INSTITUTO FEDERAL DO ESPÍRITO SANTO
CURSO SUPERIOR DE SISTEMAS DE INFORMAÇÃO

GUILHERME BODART DE OLIVEIRA
CASTRO
ANA CAROLINA CEBIN

TRABALHO FINAL - IMPLEMENTAÇÃO DE LINGUAGEM DE DOMÍNIO ESPECÍFICO (DSL)

Serra
2019

GUILHERME BODART DE OLIVEIRA
CASTRO
ANA CAROLINA CEBIN

TRABALHO FINAL - IMPLEMENTAÇÃO DE LINGUAGEM DE DOMÍNIO ESPECÍFICO (DSL)

Trabalho apresentado à disciplina de Linguagens Formais e Autômatos do Curso de Sistemas de Informação do Instituto Federal do Espírito Santo, Campus Serra, como requisito parcial de avaliação.

Orientador: Prof. Dr. Jefferson Oliveira Andrade

Serra
2019

SUMÁRIO

1 INTRODUÇÃO

Este trabalho consiste na implementação de uma Linguagem de Domínio Específico (DSL), utilizando a linguagem de programação Python e a ferramenta Lark

A DSL tem como objetivo fazer desenhos usando a biblioteca Turtle do Python, optamos por usar o Python por ter ferramentas à disposição que facilitaria o trabalho e o Lark pelo mesmo.

A DSL em questão permite utilizer nomeações de variáveis, além de construção de comandos de seleção IF-THEN-ELSE e repetição REPEAT. Permitindo também artefatos conhecidos como função, utilizados nas diversas outras linguagens de programação.

2 DEFINIÇÃO DA DSL

2.1 Gramática em EBNF

```
sttmt ::= (expr) ";"

expr ::= (assign)
        | (ifexpr)
        | (defun)
        | (atom)
        | (instruction)+
        | (conj)

instruction ::= MOVEMENT valor
            | "c" COLOR [COLOR]
            | "fill" (code_block)
            | "repeat" valor (code_block)
            | "reset"

paramlist ::= ((IDENT)? ("," (IDENT))*)

defun ::= "def" (IDENT) "(" (paramlist) ")" (expr)

ifexpr ::= "if" (conj) "then" (conj) ("else" (conj))?

code_block ::= "{" instruction+ "}"

conj ::= (variable|NUMBER) (OPERATOR (variable|NUMBER))*

rel ::= OPERATOR

MOVEMENT ::= "f"
            |"b"
            |"l"
            |"r"

OPERATOR ::= "=="
            | "!="
            | ">"
            | "=>"
            | "<"
            | "=<"

assign ::= "var"? (variable) "=" (expr)

atom ::= "-" (atom)
            | "not" (atom)
            | "(" (expr) ")"
            | NUMBER
            | (functioncall)
            | (variable)
```

functioncall::= (IDENT) (arglist)

arglist ::= "(" ((expr) ("," (expr))*)? ")"

valor ::= NUMBER|variable

variable ::= (IDENT)

IDENT ::= LETTER+

COLOR ::=LETTER+

*Obs: Algumas coisas foram definidas usando o próprio LARK, da biblioteca dele;

```
%import common.LETTER
%import common.INT -> NUMBER
%import common.WS
%ignore WS
```

2.2 Diagramas de Sintaxe

Figura 1 – arglist

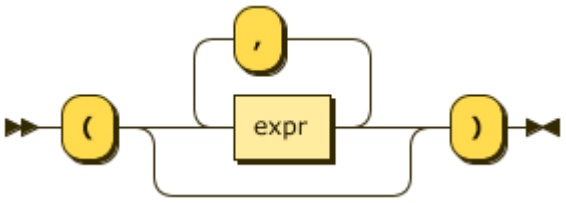

Figura 2 – assign

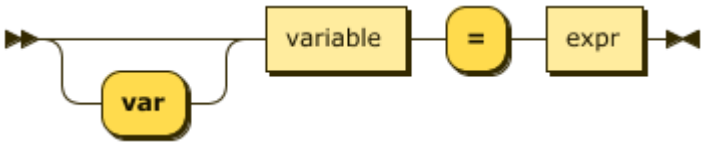

Figura 3 – code_block

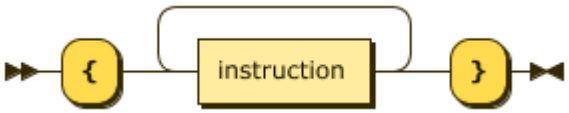

Figura 4 – conj

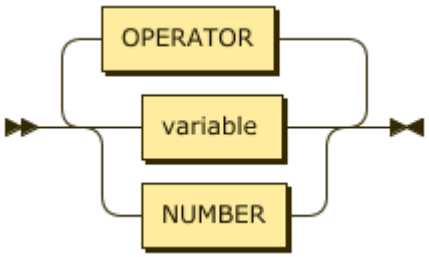

Figura 5 – defun

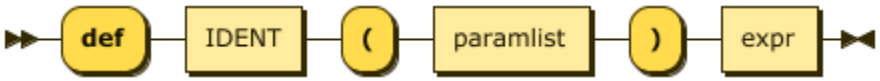

Figura 6 – IDENT

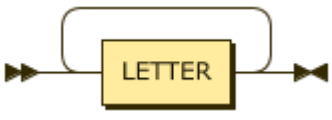

Figura 7 – functioncall

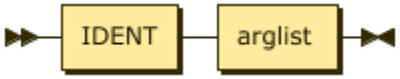

Figura 8 – ifexpr

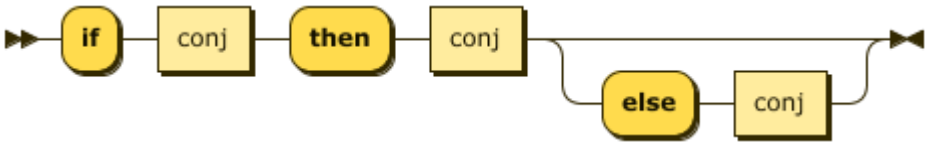

Figura 9 – expr

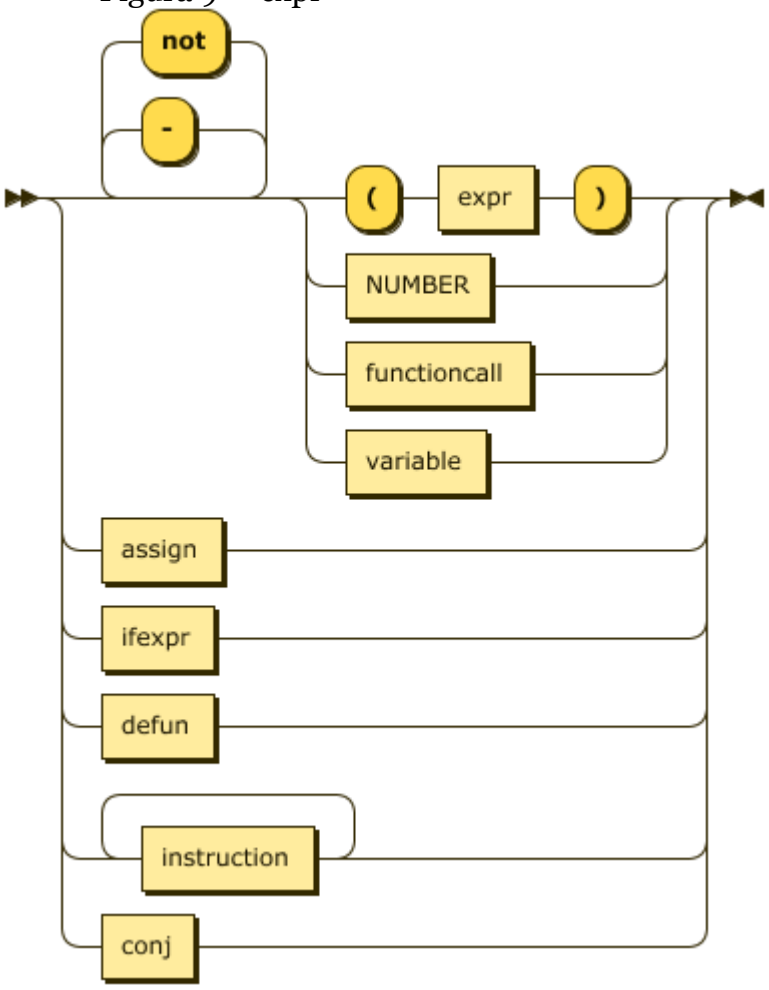

Figura 10 – instruction

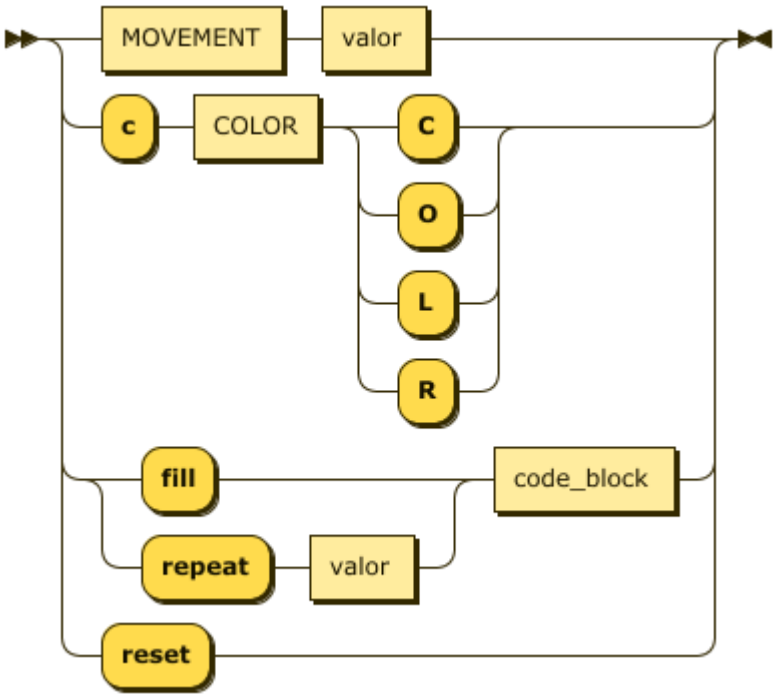

Figura 11 – MOVEMENT

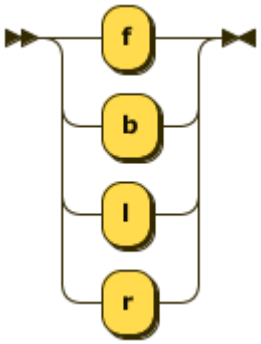

Figura 12 – OPERATOR

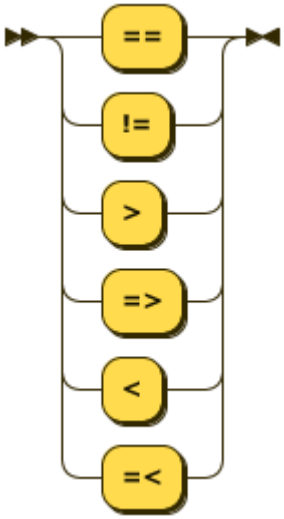

Figura 13 – paramList

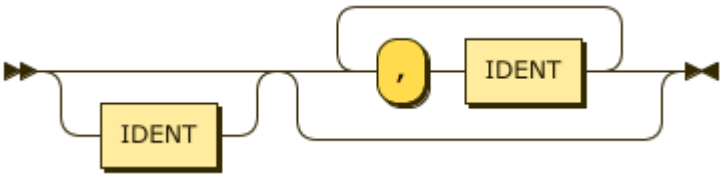

Figura 14 – rel

Figura 15 – sttmt

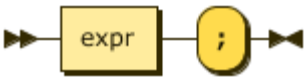

Figura 16 – valor

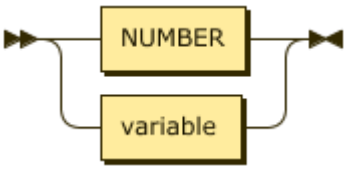

Figura 1 – variable

2.3 Exemplos de código

Quadro 1 – *Declaração de função e variáveis*

```
1   def funcao(a,b) expr;
2       a = 4;
3       b = 2;
4       def funcao() expr;
```

Quadro 2 – *Estrutura de Seleção*

```
1 if 2<3 then 2 else 3;
2 if a then a else b;
```

Quadro 3 – *Estrutura de Repetição*

1 repeat 50
2 repeat a (caso o "a" seja um número)
3 O repeat so pode, exclusivamente, ser usado dentro de uma instructrion, que é quem desenha, portanto não é possivel fazer uma função utilizado ela para que repita alguma funcionalidade;

Quadro 4 – *Estrutura de Função*

1 def funcao(a,b) if a<b then a else b;
2 def funcao(a,b,c) c red yellow fill { repeat a { f b l c}}
3 funcao(a,b,c) para utilizar a função criada

3 DEFINIÇÃO DA AST

3.1 Descrição das principais funções

Na elaboração da DSL presente neste trabalho não utilizamos orientação a objetos, portanto nosso programa não possui classes. Definimos funções em Python para lidar com o processamento adequado das cadeias de entrada de acordo com a gramática estabelecida.

- **Lark()** – Recebe a gramática analiza para usar futuramente;

- **parser.parse** – Recebe uma entrada e monta uma árvore de acordo com o que foi colocado na gramática, possíveis alterações dentro da própria gramática, como por exemplo, colocar “?” para não ler o caminha inteiro da árvore, ou “->” para mudar o nome que vai ser lido, ou atribuir o nome a um comando mais complexo;

4 CONCLUSÃO

Implementar uma Linguagem de Domínio Específico foi importante para o grupo pois assim conseguimos ver a dificuldade de fazer uma linguagem, e fizemos para uma DSL, que é de propósito reduzido comparado as outras linguagens, porém foi interessante ver, mesmo que nós tenhamos utilizado de uma biblioteca que nos ajude com a gramática, a dificuldade para fazer toda a lógica do programa ainda existia, já que fazer a gramática é a parte "fácil", mais decisões de projetos do que realmente saber fazer.

Com isso vimos que é importante antes de começar a fazer a linguagem definir muito bem para qual propósito ela será utilizada, e isso com qualquer programa que for feito posteriormente.

E o aprendizado que tiramos neste trabalho foi ver e fazer uma linguagem desde o princípio, que é o que a disciplina de LFA tenta mostrar, o funcionamento da lógica da programação como um todo.

REFERÊNCIAS

Lark: <https:// github.com/lark-parser/lark />
< https://lark-parser.readthedocs.io/en/latest/>