

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

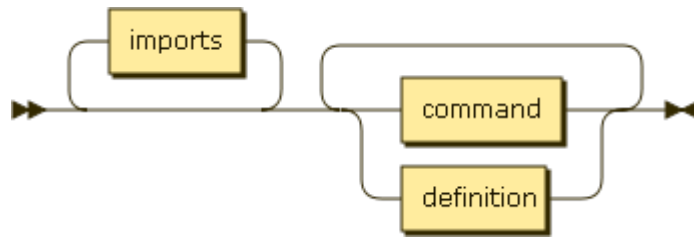
from % Para importação parcial de um módulo

```
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

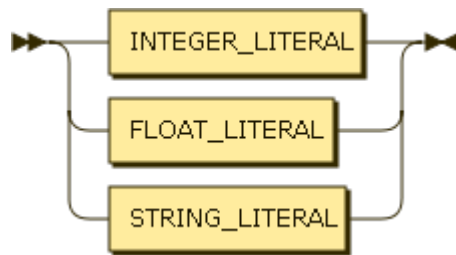
code:



```
code ::= imports* ( command | definition )+
```

Sem referências

literal:

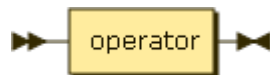


```
literal ::= INTEGER_LITERAL  
         | FLOAT_LITERAL  
         | STRING_LITERAL
```

Referenciado por:

- function_call
- operator

condition:

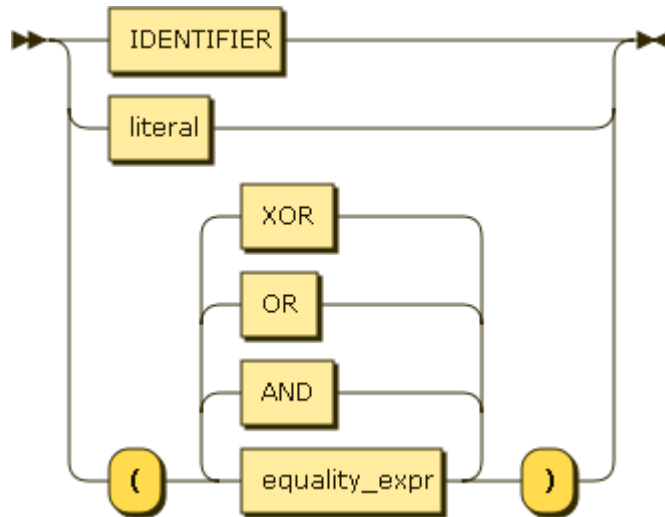


```
condition  
        ::= operator
```

Referenciado por:

- if_statement
- while_statement

operator:



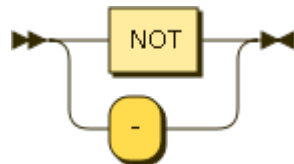
```

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

Referenciado por:

- assignment_expr
- attr_decl
- command
- condition
- for_statement
- unary_expr

unary_operator:



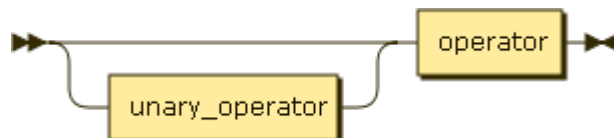
```

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

Referenciado por:

- unary_expr

unary_expr:



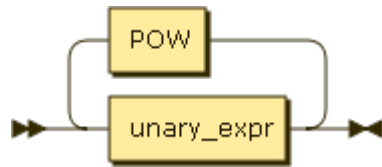
```

unary_expr
  ::= unary_operator? operator
  
```

Referenciado por:

- exponential_expr

exponential_expr:

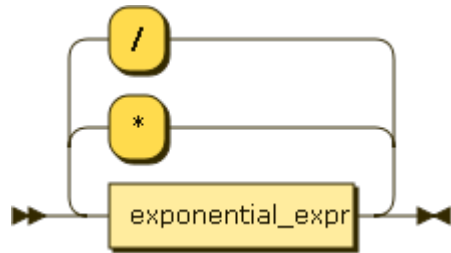


exponential_expr
::= unary_expr (POW unary_expr)*

Referenciado por:

- multiplicative_expr

multiplicative_expr:

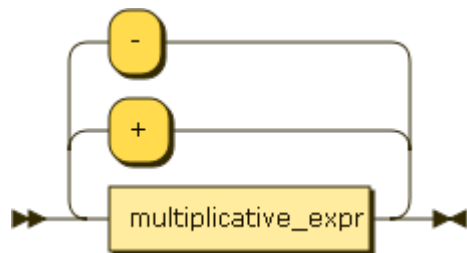


multiplicative_expr
::= exponential_expr (('*' | '/') exponential_expr)*

Referenciado por:

- additive_expr

additive_expr:

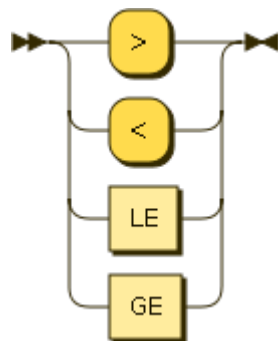


additive_expr
::= multiplicative_expr (('+' | '-') multiplicative_expr)*

Referenciado por:

- relational_expr

REL_OP:




```

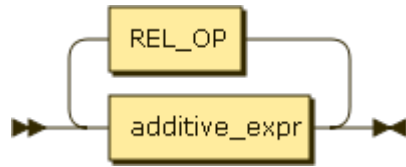
REL_OP    ::= '>'
           | '<'
           | LE
           | GE

```

Referenciado por:

- relational_expr

relational_expr:



```

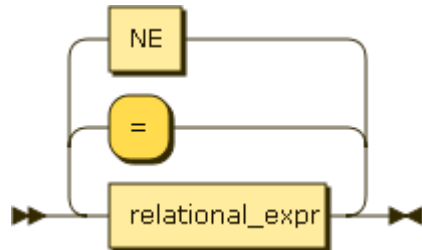
relational_expr
    ::= additive_expr ( REL_OP additive_expr )*

```

Referenciado por:

- equality_expr

equality_expr:



```

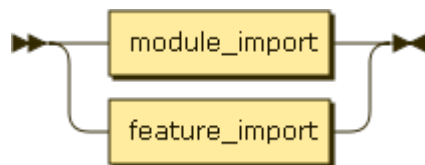
equality_expr
    ::= relational_expr ( ( '=' | NE ) relational_expr )*

```

Referenciado por:

- operator

imports:



```

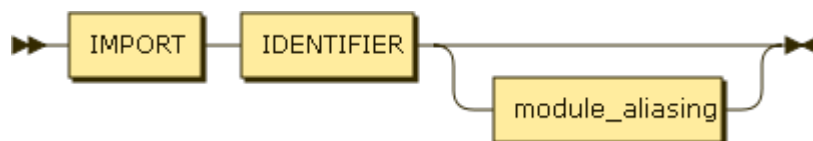
imports ::= module_import
         | feature_import

```

Referenciado por:

- code

module_import:



```

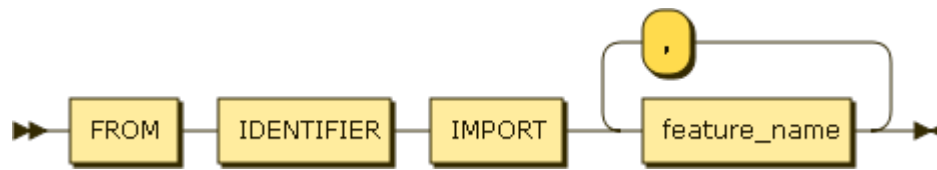
module_import
    ::= IMPORT IDENTIFIER module_aliasing?

```

Referenciado por:

- imports

feature_import:



```

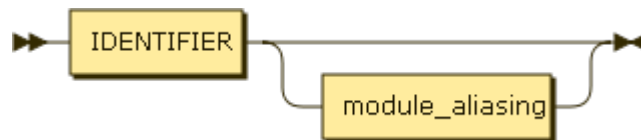
feature_import
    ::= FROM IDENTIFIER IMPORT feature_name ( ',' feature_name )*

```

Referenciado por:

- imports

feature_name:



```

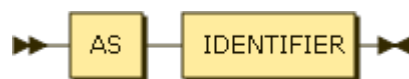
feature_name
    ::= IDENTIFIER module_aliasing?

```

Referenciado por:

- feature_import

module_aliasing:



```

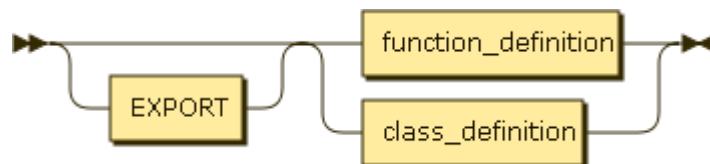
module_aliasing
    ::= AS IDENTIFIER

```

Referenciado por:

- feature_name
- module_import

definition:



```

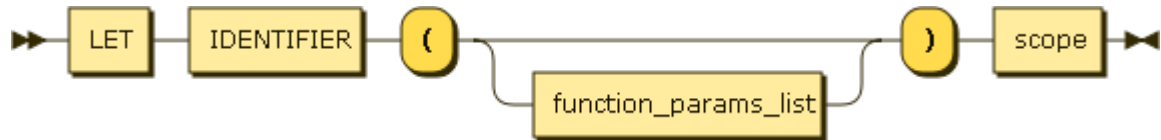
definition
    ::= EXPORT? ( function_definition | class_definition )

```

Referenciado por:

- code

function_definition:



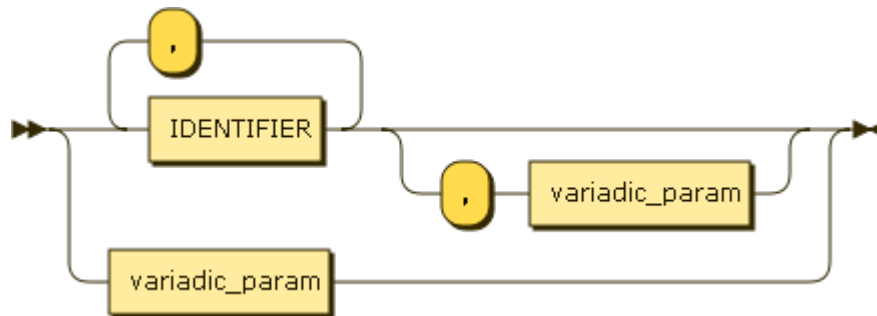
function_definition

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

Referenciado por:

- class_scope
- definition

function_params_list:



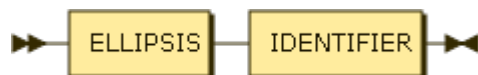
function_params_list

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

Referenciado por:

- function_definition

variadic_param:



variadic_param

::= ELLIPSIS IDENTIFIER

Referenciado por:

- function_params_list

scope:



scope ::= LINE_BREAK INDENT command+ DEDENT

Referenciado por:

- for_statement
- function_definition
- if_statement
- while_statement

class_definition:

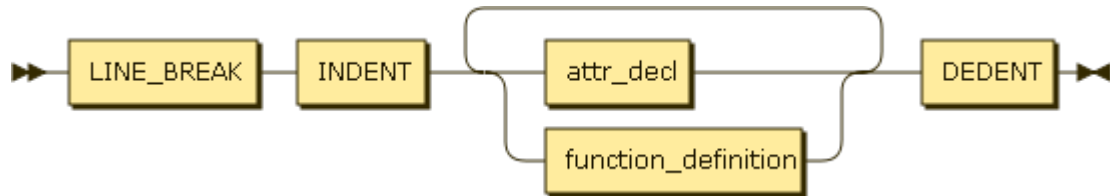


`class_definition`
 ::= `CLASS IDENTIFIER class_scope`

Referenciado por:

- `definition`

class_scope:

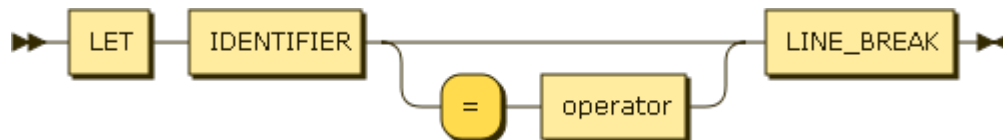


`class_scope`
 ::= `LINE_BREAK INDENT (attr_decl | function_definition)+ DEDENT`

Referenciado por:

- `class_definition`

attr_decl:

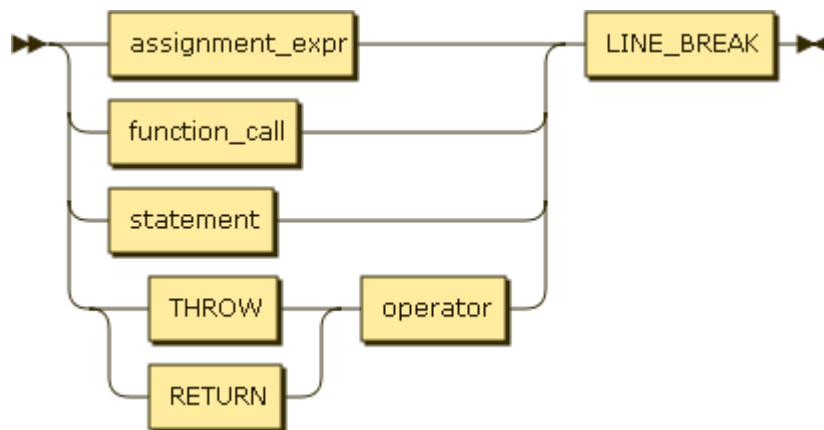


`attr_decl`
 ::= `LET IDENTIFIER ('=' operator)? LINE_BREAK`

Referenciado por:

- `class_scope`

command:



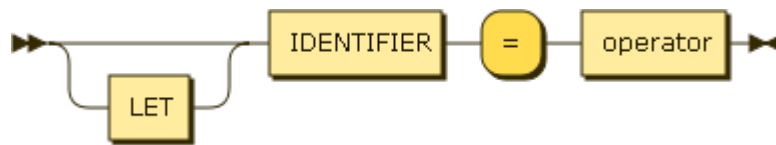
`command` ::= `(assignment_expr | function_call | statement | (THROW | RETURN) operator`

Referenciado por:

- `code`

- scope

assignment_expr:



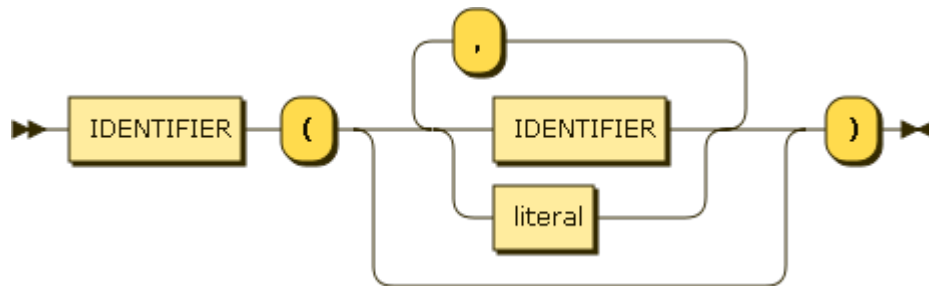
assignment_expr

::= LET? IDENTIFIER '=' operator

Referenciado por:

- command

function_call:



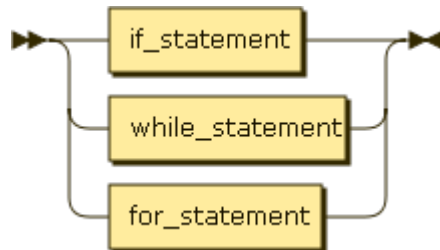
function_call

::= IDENTIFIER '(' ((IDENTIFIER | literal) (',' (IDENTIFIER | literal))) *

Referenciado por:

- command

statement:



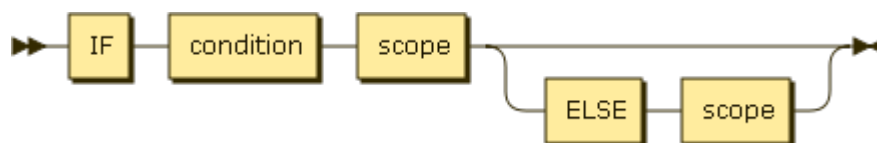
statement

::= if_statement
| while_statement
| for_statement

Referenciado por:

- command

if_statement:



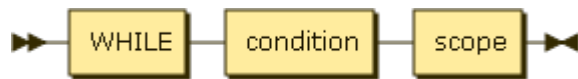
if_statement

`::= IF condition scope (ELSE scope)?`

Referenciado por:

- statement

while_statement:



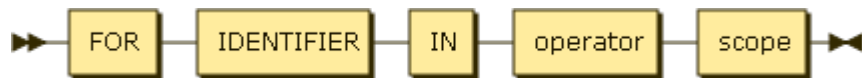
while_statement

`::= WHILE condition scope`

Referenciado por:

- statement

for_statement:



for_statement

`::= FOR IDENTIFIER IN operator scope`

Referenciado por:

- 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`.