la Grammaire du Langage

Éléments Lexicaux

Tokens

Le langage utilise les tokens suivants :

- Mots-clés : `program`, `const`, `type`, `var`, `procedure`, `function`, `if`, `then`, `else`, `while`, `do`, `repeat`, `until`, `for`, `to`, `downto`, `case`, `of`, `begin`, `end`, `write`, `read`
- Types de données : `integer`, `real`, `boolean`, `string`
- Opérateurs : `+`, `-`, `*`, `/`, `:=`, `<>`, `<`, `>`, `<=`, `>=`
- Délimiteurs : `;`, `=`, `,`, `:`, `(`, `)`, `.`

Non-Terminaux

La grammaire inclut des non-terminaux pour la structure du programme, les déclarations, les instructions et les expressions :

- Structure du programme : `PROGRAM`, `BLOCK`
- Déclarations : `CONSTS`, `CONST_DECL`, `TYPES`, `TYPE_DECL`, `VARS`, `VAR_DECL`
- Procédures et Fonctions : `PROCFUNCPART`, `PROCDECL`, `FUNCDECL`, `PARAMLIST`, `PARAMSECTION`
- Instructions : `INSTRUCTIONS`, `INSTRUCTION`, `AFFECT`, `IF`, `WHILE`, `REPEAT`, `FOR`, `CASE`, `WRITE`, `READ`, `CALL`
- Expressions : `COND`, `EXP`, `RELOP`, `TERM`, `ADDOP`, `FACT`, `CASE_ELEMENT`
- Listes : `EXPR_LIST`, `IDENT_LIST`, `ARG_LIST`
- Types de base : `BASE_TYPE`, `ID`, `NUM`, `REAL`

Règles de Production

Structure globale

La structure de tout programme suit les règles suivantes :

```
 PROGRAM -> "program" ID ";" BLOCK "."
BLOCK -> CONSTS TYPES VARS PROCFUNCPART INSTS
```

Déclarations de Constantes

```
 CONSTS -> "const" CONST_DECL { ";" CONST_DECL } ";" | ε
CONST_DECL -> ID "=" CONST_VALUE
CONST_VALUE -> NUM | REAL
```

Déclarations de Types

```
 TYPES -> "type" TYPE_DECL { ";" TYPE_DECL } ";" | ε
TYPE_DECL -> ID "=" BASE_TYPE
```

Déclarations de Variables

```
 VARS -> "var" VAR_LIST ";" | ε
VAR_LIST -> ID { "," ID } ":" BASE_TYPE
```

Procédures et Fonctions

```
 PROCFUNCPART -> PROCDECL | FUNCDECL | PROCDECL PROCFUNCPART | FUNCDECL PROCFUNCPART | ε
PROCDECL -> "procedure" ID [ "(" PARAMLIST ")" ] ";" BLOCK ";"
FUNCDECL -> "function" ID [ "(" PARAMLIST ")" ] ":" BASE_TYPE ";" BLOCK ";"
PARAMLIST -> "(" PARAMSECTION { ";" PARAMSECTION } ")" | ε
PARAMSECTION -> ID { "," ID } ":" BASE_TYPE
```

Instructions

```
 INSTRUCTIONS -> "begin" INST { ";" INST } "end"
INSTRUCTION -> AFFECT
      | IF
      | WHILE
      | REPEAT
      | FOR
      | CASE
      | WRITE
      | READ
      | CALL
```

Instructions de Base

```
 AFFECT-> ID ":=" EXPR
IF -> "if" COND "then" INST [ "else" INST ]
WHILE -> "while" COND "do" INST
REPEAT -> "repeat" INST "until" COND
FOR -> "for" ID ":=" EXPR ("to" | "downto") EXPR "do" INST
CASE -> "case" EXPR "of" CASE_ELEMENT { ";" CASE_ELEMENT } [ "else" INST ] "end"
CASE_ELEMENT -> NUM ":" INST
WRITE -> "write" "(" EXPR_LIST ")"
READ-> "read" "(" IDENT_LIST ")"
CALL -> ID "(" [ ARG_LIST ] ")"
```

Expressions et Conditions

```
 COND -> EXPR RELOP EXPR
RELOP -> "=" | "<>" | "<" | ">" | "<=" | ">="
EXPR -> TERM { ADDOP TERM }
ADDOP -> "+" | "-"
TERM -> FACT { MULOP FACT }
MULOP -> "*" | "/"
FACT -> ID [ "(" EXPR { "," EXPR } ")" ]
      | NUM
      | REAL
      | "(" EXPR ")"
```

Liste et Types de Base

```
 EXPR_LIST -> EXPR { "," EXPR }
IDENT_LIST -> ID { "," ID }
ARG_LIST -> EXPR { "," EXPR }
ARRAY_LIST -> ID "=" "[" REAL { "," REAL } "]"
BASE_TYPE -> "integer" | "real" | "boolean" | "string" | "array"
```

Définitions Lexicales

```
 ID -> lettre { lettre | chiffre | "_" }
NUM -> chiffre { chiffre }
REAL -> NUM "." NUM

lettre -> "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"
chiffre -> "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```