# PCS2056 – Linguagens e Compiladores

## 3ª parte do projeto

## Reconhecedor Sintático

## Alan Raso – 804151

### Introdução

Na partes anteriores do projeto, foram desenvolvidos a linguagem que será compilada e o analisador léxico. Este último é responsável pela criação de *tokens*, que são caracteres ou um conjunto de caracteres que normaliza o tamanho de todos os elementos escritos num texto portencilamente pertencente a linguagem.

O módulo seguinte ao analisador léxico é o analisador sintático, que é responsável por construir as chamadas árvores sintáticas, que são estruturas que relacionam diretamente o conjunto de *tokens* com a estrutura gramatical definida formalmente para a linguagem.

Algumas formas de arquitetura do sistema que forma o compilador são possíveis de serem implementadas. O analisador léxico como programa principal, por exemplo, que, ao terminar de extrair um *token* do texto, envia-o para o analisador sintático como sub-rotina. Neste projeto, no entanto, o analisador sintático é o programa principal, e sempre que necessário é solicitado ao analisador léxico um *token*.

Nesta terceira parte, foi desenvolvido o reconhecedor sintático, uma parte do analisador sintático. Ele é responsável processar a sequência de *tokens* geradas de forma a decidir se o texto que está sendo lido pertence ou não à linguagem especificada.

### Sub-máquinas

Para cada elemento da gramática formalizada na segunda parte do projeto, é apresentado um ou mais APE (Autômato de Pilha Estruturado), que são sub-máquinas que auxiliam no reconhecimento dessas estruturas quando um texto é processado.

Cada capítulo desta seção apresenta a definição em notação de Wirth do elemento analisado e, em seguida, as transiçoes geradas pela ferramenta recomendada ([*http://mc-barau.herokuapp.com*](http://mc-barau.herokuapp.com)), bem como o desenho do autômato correspondente.

Alguns elementos triviais foram considerados como se fossem terminais da linguagem para simplificação, ou seja, não foram gerados autômatos para eles:

* Tipo = bool|caract|int|pflut.
* Dig = “0” | “1” | “2” | ... | “9”.
* Letra = “A” | “B” | ... | “Z” | “a” | “b” | ... | “z”.

#### Identificador

ID = Letra{Letra|Dig}.

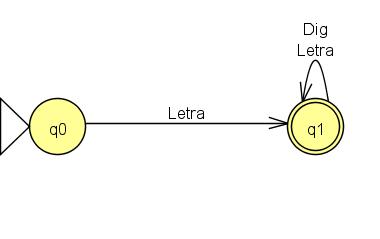
initial: 0

final: 1

(0, Letra) -> 1

(1, Letra) -> 1

(1, Dig) -> 1



#### Número

Num = “.”Dig{Dig}|Dig{Dig}[“.”Dig{Dig}].

initial: 0

final: 2, 3

(0, ".") -> 1

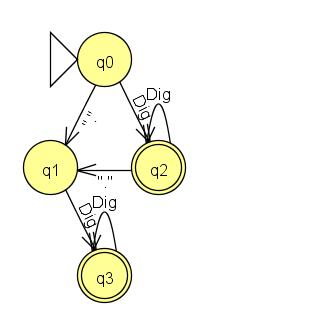
(0, Dig) -> 2

(1, Dig) -> 3

(2, ".") -> 1

(2, Dig) -> 2

(3, Dig) -> 3



#### Identificador de Arrays

IDM = ID{(“[“ExpAritimetica|"[")“]”}.

initial: 0

final: 1

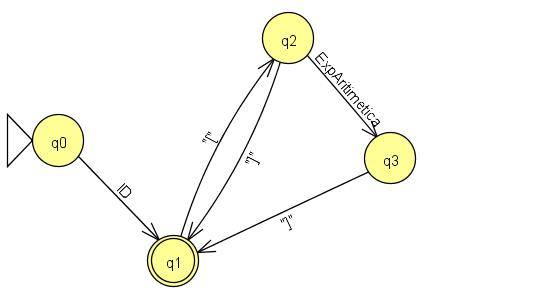
(0, ID) -> 1

(1, "[") -> 2

(2, ExpAritimetica) -> 3

(2, "]") -> 1

(3, "]") -> 1



#### Declaração

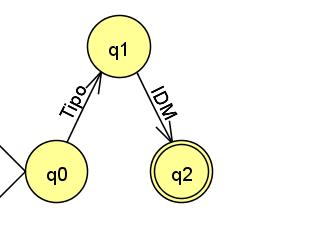
Declaracao = Tipo IDM.

initial: 0

final: 2

(0, Tipo) -> 1

(1, IDM) -> 2



#### Função

Funcao = "func"ID"("[Declaracao{","Declaracao}]")""("[Tipo]")""{"ComandoLista"}".

initial: 0

final: 12

(0, "func") -> 1

(1, ID) -> 2

(2, "(") -> 3

(3, Declaracao) -> 4

(3, ")") -> 5

(4, ",") -> 6

(4, ")") -> 5

(5, "(") -> 7

(6, Declaracao) -> 4

(7, ")") -> 8

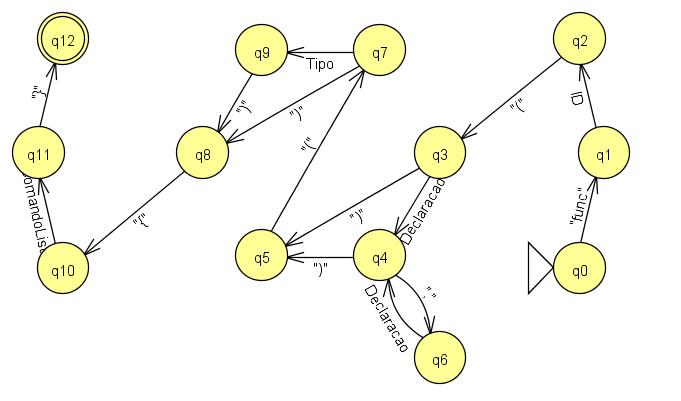
(7, Tipo) -> 9

(8, "{") -> 10

(9, ")") -> 8

(10, ComandoLista) -> 11

(11, "}") -> 12



#### Lista de comandos

ComandoLista = ((Declaracao|Atribuicao|ChamadaFunc|Retorno)“;”|Condicao|Iteracao) {(Declaracao|Atribuicao|ChamadaFunc|Retorno)“;”| Condicao|Iteracao}.

initial: 0

final: 2

(0, Declaracao) -> 1

(0, Atribuicao) -> 1

(0, ChamadaFunc) -> 1

(0, Retorno) -> 1

(0, Condicao) -> 2

(0, Iteracao) -> 2

(1, ";") -> 2

(2, Declaracao) -> 1

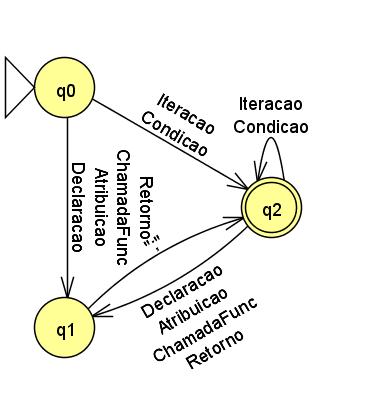
(2, Atribuicao) -> 1

(2, ChamadaFunc) -> 1

(2, Retorno) -> 1

(2, Condicao) -> 2

(2, Iteracao) -> 2



#### Atribuição

Atribuicao = IDM“:=”Exp.

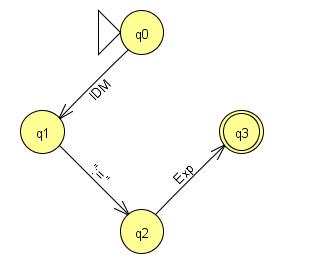
initial: 0

final: 3

(0, IDM) -> 1

(1, ":=") -> 2

(2, Exp) -> 3



#### Condição

CondicaoSimples = se“(”ExpBool“)”“{”ComandoLista“}”.

Condicao = CondicaoSimples“senao”Condicao|CondicaoSimples“senao”“{“ComandoLista“}”|CondicaoSimples.

CondicaoSimples:

initial: 0

final: 7

(0, se) -> 1

(1, "(") -> 2

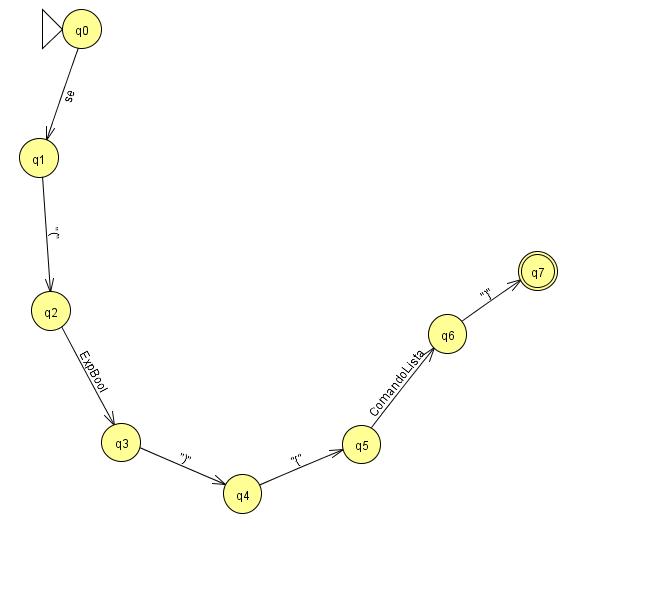
(2, ExpBool) -> 3

(3, ")") -> 4

(4, "{") -> 5

(5, ComandoLista) -> 6

(6, "}") -> 7



Condição:

initial: 0

final: 1, 3

(0, CondicaoSimples) -> 1

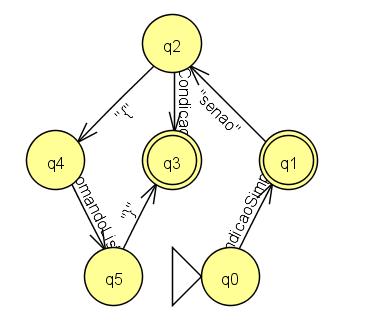
(1, "senao") -> 2

(2, Condicao) -> 3

(2, "{") -> 4

(4, ComandoLista) -> 5

(5, "}") -> 3



#### Iteração

Iteracao = “enquanto”“(”ExpBool“)”“{”ComandoLista“}” | “para”“(”Atribuicao“;”ExpBool“;”Atribuicao“)”“{”ComandoLista“}”.

initial: 0

final: 7, 15

(0, "enquanto") -> 1

(0, "para") -> 2

(1, "(") -> 3

(2, "(") -> 4

(3, ExpBool) -> 5

(4, Atribuicao) -> 6

(5, ")") -> 7

(6, ";") -> 8

(7, ComandoLista) -> 7

(8, ExpBool) -> 9

(9, ";") -> 10

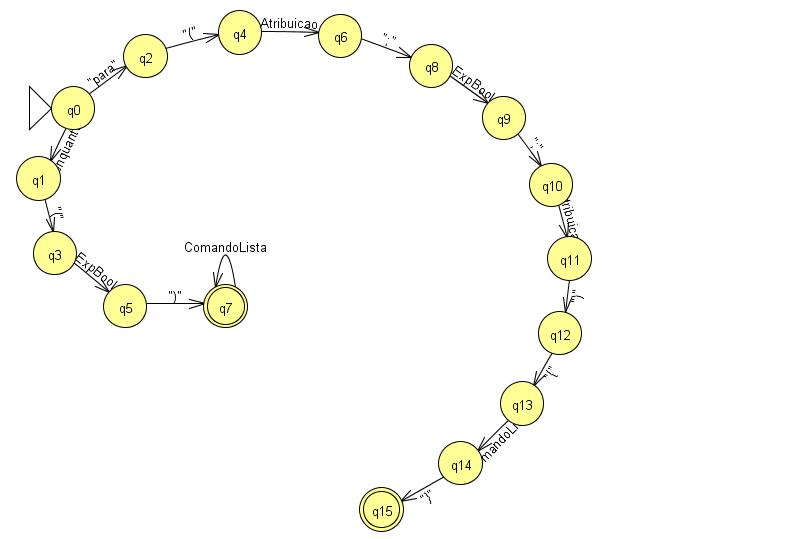
(10, Atribuicao) -> 11

(11, ")") -> 12

(12, "{") -> 13

(13, ComandoLista) -> 14

(14, "}") -> 15



#### Chamada de função

ChamadaFunc = ID (“(”Exp{","Exp}“|"(") ")”.

initial: 0

final: 4

(0, ID) -> 1

(1, "(") -> 2

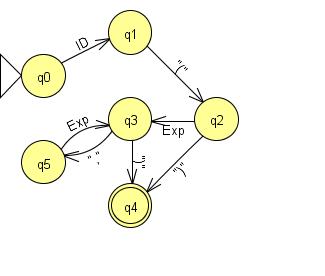
(2, Exp) -> 3

(2, ")") -> 4

(3, ",") -> 5

(3, ")") -> 4

(5, Exp) -> 3



#### Retorno

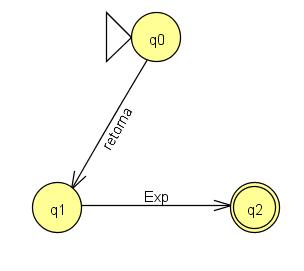
Retorno = retorna Exp.

initial: 0

final: 2

(0, retorna) -> 1

(1, Exp) -> 2



#### Comparação

Comparacao = ExpArtimetica(“<”|“>”|“=”|“!=”)ExpAritimetica.

initial: 0

final: 3

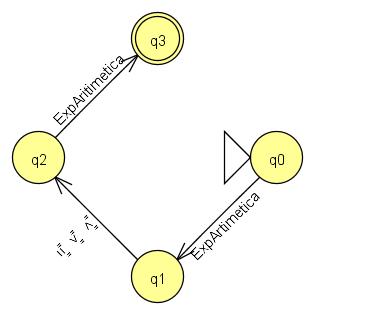
(0, ExpArtimetica) -> 1

(1, "<") -> 2

(1, ">") -> 2

(1, "=") -> 2

(2, ExpAritimetica) -> 3



#### Expressão Aritimética

##### Potência

Potencia = ChamadaFuncao|IDM|Num|“(”ExpAritimetica“)”.

initial: 0

final: 1

(0, ChamadaFuncao) -> 1

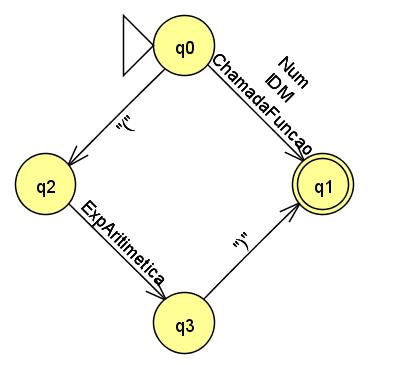
(0, IDM) -> 1

(0, Num) -> 1

(0, "(") -> 2

(2, ExpAritimetica) -> 3

(3, ")") -> 1



##### Fator

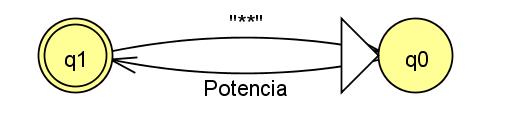
Fator = Potencia{“\*\*”Potência}.

initial: 0

final: 1

(0, Potencia) -> 1

(1, "\*\*") -> 0



##### Termo

Termo = Fator{(“\*”|“/”)Fator}.

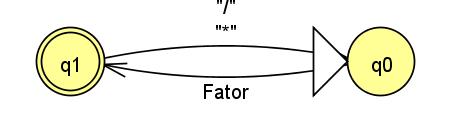
initial: 0

final: 1

(0, Fator) -> 1

(1, "\*") -> 0

(1, "/") -> 0



##### ExpAritimetica

ExpAritimetica = [“-”]Termo{(“+”|“-”)Termo}.

initial: 0

final: 2

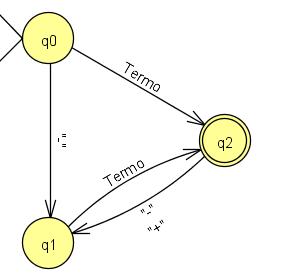
(0, "-") -> 1

(0, Termo) -> 2

(1, Termo) -> 2

(2, "-") -> 1

(2, "+") -> 1



#### Expressão Booleana

##### FatorBool

FatorBool = IDM|“!”(ExpBool|Comparacao)|“(”ExpBool“)”|Comparacao.

initial: 0

final: 1

(0, IDM) -> 1

(0, "!") -> 2

(0, Comparacao) -> 1

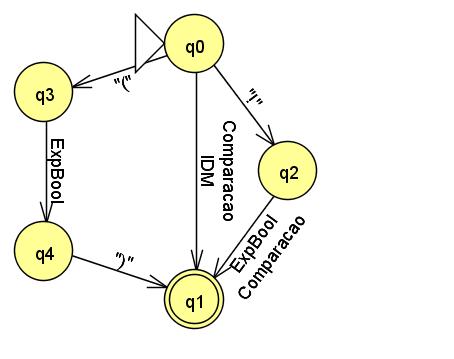
(0, "(") -> 3

(2, ExpBool) -> 1

(2, Comparacao) -> 1

(3, ExpBool) -> 4

(4, ")") -> 1



##### TermoBool

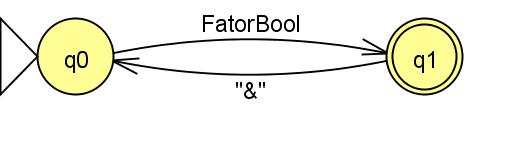
TermoBool = FatorBool{“&”FatorBool}.

initial: 0

final: 1

(0, FatorBool) -> 1

(1, "&") -> 0



##### ExpBool

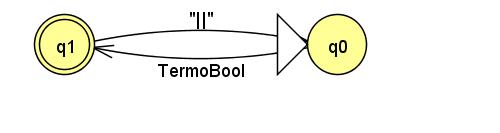
ExpBool = TermoBool{“||”TermoBool}.

initial: 0

final: 1

(0, TermoBool) -> 1

(1, "||") -> 0



#### Expressões

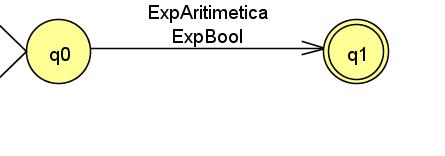
Exp = ExpBool|ExpAritimetica.

initial: 0

final: 1

(0, ExpBool) -> 1

(0, ExpAritimetica) -> 1



### Implementação

A implementação foi feita tendo como base os autômatos apresentados no item 2. Para cada autômato foi desenvolvida uma função em C, e chamada propriamente quando devido.

O programa principal tenta inicialmente resolver os autômatos de definição de função e declaração de variáveis, que são os possíveis iniciadores da linguagem.

Sempre que uma transição dos autômatos é necessária, o reconhecedor sintático solicita um token ao analisador léxico elaborado na primeira parte do projeto, e verifica se o token recebido está de acordo com o que deveria aparecer. Para resolver ambiguidades, como no caso de estados de autômatos diferentes que podem receber o mesmo token, foi utilizado *lookahead*.