Table des matières

Van De Walle Bernard
Francois Thibault Van Der Essen Frédéric

INGI2132 : Langages et traducteurs

Rapport intermédiaire 2

Prof. B. Le Charlier

Introduction

Syntaxe concraites

Nous avons définit un premier lieu cette syntaxe :

```
<Reserved_words>    ::= 'set' | 'if' | 'while' | 'write' | 'fun' | 'method'
      | 'null' | 'true' | 'false' | 'read' | 'new' | '+' | '-' | '*' | '/'
      | '%' | '|' | '&' | '!' | '<' | '>' | '<=' | '>=' | 'neg' | 'this'
      | 'return' | 'skip'
<Reserved_caracter> ::= <o_brackets> | <c_brackets> | <.>
<(> ::= '('
<)> ::= ')'
<.> ::= '.'
<digit> ::= '0..9'
<pnumber> ::=  <digit> | <number> <digit>
<number>  ::= '-' <pnumber> | <pnumber>
<caracter> ::= 'a..z' | 'A..Z' | '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*'
| '{' | '}' | '|' | '_' | ',' | '?' | '-' | '+' | '=' | '/'
| '\\' | '>' | '<' | ':' | ';' | '~' | '\'' | '"'
<sp> ::= 'lambda' | ' ' | 'NewLine' | 'Tab' | <sp> <sp>
<esp> ::= ' ' <sp>
<Id> ::= <caracter> | <Id> <caracter> | <Id> <digit>
<meth_or_fun> ::= <method> | <function>
<Program> ::=  <meth_or_fun> | <Program> <sp>  <meth_or_fun> <esp>
<function> ::= <(><sp> 'fun' <sp> <(> <sp> <Name> <esp> <arglist> <sp><)>
<sp> <instr_list> <sp><)>
<arglist> ::= <Id> | <Id> <esp> <arglist>
<methode> ::= <(><sp> 'method' <sp> <(><sp> <Number> <sp><)> <sp>
      <(> <sp> <Name> <esp> <arglist> <sp> <)> <instr_list>  <sp> <)>
<instr_list> ::= <instr> | <(> <sp> <instr_list_np> <sp> <)>
<instr_list_np> ::= <instr> | <instr> <esp> <instr_list_np>
<instr> ::= <conditional> | <while_block> | <call>
  | <(> <sp> 'return' <esp> <expr> <sp> <)>
  | <(><sp> 'skip' <sp><)>
<conditional> ::= <(><sp> 'if' <esp> <expr> <esp> <instr_list>
  <esp> <instr_list> <sp> <)>
<while_block ::= <(> <sp> 'while' <esp> <expr> <esp> <instr_list> <sp> <)>
<call> ::= <function_call> | <method_call> | <builtin_call>
<builtin_call> ::= <assignment> | <read_call> | <write_call> | <arithmetic>
<assignment> ::= <(> <sp> 'set' <esp> <left_Id> <esp> <expr> <sp> <)>
<read_call>  ::= <(> <sp> read <sp> <)>
<write_call> ::= <(> <sp> write <esp> <expr> <)>
<arithmetic> ::= <binary> | <unary>
```

```
<binary ::= <(> <sp> <bin_id> <esp> <expr> <esp> <expr> <sp> <)>
<bin_id> ::=  '+' | '-' | '*' | '/' | '%' | '|' | '&' | '<' | '>' | '<=' | '>='
<unary>  ::=  <(> <sp> <un_id> <esp> <expr> <sp> <)>
<un_id>  ::=  '!' | 'neg' | 'new'
<left_Id> ::= <Id> | <Id> <.> <expr> | 'this' <.>  <expr>
<user_call>  ::= <(> <sp> <Id> <esp> <expr_list_np> <sp> <)>
<expr_list_np> ::= 'lambda' | <expr> | <expr> <esp> <expr_list_np>
<method_call>  ::= <(> <sp> <method_id> <esp> <expr_list_np> <sp> <)>
<method_id> ::= <Id> <.> <Id> | 'super' <.> <Id>
<expr> ::= <call> | <left_Id> | <number> | 'null' | 'true' | 'false' | 'this'
```

Vu que cette syntaxe n'utilise que les parenthèses et le points comme caractère réservés, nous pouvons utiliser n'importe caractère pour définir un identifiant (pour peut qu'il ne commence pas par un chiffre). Nous pouvons donc donner des noms tel que : $-D$ ou encore ;} voir : $->$. Nous avons donc choisit d'appeler notre langage le langage Happy.

Les commentaires sont entre [] : [mon commentaire]