(a.1) Suppression des variables non productives

1) Au départ $V_0 = \{\ \}$

2)

$\langle ExprArith \rangle \rightarrow [VarName]$

$\langle Op \rangle \rightarrow +$

$\langle BinOp \rangle \rightarrow \text{and}$

$\langle Comp \rangle \rightarrow =$

$\langle Print \rangle \rightarrow print([VarName])$

$\langle Read \rangle \rightarrow read([VarName])$

$V_1 \leftarrow \{\langle ExprArith \rangle, \langle Op \rangle, \langle BinOp \rangle, \langle Comp \rangle \langle Print \rangle \langle Read \rangle\} \cup V_0$

3)

$\langle Instruction \rangle \rightarrow \langle Print \rangle$

$\langle Assign \rangle \rightarrow [VarName] := \langle ExprArith \rangle$

$\langle SimpleCond \rangle \rightarrow \langle ExprArith \rangle \langle Comp \rangle \langle ExprArith \rangle$

$V_2 \leftarrow \{\langle Instruction \rangle, \langle Assign \rangle, \langle SimpleCond \rangle\} \cup V_1$

4)

$\langle InstList \rangle \rightarrow \langle Instruction \rangle$

$\langle Cond \rangle \rightarrow \langle SimpleCond \rangle$

$V_3 \leftarrow \{\langle InstList \rangle, \langle Cond \rangle\} \cup V_2$

5)

$\langle Code \rangle \rightarrow \langle InstList \rangle$

$V_4 \leftarrow \{\langle Code \rangle\} \cup V_3$

6)

$\langle Program \rangle \rightarrow begin \langle Code \rangle end$

$\langle If \rangle \rightarrow if \langle Cond \rangle then \langle Code \rangle fi$

$\langle While \rangle \rightarrow while \langle Cond \rangle do \langle Code \rangle od$

$\langle For \rangle \rightarrow for [VarName] \text{ from } \langle ExprArith \rangle by \langle ExprArith \rangle \text{ to } \langle ExprArith \rangle do \langle Code \rangle od$

$V_5 \leftarrow \{\langle Program \rangle, \langle If \rangle, \langle While \rangle, \langle For \rangle\} \cup V_4$

7)

$V_6 \leftarrow \{\langle Program \rangle, \langle If \rangle, \langle While \rangle, \langle For \rangle, \langle Code \rangle, \langle InstList \rangle, \langle Cond \rangle, \langle Instruction \rangle, \langle Assign \rangle,$
$\langle SimpleCond \rangle, \langle ExprArith \rangle, \langle Op \rangle, \langle BinOp \rangle, \langle Comp \rangle \langle Print \rangle \langle Read \rangle\}$

En ayant $V_5 = V_6$, La boucle s'arrête.

8) $V' \leftarrow V_6$

$P' =$ L'ensemble des règles de P qui ne contiennent pas des variables dans V \ V′

Ici, $V' = V$ il n'y a plus rien a enlevé, $P' = P$ and $G' = G$

(a.2) Suppression des variables inacessible

1) Au départ $V_0 \leftarrow \{\langle Program \rangle\}$

2) $\langle Program \rangle \rightarrow begin \langle Code \rangle end, V_1 \leftarrow \{begin, \langle Code \rangle, end\} \cup V_0$

3) $\langle Code \rangle \rightarrow \langle InstList \rangle, V_2 \leftarrow \{\langle InstList \rangle\} \cup V_1$

4) $\langle InstList \rangle \rightarrow \langle Instruction \rangle ; \langle InstList \rangle, V_3 \leftarrow \{\langle Instruction \rangle, ;\} \cup V_2$

5) $\langle Instruction \rangle \rightarrow \langle Assign \rangle | \langle If \rangle | \langle While \rangle | \langle For \rangle | \langle Print \rangle | \langle Read \rangle,$
$V_4 \leftarrow \{\langle Assign \rangle, \langle If \rangle, \langle While \rangle, \langle For \rangle, \langle Print \rangle, \langle Read \rangle\} \cup V_3$

6) $\langle Assign\rangle \to [VarName] := \langle ExprArith\rangle,\ V_5 \leftarrow \{[VarName], :=, \langle ExprArith\rangle\} \cup V_4$

7) $\langle ExprArith\rangle \to [Number] \mid (\langle ExprArith\rangle) \mid -\langle ExprArith\rangle \mid \langle ExprArith\rangle \langle Op\rangle \langle ExprArith\rangle$,
$V_6 \leftarrow \{[Number], (,), -, \langle Op\rangle\} \cup V_5$

8) $\langle Op\rangle \to + \mid - \mid * \mid / ,\ V_7 \leftarrow \{+, -, *, /\} \cup V_6$

9) $\langle If\rangle \to if \langle Cond\rangle then \langle Code\rangle else \langle Code\rangle fi,\ V_7 \leftarrow \{if, \langle Cond\rangle, then, else, fi\} \cup V_6$

10) $\langle Cond\rangle \to \langle Cond\rangle \langle BinOp\rangle \langle Cond\rangle \mid not \langle SimpleCond\rangle,\ V_8 \leftarrow \{\langle BinOp\rangle, not, \langle SimpleCond\rangle\} \cup V_7$

11) $\langle SimpleCond\rangle \to \langle ExprArith\rangle \langle Comp\rangle \langle ExprArith\rangle,\ V_9 \leftarrow \{\langle Comp\rangle\} \cup V_8$

12) $\langle BinOp\rangle \to \text{and} \mid \text{or},\ V_{10} \leftarrow \{\text{and}, \text{or}\} \cup V_9$

13) $\langle Comp\rangle \to = \mid >= \mid > \mid <= \mid < \mid /=,\ V_{11} \leftarrow \{=, >=, >, <=, <, /=\} \cup V_{10}$

14) $\langle While\rangle \to while \langle Cond\rangle do \langle Code\rangle od,\ V_{12} \leftarrow \{while, do, od\} \cup V_{11}$

15) $\langle For\rangle \to for [VarName] \text{ from } \langle ExprArith\rangle \text{ by } \langle ExprArith\rangle \text{ to } \langle ExprArith\rangle \text{ do } \langle Code\rangle od,$
$V_{13} \leftarrow \{for, \text{from}, by, \text{to}\} \cup V_{12}$

16) $\langle Print\rangle \to print([VarName]),\ V_{14} \leftarrow \{print\} \cup V_{13}$

17) $\langle Read\rangle \to read([VarName]),\ V_{15} \leftarrow \{read\} \cup V_{14}$

18) $V_{16} \leftarrow \{\langle Program\rangle, begin, \langle Code\rangle, end, \langle InstList\rangle, \langle Instruction\rangle, ;, \langle Assign\rangle, \langle If\rangle, \langle While\rangle$
$\langle For\rangle, \langle Print\rangle, \langle Read\rangle, [VarName], :=, \langle ExprArith\rangle, [Number], (,), -, \langle Op\rangle, +, -, *, /, if$
$\langle Cond\rangle, then, else, fi, \langle BinOp\rangle, not, \langle SimpleCond\rangle, \langle Comp\rangle, \text{and}, \text{or}, =, >=, >, <=, <, /=, while$
$do, od, for, \text{from}, by, \text{to}, print, read\}$

Having $V_{16} = V_{15}$, the loop stops.

19) $V' \leftarrow V_{16} \cap V = V;\ T' \leftarrow V_{16} \cap T = T$
$P' <-$ L'ensemble des règles de P qui contiennent seulements des variables de V_{16}

Ici $V_{16} = V$ vu que $P' = P$ on a $G' = G$.

(b.1) Suppression de la récursivité à gauche

Dans cette grammaire, il y a quatre cas de récursivité à gauche :

$\langle ExprArith\rangle \to \langle ExprArith\rangle \langle Op'\rangle \langle ExprArith\rangle \mid \langle ExprArith\,0\rangle$
$\langle ExprArith\,0\rangle \to \langle ExprArith\,0\rangle \langle Op''\rangle \langle ExprArith\rangle \mid \langle ExprArith\,1\rangle$
$\langle Cond\rangle \to \langle Cond\rangle \langle BinOp'\rangle \langle Cond\rangle \mid \langle Cond\,0\rangle$
$\langle Cond\,0\rangle \to \langle Cond\,0\rangle \langle BinOp''\rangle \langle Cond\rangle \mid \langle Cond\,1\rangle$

Pour le première cas, introduisons une nouvelle variable $\langle ExprArith'\rangle$ et supprimons la récursivité:
$\langle ExprArith'\rangle \to \langle Op'\rangle \langle ExprArith\rangle \langle ExprArith'\rangle \mid \varepsilon$
$\langle ExprArith\rangle \to \langle ExprArith\,0\rangle \langle ExprArith'\rangle$

Pour le deuxième cas, introduisons une nouvelle variable $\langle ExprArith\,0'\rangle$ et supprimons la récursivité :
$\langle ExprArith\,0'\rangle \to \langle Op''\rangle \langle ExprArith\rangle \langle ExprArith\,0'\rangle \mid \varepsilon$
$\langle ExprArith\,0\rangle \to \langle ExprArith\,1\rangle \langle ExprArith\,0'\rangle$

Pour le troisième cas, introduisons une nouvelle variable $\langle Cond'\rangle$ et supprimons la récursivité :
$\langle Cond'\rangle \to \langle BinOp'\rangle \langle Cond\rangle \langle Cond'\rangle \mid \varepsilon$
$\langle Cond\rangle \to \langle Cond\,0\rangle \langle Cond'\rangle$

Pour le quatrième cas , introduisons une nouvelle variable $\langle Cond\,0'\rangle$ and remove the left-recursion:

$\langle Cond\,0'\rangle \rightarrow \langle BinOp''\rangle\langle Cond\rangle\langle Cond\,0'\rangle | \varepsilon$

$\langle Cond\,0\rangle \rightarrow \langle Cond\,1\rangle\langle Cond\,0'\rangle$

(b.2) Factorisation

Dans cette grammaire, il y a seulement un type de règles avec un préfixe commun

$\langle If\rangle \rightarrow if\langle Cond\rangle then\langle Code\rangle fi$

$\langle If\rangle \rightarrow if\langle Cond\rangle then\langle Code\rangle else\langle Code\rangle fi$

Pour ce premier cas, introduisons une nouvelle variable $\langle If'\rangle$ et factorisons là

$\langle If\rangle \rightarrow if\langle Cond\rangle then\langle Code\rangle\langle If'\rangle$

$\langle If'\rangle \rightarrow fi$

$\langle If'\rangle \rightarrow else\langle Code\rangle fi$

(c) Rendre la gammaire non ambigüe

Pour la règle $\langle ExprArith\rangle$, il faut appliquer les priorités suivante:

$-x$

$*\,/$

$+-$

$[VarName][Number](x)$

Réécrivons la règle $\langle Op\rangle$:

$\langle Op'\rangle \rightarrow +|-$

$\langle Op''\rangle \rightarrow *|/$

`Ensuite réécrivons les règle` $\langle ExprArith\rangle$ `s:`

$\langle ExprArith\rangle \rightarrow \langle ExprArith\rangle\langle Op'\rangle\langle ExprArith\rangle | \langle ExprArith\,0\rangle$

$\langle ExprArith\,0\rangle \rightarrow \langle ExprArith\,0\rangle\langle Op''\rangle\langle ExprArith\rangle | \langle ExprArith\,1\rangle$

$\langle ExprArith\,1\rangle \rightarrow -\langle ExprArith\rangle | [VarName] \vee [Number] \vee (\langle ExprArith\rangle)$

Pour la règle $\langle Cond\rangle$, il faut appliquer les priorités suivante

$not\,x$

$> < >= <= = /=$

and

or

Réécrivons la règle $\langle BinOp\rangle$:

$\langle BinOp'\rangle \rightarrow$ or

$\langle BinOp''\rangle \rightarrow$ and

`Pour finir, réécrivons les règles` $\langle Cond\rangle$ `:`

$\langle Cond\rangle \rightarrow \langle Cond\rangle\langle BinOp'\rangle\langle Cond\rangle | \langle Cond\,0\rangle$

$\langle Cond\,0\rangle \rightarrow \langle Cond\,0\rangle\langle BinOp''\rangle\langle Cond\rangle | \langle Cond\,1\rangle$

$\langle Cond\,1\rangle \rightarrow not\,\langle SimpleCond\rangle | \langle SimpleCond\rangle$