Analisador Léxico para a Linguagem C- utilizando Máquina de Estados de Mealy

Rubens Antonio da Silva Filho[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)

[2]R. Rosalina Maria Ferreira, 1233

1. Introdução

Um autômato é um modelo matemático composto por máquinas de estados finitas. Dado uma alfabeto de entrada, estados e uma entrada, ele é capaz de julgar a entrada como aceita ou não.

Podemos ter um autômato finito determinístico, tendo como principal característica a obrigatoriedade em haver uma, e apenas uma, transição para cada símbolo do alfabeto em cada estado existente. Ou seja, dado o alfabeto de entrada como A = {a, b} e os estados Q = {q0, q1}, é necessário que no estado q0, tenha uma transição quando for a, e uma transição quando for b; o mesmo ocorre em q1. Na figura 1 temos um exemplo de autômato finito determinístico para o alfabeto de entrada e estados citados.

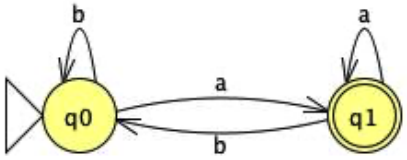

Figura 1. Exemplo de autômato finito determinístico. Fonte: Autoria própria.

Já um autômato finito não determinístico não há obrigatoriedade de transições. Nele é podemos ter um estado que não contenha uma transição para um símbolo do alfabeto, e também pode conter mais de uma transição para a mesmo símbolo. Outra característica, é que nele podemos ter transições vazias, ou seja, transições que não irão 'gastar' um símbolo da entrada. Com isso, temos que o autômato finito não determinístico possui características de paralelismo, ou seja, uma execução com uma cadeia de entrada pode estar em diversos estados ao mesmo tempo. Na figura 2 temos um exemplo de autômato finito não determinístico.

Também é possível criar um autômato com saída, podendo ser determinístico ou não. Sua principal característica é possui um saída, podendo ser na transição, Máquina de Mealy, ou no estado, Máquina de Moore.

2. Implementação

O objetivo deste trabalho é implementar um analisador léxico para a linguagem C-. Este analisador deve imprimir uma lista de tokens identificados.

Para isto, foi desenvolvido um código em *Python* que implementa um autômato determinístico com saída. Foi escolhido o uso de Máquina de Mealy, pois, as saídas precisam ser *printadas* no momento em que ocorrem as transições.

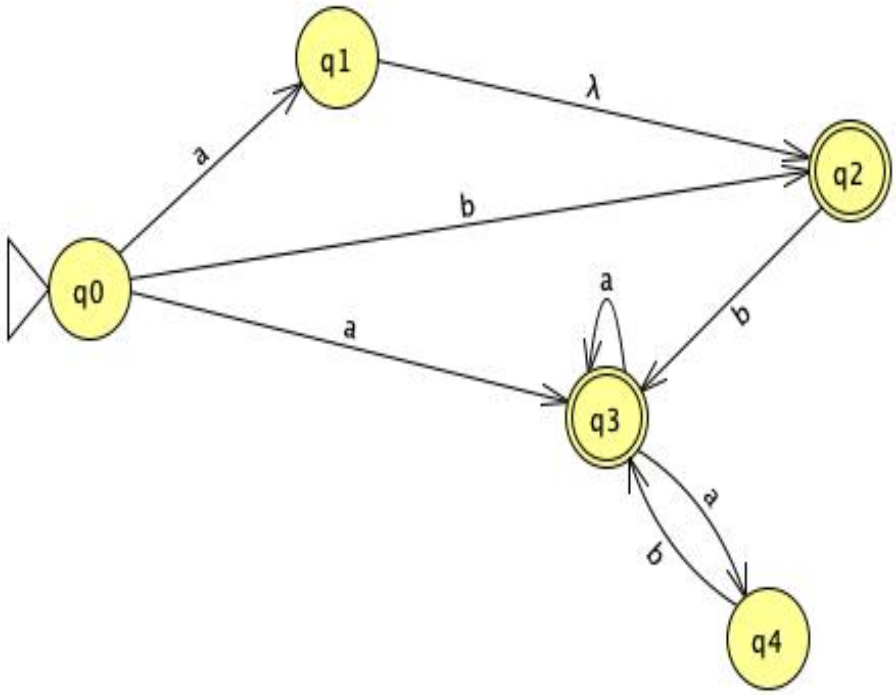

Figura 2. Exemplo de autômato finito não determinístico. Fonte: Autoria própria.

Para o auxiliar e simplificar o desenvolvimento, foi implementado 3 classes, *Transition*, *State* e *Automata*.

2.1. *Transition*

A classe *Transition* é responsável pela transição entre os estados do autômato. Ela é composta de uma *condition*, símbolo do alfabeto que é necessário para que a transição ocorra; *destinationState*, estado de destino da transição; e *output*, saída gerada pela transição.

2.2. *State*

A classe *State* é responsável pelos estados do autômato. Ela inicialmente foi composta de um *name*, nome do estado e por *transitions*, um vetor contendo as transições daquele estado. Depois foi adicionado um vetor de *conditions*, ela guarda os símbolos do alfabeto que aquele estado já possuí uma transição. Como estamos desenvolvendo um autômato determinístico, não pode haver mais de uma transição com a mesma condição, e nem haver a falta de transição com uma condição.

2.3. *Automata*

Por fim, temos a classe *Automata*, ela é responsável por criar o autômato que iremos utilizar. Ela é composta de um *inputAlphabet*, alfabeto de entrada; *initialState*, estado inicial do autômato; *defaultState* e *defaultOutput*.

Estas duas últimas variáveis auxiliaram a não repetição de código, pois, em muitos estados haviam transições que iam para um estado em comum com saídas iguais. Com o uso delas foi possível omitir essas transições. Se no estado não possui uma transição

com um símbolo da entrada, será criado uma transição para o estado padrão, com a saída padrão.

Dentro desta classe temos a função *getMealyAutomata()*, é nela aonde o autômato é em si criado. Mais detalhes da criação do autômato, será dado na seção 2.4.

2.4. Biblioteca

Para a criação do autômato foi utilizado a biblioteca *automata_python*, disponível em https://github.com/hemangsk/automata-python. Nela criamos um objeto *Mealy*, contendo um vetor com o nome dos estados, um vetor com o alfabeto de entrada, um vetor com o alfabeto de saída, um dicionário com as transições, e o nome do estado inicial.

Como já citado na seção 2, foi criado 3 classes para auxiliar no desenvolvimento, o objetivo final delas, é criar o objeto *Mealy*. Isso ocorre dentro da função *getMealyAutomata()* da classe *Automata*.

Nesta função, um *for* percorre todos os estados obtendo seu nome e adicionado em um vetor de nomes.

```
statesName = []
for state in self.states:
    statesName.append(state.name)
```

Depois, um outro *for* irá percorrer os estados, montando um dicionário similar à {simbolo_entrada: (proximo_estado, saida) } para cada estado pertencente ao autômato. O que retorna este dicionário para cada estado é a função *getDict(stateName)*. Nela um *for* percorre todos os símbolos do alfabeto de entrada, verificando se naquele estado existe transição com ele, caso não exista, cria uma transição para o estado padrão, com a saída padrão.

```
for alphabet in self.inputAlphabet:
    for state in self.states:
        if stataName == state.name:
            if alphabet in state.conditions:
                # condição existe, posso adicionar ela mesmo
                transition = state.getByCondition(alphabet)
                # print(transition)
                dic[alphabet] = (transition.destinationState.name,
                ↪  transition.output)
            else:
                 # condição não existe, adiciono o padrão
                 dic[alphabet] = (self.defaultState.name,
                 ↪  self.defaultOutput)
```

2.5. Estados

Nesta seção, temos uma tabela descrevendo a principal função de cada estado no autômato. Veja a tabela 1.

3. Exemplos

Como exemplo iremos utilizar este código.

```
int main(void){
    return(0);
}
```

Estado	Principal função
q0	*Estado inicial, nele ocorre a verificação se é a primeira letra de algum token. Por exemplo, se vier um i vai para o estado q2. Sempre que um token é identificado, voltará para este estado.*
q1	*Consome o **L** do **ELSE**.*
q2	*Consome o **N** do **INT** ou o **F** do **IF**.*
q3	*Consome o **E** do **RETURN**.*
q4	*Consome o **O** do **VOID**.*
q5	*Consome o **H** do **WHILE**.*
q6	*Consome o **S** do **ELSE**.*
q7	*Consome o segundo **E** do **ELSE**.*
q8	*Caso venha espaço ou { imprime o token **ELSE**.*
q9	*Consome o **T** do **INT**.*
q10	*Caso venha espaço imprime o token **INT**.*
q11	*Caso venha espaço ou (imprime o token **IF**.*
q12	*Consome o **I** do **VOID**.*
q13	*Consome o **D** do **VOID**.*
q14	*Caso venha espaço ou) imprime o token **VOID**.*
q15	*Consome o **I** do **WHILE**.*
q16	*Consome o **L** do **WHILE**.*
q17	*Consome o **E** do **WHILE**.*
q18	*Caso venha espaço ou (imprime o token **WHILE**.*
q19	*Consome o **T** do **RETURN**.*
q20	*Consome o **U** do **RETURN**.*
q21	*Consome o **R** do **RETURN**.*
q22	*Consome o **N** do **RETURN**.*
q23	*Caso venha espaço ou (imprime o token **RETURN**.*
q24	*Consome o **L** do **FLOAT** ou o **O** do **FOR**.*
q25	*Consome o **O** do **FLOAT**.*
q26	*Consome o **A** do **FLOAT**.*
q27	*Consome o **T** do **FLOAT**.*
q28	*Caso venha espaço imprime o token **FLOAT**.*
q29	*Consome o **R** do **FOR**.*
q30	*Caso venha espaço ou (imprime o token **FOR**.*
qlparen	*Responsável pela abertura de parenteses, imprime o token **LPAREN**.*
qlbraces	*Responsável pela abertura de chaves, imprime o token **LBRACES**.*
qrparen	*Responsável pela fechadura de parenteses, imprime o token **RPAREN**.*
qnumber	*Responsável pelos números, imprime o token **NUMBER**.*
qlessequal	*Responsável pelo ¿=, imprime o token **LESS**, caso venha espaço, e o token **LESS_EQUAL**, caso venha um =.*
qgreaterequal	*Responsável pelo ¡=, imprime o token **GREATER**, caso venha espaço, e o token **GREATER_EQUAL**, caso venha um =.*
qequal	*Responsável pelo ==, imprime o token **ATTRIBUTION**, caso venha espaço, e o token **EQUALS**, caso venha um =.*
qdifferent	*Responsável pelo !=, imprime o token **DIFFERENT**, caso venha um =.*
qo	*Estado padrão, é responsável pelos identificadores, sempre que vier algo não esparado vai para ele. Nele se vier um espaço, imprime o token **ID**.*

Tabela 1. Tabela contendo todos os estados do autômato e sua principal função. Fonte: Autoria própria.

Para ele, temos a seguinte lista de tokens.

- **INT**
- **ID**
- **LPAREN**
- **VOID**
- **RPAREN**
- **LBRACES**
- **RETURN**
- **LPAREN**
- **NUMBER**
- **RPAREN**
- **SEMICOLON**
- **RBRACES**

4. Conclusão

Ao fim do desenvolvimento temos um autômato que foi capaz de reconhecer todos os casos de testes disponibilizados.

Para validar, foi criado testes unitários que verificam a lista de tokens impressa para cada um dos 5 testes disponibilizados.

Por fim, o código fonte e como utilizar, está disponível no GitHub em https://github.com/RubinhoSilva/analisador_lexico_cmenos.

Referências

Menezes, P. (2009). *Linguagens Formais e Autômatos: Volume 3 da Série Livros Didáticos Informática UFRGS*. Bookman Editora.

Rodger, S. H. (2006). *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett Publishers, Inc., USA.

Sipser, M. (2007). *Introdução à Teoria da Computação: Tradução da 2ª edição norte-americana (trad. Ruy José Guerra Barreto de Queiroz)*. Thomson Learning, São Paulo.