

# Compilation EISE4 – TD

## Exercice 1 : Grammaire Hors-Contexte et Ambiguïté

Soit la grammaire suivante :

$G_1 = \langle \{a, b\}, \{S\}, S, R \rangle$  et  $R : S \rightarrow aSb|aS|\varepsilon$

- Quel est le langage engendré par la grammaire ?
- Montrer que cette grammaire est ambiguë
- Donner une grammaire équivalente non-ambiguë

Mêmes questions avec la grammaire  $G_2 = \langle \{a, b\}, \{S\}, S, R \rangle$  et  $R : S \rightarrow SaSaS|bS|\varepsilon$

## Exercice 2 : Analyse syntaxique

Soit le programme yacc suivant, qui permet de reconnaître les mots du langage  $x^nay^n$  :

```
1 A : 'x' A 'y'      { printf("A -> x A y\n"); }
2 | 'a'                 { printf("A -> a\n"); }
3 ;
```

- Qu'affiche ce programme appliqué à la chaîne d'entrée `xxayy` ?
- Afficher les états successifs de la pile de yacc pour la chaîne `xxayy`.

Soit le langage  $ab^*c$  engendré par la grammaire :

```
1 A : 'a' B
2 ;
3 B : 'b' B
4 | 'c'
5 ;
```

- Dessiner les états successifs de la pile de yacc pour la chaîne d'entrée `abbbc`
- Quel problème cela peut-il poser ? Donner une grammaire équivalente qui résout le problème.

## Exercice 3 : MiniC

Soit le programme MiniC suivant :

```
1 int a = 1, b;
2 bool c = true;
3 void main() {
4 }
```

- Dessiner l'arbre de ce programme à la fin de l'analyse syntaxique, en numérotant les noeuds selon l'ordre de leur création
- Dessiner l'arbre de dérivation correspondant à ce programme dans la grammaire hors-contexte de MiniC (règles qui sont prise depuis l'axiome pour arriver à ce programme)
- Pour chaque noeud de l'arbre du programme, indiquer dans l'arbre de dérivation la réduction à l'origine de la création du noeud
- Donner le code assembleur mips correspondant à ce programme

$$S \rightarrow @Sb$$

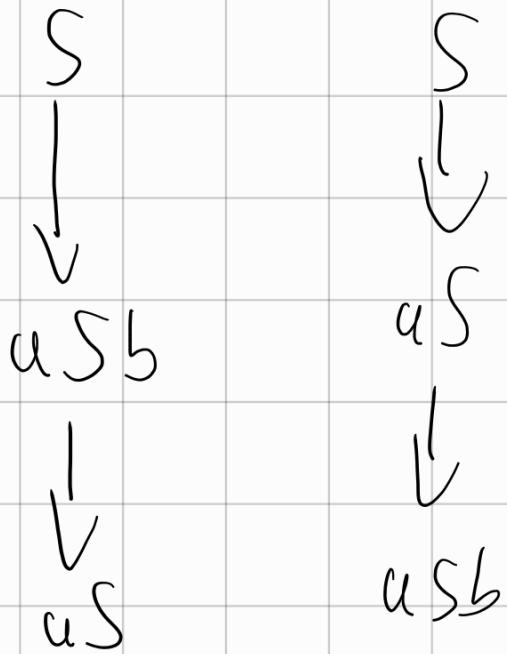
$$S \rightarrow @S$$

$$S \rightarrow \varepsilon$$

$$m * / m$$

ü ü 5

$a \in b$



Le langage est donc ambigu

$a_1' = \langle \{a, b\}, S, T \rangle$  et  $R : S \rightarrow a \setminus b$

↑

## Von ambiguel

$$\begin{array}{ccc} \overline{T} & \rightarrow & \alpha \overline{T} \\ \overline{I} & \rightarrow & \epsilon \end{array}$$

G2

$S \rightarrow S_a S_a S$

$$\left( \begin{matrix} b^4 & a & b^4 & a & b^4 \\ \end{matrix} \right)^*$$

$$S \rightarrow bS$$

5 → 6

b a a



$G_2' \quad \{S; T\}$

$$S \rightarrow bS$$

$$S \rightarrow TaTaS$$

$$S \rightarrow Sg$$

$$T \rightarrow bT$$

$$T \rightarrow \epsilon$$

### Exercice 2 : Analyse syntaxique

Soit le programme yacc suivant, qui permet de reconnaître les mots du langage  $x^nay^n$  :

```

1 A : 'x' A 'y'      { printf("A -> x A y\n"); }
2 | 'a'               { printf("A -> a\n"); }
3 ;

```

- Qu'affiche ce programme appliqué à la chaîne d'entrée **xxayy** ?
- Afficher les états successifs de la pile de yacc pour la chaîne **xxayy**.

$x \ x \ a \ y \ y$

fond

cle pile

$x \ x \ x \ x \ x \ x \ x \ / A \ y \ A \ y \ A \ A$

Shift Shift

A  $\hookrightarrow$  a

$A \rightarrow X A y$

$$A \rightarrow X A y$$

Soit le langage  $ab^*c$  engendré par la grammaire :

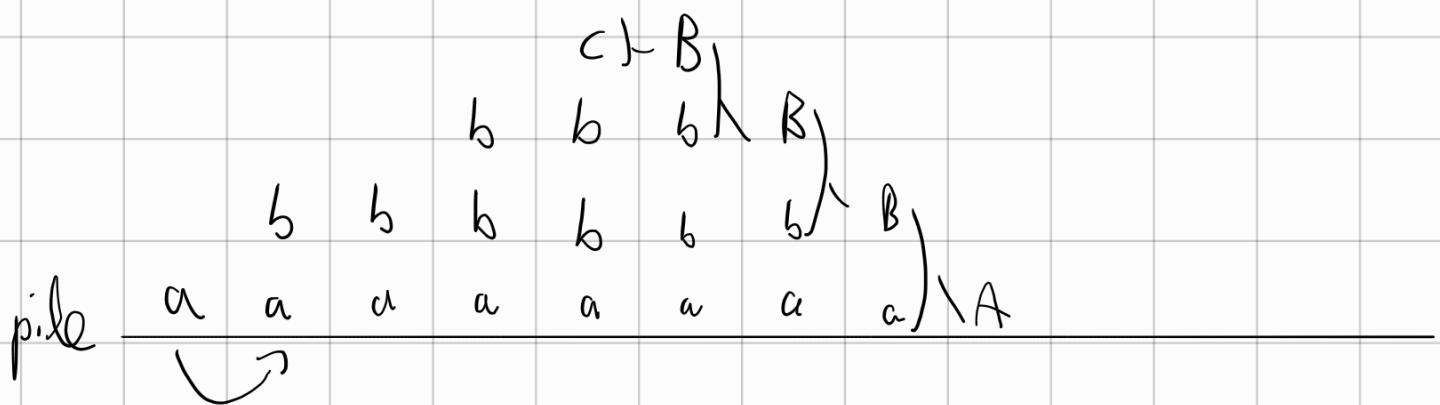
```

1 A : 'a' B
2 ;
3 B : 'b' B
4 | 'c'
5 ;

```

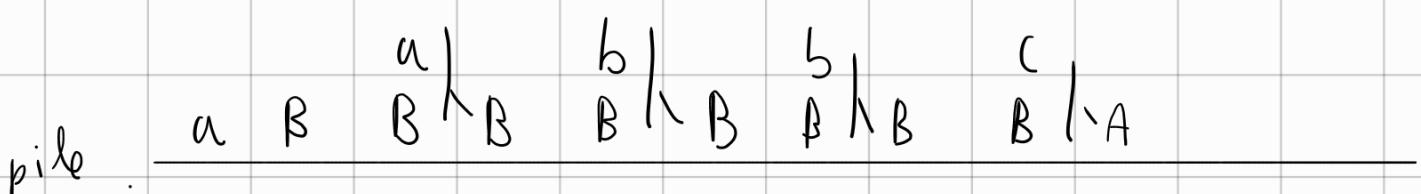
- Dessiner les états successifs de la pile de yacc pour la chaîne d'entrée **abbbc**
  - Quel problème cela peut-il poser ? Donner une grammaire équivalente qui résout le problème.

abbbc



Il faut attendre le c pour faire la réduction.

A : B' C'  
B : B' b'  
C : c'

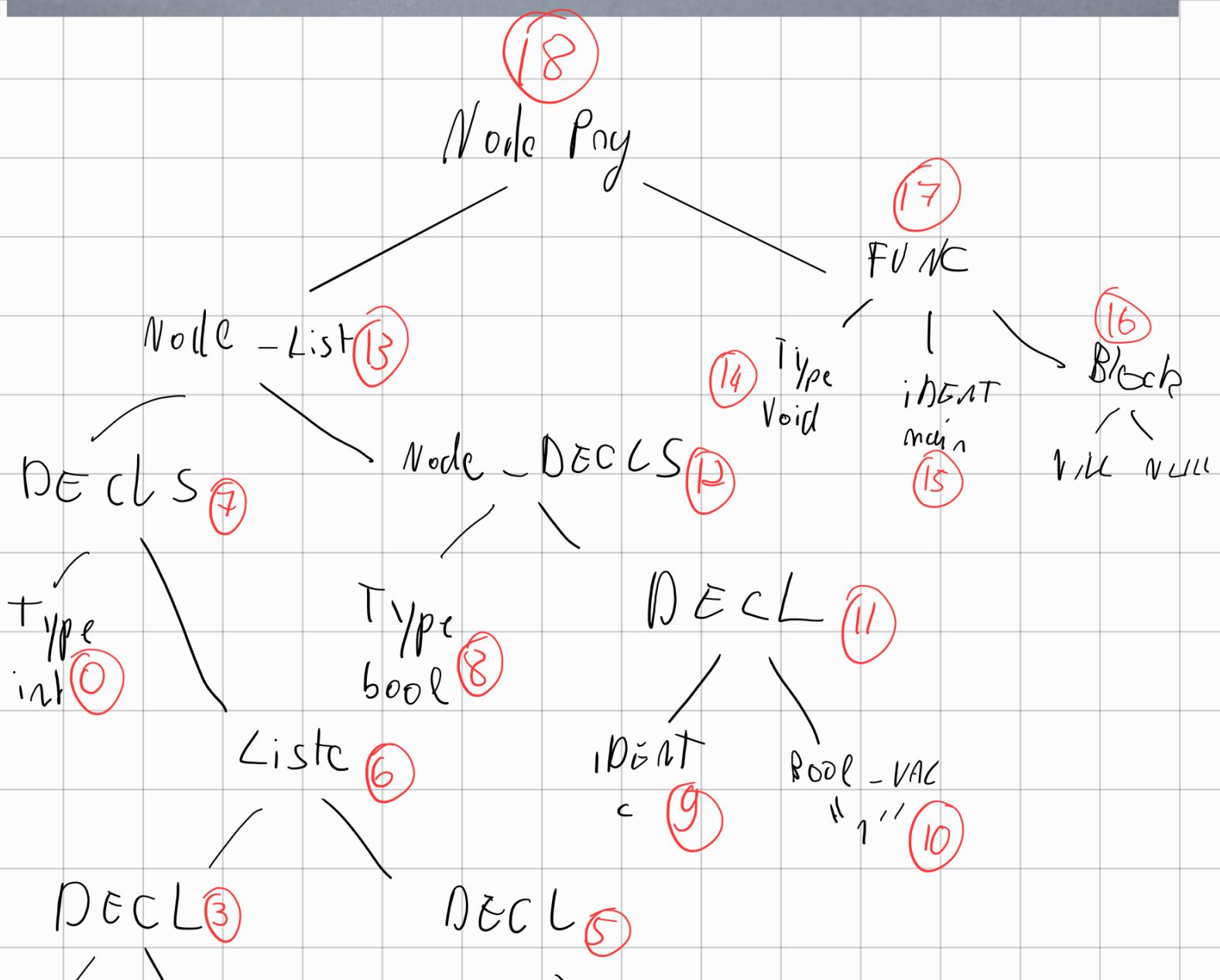


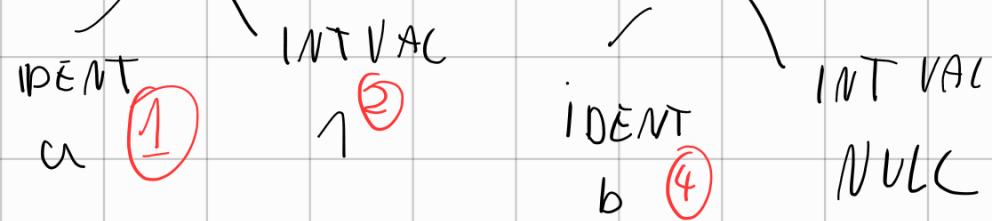
### Exercice 3 : MiniC

Soit le programme MiniC suivant :

```
1 int a = 1, b;  
2 bool c = true;  
3 void main() {  
4 }
```

- Dessiner l'arbre de ce programme à la fin de l'analyse syntaxique, en numérotant les noeuds selon l'ordre de leur création
  - Dessiner l'arbre de dérivation correspondant à ce programme dans la grammaire hors-contexte de MiniC (règles qui sont prise depuis l'axiome pour arriver à ce programme)
  - Pour chaque noeud de l'arbre du programme, indiquer dans l'arbre de dérivation la réduction à l'origine de la création du noeud
  - Donner le code assembleur mips correspondant à ce programme





0: \$\$ = create-node-type(...)

3: \$\$ = create-node-decl(..., \$1, \$3)

program

List decl<sub>nonnull</sub>

List decl<sub>nonnull</sub> vndecl

vndecl

type list type decl ScndOn

int List type decl COMA decl  
 ident  
 ||decl  
 |  
 IPENT

ident Affect tps

IPENT  
 "n"

INTVAL  
 n

• decl

a : word 1

b : .word 0

c : .word 1

.text

main:

addiu \$29,\$29,0

addiu \$29,\$29,0

ori \$2,\$0,10

Syscall

#### Exercice 4 : MiniC

Soit le programme MiniC suivant :

```
1 void main() {
2     int a = 120, b = 80;
3     if (a > b) {
4         a = a - b;
5     }
6     print("a = ", a, " - b = ", b);
7 }
```

- Dessiner l'arbre de ce programme après la première passe
- Donner toutes les conditions, explicites ou implicites, vérifiées par la grammaire attribuée MiniC pour ce programme
- Dessiner l'état de la pile au moment du **if**
- Donner le code assembleur mips correspondant à ce programme

#### Exercice 5 : Grammaire attribuée de MiniC

Donner, pour chacune des règles suivantes de la grammaire attribuée, un programme MiniC minimal ne vérifiant pas la condition de la règle :

- 1.8
- 1.12 (partie de la condition :  $type = type_1$ )
- 1.20
- 1.61

#### Exercice 6 : Génération de code assembleur Mips

Soit le programme MiniC suivant :

```
1 void main() {
2     int a = 136;
3     int b = 80;
4     while (a != b) {
5         if (a > b) {
6             a = a - b;
7         }
8     }
9 }
```

```

8     else {
9         b = b - a;
10    }
11 }
12 print(a);
13 }

```

- Que calcule ce programme ?
- Écrivez un programme assembleur mips correspondant

#### Exercice 4 : MiniC

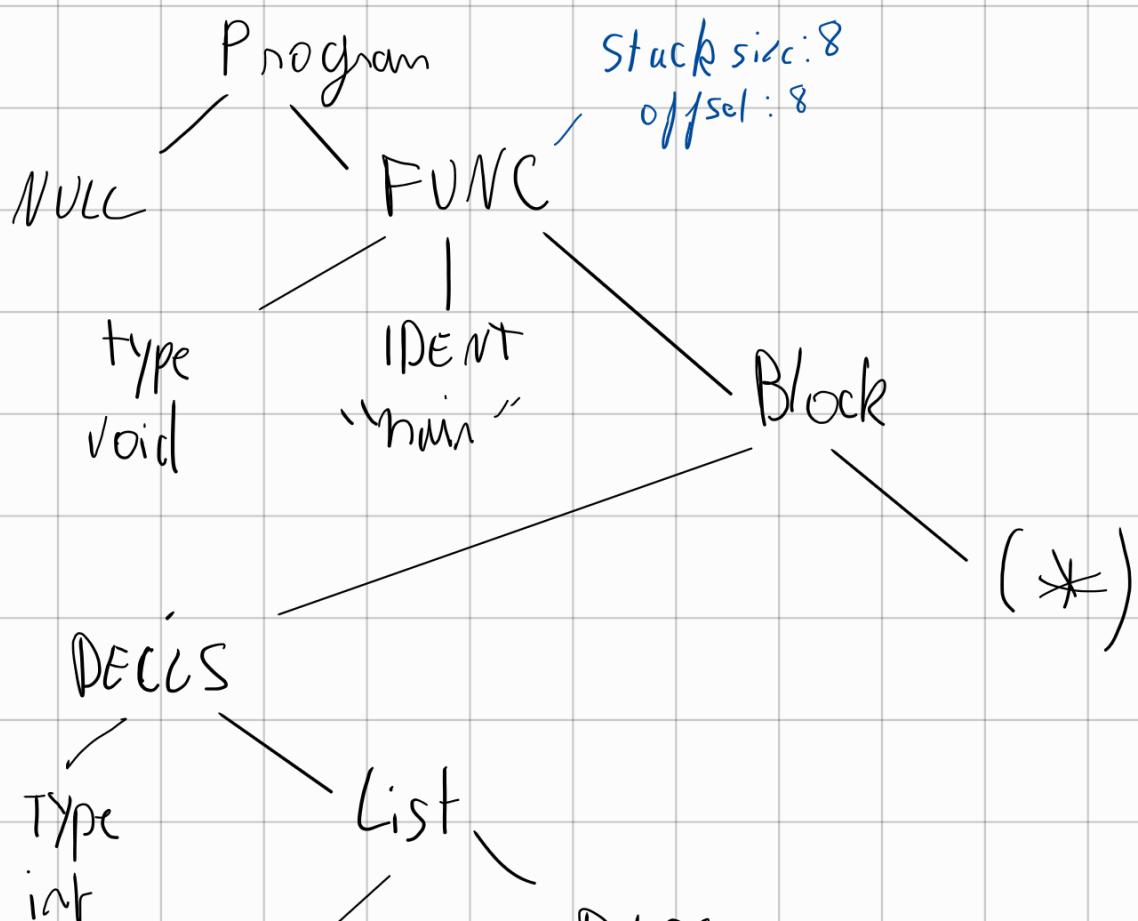
Soit le programme MiniC suivant :

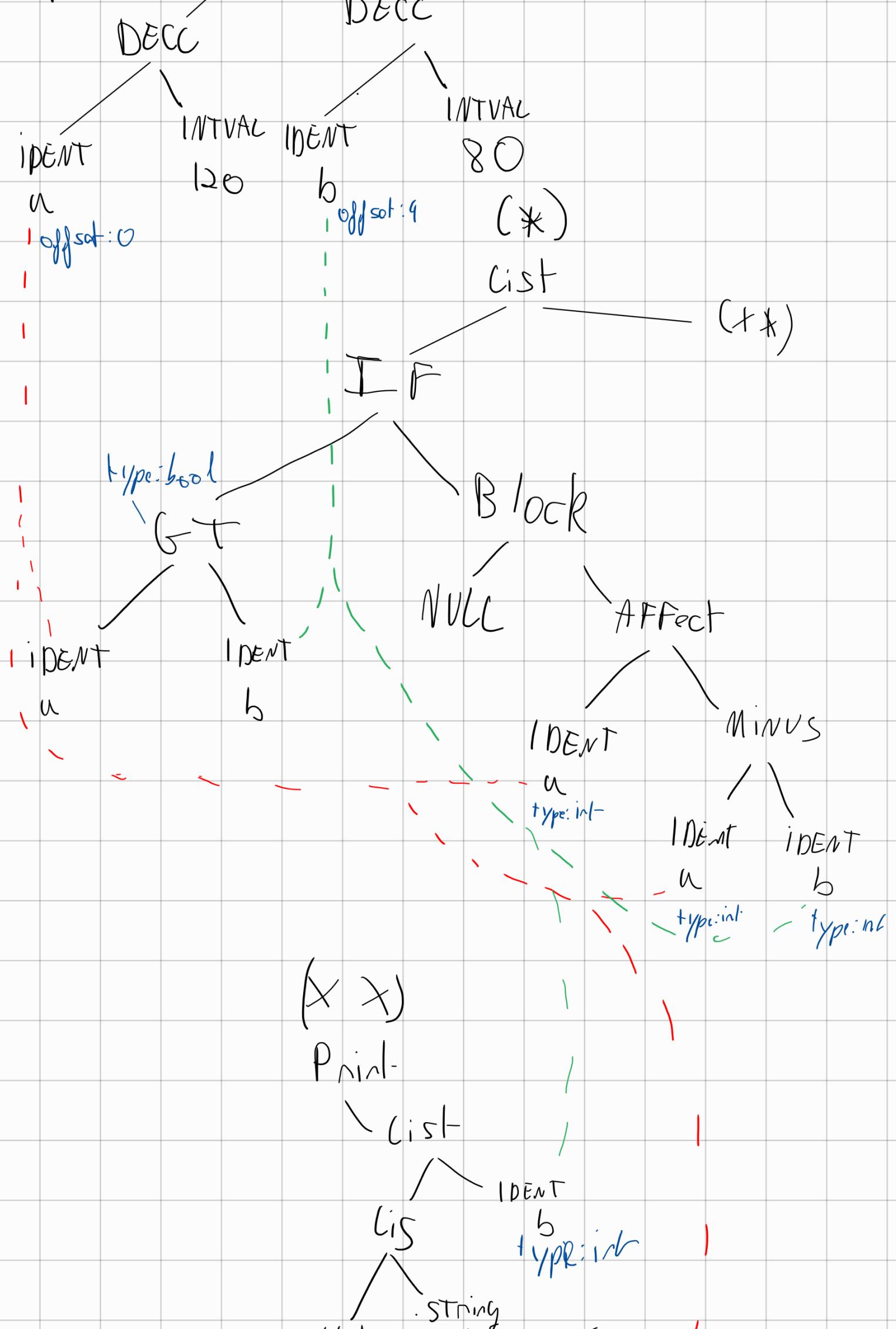
```

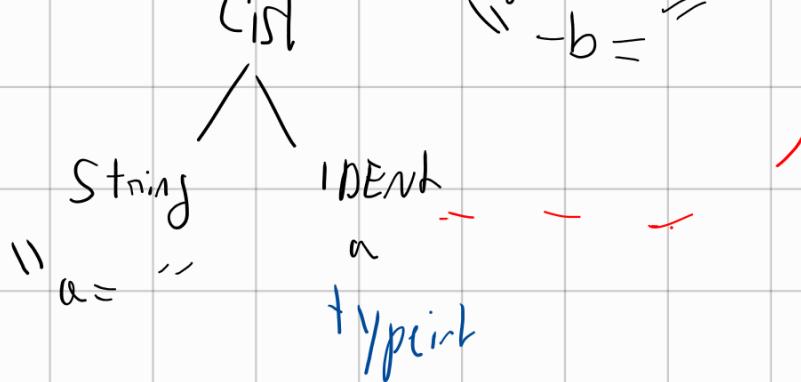
1 void main() {
2     int a = 120, b = 80;
3     if (a > b) {
4         a = a - b;
5     }
6     print("a = ", a, " - b = ", b);
7 }

```

- Dessiner l'arbre de ce programme après la première passe
- Donner toutes les conditions, explicites ou implicites, vérifiées par la grammaire attribuée de MiniC pour ce programme
- Dessiner l'état de la pile au moment du if
- Donner le code assembleur mips correspondant à ce programme



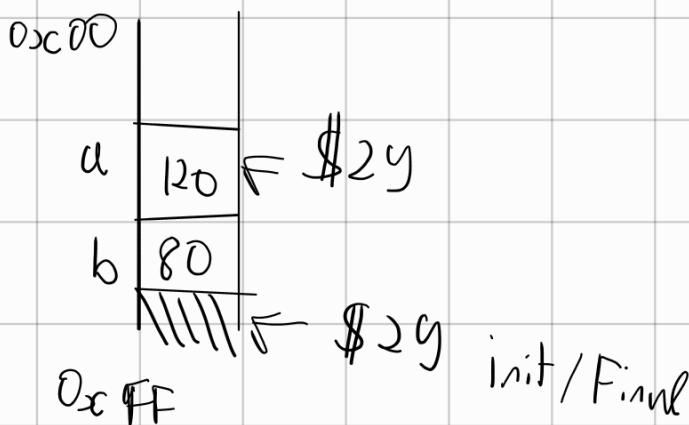




Conditions vérifiées par la passe 1

poly page 22

- Règle 1.5: condition C type = void & nom = "hair"
  - 1.8: ( $x$ ) type  $\neq$  void
  - 1.13: condition affectation
    - $\rightarrow$  rule | nom := exp + ctx<sub>0</sub> / env | type<sub>1</sub>
    - global = false et nom  $\notin$  dom(ctx<sub>0</sub>) et type = type<sub>1</sub>
    - ctx := ctx<sub>0</sub>  $\cup$  {nom  $\mapsto$  type}
  - 1.18: condition type bool
  - 1.30: implicite type op binaire
  - 1.32: type de l'exp = type de la var
  - 1.67: ( $x$ ) nom  $\in$  dom(env)



### Exercice 5 : Grammaire attribuée de MiniC

Donner, pour chacune des règles suivantes de la grammaire attribuée, un programme MiniC minimal ne vérifiant pas la condition de la règle :

- 1.8

- 1.12 (partie de la condition :  $type = type_1$ )
- 1.20
- 1.61

1.8

```
void main () {  
    void a = 15;  
}
```

1.12

```
int a = false;  
void main () {  
}
```

}

1.20

```
void main () {  
    while ( 0 ) {}  
}
```

1.61

```
void main ();  
a;  
}
```

