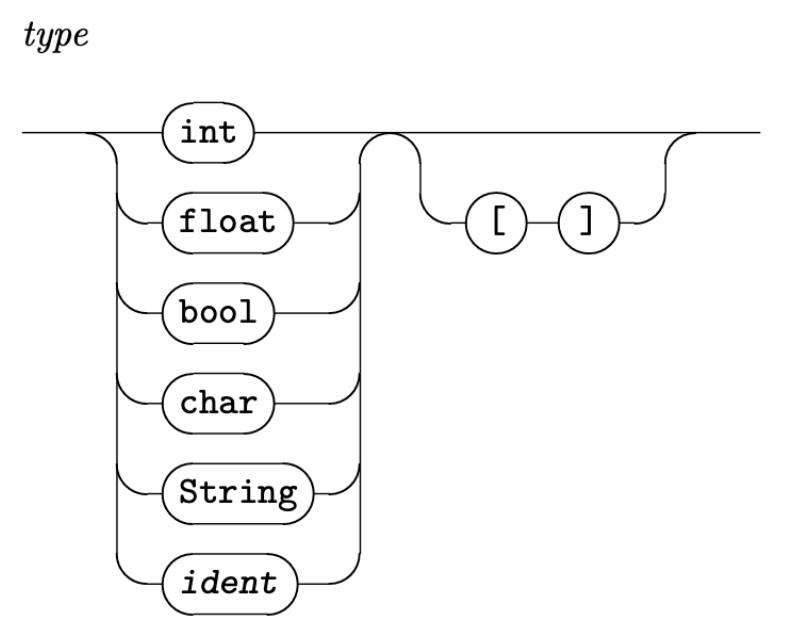
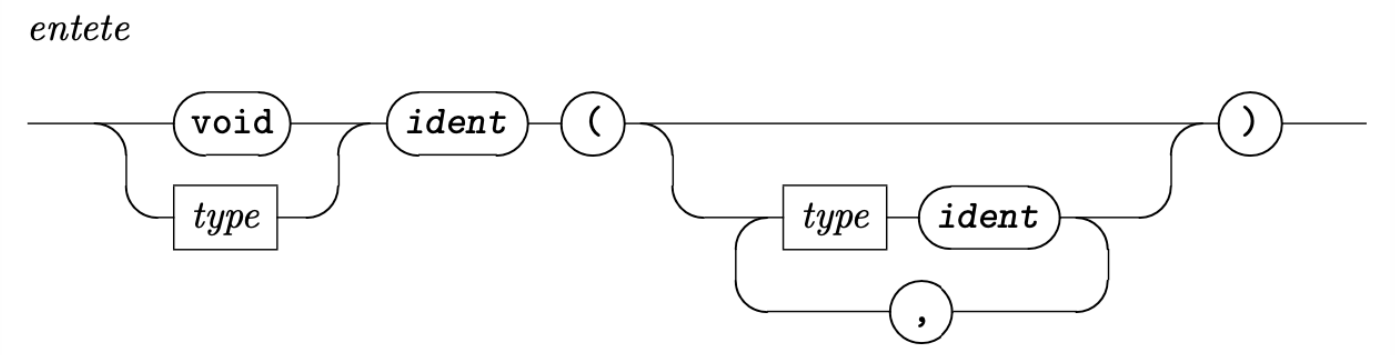
TP2: L'analyseur Sémantique

parserJava.mly

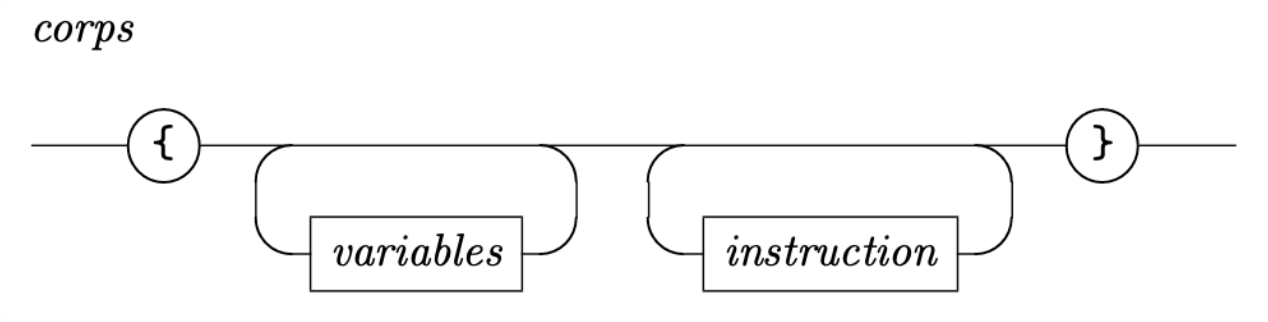



```
typeStruct : typeBase declTab { (print_endline "typeStruct : typeBase declTab") }

typeBase : INT { (print_endline "typeBase : INT") }
         | FLOAT { (print_endline "typeBase : FLOAT") }
         | BOOL { (print_endline "typeBase : BOOL") }
         | CHAR { (print_endline "typeBase : CHAR") }
         | STRING { (print_endline "typeBase : STRING") }
         | TYPEIDENT { (print_endline "typeBase : TYPEIDENT") }

declTab : /* Lambda, mot vide */ { (print_endline "declTab : /* Lambda, mot vide */") }
        | CROOUV CROFER { (print_endline "declTab : CROOUV CROFER") }
```



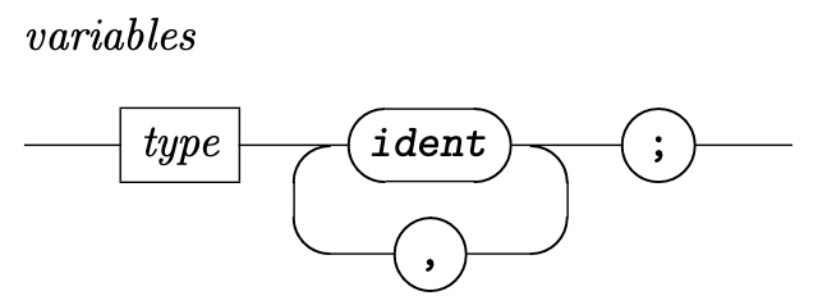

```
entete :
```

```
    typeStruct IDENT PAROUV parsFormels PARFER
      { (print_endline "entete : typeStruct IDENT PAROUV parsFormels PARFER") }
  | VOID IDENT PAROUV parsFormels PARFER
      { (print_endline "entete : VOID IDENT PAROUV parsFormels PARFER") }

parsFormels :
    /* Lambda, mot vide */
      { (print_endline "parsFormels : /* Lambda, mot vide */") }
    | typeStruct IDENT suiteParsFormels
      { (print_endline "parsFormels : typeStruct IDENT suiteParsFormels") }

suiteParsFormels :
    /* Lambda, mot vide */
      { (print_endline "suiteParsFormels : /* Lambda, mot vide */") }
    | VIRG typeStruct IDENT suiteParsFormels
      { (print_endline "suiteParsFormels : VIRG typeStruct IDENT suiteParsFormels") }
```

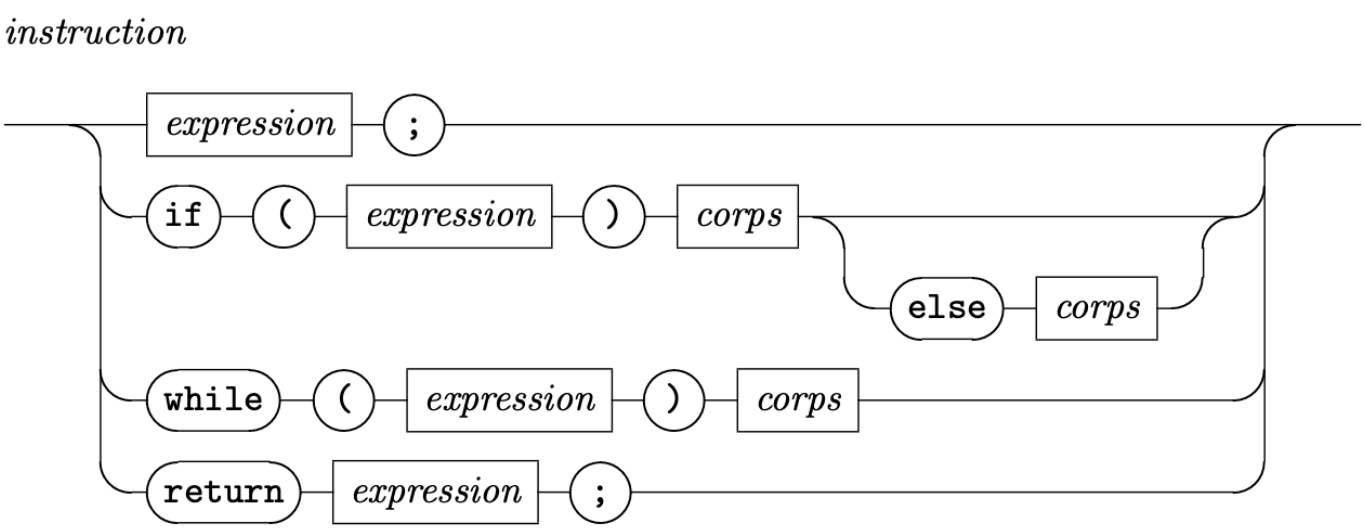



```
bloc :
    ACCOUV variables instructions ACCFER
      { (print_endline "bloc : ACCOUV variables instructions ACCFER");
        (print_string "Nombre de variables = ");
        (print_int $2); /* variables */
        (print_newline ())
      }

variables :
    /* Lambda, mot vide */
      { (print_endline "variables : /* Lambda, mot vide */");
        0
      }
    | variable /* $1 */ variables /* $2 */
      { (print_endline "variables : variable variables");
        ($2 + 1)
      }

instructions :
    /* Lambda, mot vide */
      { (print_endline "instructions : /* Lambda, mot vide */") }
    | instruction instructions
```

```
      { (print_endline "instructions : instruction instructions") }
```

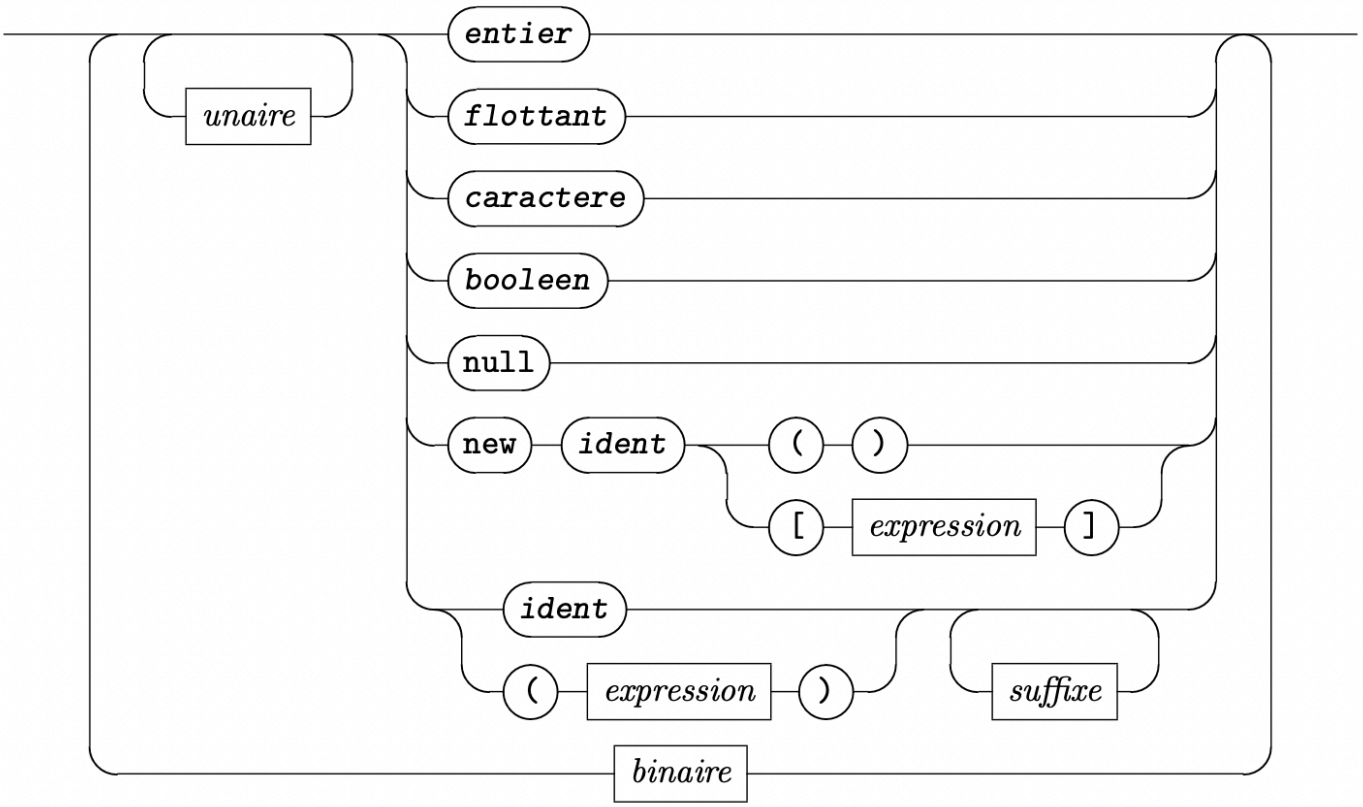



```
variable :
    typeStruct IDENT PTVIRG
      { (print_endline "variable : typeStruct IDENT PTVIRG") }
/* 我自己添加的：例如 int i = 1; 上一行不能有 "=1" 这种赋值情况存在
   只能写成如下形式：
      int i;
      i = 1;
*/
  | typeStruct IDENT binaire expression PTVIRG
      { (print_endline "variable : typeStruct IDENT binaire expression PTVIRG") }
```



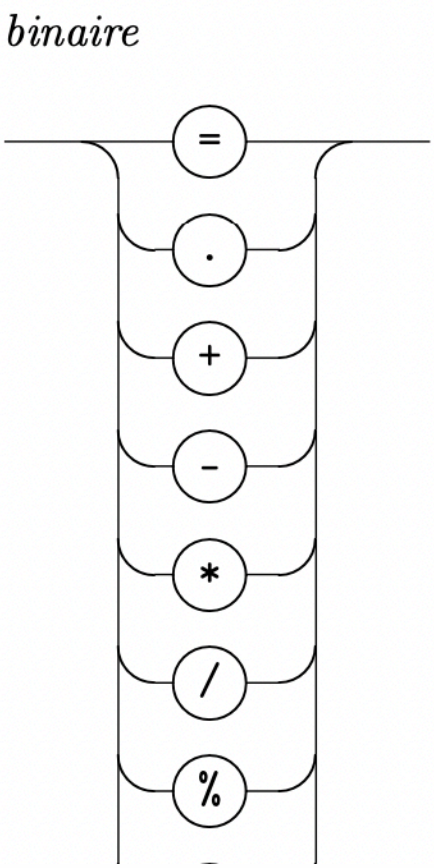
```
instruction :
    expression PTVIRG
      { (print_endline "instruction : expression PTVIRG") }
  | SI PAROUV expression PARFER bloc
      { (print_endline "instruction : SI PAROUV expression PARFER bloc") }
  | SI PAROUV expression PARFER bloc SINON bloc
      { (print_endline "instruction : SI PAROUV expression PARFER bloc SINON bloc") }
  | TANTQUE PAROUV expression PARFER bloc
      { (print_endline "instruction : TANTQUE PAROUV expression PARFER bloc") }
  | RETOUR expression PTVIRG
      { (print_endline "instruction : RETURN expression PTVIRG") }
```

expression



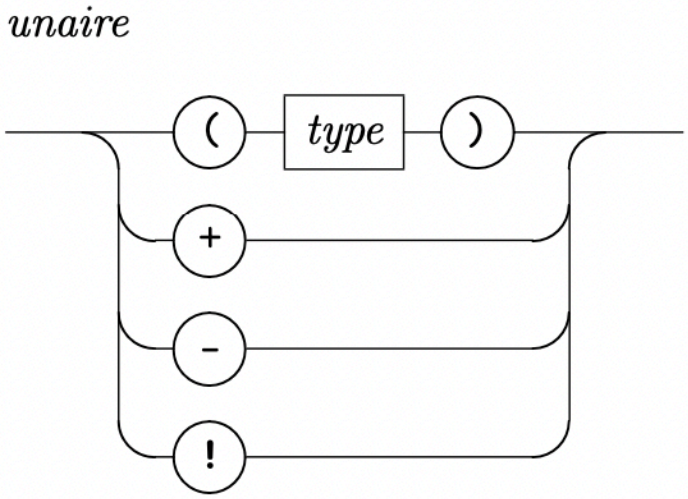

```
expression :
    unaires part_expression
      { (print_endline "expression : unaires part_expression") }
  | unaires part_expression binaire expression
      { (print_endline "expression : unaires part_expression binaire expression") }

unaires :
  /* Lambda, mot vide */
      { (print_endline "unaires : /* Lambda, mot vide */") }
  | unaire unaires
      { (print_endline "expressionvirgule : VIRG expression expressionvirgule") }
```

```
part_expression :
    ENTIER { (print_endline "part_expression : ENTIER") }
  | FLOTTANT { (print_endline "part_expression : FLOTTANT") }
  | CARACTERE { (print_endline "part_expression : CARACTERE") }
  | BOOLEEN { (print_endline "part_expression : BOOLEEN") }
  | VIDE { (print_endline "part_expression : VIDE") }
  | NOUVEAU IDENT PAROUV PARFER
      { (print_endline "part_expression : NOUVEAU IDENT PAROUV PARFER") }
  | NOUVEAU IDENT CROOUV expression CROFER
      { (print_endline "part_expression : NOUVEAU IDENT CROOUV expression CROFER") }
  | IDENT suffixes { (print_endline "part_expression : IDENT suffixes") }
  | PAROUV expression PARFER suffixes
      { (print_endline "part_expression : PAROUV expression PARFER suffixes") }

suffixes :
  /* Lambda, mot vide */ { (print_endline "suffixes : /* Lambda, mot vide */") }
  | suffixe suffixes { (print_endline "suffixes : suffixe suffixes") }
```

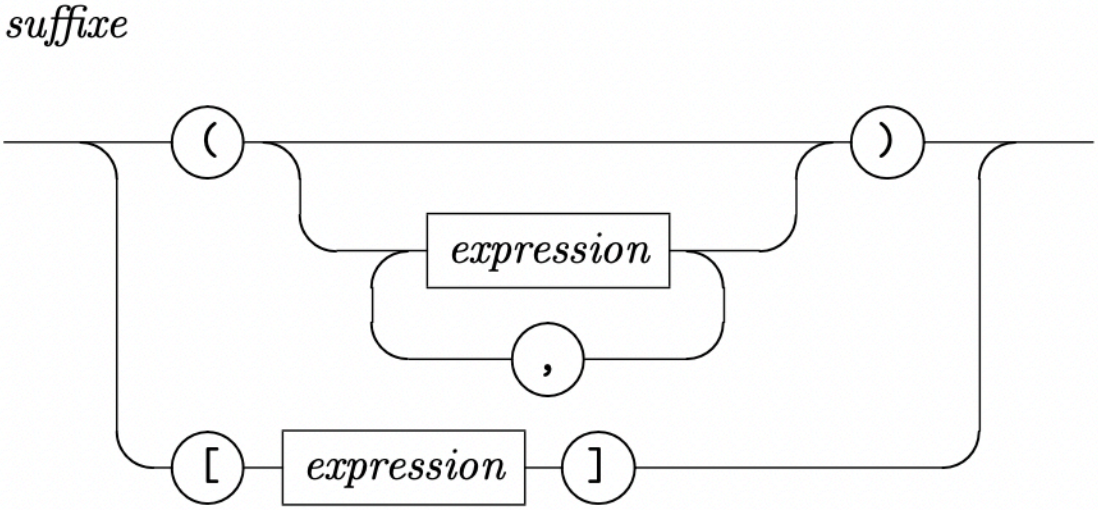



```
binaire :
      ASSIGN { (print_endline "binaire : ASSIGN") }
    | OPINF { (print_endline "binaire : OPINF") }
    | OPSUP { (print_endline "binaire : OPSUP") }
    | OPINFEG { (print_endline "binaire : OPINFEG") }
```

```
    | OPSUPEG { (print_endline "binaire : OPSUPEG") }
    | OPEG { (print_endline "binaire : OPEG") }
    | OPNONEG { (print_endline "binaire : OPNONEG") }
    | OPPLUS { (print_endline "binaire : OPPLUS") }
    | OPMOINS { (print_endline "binaire : OPMOINS") }
    | OPOU { (print_endline "binaire : OPOU") }
    | OPMULT { (print_endline "binaire : OPMULT") }
    | OPMOD { (print_endline "binaire : OPMOD") }
    | OPDIV { (print_endline "binaire : OPDIV") }
    | OPET { (print_endline "binaire : OPET") }
    | OPNON { (print_endline "binaire : OPNON") }
    | OPPT { (print_endline "binaire : OPPT") }
```




```
unaire :
    PAROUV typeBase PARFER { (print_endline "unaire : PAROUV typeBase PARFER") }
  | OPPLUS { (print_endline "unaire : OPPLUS") }
  | OPMOINS { (print_endline "unaire : OPMOINS") }
  | OPNON { (print_endline "unaire : OPNON") }
```




```
suffixe :
    PAROUV PARFER { (print_endline "suffixe : PAROUV PARFER") }
  | PAROUV expression expressionvirgule PARFER
      { (print_endline "suffixe : PAROUV expression expressionvirgule PARFER") }
  | CROOUV expression CROFER { (print_endline "suffixe : CROOUV expression CROFER") }

expressionvirgule :
  /* Lambda, mot vide */
      { (print_endline "expressionvirgule : /* Lambda, mot vide */") }
  | VIRG expression expressionvirgule
      { (print_endline "expressionvirgule : VIRG expression expressionvirgule") }
```