UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ – UNIOESTE

CAMPUS DE CASCAVEL

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS – CCET

DESCRIÇÃO DA LINGUAGEM QSL E ANALISADOR LÉXICO

BRUNO FELIPE DIEDRICH

CASCAVEL – PR
2021

1 – Especificação

A Linguagem Quenya Script Language (QSL). Quenya é uma língua presente no livro O Senhor dos Anéis, escrito por J.R.R Tolkien, é falada por elfos e alguns humanos nas terras chamadas Terra Média e Valinor. As pessoas que estudam Quenya por diversão, normalmente usam a língua para escrever poemas e canções porque é uma língua agradável e tem um som musical.

Helge K. Fauskanger é considerado um dos melhores pesquisadores das línguas de Tolkien hoje em dia, escreveu livros didáticos em Quenya e artigos sobre os idiomas criados por Tolkien. Portanto, foi utilizado o dicionário proposto por ele para a elaboração e tradução das palavras[1]. O “Script” vem da linguagem de programação Lua, pois foi a nossa linguagem adotada como base, e como Lua é uma linguagem de programação que suporta scripts, então nós adotamos o nome.

2 – Tipos de Dados Suportados

Existem três tipos básicos em QSL, sendo eles: **note**, **liltengwa** e **yulmavene**.

O tipo **note** representa os números (inteiros e floats).

O tipo **liltengwa** representa cadeias de caracteres.

O tipo **yulmavene** representa os valores *true* e *false*.

2.1 – note

Tipo: note	
Tamanho	4 bytes
Origem	Numérica
Descrição	Tipo inteiro e float
Operações Permitidas	+, -, *, /, ar, la, hela, helaer, ==, >=, <=, >, <
Chamada em Impressão	Nome da Variável
Exemplo	note var – declaração da variável sarme(var) – impressão da variável

2.2 – liltengwa

Tipo: liltengwa	
Tamanho	8 bits / 1 byte
Descrição	Representa cadeia de caracteres, pode conter caracteres de 8 bits, incluindo zeros
Origem	String
Operações Permitidas	+, -, *, /, ar, la, hela, helaer, ==, >=, <=, >, <
Chamada em Impressão	Nome da Variável
Exemplo	liltengwa var – declaração da variável sarme(var) – impressão da variável

2.3 – yulmavene

Tipo: yulmavene	
Tamanho	8 bits / 1 byte
Descrição	Representa os valores *true* e *false*
Origem	Booleano
Operações Permitidas	ar, la, hela, helaer, ==, >=, <=, >, <
Chamada em Impressão	Nome da Variável
Exemplo	yulmavene var – declaração da variável sarme(var) – impressão da variável

3 – Operadores Aritméticos Suportados

Na linguagem QSL, possuímos um total de quatro operadores aritméticos, onde os quatro são operadores binários: + (adição), - (subtração), * (multiplicação), / (divisão) e % (resto).

Operador	Operação	Sintaxe
+	Adição	A + B
-	Subtração	A - B
*	Multiplicação	A * B
/	Divisão	A / B
%	Resto da divisão	A % B

4 – Operadores Lógicos Suportados

Os operadores lógicos suportados em QSL são **ar** (AND), **la** (OR)**, hela** (NOT) e **healer** (XOR).

O operador de conjunção **ar** retorna seu primeiro argumento se este valor é ***false***, caso contrário, **ar** retorna seu segundo argumento. Utilizando dois operandos, retornará ***true*** se os valores forem ***true***.

O operador de negação **la** sempre retorna ***false*** ou ***true***. Inverte o valor booleano do operando.

O operador de disjunção **hela**, retorna seu primeiro argumento se o valor deste é diferente de ***false***, caso contrário, **hela** retorna o seu segundo argumento. Utilizando dois operadores retorna ***true*** se um dos valores lidos for ***true***.

O operador de disjunção **helaer**, retorna ***true*** apenas se os dois bits de entrada, os quais foram comparados forem diferentes. Utilizando dois operandos retorna ***true*** apenas se eles possuírem valores diferentes entre si, em caso dos valores serem iguais, retorna ***false***.

Operador	Operação	Sintaxe
ar	AND	A **ar** B
hela	OR	A **hela** B
la	NOT	**la** A
helaer	XOR	A **healer** B

5 – Operadores Relacionais Suportados

Em QSL, tais operadores sempre possuem como resultados *true* ou *false*. Os Operadores Relacionais suportados e utilizados pela linguagem QSL são:

Operador	Operação	Sintaxe
==	Igualdade: Retorna *true* se o elemento à esquerda e à direita são iguais.	A == B
>	Maior: Retorna *true* se o elemento a esquerda é maior que o elemento a direita.	A > B
<	Menor: Retorna *true* se o elemento a esquerda é menor que o elemento a direita.	A < B
>=	Maior ou Igual: Retorna *true* se o elemento a esquerda é ou maior, ou igual ao elemento a direita.	A >= B
<=	Menor ou Igual: Retorna *true* se o elemento a esquerda é ou menor, ou igual ao elemento a direita.	A <= B

Os operadores relacionais, suportam todos os três tipos existentes na linguagem: **note**, **liltengwa** e **yulmavene**.

6 – Delimitadores

Delimitadores são utilizados para separar partes de um código. Podem ser utilizados para delimitar blocos ou dar prioridade a operações.

Identificador	Identificação	Significado
()	Parênteses	Quando usado, irá executar a operação solicitada antes das demais, ignorando a regra de precedência.
‘’	Aspas simples	Utilizada para delimitar a utilização de cadeia de caracteres.
;	Ponto e Vírgula	Delimita o final de uma linha/sentença/cadeia de caracteres

7 – Variáveis e Atribuição

As declarações de variáveis podem ser feitas em qualquer região do programa. Sua forma de declaração deve sempre corresponder ao tipo da variável seguido do nome da variável que pode ser uma cadeia de caracteres do tipo [a-Z].

As atribuições em QSL são simples, e não permitem nenhum outro tipo de atribuição. Durante a declaração da variável, esta pode ser inicializada utilizando o símbolo para atribuição “:” seguido pelo valor desejado. O operador de atribuição simples atribui o operando à direita ao operando à esquerda, ou seja, “1” é atribuído ao operando “a”, ‘Cadeia de Caracteres’ ao operando “b” e “true” ao operando “c”. Para que essa atribuição ocorra é necessário que o valor a esquerda seja um valor modificável. Caso o operando à direita for de um tipo diferente ao operando da esquerda, a atribuição gera um erro, pois não existe conversão na QSL.

```
note a=1;

liltengwa b='Cadeia de Caracteres';

yulmavene c=true;
```

Com relação a regra de nomes para as variáveis, utiliza-se somente letras, considerando a necessidade de manter a QSL uma linguagem com tom sonoro. As variáveis podem ser quaisquer combinações do tipo [a-Z], porém não deve ser uma palavra reservada.

```
note variavel= 10;                  -- Correto
note variavel1= 10                  -- Errado
note variavel_= 10                  -- Errado
```

8 – Comandos de Entrada e Saída

Os comandos de entrada e saída (I/O) na linguagem são realizados utilizando as palavras *sarme* para escrita e *ista* para leitura, seguidos por parênteses.

O comando ***ista*** receberá um valor para uma variável que foi declarada anteriormente.

O comando ***sarme*** escreve e pode imprimir uma cadeia de caracteres delimitadas por aspas simples, também pode mostrar só o valor da variável, e também pode imprimir ambos, delimitados por uma vírgula.

```
ista(a)                             -- leitura
sarme('cadeia de caractere')        -- escrita
sarme(a)                            -- escrita
sarme('cadeia de caractere', a)     -- escrita
```

9 – Estruturas de Repetição

Nome	Descrição	Exemplo

an	O laço **an** precisa de uma variável para controlar as voltas (loops). Em an, essa variável deverá ser inicializada, indicando seu critério de parada e a forma de incremento ou decremento. Ou seja, an precisa de três condições, essas condições são separadas vírgula(,).	`an i = 10,1,-1`
		`car`
		`  sarme(i)`
		`metta`
		
		`-- o exemplo imprime valores de 10 à 1`

sive	O laço **sive** repete a execução de um bloco de sentenças enquanto uma condição permanecer verdadeira. Na primeira vez que a condição se tornar falsa, o sive não repetirá a execução do bloco, e a execução continuará com a sentença ou comando que vem logo após o bloco do sive, na sequência do programa.	`a = 10`
		
		`sive(a < 20)`
		`car`
		`  sarme("value of a: ", a)`
		`  a = a+1`
		`metta`
		
		`-- o exemplo imprime valores de 10 à 20`

vorima...tenna	Ao contrário dos laços an e sive, que testam a condição do loop no início, o laço vorima...tenna verifica a condição no final do laço.	`a = 10`
		
		`vorima`
		`  sarme("value of a: ", a)`
		`  a = a + 1`
		`tenna(a > 15)`
		
		`-- o exemplo imprime valores de 10 à 15`

10 – Estruturas de Salto

10.1 – Saltos Condicionais

10.1.1 – qui

Caso a expressão de condição retorne valor *true*, ou seja, verdadeiro, é executado o código que se encontra no próximo bloco. Note que é necessário metta para finalizar o qui.

```
qui line > MAXLINES ta
        showpage()
        line = 0
metta
```

10.1.2 – cela

Deve estar após um qui, normalmente encontra-se no fim do bloco. Se a sentença qui for *false*, ou seja, falsa, o bloco de código posterior ao cela será executado. Atenta-se que o cela não deve conter expressões.

```
qui a<b ta entulesse a cela entulesse b metta
```

10.1.3 – celace

Deve estar após um qui, caso a sentença do qui for falsa, o bloco de código posterior ao celace é executado. Não são permitidas expressões vazias para essa ocasião.

```
qui op == "+" ta
    r = a + b
celace op == "-" ta
    r = a - b
celace op == "*" ta
    r = a*b
celace op == "/" ta
    r = a/b
cela
    error("invalid operation")
metta
```

11 – Palavras Reservadas

Em QSL, *identificadores* podem ser qualquer cadeia de letras. Identificadores são usados para nomear variáveis e campos de tabelas. As seguintes *palavras-chave* são reservadas e não podem ser utilizadas como nomes:

Nome	Função
note	Define o tipo NUMÉRICO

liltengwa	Define o tipo STRING
yulmavene	Define o tipo LÓGICO (BOOLEAN)
ista	Leitura do teclado
sarme	Escreve em tela
an	Início do laço PARA
car	Início do laço DO
metta	Finaliza laços e condicionais (END)
sive	Início do laço WHILE
vorima	Início do laço DO...WHILE
tenna	Fim do laço e condicional para DO...WHILE
qui	Início do laço IF
cela	Condição alternativa ELSE
celace	Condição alternativa ELSE IF
ta	Condicional THEN
ar	Operador Lógico AND
hela	Operador Lógico OR
la	Operador Lógico NOT
helaer	Operador Lógico XOR
entulesse	Equivalente a return
hyaline	Define uma Function

As seguintes cadeias denotam outros itens léxicos:

+ - * / = ; :

( ) { } [ ]

<= => < > , .

Um comentário começa com um hífen duplo (--) em qualquer lugar, desde que fora de uma cadeia de caracteres. Comentários se estendem até o fim da linha.

12 – Precedência entre Operadores

Define a ordem em que os operadores serão executados em algum tipo de operação. Quanto menor o número maior a prioridade.

1	()
2	la
3	* /
4	< <= => >
5	==
6	haelar
7	ar
8	hela
9	:

13 – Estrutura Geral do Programa (Sintaxe) com Exemplos

13.1 – Imprimir “Olá Mundo”

`sarme “Olá, Mundo!”`	`print “Olá, Mundo!”`

13.2 – Função fatorial recursiva

`hyaline fact(n)`
`    qui n == 0 ta`
`        entulesse 1`
`    cela`
`        entulesse n * fact(n - 1)`
`    metta`
`metta`
	`function fact(n)`
	`    if n == 0 then`
	`        return 1`
	`    else`
	`        return n * fact(n - 1)`
	`    end`
	`end`

13.3 – Declaração, Atribuição e Leitura de Variáveis

`note  a;`
`note  b = 10;`
`note  c`
 	
`ista(a)`
`ista(b)`
`ista(c)`
	`int a;`
	`int b = 10;`
	`int c;`
	
	`read(a);`
	`read(b);`
	`read(c);`

14 – Expressões Regulares

14.1 – Números

nat = [0-9]+

number = nat (‘.’ nat)

14.2 – Identificadores

letter = [a-zA-Z]

digits = [0-9]

id = letter (letter | digits)*

14.3 – Palavras Reservadas

reserved_list = note | liltengwa | yulmavene | ista | sarme | an | car | metta | sive | vorima | tenna | qui | metta | cela | celace | ta | ar | hela | la | helaer | entulesse

| hyaline

14.4 – Comentários

comment = -- (~newline)*

14.5 – Operadores Relacionais, Aritméticos e Delimitadores

op_arit: ‘+’ | ‘-’ | ‘/’ | ‘*’ | ‘%’

op_rel: ‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘==’

del: ‘,’ | ‘;’ | ‘(‘ | ‘)’

postfix: ‘++’ | ‘--’

operators: op_arit | op_rel | del | postfix

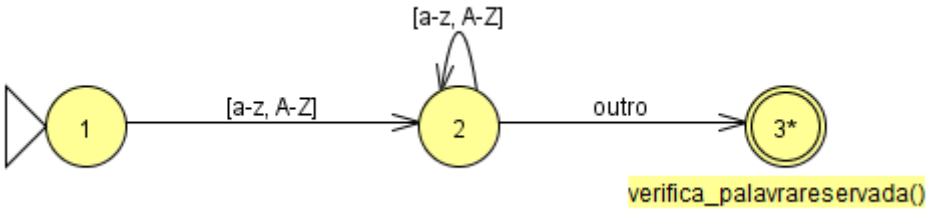
14.6 – Strings

string = ‘ ‘[0-9] | [a-zA-Z]’ ’

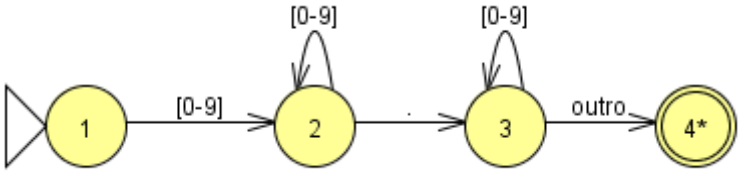
15 – Autômatos para cada Token

Os autômatos utilizados são do tipo finitos e determinísticos.

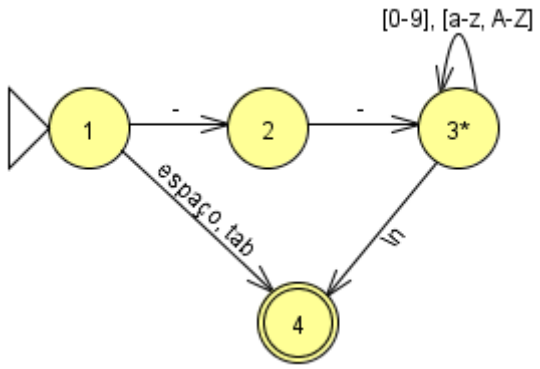
15.1 – Identificadores e Palavras Reservadas



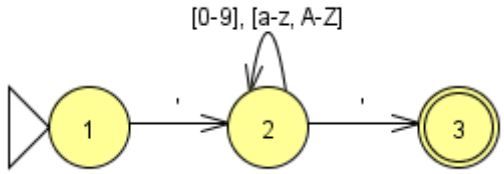
15.2 – Números Inteiros e Reais



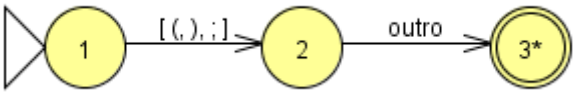
15.3 – Comentários, Espaços e Tabulações



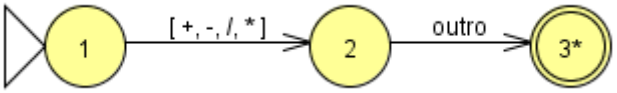
15.4 – Strings



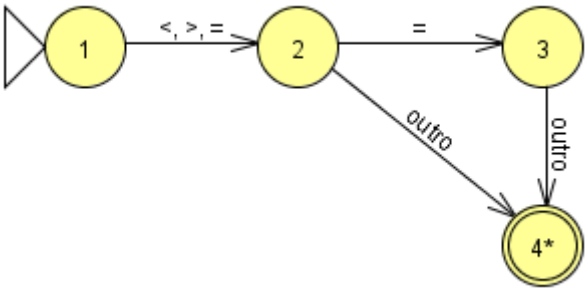
15.5 – Delimitadores



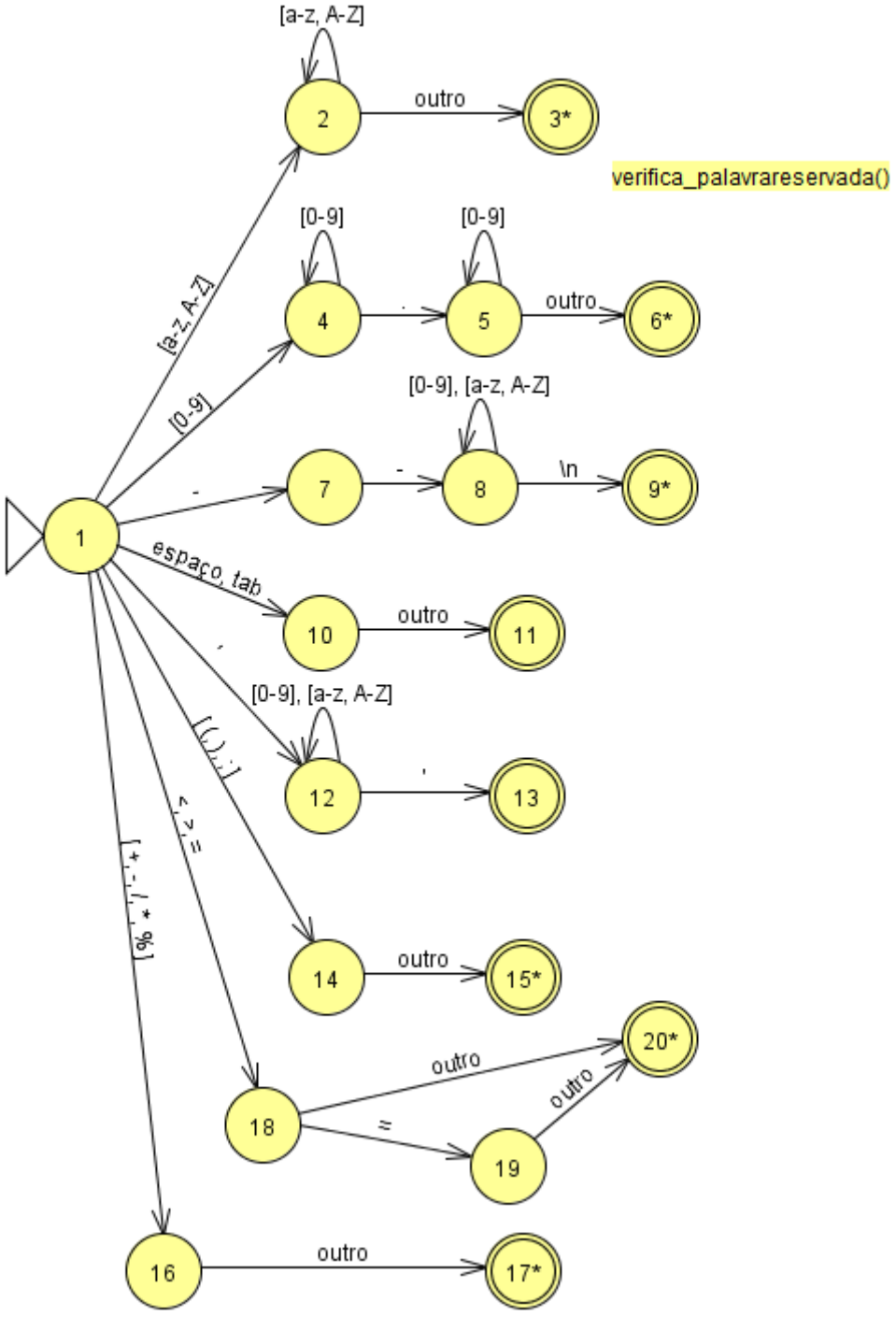
15.6 – Operadores Aritméticos



15.7 – Operadores Relacionais



15.8 – Autômato Completo



16 – EBNF

<program> → <statements>

<statements> → { <statement> }

<statement> → qui ‘(‘ <expr> ‘)’ ta <command> metta

| qui ‘(‘ <expr> ‘)’ ta entulesse <command> [cela entulesse <command> metta]

| qui ‘(‘ <expr> ’)’ ta <command> [{ celace <expr> ta <command> }] metta

| an ‘(‘ <expr> ‘)’ car <command> metta

| sive ‘(‘ <expr> ’)’ car <command> metta

| vorima <command> tenna ‘(‘ <expr> ‘)’

| ‘{‘ <statement>* }

| <expr> ‘;’

| ‘;’

<expr> → <operators> | id ‘=’ <expr>

<operators> → <arit> | <rel> | <postfix> | <del>

<arit> → <term> | <number> <op_arit> <term>

<rel> → <arit> | <arit> <op_rel> <arit> | <arit> id <arit>

<type> → note | liltengwa | yulmavene

<term> → id | ‘(‘ <expr> ‘)’