Aplicação para reconhecimento de sequência de símbolos por autômato de pilha indeterminístico

Edson Lemes da Silva[1]

[1]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.802-112 – Chapecó – SC – Brasil

edson_lemes@live.com

Abstract. *This paper describes some theoretical aspects for pushdown automata. Conversely, it presents an application for checking patterns recognized by a pushdown automata according to its transitions. The form of acceptance happens by empty stack.*

Resumo. *Este trabalho descreve alguns aspectos teóricos referentes aos autômatos de pilhas. Por outro lado, apresenta uma aplicação para a verificação de padrões reconhecidos por um autômato de pilha conforme as suas transições. A forma de aceitação acontece por pilha vazia.*

1. Introdução

Um autômato de pilha não-determinístico (APND), permite reconhecer sentenças geradas por uma gramática livre de contexto. Nesta ideia, a verificação ocorre levando em conta as suas transições, consegue-se analisar se uma sequência de símbolos ou tokens é reconhecida ou não por um autômato de pilha.

A aplicação descrita aqui atende a leitura de transições de um autômato de pilha não-determinístico, juntamente com uma sequência de símbolos para ser testada. A condição de aceitação implementada por este trabalho, é sobre pilha vazia. A linguagem de programação usada é o Python, juntamente com algumas classes que já estão implementadas.

2. Autômatos de pilha

De maneira formal um autômato de pilha não-determinístico (APND) pode ser representado como uma 7-tupla, definida por $M = (k, \Sigma, A, \delta, i, S, F)$. Onde:

- k: conjunto de estados;
- Σ: alfabeto da linguagem;
- A: alfabeto da pilha;
- δ: função de transição;
- i: estado inicial;
- S: símbolo inicial da pilha;
- F: estados finais, tais que $F \in k$.

Os autômatos de pilha são usados como reconhecedores para gramáticas livres de contexto. Conforme destacado por [Menezes 2000, p. 201], os algoritmos são do tipo `top-down`, e assim analisam a árvore de derivação para reconhecer uma palavra específica.

$$ef(A) = \{\alpha \in \Sigma^* \mid [i, \alpha, S] \vdash^* [f, \epsilon, y], f \in F, y \in A^*\} \quad (1)$$

$$pv(A) = \{\alpha \in \Sigma^* \mid [i, \alpha, S] \vdash^* [f, \epsilon, \epsilon], f \in k\} \quad (2)$$

Existem duas formas de reconhecimento com autômato de pilha, são elas : por estado final de aceitação (equação 1) e por pilha vazia (equação 2); a primeira irá convergir para um estado de aceitação quando alcançar um estado final de reconhecimento. E a outra forma é através de pilha vazia, onde o conteúdo da fita e da pilha estão vazias.

A construção deste algoritmo ocorrem de forma não-determinística, com o intuito de testar as diversas produções de uma gramática. A forma mais simples de fazer o reconhecimento por autômato de pilha, é através de sua de forma descendente, conforme [Menezes 2000, p. 202], este algoritmo simula a derivação mais à esquerda fazendo a substituição do elemento do topo da pilha. Para tal, leva-se em conta se o símbolo é um terminal ou não.

Algoritmo 1: AUTÔMATO COM PILHA DESCENDENTE

Entrada: Transições, fita
Saída: Sequência reconhecida
1 **início**
2 Empilha o símbolo inicial.
3 Para toda variável no topo da pilha, substitui esta por suas produções.
4 Se o topo da pilha for um terminal, remove-o caso for o mesmo símbolo no topo da fita de entrada.
5 **fim**
6 **retorna** *Sequência reconhecida*

3. A aplicação

Este projeto visa aplicar os conceitos relativos aos autômatos de pilha não determinísticos. Essa aplicação refere-se ao processo de aceitação ou não de uma sequência de símbolos. Deste modo, é realizado a partir de um arquivo de texto a leitura das transições de um APND, seguido da sequência a ser testada.

Durante o processo de análise, são construídos estados relativos a cada derivação na árvore, assim, através de uma busca em largura, é feito a computação para os diferentes ramos desta árvore. Portanto, como a verificação dos estados é feito por nível, se existe um ramo que leva ao estado de aceitação, este será encontrado.

4. A implementação

Inicialmente, é preciso descrever o ambiente de programação. A linguagem utilizada para a aplicação, foi o Python; utilizando-se de alguns módulos (classes) já implementados para esta linguagem.

Como forma de compatibilidade, esta aplicação foi desenvolvida sob a versão 2.7.12 do python. Os único módulos utilizados refere-se as operações envolvendo listas.

Internamente, existem 3 funções principais. As demais são consideradas secundárias, pois são processos chamados dentro destas funções. Assim, estarão descrita aqui as funções em ordem de execução.

Na implementação, estas funções estão dentro de uma classe, chamada de `Ap`. Uma instância desta classe, recebe um parâmetro do tipo `String`, que representa o nome e o caminho do arquivo de leitura (onde estão descritos as transições e a sequência passada). E também um valor para impressão dos resultados durante o processo de reconhecimento. Os estados e as transições da Pilha são armazenados em listas, assim como a fila utilizada na implementação, que é uma lista de instâncias do tipo Pilha.

A outra classe utilizada nesta aplicação chama-se `Pilha`, ela representa uma pilha, e cada instância representa um ramo de computação da árvore de derivação (considerando o nível atual). Os métodos implementados por esta classe são as funções padrões de uma pilha: empilha, desempilha e topo.

Primeiramente, o formato do arquivo de leitura deve ser especificado, pois para fins de implementação, algumas restrições foram impostas neste arquivo de entrada. A primeira situação a ser destacada refere-se ao formato de cada transição, onde o estado de cada uma delas são antecedidos por um símbolo "::=", assim, o número do estado, conteúdo da fita e o estado da pilha devem ser separados por vírgula. Em referência ao `epsilon`, ignora-se o espaço, isto é acrescenta um espaço em branco, ou uma vírgula para separar (caso o `epsilon` for o conteúdo da fita).

Seguindo na análise do arquivo de entrada, os dados que estiverem após o símbolo de "::=" são considerados transições para a variável sendo verificada. Como o autômato é não-determinístico, se um estado possui mais de uma transição possível, deve-se separá-las usando o símbolo " $|$". Desta maneira, cada uma destas transições possuem um número (próximo estado) e algo para ser incluído no topo da pilha, este conteúdo que irá na pilha, deve estar separado por espaços, isto é, os símbolos que fazem parte do alfabeto da linguagem deve estar separado por um espaço dos símbolos que são estados. Esta situação facilita a diferenciação de cada elemento na hora de análise no autômato de pilha.

```
1
0,,S::=0,if E then O
3 0,if,if::=0,
0,,E::=0,a P a
5 0,a,a::=0,
0,,P::=0,>|0,<|0,=
7 0,>,>::=0,
0,<,<::=0,
9 0,=,=::=0,
0,then,then::=0,
11 0,,O::=0,B|0,S
0,,B::=0,B + B|0,B * B|0,a
13 0,+,+::=0,
0,*,*::=0,
15 if a = a then if a > a then a * a
```

Exemplo 1. Arquivo de entrada

Conforme destaca o Exemplo 1, cada transição da pilha esta escrita em uma única

linha. Sendo que onde há o `epsilon`, ignora-se este espaço. As próximas transições dado um estado são destacados após o símbolo de "::=". Novamente usa-se vírgulas para separa os elementos, sendo que o conteúdo a ser incluído no topo da pilha deve ser separado com espaços em relações aos símbolos, tokens ou estados. A última linha desse arquivo de entrada, deve se acompanhado por uma sequência de símbolos ou tokens. Deste modo, também deve ser separado por espaços em símbolos ou tokens.

A primeira função (método), é chamada de `readFile`, ela é a responsável por fazer a leitura do arquivo de entrada. Assim, é feito o processamento para identificação dos itens que fazem parte do texto. A função lê linha por linha (até encontrar um separador), e identifica o estado e suas próximas transições em relação ao autômato de pilha. A última linha reserva-se para uma sequência de símbolos que devem estar contida no arquivo de entrada. Assim, para separação dos tokens e/ou símbolos utiliza-se um "$" para separação.

A função `execute` inicializa a fila com o estado inicial da árvore de derivação. Em seguida ela chama uma função de busca em largura em relação a árvore. Como parâmetro recebe um valor inteiro, este valor representa as iterações limites para o algoritmo de busca, pois a navegação é feita em nível, e caso não haja nenhum ramo da árvore que conduz a um estado de aceitação, pode haver ramificações que nunca geram símbolos terminais, ou seja, o algoritmo não consegue determinar estes caso.

Deste modo, chama-se a função `bfs` que realiza a busca. Há duas condições de parada, uma caso a fila esteja vazia e a outra caso o tamanho da fila (iterações) ultrapasse o limite recebido por parâmetro. Assim, o algoritmo escolhe o próximo elemento que está na pilha. Para ele, verifica o estado correspondente, o conteúdo da fita e o conteúdo do topo da pilha. Busca qual é o estado representado por estes dados no topo, então, para cada transição concatena-se o APND com a próxima transição. Deste modo, irá percorrendo a fila verificando os níveis e procurando uma computação que leve a um reconhecimento por pilha vazia.

4.1. Execução do algoritmo

A execução do algoritmo é feita a partir do interpretador Python. Primeiramente, deve-se importar o arquivo que contém a classe implementada, que esta no arquivo "apnd.py". O próximo passo é criar uma instância para esta classe, passando como parâmetro o nome do arquivo de leitura.

Assim, após a instanciação, deve-se chamar o método (função) `execute`, ele se encarrega de chamar as demais funções do algoritmo, execução da busca em largura.

A resposta em caso de aceitação, é exibida em forma de lista, onde cada posição desta lista representa uma execução em relação as transições do autômato de pilha. Os casos que não são verificados, mostra-se uma mensagem de rejeição. Como exemplo, o arquivo "test.py" executa o algoritmo para um caso específico.

Para visualizar os dados de saída, permite-se imprimir o elemento do topo da pilha em cada execução. Para isso, deve-se passar o valor booleano `True` para o construtor da classe `Ap`.

```
from apnd import *

test = Ap('test2.in',True)
test.execute()
```

Exemplo 2. Execução do algoritmo.

5. Conclusão

A implementação deste algoritmo possui umas restrições que precisam ser seguidas para o funcionamento correto.

A principal dificuldade encontrada refere-se no quesito de rejeição de uma cadeia de símbolos. Uma vez que a busca em largura permite verificar as derivações em níveis, e pode haver ramos que são sempre reconhecidos pelas transições da pilha, mas que nunca geram um símbolo terminal.

A documentação sobre a implementação descreveu as funções principais de forma detalhada, assim como no código-fonte, onde estão descritas de forma resumida as funcionalidades de cada uma delas.

Referências

Menezes, P. F. B. (2000). *Linguagens formais e autômatos*. Editora Sagra Luzzatto, 3rd edition.