siunitx

On change les règles liés aux opérations pour intégrer dans ces règles l'associativité à gauche et la priorité des multiplication et division.

On change les règles :

$$
\begin{aligned}
< expr > \quad & \rightarrow < entier > \\
& \rightarrow < caractère > \\
& \rightarrow \text{true} \\
& \rightarrow \text{false} \\
& \rightarrow \text{null} \\
& \rightarrow (< expr >) \\
& \rightarrow < acces > \\
& \rightarrow < expr > < operateur > < expr > \\
& \rightarrow \text{not} < expr > \\
& \rightarrow \text{-} < expr > \\
& \rightarrow \text{new} < ident > \\
& \rightarrow < ident > (< expr >_{,}^{+}) \\
& \rightarrow \text{character ' val} (< expr >)
\end{aligned}
$$

$$
\begin{aligned}
< operateur > \rightarrow & = \\
& \rightarrow \ / = \\
& \rightarrow < \\
& \rightarrow <= \\
& \rightarrow > \\
& \rightarrow >= \\
& \rightarrow \ + \\
& \rightarrow \ - \\
& \rightarrow \ * \\
& \rightarrow \ / \\
& \rightarrow \ rem
\end{aligned}
$$

Par :

$$
\begin{aligned}
< expr > &\rightarrow T + < expr > \\
&\rightarrow T - < expr > \\
&\rightarrow \mathrm{T}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{T} &\rightarrow \mathrm{I} * \mathrm{T} \\
&\rightarrow \mathrm{I} \ / \ \mathrm{T} \\
&\rightarrow \mathrm{I}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{I} &\rightarrow \mathrm{F} = \mathrm{I} \\
&\rightarrow \mathrm{F} \ /= \mathrm{I} \\
&\rightarrow F < I \\
&\rightarrow F <= I \\
&\rightarrow F > I \\
&\rightarrow F >= I \\
&\rightarrow \mathrm{F\ rem\ I} \\
&\rightarrow \mathrm{I}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{F} &\rightarrow \mathrm{P} \\
&\rightarrow \text{-P} \\
&\rightarrow \mathrm{not\ \ P}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{P} &\rightarrow < entier > \\
&\rightarrow < caractere > \\
&\rightarrow true \\
&\rightarrow false \\
&\rightarrow null \\
&\rightarrow < acces > \\
&\rightarrow new < ident > \\
&\rightarrow < ident > (< expr >_{,}^{+}) \\
&\rightarrow character\,' val\ (< expr >) \\
&\rightarrow (< expr >)
\end{aligned}
$$

En factorisant, on obtient.

$$
\begin{array}{ll}
< expr > & \rightarrow T < expr >_1 \\
 & \\
< expr >_1 & \rightarrow + < expr > \\
 & \rightarrow - < expr > \\
 & \rightarrow \wedge \\
 & \\
\mathrm{T} & \rightarrow \mathrm{I}\ T_1 \\
 & \\
T_1 & \rightarrow * \ \mathrm{T} \\
 & \rightarrow / \ \mathrm{T} \\
 & \rightarrow \wedge \\
 & \\
\mathrm{I} & \rightarrow \mathrm{F}\ I_1 \\
 & \\
I_1 & \rightarrow = \mathrm{I} \\
 & \rightarrow /= \mathrm{I} \\
 & \rightarrow < I \\
 & \rightarrow <= I \\
 & \rightarrow > I \\
 & \rightarrow >= I \\
 & \rightarrow \mathrm{rem\ I} \\
 & \rightarrow \wedge \\
 & \\
\mathrm{F} & \rightarrow \mathrm{P} \\
 & \rightarrow \text{-P} \\
 & \rightarrow \mathrm{not\ \ P} \\
 & \\
\mathrm{P} & \rightarrow < entier > \\
 & \rightarrow < caractere > \\
 & \rightarrow true \\
 & \rightarrow false \\
 & \rightarrow null \\
 & \rightarrow < acces > \\
 & \rightarrow new < ident > \\
 & \rightarrow < ident > (< expr >_{,}^{+}) \\
 & \rightarrow character\ ' \ val\ (< expr >) \\
 & \rightarrow (< expr >)
\end{array}
$$

On injecte les règles de $< acces >$ dans les règles ci-dessous pour supprimer la règle $< acces >$ et ainsi résoudre des conflits :

$$\begin{aligned} \text{P} &\rightarrow < acces > \\ < instr > &\rightarrow < acces > := < expr >; \end{aligned}$$

On obtient les règles :

$$\begin{aligned} < instr > &\rightarrow < ident > := < expr >; \\ &\rightarrow < expr > . < ident > := < expr >; \end{aligned}$$

$$\begin{aligned} P &\rightarrow < ident > \\ &\rightarrow < expr > . < ident > \end{aligned}$$

La règle $P \rightarrow \langle expr \rangle$. $\langle ident \rangle$ offre la possibilité de finir une expression par . $\langle ident \rangle$. Or, une expression se termine toujours par l'une des parties droites des règles $P \rightarrow$ et ces parties droites terminent toujours une expression.

Donc, on peut remplacer :
$P \rightarrow \langle expr \rangle$. $\langle ident \rangle$

Par :

P $\rightarrow \langle$entier$\rangle$. $\langle$ident$\rangle$
$\rightarrow \langle$caractere$\rangle$. $\langle$ident$\rangle$
$\rightarrow$ *true* . $\langle$ident$\rangle$
$\rightarrow$ *false* . $\langle$ident$\rangle$
$\rightarrow$ *null* . $\langle$ident$\rangle$
$\rightarrow \langle$ident$\rangle$. $\langle$ident$\rangle$
$\rightarrow$ *new* $\langle$ident$\rangle$. $\langle$ident$\rangle$
$\rightarrow \langle$ident$\rangle$ ($\langle$expr$\rangle_{,}^{+}$) . $\langle$ident$\rangle$
$\rightarrow$ *character* ′ val ($\langle$expr$\rangle$) . $\langle$ident$\rangle$
$\rightarrow$ ($\langle$expr$\rangle$) . $\langle$ident$\rangle$

On développe la grammaire pour ensuite appliquer la dérécursivation :

$< fichier > \rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is begin $< instr >^+$ end; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is begin $< instr >^+$ end $< ident >$; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$; EOF

$< decl > \rightarrow$ type $< ident >$;
$\rightarrow$ type $< ident >$ is access $< ident >$;
$\rightarrow$ type $< ident >$ is record $< champs >^+$ end record;
$\rightarrow$ type $< ident >^+_{,} : < type >$;
$\rightarrow$ type $< ident >^+_{,} : < type > (:=< expr >)$;
$\rightarrow$ procedure $< ident >$ is begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident > < param >$ is begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident > < param >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident > < param >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident > < param >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident >$ return $< type >$ is begin $< instr >^+$ end;
$\rightarrow$ function $< ident >$ return $< type >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ function $< ident >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is begin $< instr >^+$ end;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;

$< champs > \rightarrow < ident >^+_{,} : < type >$;

$< type > \rightarrow < ident >$
$\rightarrow$ access $< ident >$

$< params > \rightarrow (< param >^+_{;})$

$< param > \rightarrow < ident >^+_{,} : < type >$
$\rightarrow < ident >^+_{,} : < mode > < type >$

$< mode > \rightarrow$ in
$\rightarrow$ in out

$< expr > \rightarrow T < expr >_1$

$< expr >_1 \rightarrow + < expr >$
$\rightarrow - < expr >$
$\rightarrow \wedge$

T $\rightarrow$ I T_1

T_1 $\rightarrow$ * T
$\rightarrow$ / T
$\rightarrow \wedge$

I $\rightarrow$ F I_1

I_1 $\rightarrow$ = I
$\rightarrow$ /= I
$\rightarrow < I$
$\rightarrow <= I$
$\rightarrow > I$
$\rightarrow >= I$
$\rightarrow$ rem I
$\rightarrow \wedge$

F $\rightarrow$ P
$\rightarrow$ -P
$\rightarrow$ not P

P $\rightarrow < entier >$
$\rightarrow < caractere >$
$\rightarrow true$
$\rightarrow false$
$\rightarrow null$
$\rightarrow < ident >$
$\rightarrow new < ident >$
$\rightarrow < ident > (< expr >^{+}_{,})$
$\rightarrow character\ '\ val\ (< expr >)$
$\rightarrow (< expr >)$
$\rightarrow < entier > . < ident >$
$\rightarrow < caractere > . < ident >$
$\rightarrow true . < ident >$
$\rightarrow false . < ident >$
$\rightarrow null . < ident >$
$\rightarrow < ident > . < ident >$

$\rightarrow new < ident > . < ident >$
$\rightarrow < ident > (< expr >^{+}_{,}) . < ident >$
$\rightarrow character\ ' val\ (< expr >) . < ident >$
$\rightarrow (< expr >) . < ident >$

$< instr >$ $\rightarrow < ident > := < expr >;$
$\rightarrow < expr > . < ident > := < expr >;$
$\rightarrow < ident >;$
$\rightarrow < ident > (< expr >^{+}_{,});$
$\rightarrow$ return;
$\rightarrow$ return $< expr >;$
$\rightarrow$ begin $< instr >^{+}$ end;
$\rightarrow$ if $< expr >$ then $< instr >^{+}$ end if;
$\rightarrow$ if $< expr >$ then $< instr >^{+}$ (else $< instr >^{+}$) end if;
$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ end if;
$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ (else $< instr >^{+}$) end if;
$\rightarrow$ for $< ident >$ in $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;
$\rightarrow$ for $< ident >$ in reverse $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;
$\rightarrow$ while $< expr >$ loop $< instr >^{+}$ end loop;

$< instr >^{+}$ $\rightarrow < instr > < instr >^{+}$
$\rightarrow < instr >$
$< decl >^{+}$ $\rightarrow < decl > < decl >^{+}$
$\rightarrow < decl >$
$< champs >^{+}$ $\rightarrow < champs > < champs >^{+}$
$\rightarrow < champs >$
$< ident >^{+}_{,}$ $\rightarrow < ident > , < ident >^{+}_{,}$
$\rightarrow < ident >;$
$< param >^{+}_{;}$ $\rightarrow < param > ; < param >^{+}_{;}$
$\rightarrow < param >$
$< expr >^{+}_{,}$ $\rightarrow < expr > , < expr >^{+}_{,}$
$\rightarrow < expr >$
$< elsif >^{+}$ $\rightarrow$ elsif $< expr >$ then $< instr >^{+} < elsif >^{+}$
$\rightarrow$ elsif $< expr >$ then $< instr >^{+}$

On n'a plus de récursivité à droite.

On factorise la grammaire et on numérote les règles.

$r_1 :< fichier > \rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< fichier >_2$

$r_2 :< fichier >_2 \rightarrow$ begin $< instr >^+$ end $< fichier >_3$
$r_3 : \rightarrow < decl >^+$ begin $< instr >^+$ end $< fichier >_3$

$r_4 :< fichier >_3 \rightarrow$; EOF
$r_5 : \rightarrow < ident >$; EOF

$r_6 :< decl > \rightarrow$ type $< ident > < decl >_{11}$
$r_7 : \rightarrow$ procedure $< ident > < decl >_{21}$
$r_8 : \rightarrow$ function $< ident > < decl >_{31}$
$r_9 : \rightarrow < ident >_{,}^+ : < type > < decl >_{12}$

$r_{10} :< decl >_{11} \rightarrow$;
$r_{11} : \rightarrow$ is $< decl >_{13}$

$r_{12} :< decl >_{12} \rightarrow$;
$r_{13} : \rightarrow :=< expr >$;

$r_{14} :< decl >_{13} \rightarrow$ access $< ident >$;
$r_{15} : \rightarrow$ record $< champs >^+$ end record;

$r_{16} :< decl >_{21} \rightarrow$ is $< decl >_{22}$
$r_{17} : \rightarrow < params >$ is $< decl >_{22}$

$r_{18} :< decl >_{22} \rightarrow$ begin $< instr >^+$ end $< decl >_{23}$
$r_{19} : \rightarrow < decl >^+$ begin $< instr >^+$ end $< decl >_{23}$

$r_{20} :< decl >_{23} \rightarrow$;
$r_{21} : \rightarrow < ident >$;

$r_{22} :< decl >_{31} \rightarrow$ return $< type >$ is $< decl >_{22}$
$r_{23} : \rightarrow < params >$ return $< type >$ is $< decl >_{22}$

$r_{24} :< champs > \rightarrow < ident >_{,}^+ : < type >$;

$r_{25} :< type > \rightarrow < ident >$
$r_{26} : \rightarrow$ access $< ident >$

$r_{27} :< params > \rightarrow (< param >_{;}^+)$

r_{28} :$< param >$ $\rightarrow < ident >^{+}_{,} : < param >_2$

r_{29} :$< param >_2$ $\rightarrow < type >$
r_{30} : $\rightarrow < mode > < type >$

r_{31} :$< mode >$ $\rightarrow$ in $< mode >_1$

r_{32} :$< mode >_1$ $\rightarrow$ out
r_{33} : $\rightarrow \wedge$

r_{34} :$< expr >$ $\rightarrow T < expr >_1$

r_{35} :$< expr >_1$ $\rightarrow + < expr >$
r_{36} : $\rightarrow - < expr >$
r_{37} : $\rightarrow \wedge$

$r_{38} : T$ $\rightarrow I\ T_1$

$r_{39} : T_1$ $\rightarrow *T$
r_{40} : $\rightarrow /T$
r_{41} : $\rightarrow \wedge$

$r_{42} : I$ $\rightarrow F\ I_1$

$r_{43} : I_1$ $\rightarrow = T$
r_{44} : $\rightarrow = / T$
r_{45} : $\rightarrow < T$
r_{46} : $\rightarrow <= T$
r_{47} : $\rightarrow > T$
r_{48} : $\rightarrow >= T$
r_{49} : $\rightarrow remT$
r_{50} : $\rightarrow \wedge$

$r_{51} : F$ $\rightarrow P$
r_{52} : $\rightarrow -P$
r_{53} : $\rightarrow not\ P$

$r_{54} : P \quad \rightarrow < entier > P_1$

$r_{55} : \quad \rightarrow < caractere > P_1$

$r_{56} : \quad \rightarrow true\ P_1$

$r_{57} : \quad \rightarrow false\ P_1$

$r_{58} : \quad \rightarrow null\ P_1$

$r_{59} : \quad \rightarrow$ new $< ident > P_1$

$r_{60} : \quad \rightarrow$ character ' val $(< expr)\ P_1$

$r_{61} : \quad \rightarrow (< expr >)\ P_1$

$r_{62} : \quad \rightarrow < ident > P_2$

$r_{63} : P_1 \quad \rightarrow . < ident >$

$r_{64} : \quad \rightarrow \wedge$

$r_{65} : P_2 \quad \rightarrow . < ident >$

$r_{66} : \quad \rightarrow (< expr >_{,}^{+})\ P_1$

$r_{67} : \quad \rightarrow \wedge$

$r_{68} :< instr > \quad \rightarrow < ident > < instr >_1$

$r_{69} : \quad \rightarrow < expr > . < ident > := < expr >;$

$r_{70} : \quad \rightarrow$ return $< instr >_2$

$r_{71} : \quad \rightarrow$ begin $< instr >^{+}$ end;

$r_{72} : \quad \rightarrow$ if $< expr >$ then $< instr >^{+} < instr >_3$

$r_{73} : \quad \rightarrow$ for $< ident >$ in $< instr >_4$

$r_{74} : \quad \rightarrow$ while $< expr >$ loop $< instr >^{+}$ end loop;

$r_{75} :< instr >_1 \rightarrow := < expr >;$

$r_{76} : \quad \rightarrow ;$

$r_{77} : \quad \rightarrow (< expr >_{,}^{+});$

$r_{78} :< instr >_2 \rightarrow < expr >;$

$r_{79} : \quad \rightarrow ;$

$r_{80} :< instr >_3 \rightarrow$ end if;

$r_{81} : \quad \rightarrow$ else $< instr >^{+}$ end if;

$r_{82} : \quad \rightarrow < elsif >^{+} < instr >_3$

$r_{83} :< instr >_4 \rightarrow < expr > .. < expr >$ loop $< instr >^{+}$ end loop;

$r_{84} : \quad \rightarrow$ reverse $< expr > .. < expr >$ loop $< instr >^{+}$ end loop;

$r_{85} :< instr >^{+} \quad \rightarrow < instr > < instr >_{1}^{+}$

$r_{86} :< instr >_{1}^{+} \quad \rightarrow < instr >^{+}$

$r_{87} : \quad \rightarrow \wedge$

$r_{88} :< decl >^{+} \quad \rightarrow < decl > < decl >_{1}^{+}$

$r_{89} :< decl >_{1}^{+} \quad \rightarrow < decl >^{+}$

$r_{90} : \quad \rightarrow \wedge$

$r_{91} :< champs >^{+} \rightarrow < champs > < champs >_{1}^{+}$

$r_{92} :< champs >_{1}^{+} \rightarrow < champs >^{+}$

$r_{93} : \quad \rightarrow \wedge$

$r_{94} :< ident >_{,}^{+} \quad \rightarrow < ident > < ident >_{,1}^{+}$

$r_{95} :< ident >_{,1}^{+} \quad \rightarrow , < ident >_{,}^{+}$

$r_{96} : \quad \rightarrow \wedge$

$r_{97} :< param >_{;}^{+} \quad \rightarrow < param > < param >_{;1}^{+}$

$r_{98} :< param >_{;1}^{+} \quad \rightarrow ; < param >_{;}^{+}$

$r_{99} : \quad \rightarrow \wedge$

$r_{100} :< expr >_{,}^{+} \quad \rightarrow < expr > < expr >_{,1}^{+}$

$r_{101} :< expr >_{,1}^{+} \quad \rightarrow , < expr >_{,}^{+}$

$r_{102} : \quad \rightarrow \wedge$

$r_{103} :< elsif >^{+} \quad \rightarrow \text{elsif} < expr > \text{then} < instr >^{+} < elsif >_{1}^{+}$

$r_{104} :< elsif >_{1}^{+} \quad \rightarrow < elsif >^{+}$

$r_{105} : \quad \rightarrow \wedge$

On injecte les règles de $< expr >$ dans la règle :

$$< instr > \rightarrow < expr > . < ident > := < expr >;$$

On obtient :

$$< instr > \rightarrow T < expr >_1 . < ident > := < expr >;$$

On injecte les règles de T dans la règle :

$$< instr > \rightarrow T < expr >_1 . < ident > := < expr >;$$

On obtient :

$$< instr > \rightarrow I\ T_1 < expr >_1 . < ident > := < expr >;$$

On injecte les règles de I dans la règle :

$$< instr > \rightarrow I\ T_1 < expr >_1 . < ident > := < expr >;$$

On obtient :

$$< instr > \rightarrow FI_1\ T_1 < expr >_1 . < ident > := < expr >;$$

On injecte les règles de F dans la règle :

$$< instr > \rightarrow FI_1\ T_1 < expr >_1 . < ident > := < expr >;$$

On obtient :

$$\begin{aligned} < instr > &\rightarrow P\ I_1\ T_1 < expr >_1 . < ident > := < expr >; \\ &\rightarrow -P\ I_1\ T_1 < expr >_1 . < ident > := < expr >; \\ &\rightarrow not\ P\ I_1\ T_1 < expr >_1 . < ident > := < expr >; \end{aligned}$$

On injecte les règles de P dans la règle :

$$< instr > \rightarrow P\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$$

On obtient :

$$
\begin{aligned}
< instr > \rightarrow & < entier > P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & < caractere > P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & true\ P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & false\ P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & null\ P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & new < ident > P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & character'val(< expr >) P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & (< expr >) P_1\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & < ident > P_2\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >;
\end{aligned}
$$

On injecte les règles de P_2 dans la règle :

$$< instr > \rightarrow < ident > P_2\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >;$$

On obtient :

$$
\begin{aligned}
< instr > \rightarrow & < ident > . < ident >\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & < ident > (< expr >_{,}^{+})\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >; \\
\rightarrow & < ident >\ I_1\ T_1 < expr >_1 \ .\ < ident > := < expr >;
\end{aligned}
$$

On injecte les règles de $< instr >_1$ dans la règle :

$$< instr > \rightarrow < ident > < instr >_1$$

On obtient :

$$
\begin{aligned}
< instr > \rightarrow & < ident > := < expr > ; \\
\rightarrow & < ident > ; \\
\rightarrow & < ident > (< expr >_{,}^{+});
\end{aligned}
$$

On a au final :

$$
\begin{aligned}
< instr > &\rightarrow < ident > := < expr > ; \\
&\rightarrow < ident > ; \\
&\rightarrow < ident > (< expr >_{,}^{+}); \\
&\rightarrow < ident > . < ident > \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow < ident > (< expr >_{,}^{+}) \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow < ident > \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow -P \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow not \ P \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow < entier > P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow < caractere > P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow true \ P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow false \ P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow null \ P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow new < ident > P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow character'val(< expr >) P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow (< expr >) P_1 \ I_1 \ T_1 < expr >_1 \ . \ < ident > := < expr >; \\
&\rightarrow \text{return} < instr >_2 \\
&\rightarrow \text{begin} < instr >^{+} \text{end;} \\
&\rightarrow \text{if} < expr > \text{then} < instr >^{+} < instr >_3 \\
&\rightarrow \text{for} < ident > \text{in} < instr >_4 \\
&\rightarrow \text{while} < expr > \text{loop} < instr >^{+} \text{end loop;}
\end{aligned}
$$

En factorisant de nouveau, on a :

$r_1 :< fichier > \rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< fichier >_2$

$r_2 :< fichier >_2 \rightarrow$ begin $< instr >^+$ end $< fichier >_3$
$r_3 : \rightarrow < decl >^+$ begin $< instr >^+$ end $< fichier >_3$

$r_4 :< fichier >_3 \rightarrow$; EOF
$r_5 : \rightarrow < ident >$; EOF

$r_6 :< decl > \rightarrow$ type $< ident > < decl >_{11}$
$r_7 : \rightarrow$ procedure $< ident > < decl >_{21}$
$r_8 : \rightarrow$ function $< ident > < decl >_{31}$
$r_9 : \rightarrow < ident >_,^+ : < type > < decl >_{12}$

$r_{10} :< decl >_{11} \rightarrow$;
$r_{11} : \rightarrow$ is $< decl >_{13}$

$r_{12} :< decl >_{12} \rightarrow$;
$r_{13} : \rightarrow :=< expr >$;

$r_{14} :< decl >_{13} \rightarrow$ access $< ident >$;
$r_{15} : \rightarrow$ record $< champs >^+$ end record;

$r_{16} :< decl >_{21} \rightarrow$ is $< decl >_{22}$
$r_{17} : \rightarrow < params >$ is $< decl >_{22}$

$r_{18} :< decl >_{22} \rightarrow$ begin $< instr >^+$ end $< decl >_{23}$
$r_{19} : \rightarrow < decl >^+$ begin $< instr >^+$ end $< decl >_{23}$

$r_{20} :< decl >_{23} \rightarrow$;
$r_{21} : \rightarrow < ident >$;

$r_{22} :< decl >_{31} \rightarrow$ return $< type >$ is $< decl >_{22}$
$r_{23} : \rightarrow < params >$ return $< type >$ is $< decl >_{22}$

$r_{24} :< champs > \rightarrow < ident >_,^+ : < type >$;

$r_{25} :< type > \rightarrow < ident >$
$r_{26} : \rightarrow$ access $< ident >$

$r_{27} :< params > \rightarrow (< param >_;^+)$

r_{28} :$< param >$ $\rightarrow < ident >^{+}_{,} : < param >_2$

r_{29} :$< param >_2$ $\rightarrow < type >$
r_{30} : $\rightarrow < mode > < type >$

r_{31} :$< mode >$ $\rightarrow$ in $< mode >_1$

r_{32} :$< mode >_1$ $\rightarrow$ out
r_{33} : $\rightarrow \wedge$

r_{34} :$< expr >$ $\rightarrow T < expr >_1$

r_{35} :$< expr >_1$ $\rightarrow + < expr >$
r_{36} : $\rightarrow - < expr >$
r_{37} : $\rightarrow \wedge$

$r_{38} : T$ $\rightarrow I\ T_1$

$r_{39} : T_1$ $\rightarrow *T$
r_{40} : $\rightarrow /T$
r_{41} : $\rightarrow \wedge$

$r_{42} : I$ $\rightarrow F\ I_1$

$r_{43} : I_1$ $\rightarrow = T$
r_{44} : $\rightarrow = / T$
r_{45} : $\rightarrow < T$
r_{46} : $\rightarrow <= T$
r_{47} : $\rightarrow > T$
r_{48} : $\rightarrow >= T$
r_{49} : $\rightarrow remT$
r_{50} : $\rightarrow \wedge$

$r_{51} : F$ $\rightarrow P$
r_{52} : $\rightarrow -P$
r_{53} : $\rightarrow not\ P$

$r_{54} : P \quad \rightarrow < entier > P_1$

$r_{55} : \quad \rightarrow < caractere > P_1$

$r_{56} : \quad \rightarrow true\ P_1$

$r_{57} : \quad \rightarrow false\ P_1$

$r_{58} : \quad \rightarrow null\ P_1$

$r_{59} : \quad \rightarrow \text{new} < ident > P_1$

$r_{60} : \quad \rightarrow \text{character ' val} (< expr)\ P_1$

$r_{61} : \quad \rightarrow (< expr >)\ P_1$

$r_{62} : \quad \rightarrow < ident > P_2$

$r_{63} : P_1 \quad \rightarrow . < ident >$

$r_{64} : \quad \rightarrow \wedge$

$r_{65} : P_2 \quad \rightarrow . < ident >$

$r_{66} : \quad \rightarrow (< expr >_{,}^{+})\ P_1$

$r_{67} : \quad \rightarrow \wedge$

$r_{68} :< instr > \quad \rightarrow < ident > < instr >_1$

$r_{69} : \quad \rightarrow -P\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{70} : \quad \rightarrow not\ P\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{71} : \quad \rightarrow < entier > P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{72} : \quad \rightarrow < caractere > P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{73} : \quad \rightarrow true\ P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{74} : \quad \rightarrow false\ P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{75} : \quad \rightarrow null\ P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{76} : \quad \rightarrow new < ident > P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{77} : \quad \rightarrow character'val(< expr >)P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{78} : \quad \rightarrow (< expr >)P_1\ I_1\ T_1 < expr >_1 . < ident > := < expr >;$

$r_{79} : \quad \rightarrow \text{return} < instr >_2$

$r_{80} : \quad \rightarrow \text{begin} < instr >^{+} \text{end;}$

$r_{81} : \quad \rightarrow \text{if} < expr > \text{then} < instr >^{+} < instr >_3$

$r_{82} : \quad \rightarrow \text{for} < ident > \text{in} < instr >_4$

$r_{83} : \quad \rightarrow \text{while} < expr > \text{loop} < instr >^{+} \text{end loop;}$

$r_{84} :< instr >_1 \quad \rightarrow := < expr >;$

$r_{85} : \quad \rightarrow ;$

$r_{86} : \quad \rightarrow (< expr >_{,}^{+}) < instr >_{11}$

$r_{87} : \quad \rightarrow . < ident > I_1\ T_1 < expr >_1 . < ident > := < expr > ;$

$r_{88} : \quad \rightarrow I_1\ T_1 < expr >_1 . < ident > := < expr > ;$

$r_{89} :< instr >_{11} \rightarrow ;$

$r_{90} : \quad \rightarrow I_1\ T_1 < expr >_1 . < ident > := < expr > ;$

r_{91} :$< instr >_2$ $\rightarrow < expr >$;

r_{92} : $\rightarrow$;

r_{93} :$< instr >_3$ $\rightarrow$ end if;

r_{94} : $\rightarrow$ else $< instr >^+$ end if;

r_{95} : $\rightarrow < elsif >^+ < instr >_3$

r_{96} :$< instr >_4$ $\rightarrow < expr >$ .. $< expr >$ loop $< instr >^+$ end loop;

r_{97} : $\rightarrow$ reverse $< expr >$ .. $< expr >$ loop $< instr >^+$ end loop;

r_{98} :$< instr >^+$ $\rightarrow < instr > < instr >_1^+$

r_{99} :$< instr >_1^+$ $\rightarrow < instr >^+$

r_{100} : $\rightarrow \wedge$

r_{101} :$< decl >^+$ $\rightarrow < decl > < decl >_1^+$

r_{102} :$< decl >_1^+$ $\rightarrow < decl >^+$

r_{103} : $\rightarrow \wedge$

r_{104} :$< champs >^+$ $\rightarrow < champs > < champs >_1^+$

r_{105} :$< champs >_1^+$ $\rightarrow < champs >^+$

r_{106} : $\rightarrow \wedge$

r_{107} :$< ident >_,^+$ $\rightarrow < ident > < ident >_{,1}^+$

r_{108} :$< ident >_{,1}^+$ $\rightarrow$, $< ident >_,^+$

r_{109} : $\rightarrow \wedge$

r_{110} :$< param >_;^+$ $\rightarrow < param > < param >_{;1}^+$

r_{111} :$< param >_{;1}^+$ $\rightarrow$; $< param >_;^+$

r_{112} : $\rightarrow \wedge$

r_{113} :$< expr >^{+}_{,} \rightarrow < expr > < expr >^{+}_{,1}$

r_{114} :$< expr >^{+}_{,1} \rightarrow , < expr >^{+}_{,}$

r_{115} : $\rightarrow \wedge$

r_{116} :$< elsif >^{+} \rightarrow$ elsif $< expr >$ then $< instr >^{+} < elsif >^{+}_{1}$

r_{117} :$< elsif >^{+}_{1} \rightarrow < elsif >^{+}$

r_{118} : $\rightarrow \wedge$

$$P_\wedge(G) = \{< mode >_1, < expr >_1, < acces >_1, < instr >_1^+, < decl >_1^+, < champs >_1^+, < ident >_{,1}^+, < param >_{;1}^+, < expr >_{,1}^+, < elsif >_1^+\}$$

Non terminal gauche	Règle	Symbole Directeur
$< fichier >$	r_1	with
$< fichier >_2$	r_2	begin
$< fichier >_2$	r_3	type , procedure , function, $< ident >$
$< fichier >_3$	r_4	;
$< fichier >_3$	r_5	$< ident >$
$< decl >$	r_6	type
$< decl >$	r_7	procedure
$< decl >$	r_8	function
$< decl >$	r_9	$< ident >$
$< decl >_{11}$	r_{10}	;
$< decl >_{11}$	r_{11}	is
$< decl >_{12}$	r_{12}	;
$< decl >_{12}$	r_{13}	:=
$< decl >_{13}$	r_{14}	access
$< decl >_{13}$	r_{15}	record
$< decl >_{21}$	r_{16}	is
$< decl >_{21}$	r_{17}	(
$< decl >_{22}$	r_{18}	begin
$< decl >_{22}$	r_{19}	type , procedure , function, $< ident >$
$< decl >_{23}$	r_{20}	;
$< decl >_{23}$	r_{21}	$< ident >$
$< decl >_{31}$	r_{22}	return
$< decl >_{31}$	r_{23}	(
$< champs >$	r_{24}	$< ident >$
$< type >$	r_{25}	$< ident >$
$< type >$	r_{26}	access
$< params >$	r_{27}	(
$< param >$	r_{28}	$< ident >$
$< param >_2$	r_{29}	$< ident >$, access
$< param >_2$	r_{30}	in
$< mode >$	r_{31}	in
$< mode >_1$	r_{32}	out
$< mode >_1$	r_{33}	$< ident >$, access
$< expr >$	r_{34}	$-, not, < entier >, < caractère >, < ident >, true, false, null, new, character, ($
$< expr >_1$	r_{35}	+
$< expr >_1$	r_{36}	−
$< expr >_1$	r_{37}	$;,), ., ,, then, loop, ..$
T	r_{38}	$-, not, < entier >, < caractère >, < ident >, true, false, null, new, character, ($
T_1	r_{39}	/
T_1	r_{40}	*
T_1	r_{41}	$;,), ., ,, then, loop, .., +, -$
I	r_{42}	$-, not, < entier >, < caractère >, < ident >, true, false, null, new, character, ($
I_1	r_{43}	=
I_1	r_{44}	= /
I_1	r_{45}	<
I_1	r_{46}	<=
I_1	r_{47}	>
I_1	r_{48}	>=
I_1	r_{49}	rem

I_1	r_{50}	$;,), ., , , then, loop, .., +, -, *, /$
F	r_{51}	$< entier >, < caractère >, < ident >, true, false, null, new, character, ($
F	r_{52}	$-$
F	r_{53}	not
P	r_{54}	$< entier >$
P	r_{55}	$< caractere >$
P	r_{56}	$true$
P	r_{57}	$false$
P	r_{58}	$null$
P	r_{59}	new
P	r_{60}	$character$
P	r_{61}	$($
P	r_{62}	$< ident >$
P_1	r_{63}	.
P_1	r_{64}	$=, / =, <, <=, >, >=, rem, *, /, +, -, ;,), ., , , then, loop, ..,$
P_2	r_{65}	.
P_2	r_{66}	$($
P_2	r_{67}	$=, / =, <, <=, >, >=, rem, *, /, +, -, ;,), ., , , then, loop, ..,$
$< instr >$	r_{68}	$< ident >$
$< instr >$	r_{69}	$-$
$< instr >$	r_{70}	not
$< instr >$	r_{71}	$< entier >$
$< instr >$	r_{72}	$< caractere >$
$< instr >$	r_{73}	$true$
$< instr >$	r_{74}	$false$
$< instr >$	r_{75}	$null$
$< instr >$	r_{76}	new
$< instr >$	r_{77}	$character$
$< instr >$	r_{78}	$($
$< instr >$	r_{79}	$return$
$< instr >$	r_{80}	$begin$
$< instr >$	r_{81}	if
$< instr >$	r_{82}	for
$< instr >$	r_{83}	$while$
$< instr >_1$	r_{84}	$:=$
$< instr >_1$	r_{85}	;
$< instr >_1$	r_{86}	(
$< instr >_1$	r_{87}	.
$< instr >_1$	r_{88}	., +, -, *, /, =, =/, <, <=, >, >=, rem
$< instr >_{11}$	r_{89}	;
$< instr >_{11}$	r_{90}	., +, -, *, /, =, =/, <, <=, >, >=, rem
$< instr >_2$	r_{91}	$< entier >$
$< instr >_2$	r_{92}	;
$< instr >_3$	r_{93}	end
$< instr >_3$	r_{94}	$else$
$< instr >_3$	r_{95}	elsif
$< instr >_4$	r_{96}	$< entier >, < caractere >, true, false, null, new, character, (, < ident >$
$< instr >_4$	r_{97}	reverse
$< instr >^+$	r_{98}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character, return, begin, if, for, while
$< instr >_1^+$	r_{99}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, new, character, return, begin, if, for, while
$< instr >_1^+$	r_{100}	end, (,)
$< decl >^+$	r_{101}	type, procedure, function

$< decl >^{+}_{1}$	r_{102}	type, procedure, function
$< decl >^{+}_{1}$	r_{103}	begin
$< champs >^{+}$	r_{104}	$< ident >$
$< champs >^{+}_{1}$	r_{105}	$< ident >$
$< champs >^{+}_{1}$	r_{106}	end
$< ident >^{+}_{,}$	r_{107}	$< ident >$
$< ident >^{+}_{,1}$	r_{108}	,
$< ident >^{+}_{,1}$	r_{109}	:
$< param >^{+}_{;}$	r_{110}	$< ident >$
$< param >^{+}_{;1}$	r_{111}	;
$< param >^{+}_{;1}$	r_{112}	)
$< expr >^{+}_{,}$	r_{113}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character
$< expr >^{+}_{,1}$	r_{114}	,
$< expr >^{+}_{,1}$	r_{115}	)
$< elsif >^{+}$	r_{116}	elsif
$< elsif >^{+}_{1}$	r_{117}	elsif
$< elsif >^{+}_{1}$	r_{118}	end, else

Si problème, on infecte. (règle 69)