

Sucuri

Uma linguagem baseada em Python

João Paulo T. I. Z., Ranieri S. A., William K. A.

22 de Agosto de 2017

A linguagem

A linguagem é planejada tendo como base algumas ideias de Python, Javascript e Haskell. Para geração do analisador léxico, foi utilizada as ferramentas FLEX (para especificação do léxico) e BISON (para gerar o código-fonte do analisador).

Exemplo de código válido na linguagem Sucuri:

```
# Geometry example module .

# Simple Point class
export class Point
  let x = 0
  let y = 0

  let new(self, x, y)
    self.x = x
    self.y = y

  let __sub__(self, b)
    return Vector(b.x - self.x, b.y - self.y)

# Alias example
export let Vector = Point

# Distance from a to b
export let distance(a, b)
  return b - a

# Simple Rectangle class
export class Rectangle
  let top_left = Point(0, 0)
  let bottom_right = Point(0, 0)

  let new(self, top_left, bottom_right)
    self.top_left = top_left
    self.bottom_right = bottom_right

  let width(self)
    return (self.bottom_right - self.top_left).x

  let height(self)
    return (self.bottom_right - self.top_left).y
```

Especificação Léxica

Inicialmente são definidas algumas regex de apoio:

```
D [0-9] % Reconhece dígitos

L [a-zA-Z_!@${?}] % Reconhece qualquer símbolo
                  % possível em um identificador

NO_SQUOTE_STRING_LITERAL [^']* % Qualquer _string literal_
                              % que não possua aspas simples

NO_DQUOTE_STRING_LITERAL [^"]* % Qualquer _string literal_
                              % que não possua aspas duplas
```

Além de duas funções, `count()`, que realiza contagem de colunas para gerar uma melhor mensagem de erro (caso ocorra), e `indent_level()`, que informa o nível de indentação atual.

Identificadores

Identificadores são compostos por qualquer sequência de L ou D não separados por espaços, podendo conter “.” (não no início, no final nem sucedidos por dígitos).

Literais

São assumidos como literais de inteiros qualquer construção de somente dígitos:

Inteiros válidos:

```
1
10
0
0000 % Tratado como 0
300
-10 % É reconhecido o "10" como literal inteiro e o "-" como operador unário
    % operado sobre o "10"
```

Assim, sua *regex* se torna `{D}+` (1 ou mais dígitos consecutivos).

São assumidos como ponto-flutuante todo literal composto por números e que tenha um “.” no início, meio ou fim do literal:

```
1 % Inteiro
1. % Float
.1 % Float
-1. % "1." reconhecido como literal float, unário "-" operado em "1."

. % Erro léxico
```

Assim, sua *regex* é separada em duas:

- `"{D}+` — Reconhece *floats* iniciados em “.”;
- `{D}+"{D}*` — Reconhece *floats* com “.” no meio ou final;

String literals são compostos de qualquer sequência de caracteres que:

- Estão entre aspas simples (') e não possuem outra aspa simples no meio (reconhecido pela *regex* `''{NO_SQUOTE_STRING_LITERAL}''`).
- Estão entre aspas dupla (") e não possuem outra aspa dupla no meio (reconhecido pela *regex* `\"{NO_DQUOTE_STRING_LITERAL}\"`).

Operadores:

% Unários

not

-

% Comparativos

!=

=

<

<=

>

>=

% Matemáticos

+

-

*

**

/

% Lógicos

and

or

xor

% Outros

(

)

Palavras reservadas:

% Estruturas de controle

class % Define um novo tipo

if

else

for

while

% Retornos

return

throw

as % Serve para alias

export % Define o elemento como público

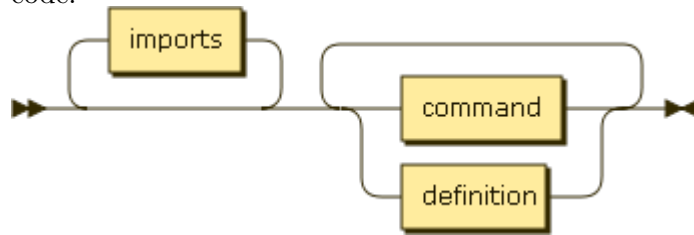
from % Para importação parcial de um módulo

```
import % Para importar um módulo  
in      % Para iteração (for i in set)  
let      % Definição
```

Há também a definição de `ellipse (...)` para parâmetros variádicos.

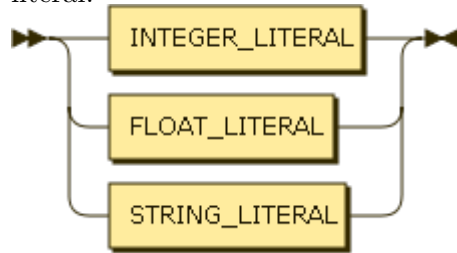
Grafo de sintaxe e especificação EBNF

code:



code ::= imports* (command | definition)+
no references

literal:

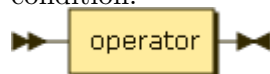


literal ::= INTEGER_LITERAL
| FLOAT_LITERAL
| STRING_LITERAL

referenced by:

- operator

condition:



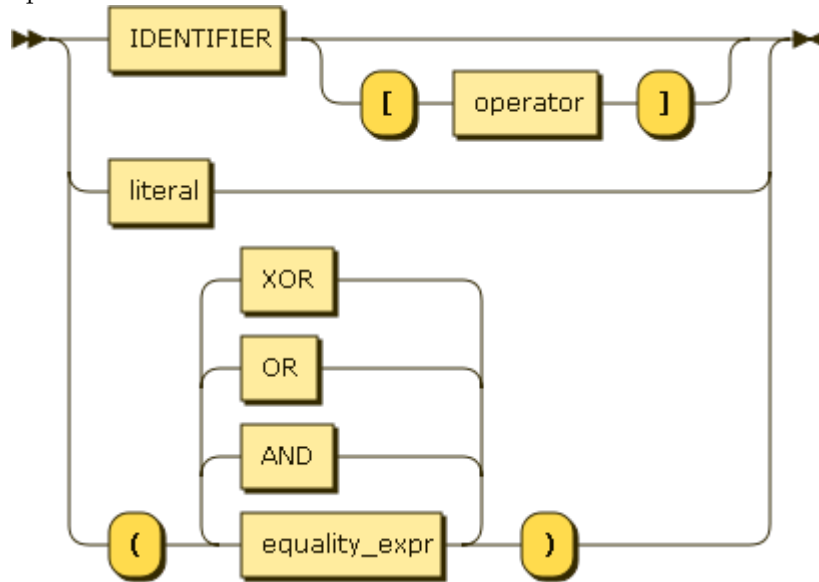
condition

::= operator

referenced by:

- if_statement
- while_statement

operator:

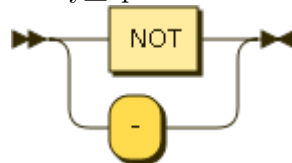


```
operator ::= IDENTIFIER ( '[' operator ']' )?
          | literal
          | '(' equality_expr ( ( AND | OR | XOR ) equality_expr )* ')'
```

referenced by:

- assignment_expr
- attr_decl
- command
- condition
- for_statement
- function_call
- function_params_list
- operator
- unary_expr

unary_operator:

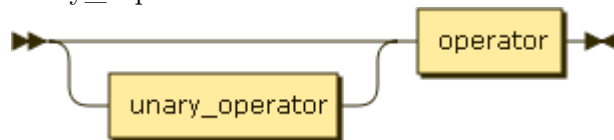


```
unary_operator
    ::= NOT
    | '-'
```

referenced by:

- unary_expr

unary_expr:

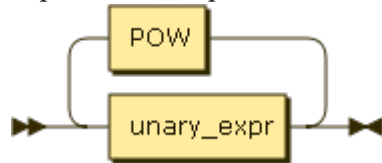


```
unary_expr
    ::= unary_operator? operator
```

referenced by:

- exponential_expr

exponential_expr:



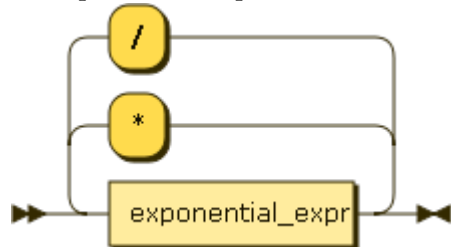
exponential_expr

$::= \text{unary_expr (POW unary_expr)}^*$

referenced by:

- multiplicative_expr

multiplicative_expr:



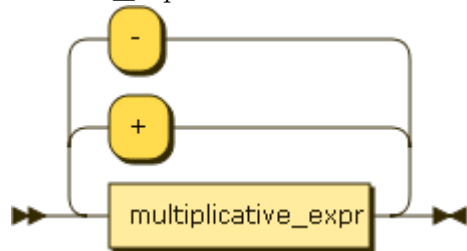
multiplicative_expr

$::= \text{exponential_expr (('*' | '/') exponential_expr)}^*$

referenced by:

- additive_expr

additive_expr:



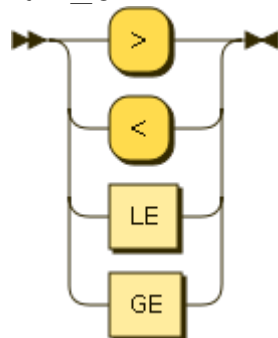
additive_expr

$::= \text{multiplicative_expr (('+' | '-') multiplicative_expr)}^*$

referenced by:

- relational_expr

REL_OP:



REL_OP $::=$ `'>'`
 | `'<'`
 | `LE`
 | `GE`

referenced by:

- relational_expr

relational_expr:



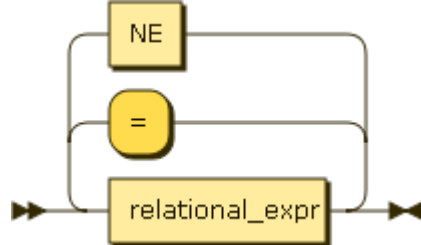
relational_expr

`::= additive_expr (REL_OP additive_expr)*`

referenced by:

- equality_expr

equality_expr:



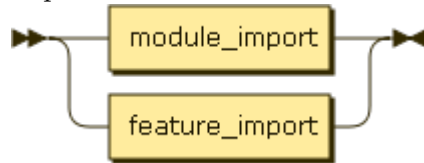
equality_expr

`::= relational_expr (('=' | NE) relational_expr)*`

referenced by:

- operator

imports:

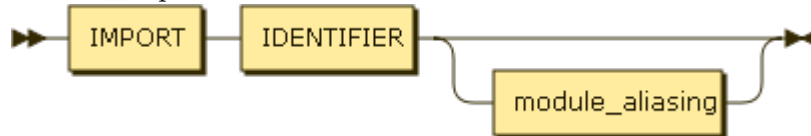


imports `::= module_import`
`| feature_import`

referenced by:

- code

module_import:



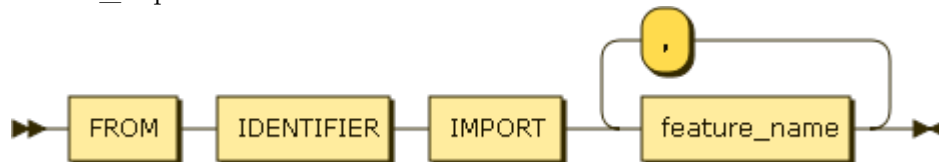
module_import

`::= IMPORT IDENTIFIER module_aliasing?`

referenced by:

- imports

feature_import:



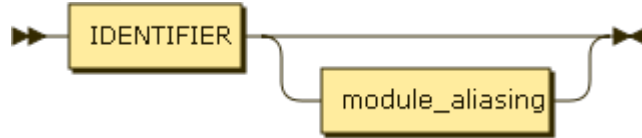
feature_import

`::= FROM IDENTIFIER IMPORT feature_name (',' feature_name)*`

referenced by:

- imports

feature_name:



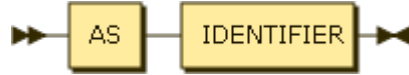
feature_name

::= IDENTIFIER module_aliasing?

referenced by:

- feature_import

module_aliasing:



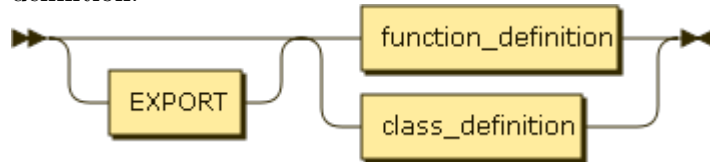
module_aliasing

::= AS IDENTIFIER

referenced by:

- feature_name
- module_import

definition:



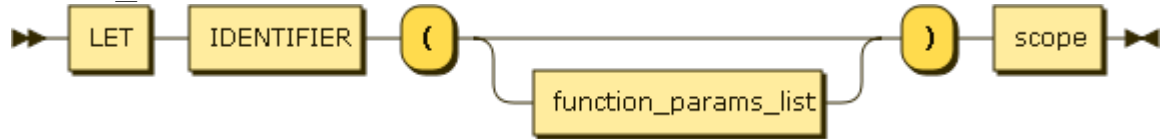
definition

::= EXPORT? (function_definition | class_definition)

referenced by:

- code

function_definition:



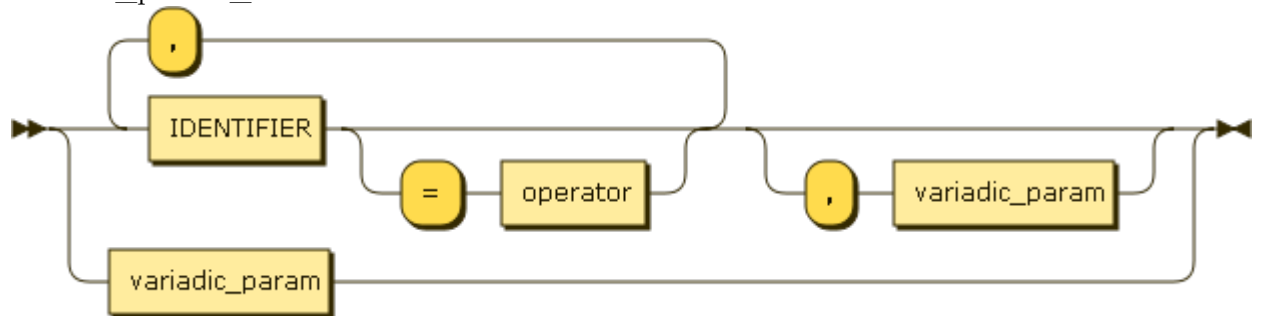
function_definition

::= LET IDENTIFIER '(' function_params_list? ')' scope

referenced by:

- class_scope
- definition

function_params_list:



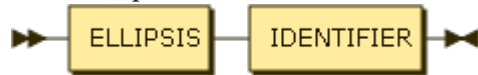
function_params_list

::= IDENTIFIER ('=' operator)? (',' IDENTIFIER ('=' operator)?)* (',' variadic_param | variadic_param

referenced by:

- function_definition

variadic_param:



variadic_param

::= ELLIPSIS IDENTIFIER

referenced by:

- function_params_list

scope:



scope ::= LINE_BREAK INDENT command+ DEDENT

referenced by:

- for_statement
- function_definition
- if_statement
- while_statement

class_definition:



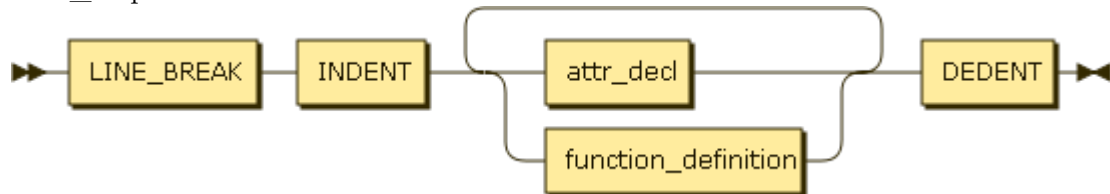
class_definition

::= CLASS IDENTIFIER class_scope

referenced by:

- definition

class_scope:



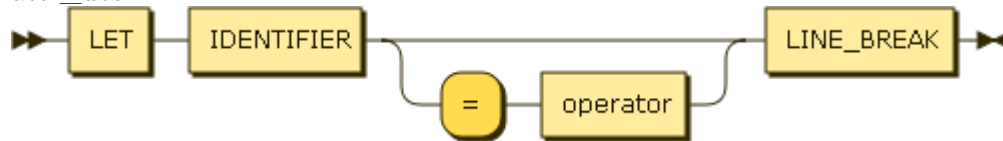
class_scope

::= LINE_BREAK INDENT (attr_decl | function_definition)+ DEDENT

referenced by:

- class_definition

attr_decl:



attr_decl

::= LET IDENTIFIER ('=' operator)? LINE_BREAK

referenced by:

- class_scope

```

graph LR
    A[assignment_expr] --- J1(( ))
    J1 --- J2(( ))
    J2 --- J3(( ))
    J3 --- J4(( ))
    J4 --- J5(( ))
    J5 --- J6(( ))
    J6 --- J7(( ))
    J7 --- J8(( ))
    J8 --- J9(( ))
    J9 --- J10(( ))
    J10 --- J11(( ))
    J11 --- J12(( ))
    J12 --- J13(( ))
    J13 --- J14(( ))
    J14 --- J15(( ))
    J15 --- J16(( ))
    J16 --- J17(( ))
    J17 --- J18(( ))
    J18 --- J19(( ))
    J19 --- J20(( ))
    J20 --- J21(( ))
    J21 --- J22(( ))
    J22 --- J23(( ))
    J23 --- J24(( ))
    J24 --- J25(( ))
    J25 --- J26(( ))
    J26 --- J27(( ))
    J27 --- J28(( ))
    J28 --- J29(( ))
    J29 --- J30(( ))
    J30 --- J31(( ))
    J31 --- J32(( ))
    J32 --- J33(( ))
    J33 --- J34(( ))
    J34 --- J35(( ))
    J35 --- J36(( ))
    J36 --- J37(( ))
    J37 --- J38(( ))
    J38 --- J39(( ))
    J39 --- J40(( ))
    J40 --- J41(( ))
    J41 --- J42(( ))
    J42 --- J43(( ))
    J43 --- J44(( ))
    J44 --- J45(( ))
    J45 --- J46(( ))
    J46 --- J47(( ))
    J47 --- J48(( ))
    J48 --- J49(( ))
    J49 --- J50(( ))
    J50 --- J51(( ))
    J51 --- J52(( ))
    J52 --- J53(( ))
    J53 --- J54(( ))
    J54 --- J55(( ))
    J55 --- J56(( ))
    J56 --- J57(( ))
    J57 --- J58(( ))
    J58 --- J59(( ))
    J59 --- J60(( ))
    J60 --- J61(( ))
    J61 --- J62(( ))
    J62 --- J63(( ))
    J63 --- J64(( ))
    J64 --- J65(( ))
    J65 --- J66(( ))
    J66 --- J67(( ))
    J67 --- J68(( ))
    J68 --- J69(( ))
    J69 --- J70(( ))
    J70 --- J71(( ))
    J71 --- J72(( ))
    J72 --- J73(( ))
    J73 --- J74(( ))
    J74 --- J75(( ))
    J75 --- J76(( ))
    J76 --- J77(( ))
    J77 --- J78(( ))
    J78 --- J79(( ))
    J79 --- J80(( ))
    J80 --- J81(( ))
    J81 --- J82(( ))
    J82 --- J83(( ))
    J83 --- J84(( ))
    J84 --- J85(( ))
    J85 --- J86(( ))
    J86 --- J87(( ))
    J87 --- J88(( ))
    J88 --- J89(( ))
    J89 --- J90(( ))
    J90 --- J91(( ))
    J91 --- J92(( ))
    J92 --- J93(( ))
    J93 --- J94(( ))
    J94 --- J95(( ))
    J95 --- J96(( ))
    J96 --- J97(( ))
    J97 --- J98(( ))
    J98 --- J99(( ))
    J99 --- J100(( ))
    J100 --- J101(( ))
    J101 --- J102(( ))
    J102 --- J103(( ))
    J103 --- J104(( ))
    J104 --- J105(( ))
    J105 --- J106(( ))
    J106 --- J107(( ))
    J107 --- J108(( ))
    J108 --- J109(( ))
    J109 --- J110(( ))
    J110 --- J111(( ))
    J111 --- J112(( ))
    J112 --- J113(( ))
    J113 --- J114(( ))
    J114 --- J115(( ))
    J115 --- J116(( ))
    J116 --- J117(( ))
    J117 --- J118(( ))
    J118 --- J119(( ))
    J119 --- J120(( ))
    J120 --- J121(( ))
    J121 --- J122(( ))
    J122 --- J123(( ))
    J123 --- J124(( ))
    J124 --- J125(( ))
    J125 --- J126(( ))
    J126 --- J127(( ))
    J127 --- J128(( ))
    J128 --- J129(( ))
    J129 --- J130(( ))
    J130 --- J131(( ))
    J131 --- J132(( ))
    J132 --- J133(( ))
    J133 --- J134(( ))
    J134 --- J135(( ))
    J135 --- J136(( ))
    J136 --- J137(( ))
    J137 --- J138(( ))
    J138 --- J139(( ))
    J139 --- J140(( ))
    J140 --- J141(( ))
    J141 --- J142(( ))
    J142 --- J143(( ))
    J143 --- J144(( ))
    J144 --- J145(( ))
    J145 --- J146(( ))
    J146 --- J147(( ))
    J147 --- J148(( ))
    J148 --- J149(( ))
    J149 --- J150(( ))
    J150 --- J151(( ))
    J151 --- J152(( ))
    J152 --- J153(( ))
    J153 --- J154(( ))
    J154 --- J155(( ))
    J155 --- J156(( ))
    J156 --- J157(( ))
    J157 --- J158(( ))
    J158 --- J159(( ))
    J159 --- J160(( ))
    J160 --- J161(( ))
    J161 --- J162(( ))
    J162 --- J163(( ))
    J163 --- J164(( ))
    J164 --- J165(( ))
    J165 --- J166(( ))
    J166 --- J167(( ))
    J167 --- J168(( ))
    J168 --- J169(( ))
    J169 --- J170(( ))
    J170 --- J171(( ))
    J171 --- J172(( ))
    J172 --- J173(( ))
    J173 --- J174(( ))
    J174 --- J175(( ))
    J175 --- J176(( ))
    J176 --- J177(( ))
    J177 --- J178(( ))
    J178 --- J179(( ))
    J179 --- J180(( ))
    J180 --- J181(( ))
    J181 --- J182(( ))
    J182 --- J183(( ))
    J183 --- J184(( ))
    J184 --- J185(( ))
    J185 --- J186(( ))
    J186 --- J187(( ))
    J187 --- J188(( ))
    J188 --- J189(( ))
    J189 --- J190(( ))
    J190 --- J191(( ))
    J191 --- J192(( ))
    J192 --- J193(( ))
    J193 --- J194(( ))
    J194 --- J195(( ))
    J195 --- J196(( ))
    J196 --- J197(( ))
    J197 --- J198(( ))
    J198 --- J199(( ))
    J199 --- J200(( ))
    J200 --- J201(( ))
    J201 --- J202(( ))
    J202 --- J203(( ))
    J203 --- J204(( ))
    J204 --- J205(( ))
    J205 --- J206(( ))
    J206 --- J207(( ))
    J207 --- J208(( ))
    J208 --- J209(( ))
    J209 --- J210(( ))
    J210 --- J211(( ))
    J211 --- J212(( ))
    J212 --- J213(( ))
    J213 --- J214(( ))
    J214 --- J215(( ))
    J215 --- J216(( ))
    J216 --- J217(( ))
    J217 --- J218(( ))
    J218 --- J219(( ))
    J219 --- J220(( ))
    J220 --- J221(( ))
    J221 --- J222(( ))
    J222 --- J223(( ))
    J223 --- J224(( ))
    J224 --- J225(( ))
    J225 --- J226(( ))
    J226 --- J227(( ))
    J227 --- J228(( ))
    J228 --- J229(( ))
    J229 --- J230(( ))
    J230 --- J231(( ))
    J231 --- J232(( ))
    J232 --- J233(( ))
    J233 --- J234(( ))
    J234 --- J235(( ))
    J235 --- J236(( ))
    J236 --- J237(( ))
    J237 --- J238(( ))
    J238 --- J239(( ))
    J239 --- J240(( ))
    J240 --- J241(( ))
    J241 --- J242(( ))
    J242 --- J243(( ))
    J243 --- J244(( ))
    J244 --- J245(( ))
    J245 --- J246(( ))
    J246 --- J247(( ))
    J247 --- J248(( ))
    J248 --- J249(( ))
    J249 --- J250(( ))
    J250 --- J251(( ))
    J251 --- J252(( ))
    J252 --- J253(( ))
    J253 --- J254(( ))
    J254 --- J255(( ))
    J255 --- J256(( ))
    J256 --- J257(( ))
    J257 --- J258(( ))
    J258 --- J259(( ))
    J259 --- J260(( ))
    J260 --- J261(( ))
    J261 --- J262(( ))
    J262 --- J263(( ))
    J263 --- J264(( ))
    J264 --- J265(( ))
    J265 --- J266(( ))
    J266 --- J267(( ))
    J267 --- J268(( ))
    J268 --- J269(( ))
    J269 --- J270(( ))
    J270 --- J271(( ))
    J271 --- J272(( ))
    J272 --- J273(( ))
    J273 --- J274(( ))
    J274 --- J275(( ))
    J275 --- J276(( ))
    J276 --- J277(( ))
    J277 --- J278(( ))
    J278 --- J279(( ))
    J279 --- J280(( ))
    J280 --- J281(( ))
    J281 --- J282(( ))
    J282 --- J283(( ))
    J283 --- J284(( ))
    J284 --- J285(( ))
    J285 --- J286(( ))
    J286 --- J287(( ))
    J287 --- J288(( ))
    J288 --- J289(( ))
    J289 --- J290(( ))
    J290 --- J291(( ))
    J291 --- J292(( ))
    J292 --- J293(( ))
    J293 --- J294(( ))
    J294 --- J295(( ))
    J295 --- J296(( ))
    J296 --- J297(( ))
    J297 --- J298(( ))
    J298 --- J299(( ))
    J299 --- J300(( ))
    J300 --- J301(( ))
    J301 --- J302(( ))
    J302 --- J303(( ))
    J303 --- J304(( ))
    J304 --- J305(( ))
    J305 --- J306(( ))
    J306 --- J307(( ))
    J307 --- J308(( ))
    J308 --- J309(( ))
    J309 --- J310(( ))
    J310 --- J311(( ))
    J311 --- J312(( ))
    J312 --- J313(( ))
    J313 --- J314(( ))
    J314 --- J315(( ))
    J315 --- J316(( ))
    J316 --- J317(( ))
    J317 --- J318(( ))
    J318 --- J319(( ))
    J319 --- J320(( ))
    J320 --- J321(( ))
    J321 --- J322(( ))
    J322 --- J323(( ))
    J323 --- J324(( ))
    J324 --- J325(( ))
    J325 --- J326(( ))
    J326 --- J327
```

- code
- scope

```

graph LR
    S1[LET IDENTIFIER] --> S2[IDENTIFIER]
    S2 --> S3[IDENTIFIER [= operator]]
    S3 --> S4[IDENTIFIER [operator]]
    S4 --> S5[ ]
  
```

- command

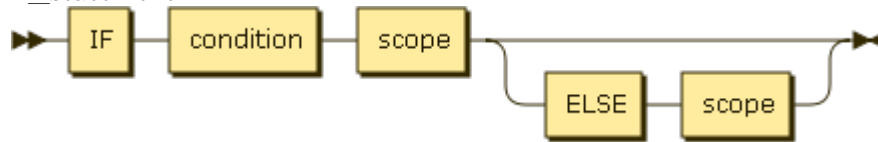
A diagram illustrating a function call grammar rule. It consists of a sequence of tokens: an identifier, an opening parenthesis '(', an operator, and a closing parenthesis ')'. The tokens are represented by yellow boxes and circles. The identifier is a rectangle, while the parentheses and operator are circles. Arrows indicate the flow from left to right. A curved arrow points from the opening parenthesis to the operator, and another curved arrow points from the operator to the closing parenthesis, suggesting a loop or a specific relationship between these elements.

- command

referenced by:

- `command`

if_statement:



if_statement

::= IF condition scope (ELSE scope)?

referenced by:

- statement

while_statement:



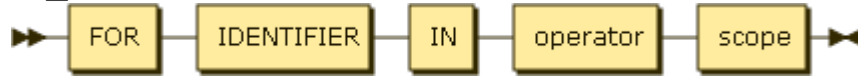
while_statement

::= WHILE condition scope

referenced by:

- statement

for_statement:



for_statement

::= FOR IDENTIFIER IN operator scope

referenced by:

- statement
-

Arquivos

Os arquivos FLEX e BISON são respectivamente `sucuri.l` e `sucuri.y`. Exemplos de programas válidos se encontram na pasta `examples/`. O código fonte do analisador é `sucuri.yy.c`. Os logs de saída aplicados no exemplo `examples/geometry.scr` estão no arquivo `geometry-parse.ylog`.