Chapitre : Conception

1. choix de l'approche AST -> php

Jérôme Vouillon et Vincent Balat dans leur projet From bytecode to javascript : the js_of_ocaml Compiler, En partant du bytecode pour générer du code javascript était un choix judicieux car les machines virtuelles offrent des API stables, il n'est pas nécessaire de modifier le compilateur à chaque nouvelle version du langage de programmation, php_of_ocaml arrive avec une autre approche en basant sur la chaîne de compilation d'un programme Ocaml, à partir de l'AST d'un programme Ocaml on va générer un programme PHP équivalent sémantiquement et qu'il soit lisible pour des programmeurs PHP.

2. Grammaire du langage :

pseudo-Ocaml :
nous avons opté pour la traduction de certains traits du langage Ocaml on citera ci-dessous la grammaire du langage :

```
NUMS = [0-9]+ ("." [0-9]+)?
IDENT = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9']*

TYPE ::= INT
        | FLOAT
        | STRING
        | CHAR

STATM ::= 'let' IDENT '=' EXPR
        | IDENT '=' NUMS

EXPR ::= STATM
        | IDENT ':' TYPE
        | 'if' EXPR 'then' EXPR 'else' EXPR (optional)
        | 'for' IDENT '=' EXPR 'to' EXPR 'do' ( EXPR )* 'done'
        | 'for' IDENT '=' EXPR 'downto' EXPR 'do' (EXPR)+ 'done'
        | 'while' BOOL 'do' (EXPR)+ 'done'
        | IDENT
        | CONSTRUCT
        | NUMS
        | FUNC

CONSTRUCT ::= TRUE
            | FALSE
            | '[]'
            | '[' ( IDENT )+  (';' IDENT )* ']'
            | '[' ( NUMS )+  (';' NUMS )* ']'
            | CONSTRUCT :: CONSTRUCT * :: '[' CONSTRUCT ']'
```

le type de CONSTRUCT doit être le même l'expression CONSTRUCT :: CONSTRUCT * :: '[' CONSTRUCT ']'

FUNCT ::= 'let' 'rec'? IDENT (' ' IDENT)* '=' EXPR

3. Squelette de traduction :

- Traduction d'une affectation :

let IDENT = EXPR
$[<IDENT>] = [EXPR]

- Traduction de la boucle FOR :

for IDENT = EXPR to EXPR do (EXPR)* done
for '(' $[<IDENT>] = [EXPR] ; $[<IDENT>] '<=' EXPR; $[<IDENT>]'++')' '{' (EXPR)* '}'

- Traduction de la boucle WHILE :

WHILE (EXPR) do (EXPR)* done
do '{' [<EXPR>] '}' WHILE '('[<EXPR>]')'

- Traduction des ARRAY :

let IDENT = '[|' (IDENT || NUMS)? (';' (IDENT || NUMS)) * '|]'
$[<IDENT>] = array '(' ([<IDENT>] || [<NUMS>])? (',' ([<IDENT>] || [<NUMS>])) * ')'

- Traduction des TUPLE :

let IDENT = '[|' (IDENT || NUMS)? (',' (IDENT || NUMS)) * '|]'
$[<IDENT>] = array '(' ([<IDENT>] || [<NUMS>])? (',' ([<IDENT>] || [<NUMS>])) * ')'

- Traduction des fonction :

let 'rec'? IDENT (' ' IDENT)* = EXPR
function [<IDENT>] '(' [<IDENT>] ',' ([<IDENT>])* ')' '{' [<EXPR>] '}'