Universidade Federal de Uberlândia
Faculdade de Computação
Construção de Compiladores

Alunos: Matheus José da Costa **Matrícula:** 11711BCC008
Paulo Vitor Costa Silva 12011BCC045
Pedro Henrique Resende Ribeiro 12011BCC004

1ª ETAPA DO PROJETO – ESPECIFICAÇÃO DA LINGUAGEM

Seja uma gramática livre de contexto (GLC) G uma quádrupla (V, T, P, S), onde:

- V – é um subconjunto finito de variáveis ou símbolos não terminais
- T – é um conjunto de terminais (é um subconjunto de V)
- P – é um conjunto de regras de produção
- S – é o símbolo inicial e é um elemento de V

Dessa forma, o símbolo inicial da gramática é o identificador do começo do programa, neste caso, definido por function. Por convenção, adotou-se:
nome_programa → identificador

```
S  →  function identificador () bloco
```

O identificador do nome do programa, dado por nome_programa, será definido pela expressão regular de identificadores mostrada mais adiante. Um bloco de código é delimitado pela abertura de chave, seguida ou não de declarações de variáveis e sequências de comandos. A estrutura é mostrada abaixo.

```
bloco  →  { declaracoes_variaveis sequencia_comandos }
```

A declaração de variáveis, dada por declaracoes_variaveis, é construída informando o tipo da variável, seguida de uma lista de ids que pode conter um ou mais identificadores.

```
declaracoes_variaveis  →  ε | tipo : lista_ids ; declaracoes_variaveis
```

Pela especificação do projeto, o tipo pode ser int, char ou float. Além disso, a definição de lista_ids é dada por:

```
lista_ids  →  identificador | identificador , lista_ids
```

Após a declaração das variáveis, tem-se a sequência de comandos. A regra de produção de uma sequência de comandos é mostrada abaixo.

```
sequencia_comandos  →  ε | comando sequencia_comandos
```

Para os comandos tem-se: comandos de seleção, comentários, comandos de repetição, comandos de atribuição, condições e expressões.

```
comando  →  selecao | comentarios | laco_repita |
            laco_enquanto | atribuicao
```

O comando selecao é composto por uma estrutura se (condicao) entao, podendo ter ou não um comando senao. A regra é mostrada abaixo.

```
estrutura  →  comando | bloco

selecao  →  se (condicao) entao estrutura |
            se (condicao) entao estrutura senao estrutura
```

O comando de comentário é dado pela seguinte regra:

```
comentario  →  /* texto_comentario */
```

Os comandos de repetição podem ser de dois tipos: enquanto (condicao) faca ou repita ate (condicao). As regras são mostradas abaixo.

```
laco_enquanto  →  enquanto (condicao) faca estrutura

laco_repita  →  repita estrutura ate (condicao)
```

Por fim, o comando de atribuição é dado pela seguinte regra de produção:

```
atribuicao  →  identificador = expressao
```

As regras de produção para as condições são mostradas abaixo.

```
condicao  →  expressao operador_relacional expressao

expressao  →  identificador | constante | (expressao) |
              expressao operador_aritmetico expressao

constante  →  numero | caractere
```

Identificação dos tokens

Nome do token	Atributos	Expressão regular
function	Não possui	function
identificador	Possui	[a-zA-Z_](a-zA-Z0-9_)*
se	Não possui	se
entao	Não possui	entao
senao	Não possui	senao
tipo	Possui	(char | int | float)
{	Não possui	{
}	Não possui	}
;	Não possui	;
:	Não possui	:
,	Não possui	,
(	Não possui	(
)	Não possui	)
enquanto	Não possui	enquanto
faca	Não possui	faca
repita	Não possui	repita
ate	Não possui	ate
operador_relacional	Possui	(==|<>|<|>|<=|>=)
operador_aritmetico	Possui	(+|-|*|/|^)
numero	Possui	[0-9](.[0-9])?(E[+-]?[0-9])?

caractere	Possui	‘.’
=	Não possui	=

2ª ETAPA DO PROJETO – ANÁLISE LÉXICA (ESPECIFICAÇÃO)
ELABORAÇÃO DOS DIAGRAMAS DE TRANSIÇÃO

Nesta etapa, serão construídos os diagramas de transição para cada token presente na linguagem analisada. Em seguida, será construído o diagrama unificado para a linguagem. Por fim será criado o diagrama determinístico. As imagens estão no arquivo .zip enviado junto com o relatório.

3ª ETAPA DO PROJETO – ANÁLISE SINTÁTICA (ESPECIFICAÇÃO)
GRAMÁTICA LL(1), FIRST, FOLLOW E TABELA DE ANÁLISE PREDITIVA

Gramáticas LL(1) não possuem recursão a esquerda, são fatoradas e não possuem ambiguidade. Considerando a linguagem descrita na 1ª etapa do projeto, pode-se perceber que são necessários ajustes para que a gramática seja LL(1).

No caso de ambiguidade, a gramática pode ser considerada ambígua, o que significa que existem cadeias de símbolos que podem ser derivadas de mais de uma maneira, levando a diferentes árvores de análise. Por exemplo, considere a seguinte produção:

```
selecao  →  se (condicao) entao estrutura |
            se (condicao) entao estrutura senao estrutura
```

Nesse caso, a sequência “`senao se (condicao) entao estrutura`” pode ser interpretada de duas maneiras diferentes:

- Interpretando “`senao`” como parte da estrutura anterior: “`se (condicao) entao estrutura senao (se (condicao) entao estrutura)`”.
- Interpretando “`senao`” como o início de uma nova estrutura: “`se (condicao) entao estrutura (senao se (condicao) entao estrutura)`”.

Para o caso descrito acima, mesmo após a fatoração, a gramática continuará ambígua. Dessa forma, a solução é considerar o “senao” associado ao “entao” mais próximo, ou seja, da estrutura anterior.

Além dos casos mostrados acima, também há ambiguidade na regra mostrada abaixo:

```
lista_ids  →  identificador | identificador, lista_ids
```

A ambiguidade ocorre quando temos múltiplos identificadores separados por vírgula. Considere o exemplo: “a, b, c”. Nesse caso, podemos interpretar a sequência como:

- Uma lista com três identificadores: “identificador = a, identificador = b, identificador = c”
- Ou como um identificador seguido de uma lista de identificadores: “identificador = a, lista_ids = b, identificador = c”

Para remover a ambiguidade, pode-se reescrever a regra "lista_ids" para garantir que ela seja não ambígua. Dessa forma, a regra ficaria da seguinte forma:

```
lista_ids  →  identificador lista_ids’

lista_ids’ →  ε | , identificador lista_ids’
```

Essa modificação garante que cada identificador seja seguido ou por uma sequência separada por vírgula (lista_ids') ou por nada (ε), evitando ambiguidade na interpretação da lista de identificadores.

A gramática final, no formato LL(1) é mostrada abaixo.

```
S  →  function identificador () bloco

bloco  →  { declaracoes_variaveis sequencia_comandos }

declaracoes_variaveis  →  ε | tipo : lista_ids ; declaracoes_variaveis

lista_ids  →  identificador lista_ids’
```

lista_ids’ → , identificador lista_ids’ | ε

sequencia_comandos → ε | comando sequencia_comandos

comando → selecao | laco_repita | laco_enquanto | atribuicao

estrutura → comando | bloco

selecao → se (condicao) entao estrutura selecao'

selecao’ → senao estrutura | ε

laco_enquanto → enquanto (condicao) faca estrutura

laco_repita → repita estrutura ate (condicao)

atribuicao → identificador = expressao ;

condicao → expressao operador_relacional expressao

termo → identificador | constante

expressao → termo expressao’

expressao' → operador_aritmetico termo expressao’ | ε

constante → numero | caractere

tipo → char | int | float

operador_aritmetico → + | - | * | / | ^

operador_relacional → == | <> | < | > | <= | >=

First dos símbolos da gramática

Da teoria, sabe-se que o conjunto `First` de um símbolo não-terminal é composto pelos terminais que podem aparecer como o primeiro símbolo nas produções desse não-terminal.

A seguir estão os conjuntos `First` para cada não-terminal na gramática:

```
First(S) = {function}

First(bloco) = {{}

First(declaracoes_variaveis) = First(tipo) = {char, int, float, ε}

First(lista_ids) = {identificador}

First(lista_ids') = {‘,’, ε}

First(sequencia_comandos) = {ε, se, repita, enquanto, identificador}

First(comando) = {se, repita, enquanto, identificador}

First(selecao) = {se}

First(selecao') = {senao, ε}

First(laco_enquanto) = {enquanto}

First(laco_repita) = {repita}

First(atribuicao) = {identificador}

First(condicao) = First(expressao)
    = {identificador, numero, caractere}
```

```
First(termo) = {identificador, numero, caractere}

First(expressao) = {identificador, numero, caractere}

First(expressao') = {+, -, *, /, ^, ε}

First(constante) = {numero, caractere}

First(tipo) = {char, int, float}

First(operador_aritmetico) = {+, -, *, /, ^}

First(operador_relacional) = {==, <>, <, >, <=, >=}

First(estrutura) = First(comando) + First(bloco)
    = {se, repita, enquanto, identificador, {}
```

Follow dos símbolos da gramática

O conjunto Follow de um não-terminal T é composto pelos terminais que podem aparecer imediatamente após o símbolo de alguma produção.

A seguir estão os conjuntos Follow para cada não-terminal na gramática:

```
Follow(S) = {$}

Follow(bloco) = {$}

Follow(declaracoes_variaveis) = First(sequencia_comandos) – ε
    = {se, repita, enquanto, identificador}

Follow(lista_ids) = {;}

Follow(lista_ids’) = Follow(lista_ids) + Follow(lista_ids’) = {;}
```

Follow(sequencia_comandos) = Follow(sequencia_comandos) + {}} = {}}

Follow(comando) = First(sequencia_comandos)
    = {ε, se, repita, enquanto, identificador}

Follow(selecao) = Follow(comando)
    = {ε, se, repita, enquanto, identificador}

Follow(selecao’) = Follow(selecao)
    = {ε, se, repita, enquanto, identificador}

Follow(laco_enquanto) = Follow(comando)
    = {ε, se, repita, enquanto, identificador}

Follow(laco_repita) = Follow(comando)
    = {ε, se, repita, enquanto, identificador}

Follow(atribuicao) = Follow(comando)
    = {ε, se, repita, enquanto, identificador}

Follow(condicao) = {)}

Follow(termo) = First(expressao’) - ε = {+, -, *, /, ^}

Follow(expressao) = {;} + First(operador_relacional) – ε +
                    Follow(condicao)
    = {;} + {==, <>, <, >, <=, >=} + {)}
    = {;, ==, <>, <, >, <=, >=,)}

Follow(expressao’) = Follow(expressao) + Follow(expressao’)
    = {;, ==, <>, <, >, <=, >=,)}

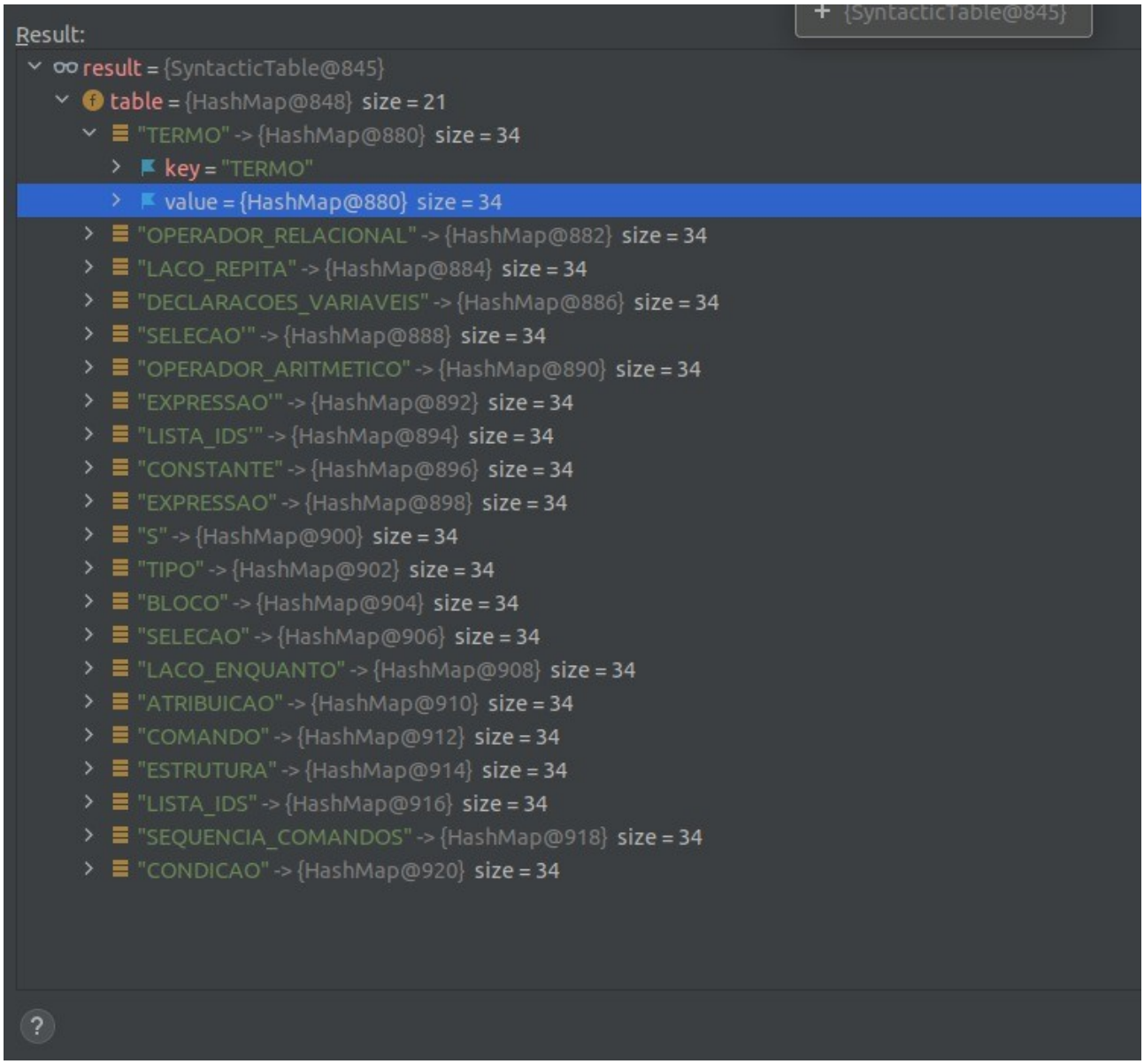
Follow(tipo) = {:}

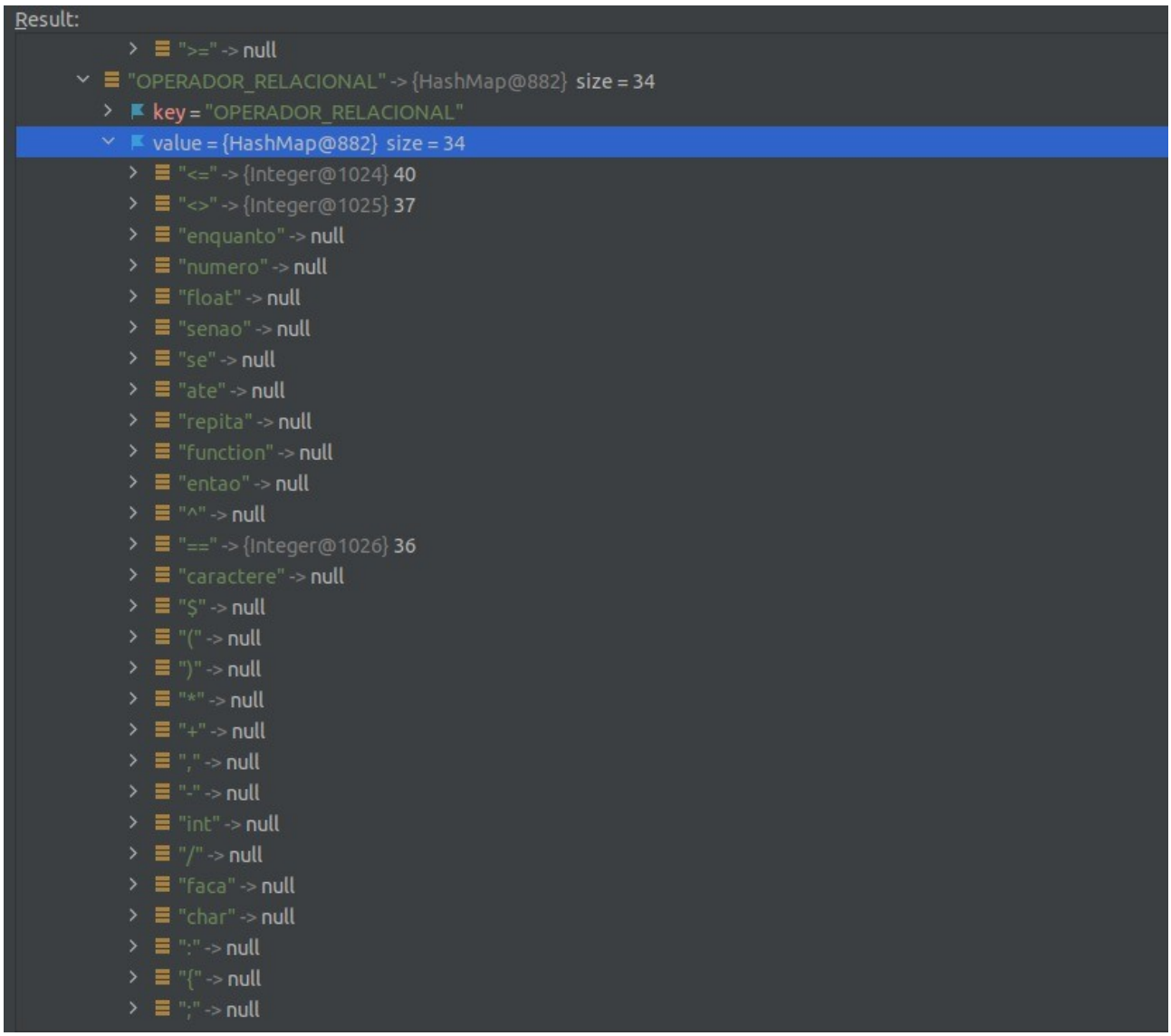
Follow(operador_aritmetico) = First(expressao’) - ε = {+, -, *, /, ^}

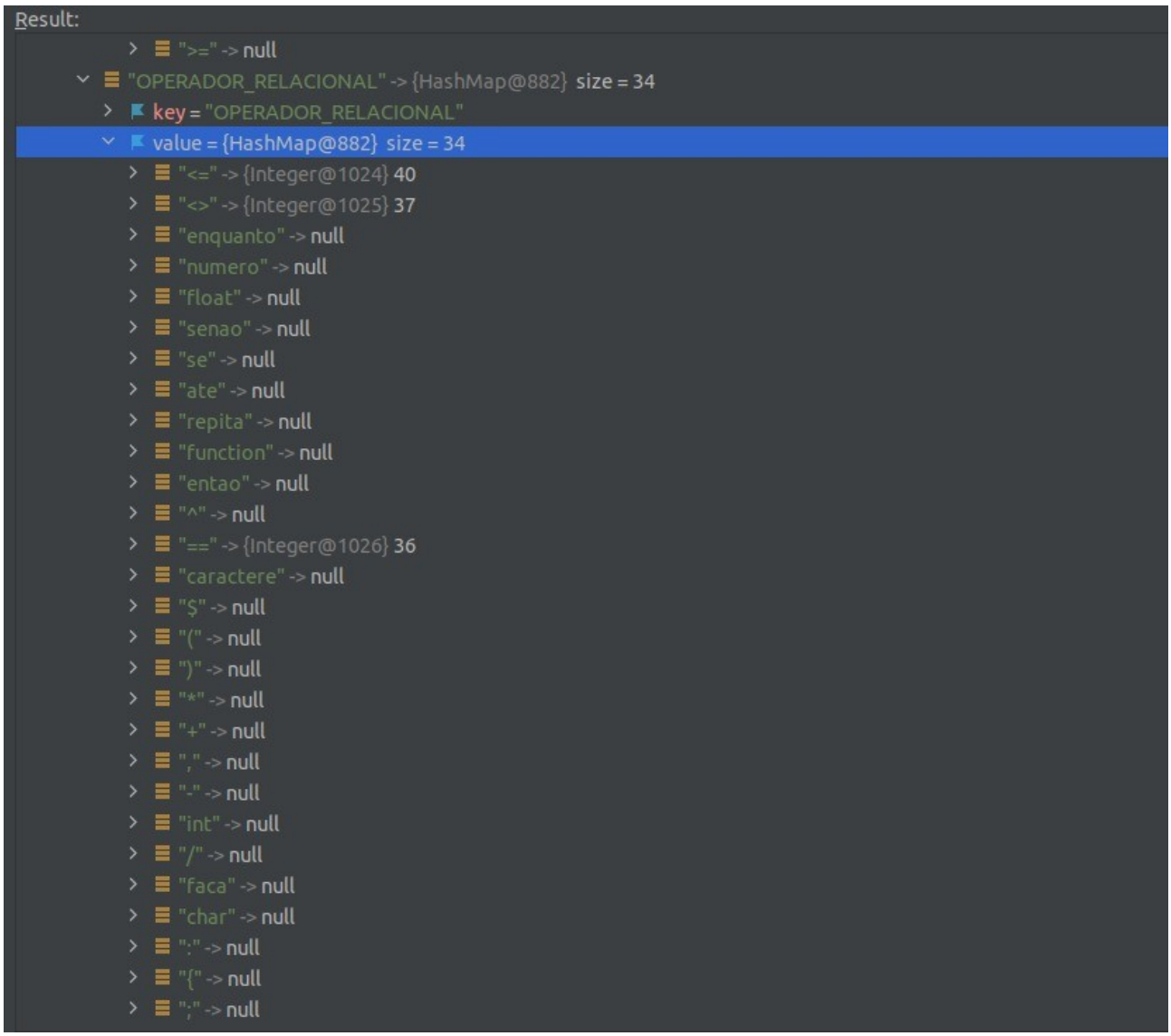
Follow(operador_relacional) = First(expressao) – ε
= {identificador, constante, (}

Follow(estrutura) = First(selecao’) – ε + {ate} + Follow(laco_enquanto)
= {senao, ate, ε, se, repita, enquanto, identificador}

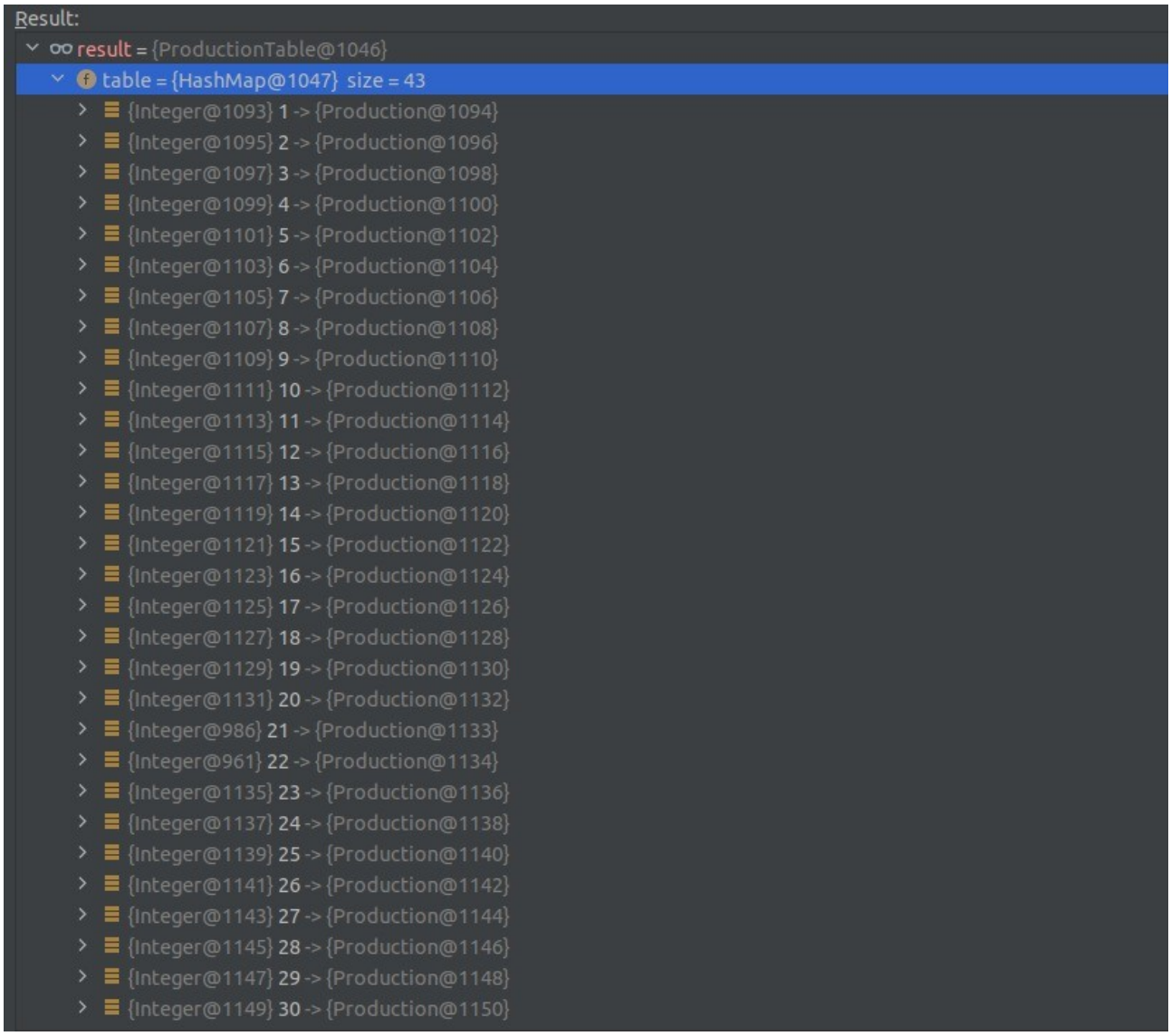
ANEXO A – REPRESENTAÇÃO DA TABELA SINTÁTICA NO CÓDIGO

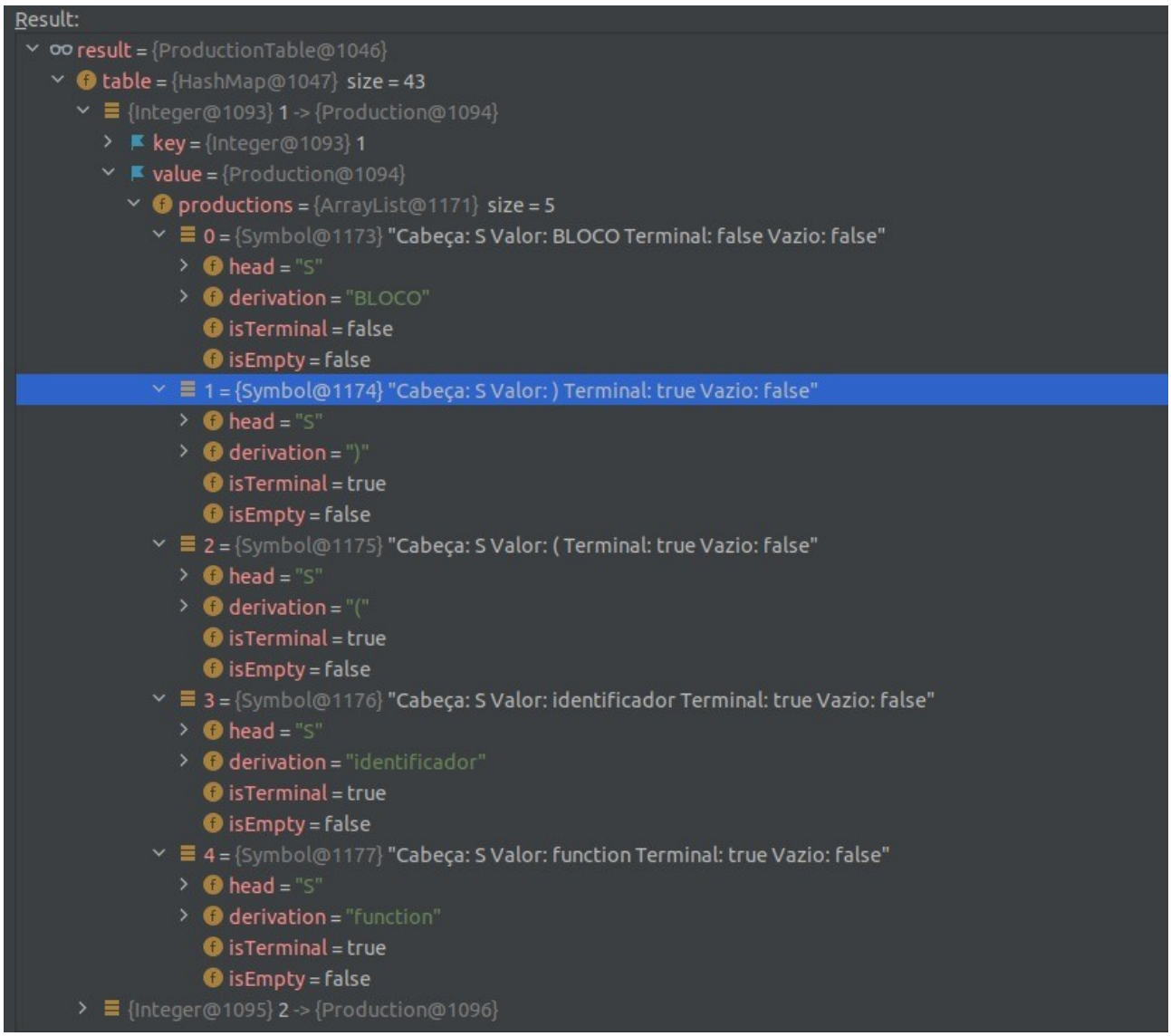


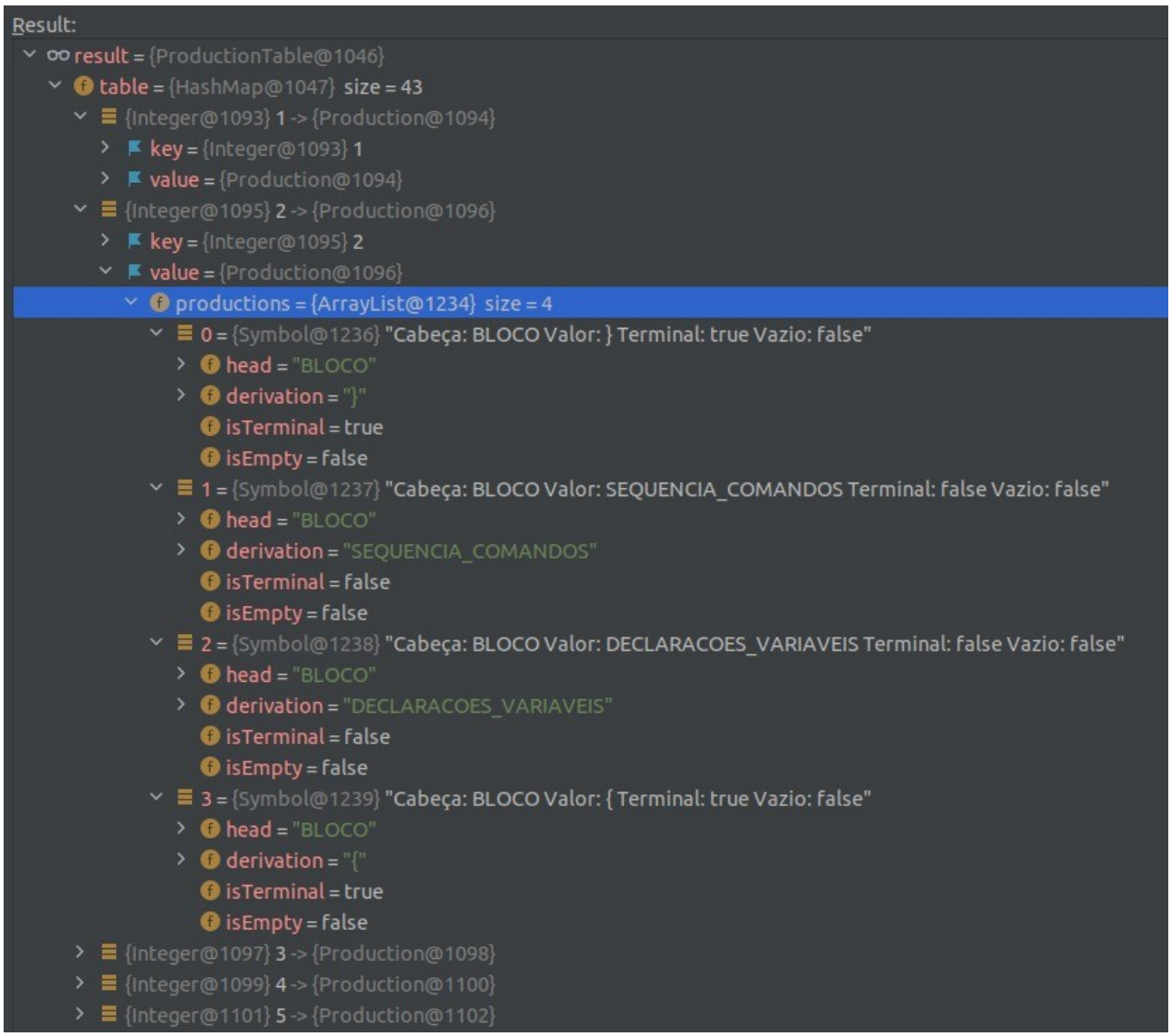

Result:
">=" -> null
"OPERADOR_RELACIONAL" -> {HashMap@882} size = 34
key = "OPERADOR_RELACIONAL"
value = {HashMap@882} size = 34
"<=" -> {Integer@1024} 40
"<>" -> {Integer@1025} 37
"enquanto" -> null
"numero" -> null
"float" -> null
"senao" -> null
"se" -> null
"ate" -> null
"repita" -> null
"function" -> null
"entao" -> null
"^" -> null
"==" -> {Integer@1026} 36
"caractere" -> null
"$" -> null
"(" -> null
")" -> null
"*" -> null
"+" -> null
"," -> null
"-" -> null
"int" -> null
"/" -> null
"faca" -> null
"char" -> null
":" -> null
"{" -> null
";" -> null


Result:
">=" -> null
"OPERADOR_RELACIONAL" -> {HashMap@882} size = 34
key = "OPERADOR_RELACIONAL"
value = {HashMap@882} size = 34
"<=" -> {Integer@1024} 40
"<>" -> {Integer@1025} 37
"enquanto" -> null
"numero" -> null
"float" -> null
"senao" -> null
"se" -> null
"ate" -> null
"repita" -> null
"function" -> null
"entao" -> null
"^" -> null
"==" -> {Integer@1026} 36
"caractere" -> null
"$" -> null
"(" -> null
")" -> null
"*" -> null
"+" -> null
"," -> null
"-" -> null
"int" -> null
"/" -> null
"faca" -> null
"char" -> null
":" -> null
"{" -> null
";" -> null

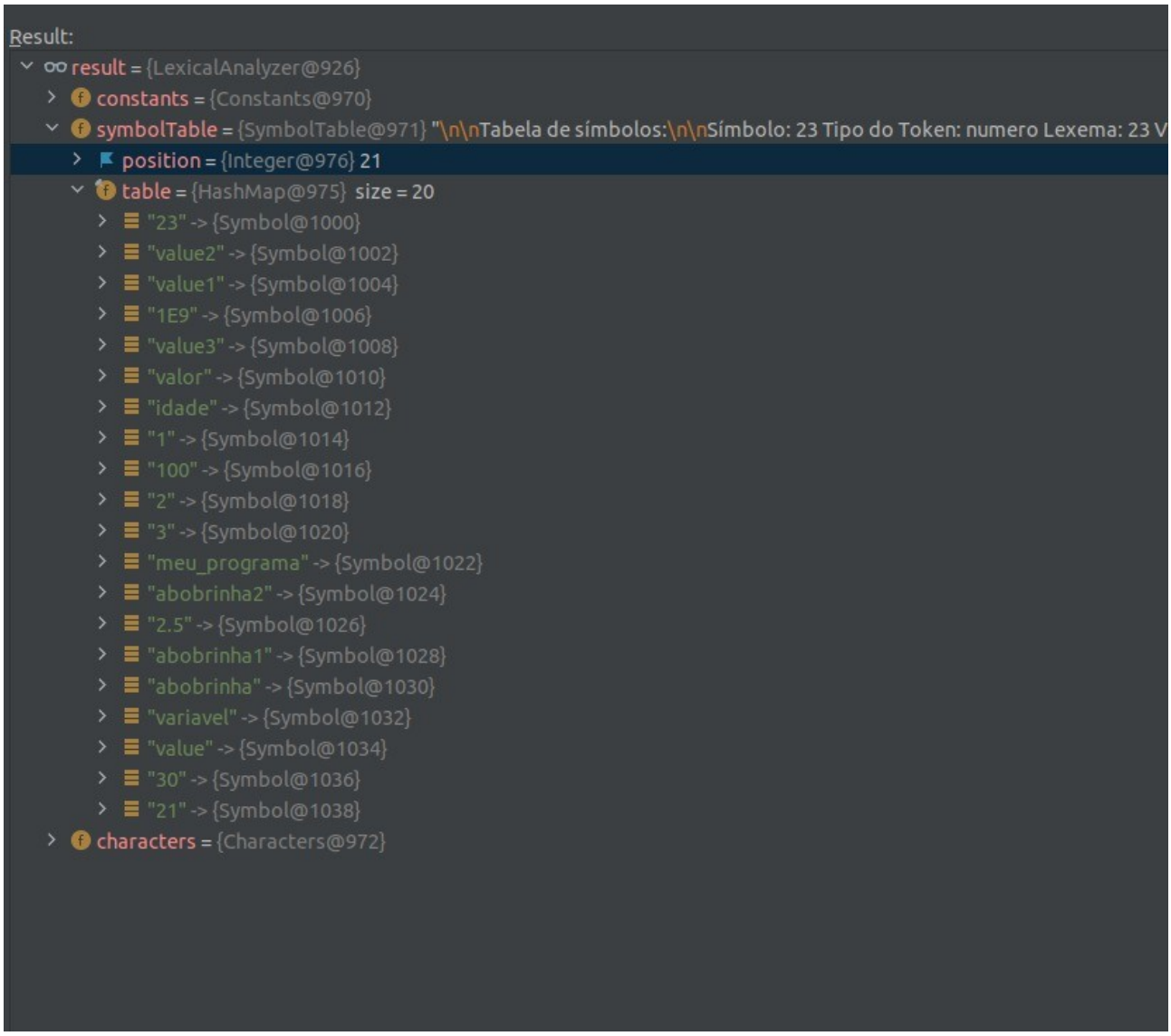
ANEXO B – TABELA DE PRODUÇÕES

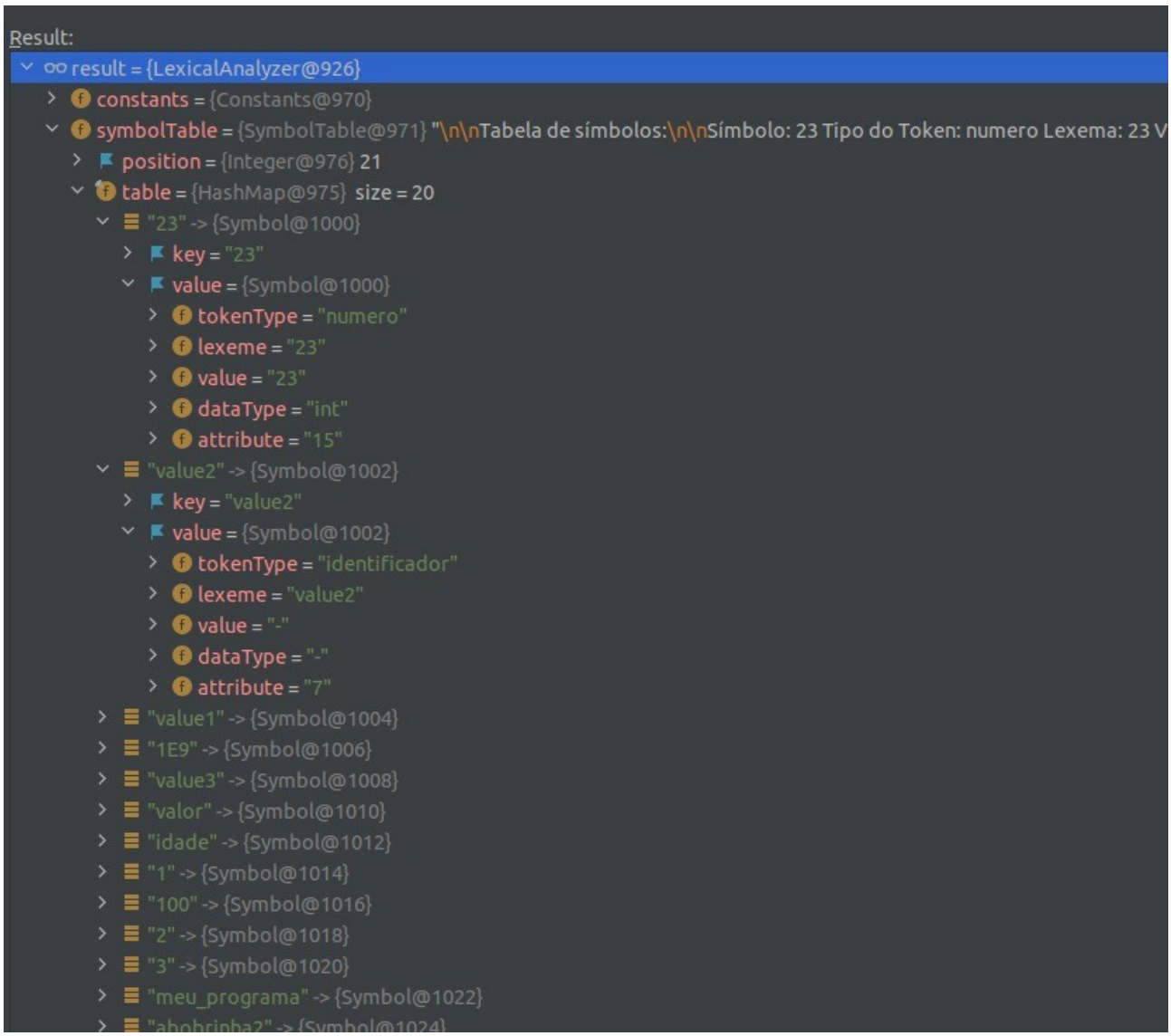



Result:
result = {ProductionTable@1046}
table = {HashMap@1047} size = 43
{Integer@1093} 1 -> {Production@1094}
key = {Integer@1093} 1
value = {Production@1094}
productions = {ArrayList@1171} size = 5
0 = {Symbol@1173} "Cabeça: S Valor: BLOCO Terminal: false Vazio: false"
head = "S"
derivation = "BLOCO"
isTerminal = false
isEmpty = false
1 = {Symbol@1174} "Cabeça: S Valor:) Terminal: true Vazio: false"
head = "S"
derivation = ")"
isTerminal = true
isEmpty = false
2 = {Symbol@1175} "Cabeça: S Valor: (Terminal: true Vazio: false"
head = "S"
derivation = "("
isTerminal = true
isEmpty = false
3 = {Symbol@1176} "Cabeça: S Valor: identificador Terminal: true Vazio: false"
head = "S"
derivation = "identificador"
isTerminal = true
isEmpty = false
4 = {Symbol@1177} "Cabeça: S Valor: function Terminal: true Vazio: false"
head = "S"
derivation = "function"
isTerminal = true
isEmpty = false
{Integer@1095} 2 -> {Production@1096}


Result:
result = {ProductionTable@1046}
table = {HashMap@1047} size = 43
{Integer@1093} 1 -> {Production@1094}
key = {Integer@1093} 1
value = {Production@1094}
{Integer@1095} 2 -> {Production@1096}
key = {Integer@1095} 2
value = {Production@1096}
productions = {ArrayList@1234} size = 4
0 = {Symbol@1236} "Cabeça: BLOCO Valor: } Terminal: true Vazio: false"
head = "BLOCO"
derivation = "}"
isTerminal = true
isEmpty = false
1 = {Symbol@1237} "Cabeça: BLOCO Valor: SEQUENCIA_COMANDOS Terminal: false Vazio: false"
head = "BLOCO"
derivation = "SEQUENCIA_COMANDOS"
isTerminal = false
isEmpty = false
2 = {Symbol@1238} "Cabeça: BLOCO Valor: DECLARACOES_VARIAVEIS Terminal: false Vazio: false"
head = "BLOCO"
derivation = "DECLARACOES_VARIAVEIS"
isTerminal = false
isEmpty = false
3 = {Symbol@1239} "Cabeça: BLOCO Valor: { Terminal: true Vazio: false"
head = "BLOCO"
derivation = "{"
isTerminal = true
isEmpty = false
{Integer@1097} 3 -> {Production@1098}
{Integer@1099} 4 -> {Production@1100}
{Integer@1101} 5 -> {Production@1102}

ANEXO C – TABELA DE SÍMBOLOS




Result:
result = {LexicalAnalyzer@926}
constants = {Constants@970}
symbolTable = {SymbolTable@971} "\n\nTabela de símbolos:\n\nSímbolo: 23 Tipo do Token: numero Lexema: 23 V
position = {Integer@976} 21
table = {HashMap@975} size = 20
"23" -> {Symbol@1000}
key = "23"
value = {Symbol@1000}
tokenType = "numero"
lexeme = "23"
value = "23"
dataType = "int"
attribute = "15"
"value2" -> {Symbol@1002}
key = "value2"
value = {Symbol@1002}
tokenType = "identificador"
lexeme = "value2"
value = "-"
dataType = "-"
attribute = "7"
"value1" -> {Symbol@1004}
"1E9" -> {Symbol@1006}
"value3" -> {Symbol@1008}
"valor" -> {Symbol@1010}
"idade" -> {Symbol@1012}
"1" -> {Symbol@1014}
"100" -> {Symbol@1016}
"2" -> {Symbol@1018}
"3" -> {Symbol@1020}
"meu_programa" -> {Symbol@1022}