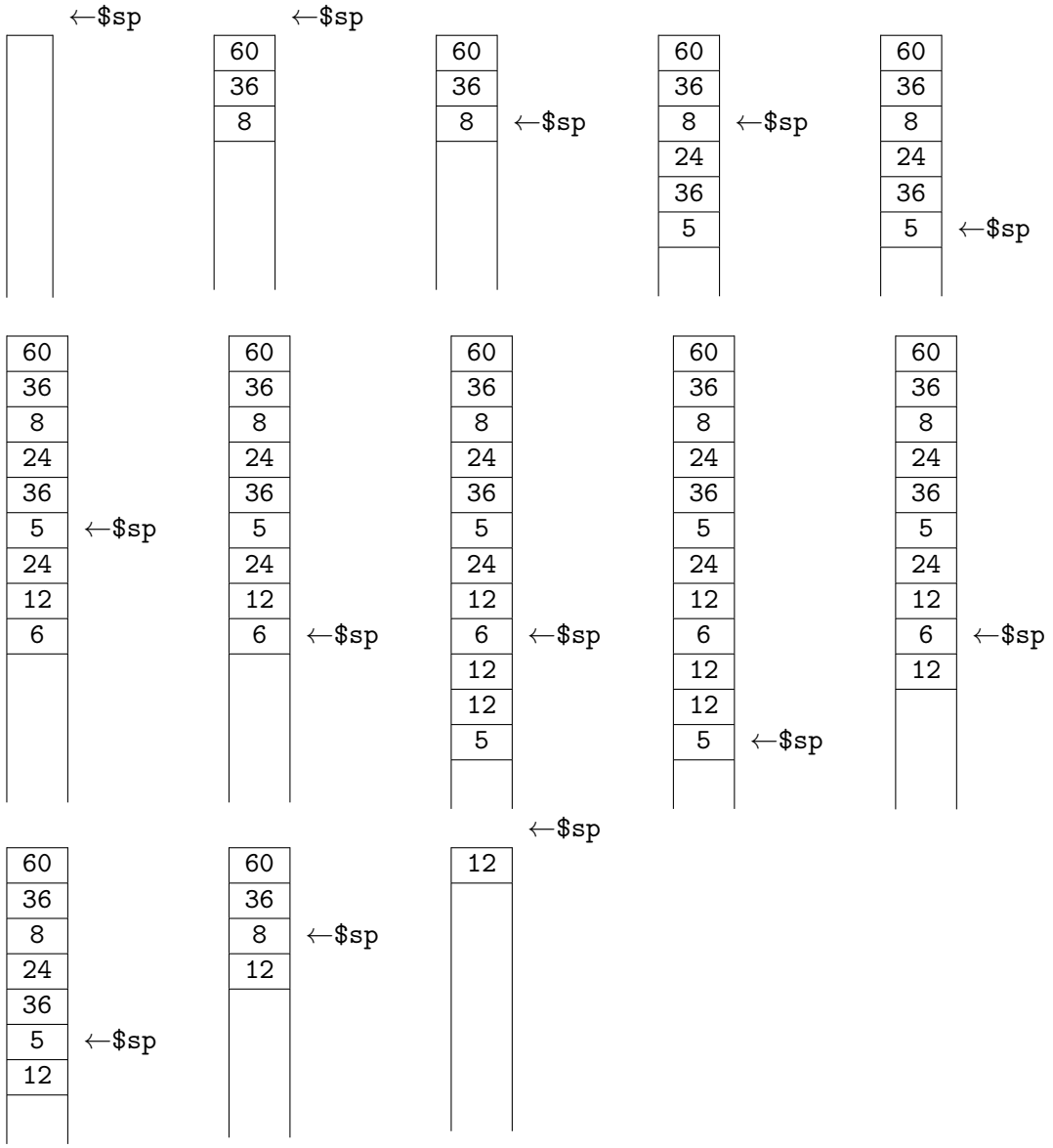
Exercice 5 : Convention d'appel (5 points)

Convention par pile

1.

x
y
adresse de retour

2.



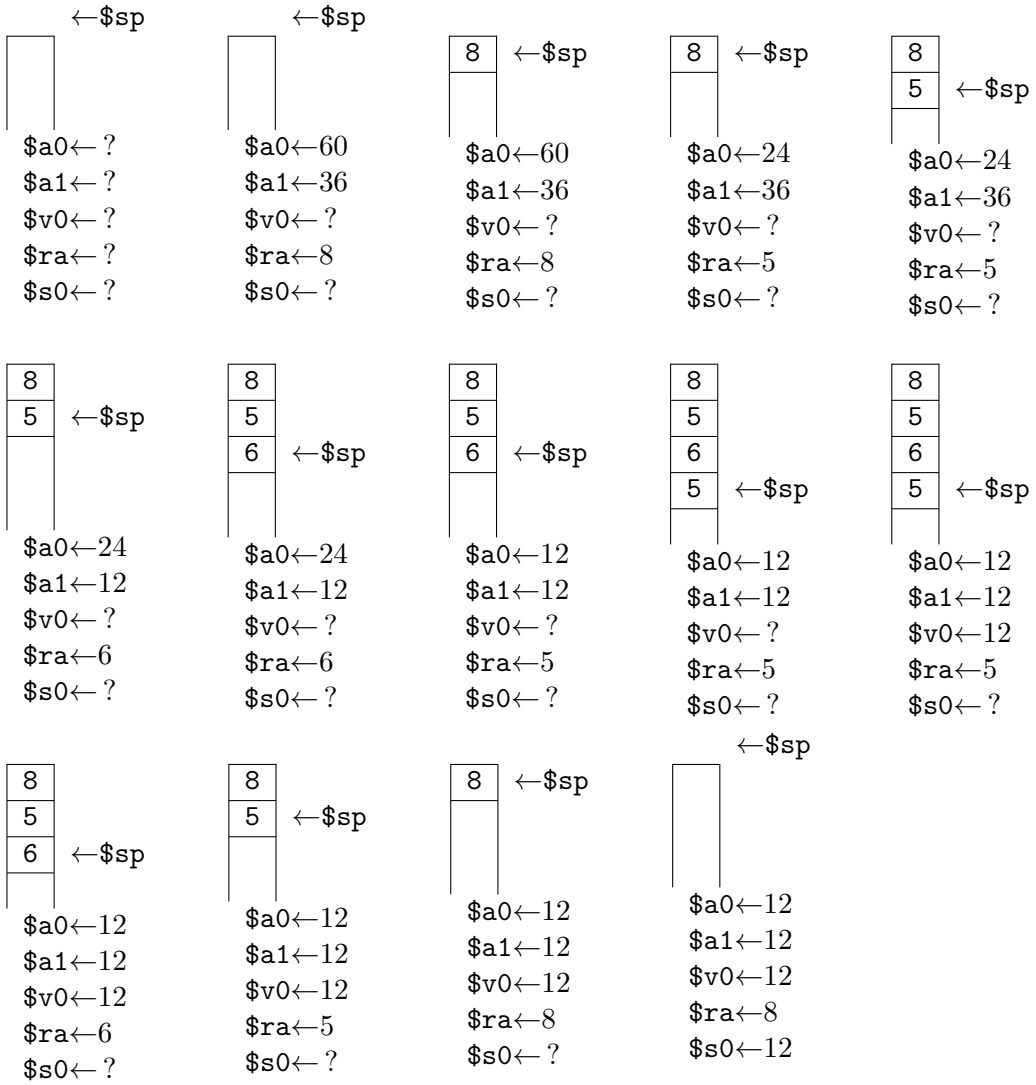
Convention d'appel MIPS

3. `pgcd` modifie le registre *callee-saved* `$ra` lors des appels récursifs, elle doit donc le sauvegarder. Les autres registres *callee-saved* ne sont pas modifiés, et ne doivent donc pas être sauvegardés.

   `pgcd` modifie les registres *caller-saved* `$a0` et `$a1` lors des appels récursifs, mais elle n'a plus besoin de leur valeur de départ après. Il n'est donc pas nécessaire de les sauvegarder. Les autres registres *caller-saved* ne sont pas modifiés.

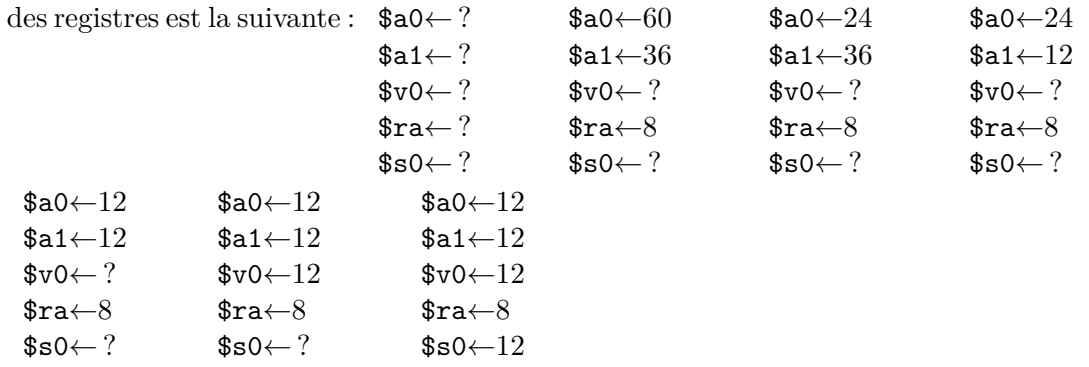
   La trame de `pgcd` est donc : | sauvegarde de `$ra` |

4.



5. Les appels récursifs, qui sont terminaux, se font alors à l'aide de `j` et non plus `jal`. La valeur de `$ra` n'est donc pas modifiée ; il n'est plus nécessaire de la sauvegarder.

   La pile reste donc vide durant l'exécution du programme. L'évolution du contenu des registres est la suivante :

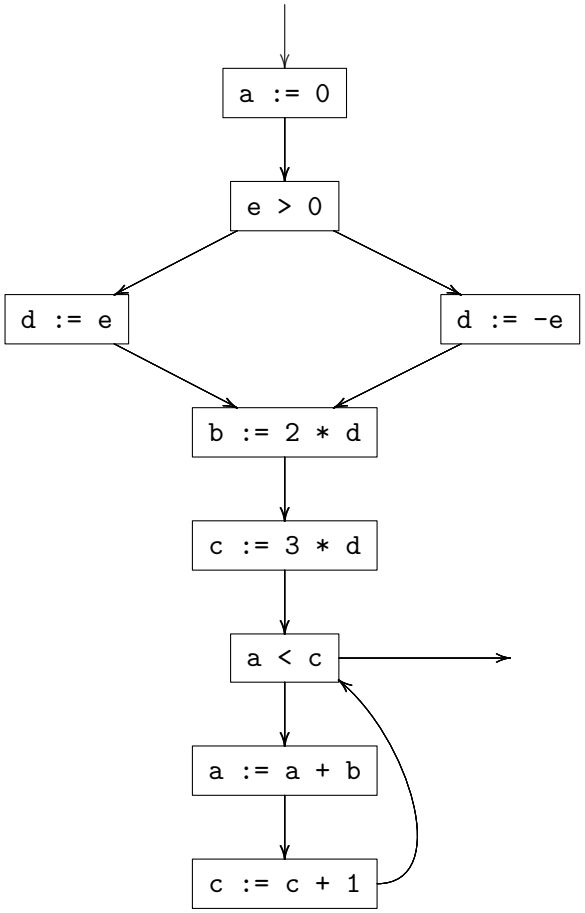


`pgcd` est alors équivalente à la fonction impérative :

```
function pgcd(x : integer, y : integer)
begin
  while x <> y do
    if x > y
    then x := x - y
    else y := y - x;
  pgcd := x
end
```

Exercice 6 : Allocation de registres (4 points)

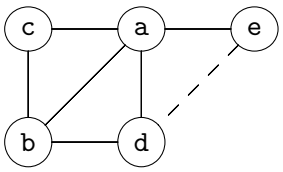
1.



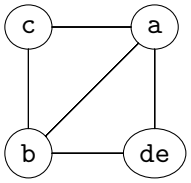
2.

point	vivantes après	vivantes avant
1	a e	e
2	a e	a e
3	a d	a e
4	a d	a e
5	a b d	a d
6	a b c	a b d
7	a b c	a b c
8	a b c	a b c
9	a b c	a b c

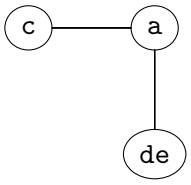
3.



4. On ne peut pas simplifier, car le seul nœud trivialement colorable, `e`, possède une arête de préférence.
   On peut fusionner `e` et `d` en utilisant le critère de George (les voisins non trivialement colorable de `e` sont inclus dans ceux de `d`), celui de Briggs ne marche pas ici.



   Aucun nœud n'est trivialement colorable, et il n'y a pas d'arête de préférence. On spille par exemple `b`.



   On peut simplifier `c`, puis `de`, puis `a`.
   En remontant, on attribue la première couleur à `a`, la deuxième à `de` et `c`. On voit qu'il n'est pas possible d'attribuer une couleur à `b`, on n'est donc vraiment obliger de la spiller. On obtient donc au final le coloriage :