5. Autómatos de Pilha (“Pushdown automata”)

Copiar link

Gramáticas Livres de Contexto

Vimos em +4. Gramáticas e Expressões Regulares que as gramáticas permitem descrever de forma sistemática um processo de derivação de palavras de uma linguagem, através das chamadas *regras de produção.* Vimos também que as *linguagens regulares*, que temos focado até este ponto, são descritas por gramáticas regulares, em que todas as regras de produção $\alpha \to \beta$ obedecem às seguintes restrições:

- α é apenas um símbolo não-terminal, e
- β é apenas:
  - um símbolo terminal,
  - OU um símbolo terminal seguido de um símbolo não terminal,
  - OU ε

Por exemplo a linguagem regular $L_{01Rep} = \{(01)^i \mid i \geq 0\}$ pode ser descrita pela gramática constituída pelas regras:
  - $S \to 0A$
  - $A \to 1S$
  - $S \to \varepsilon$

em que S, A são os símbolos não-terminais, sendo S o inicial.

Como exemplo de uma derivação temos
$S \to 0A \to 01S \to 010A \to 0101S \to 0101$

Uma primeira liberalização possível desta noção de gramática consiste em permitir que nas regras de produção $\alpha \to \beta$ não haja qualquer restrição sobre a palavra β, mantendo-se a restrição de α ser apenas um símbolo não-terminal. Por outras palavras, não podem existir restrições relativas ao *contexto* necessário para a aplicação das regras. Designamos então estas gramáticas como *livres de contexto*.

Exemplos

1. Um exemplo muito típico de linguagem livre de contexto (que não é regular) é a linguagem de parêntesis, de todas as palavras constituídas por parêntesis “equilibrados”, como por exemplo ((( ))) (( ) ( )). As generalizações desta linguagem contendo diferentes símbolos são conhecidas por *linguagens de Dyck*.

   Esta linguagem é gerada pela gramática com símbolos terminais ‘(‘ e ‘)’ e símbolo inicial S, e as seguintes regras:
   - $S \to (S)$
   - $S \to SS$
   - $S \to \varepsilon$

2. A linguagem $\{a^i b^j c^{i+j} \mid i, j \geq 0\}$ pode ser descrita pela seguinte gramática livre de contexto:
   - $S \to aSc$
   - $S \to B$
   - $B \to bBc$
   - $B \to \varepsilon$

   Por exemplo: $S \to aSc \to aaScc \to aaaSccc \to aaaBccc \to aaabBcccc \to aaabbBccccc \to aaabbccccc$

De Volta aos Autómatos: Configurações e Transições

Vimos que a execução de um autómato $(Q, \Sigma, \delta, q_0, F)$ e a noção relacionada de aceitação de palavras eram definidas como base numa função $\hat{\delta}$ que calculava o estado $q' = \hat{\delta}(q, w)$ em que o autómato se encontra quando processa todos os símbolos de uma palavra w a partir do estado q.

Para a próxima classe de autómatos que introduziremos será útil definir uma noção de *configuração*, um par (q, w) constituído por um estado $q \in Q$ e uma palavra $w \in \Sigma^*$. Definimos depois uma *relação de transição* $\to$ *sobre configurações:*

$(q, w) \rightarrow (q', w')$ se $w = xw'$ e $\delta(q, x) = q'$

De forma mais compacta:

$(q, xw) \rightarrow (\delta(q, x), w)$

Note-se que a definição contempla o caso especial $x = \varepsilon$, em que nenhum símbolo é lido da palavra.

Definimos também o *fecho reflexivo e transitivo* $\rightarrow^*$ desta relação pelas seguintes regras:

1. $(q, w) \rightarrow^* (q, w)$
2. Se $(q, w) \rightarrow^* (q', w')$ e $(q', w') \rightarrow^* (q'', w'')$, então $(q, w) \rightarrow^* (q'', w'')$

A frase $(q, w) \rightarrow^* (q', w')$ significa que o autómato transita do estado q para o estado q' *em 0 ou mais passos*, consumindo os primeiros símbolos de w, e sendo w' constituída pelos símbolos restantes de w, ainda não consumidos.

É intuitivo então entender que o DFA $(Q, \Sigma, \delta, q_0, F)$ aceita a palavra w sse $(q_0, w) \rightarrow^* (q, \varepsilon)$ para algum $q \in F$, logo a linguagem por ele reconhecida pode ser definida alternativamente como:

$$L = \{w \mid w \in \Sigma^* \wedge (q_0, w) \rightarrow^* (q, \varepsilon) \text{ com } q \in F\}$$

Esta definição alternativa é viável também no caso dos autómatos não-determinísticos. Neste caso a definição da relação de transição será:

$(q, w) \rightarrow (q', w')$ se $w = xw'$ e $q' \in \delta(q, x)$

e a linguagem reconhecida será:

$$L = \{w \mid w \in \Sigma^* \wedge (q_0, w) \rightarrow^* (q, \varepsilon) \text{ para algum } q_0 \in Q_0 \text{ e } q \in F\}$$

Nesta perspectiva não vemos a execução como sendo baseada em múltiplos tokens presentes no autómato, mas sim em em múltiplas sequências

Autómatos PDA

Os autómatos de pilha, ou *pushdown automata*, generalizam a noção de autómato não-determinístico, acrescentando-lhes uma estrutura de dados, uma *pilha de símbolos.*

Uma pilha (*stack*) é uma estrutura de dados que é essencialmente uma lista em que o acesso só é possível à cabeça. A operação de inserção chama-se *push,* e a de remoção chama-se *pop.*

Utilizaremos notação da linguagem Haskell para representar pilhas:
- A pilha A : B : C : [] contém 3 elementos, sendo A o que se encontra no topo. Poderemos também escrever esta pilha como [A,B,C]
- A única forma de construir esta pilha é fazendo **push** de A na pilha [B,C]
- a operação **pop** nesta pilha lê o símbolo A, e altera a pilha para [B,C]
- o símbolo ++ será usado para denotar de forma conveniente uma sequência de **push**. Por exemplo [X,Y]++[A,B,C] denota a pilha [X,Y,A,B,C], o resultado de se fazer **push** sucessivamente de Y e de X em [A,B,C]

Um autómato de pilha é um tuplo $(Q, \Sigma, \Gamma, \delta, q_0, F)$ em que:

- Q é um conjunto *finito* e *não vazio* de estados
- Σ é o habitual alfabeto (*finito* e *não vazio*) de símbolos lidos pelo autómato
- Γ é também um alfabeto (igualmente *finito* e *não vazio*), dos símbolos que podem ser inseridos na pilha
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$ é uma função de transição
- $q_0 \in Q$ é o estado inicial do autómato
- $F \subseteq Q$ é um conjunto *finito* e *possivelmente vazio* de estados finais

Para se perceber o funcionamento destes autómatos atente-se no tipo da função de transição:

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$$

A função δ especifica, para cada estado do autómato e símbolos lidos da palavra e da pilha, um conjunto possível de transições: em cada execução poderá ocorrer uma transição diferente, escolhida de forma não-determinística, (não é habitual recorrer-se à visão da execução baseada em tokens, como nos NFAs). Os pontos $\delta(\ldots, \varepsilon, \ldots)$ especificam o comportamento quando não é lido nenhum símbolo da palavra.

Encontrando-se num determinado estado q, o autómato:

1. lê um símbolo x de uma palavra (ou, opcionalmente, nenhum símbolo, o que corresponde ao que temos designado por transições espontâneas);
2. faz ***pop*** do símbolo Y que se encontra no topo da pilha);
3. selecciona de forma não-determinística um dos possíveis pares $(q', S') \in \delta(q, x, Y)$;
4. transita para o estado q';
5. faz ***push*** da sequência de símbolos S' para a pilha (possivelmente nenhum).

Relação de Transição

As *configurações* são agora triplos (q, w, S) constituídos por um estado $q \in Q$, uma palavra $w \in \Sigma^*$, e uma pilha S constituída por símbolos de Γ.

Definimos depois a *relação de transição* $\to$ *sobre configurações* da seguinte forma:

Se $(q', S') \in \delta(q, x, Y)$ então $(q, xw, Y : S) \to (q', w, S' {+}{+} S)$

Note-se que esta definição obriga a que seja sempre **pop**ped (retirado) um símbolo da pilha, e como tal não haverá transições quando a pilha está vazia. Isto coloca um problema no arranque do autómato, em que a pilha está naturalmente vazia, mas a questão é resolvida inicializando a pilha com um qualquer símbolo, normalmente denotado por $\#$.

> *Alguns autores consideram uma definição alternativa em que são possíveis transições que não fazem **pop** da pilha.*

Linguagens de um Autómato de Pilha

Estes autómatos reconhecem duas linguagens diferentes, usando dois critérios diferentes.

A primeira linguagem corresponde à noção que usávamos nos DFA e NFA: uma palavra é aceite se o autómato atinge um estado final depois de a consumir. Esta noção é conhecida por *final state acceptance:*

$$L_{fs} = \{w \mid w \in \Sigma^* \wedge (q_0, w, [\#]) \to^* (q, \varepsilon, S) \text{ para alguma pilha } S \text{ e estado } q \in F\ \}$$

A alternativa é utilizar como critério de aceitação ser obtida uma pilha vazia depois de consumida a palavra (*empty stack acceptance*):

$$L_{es} = \{w \mid w \in \Sigma^* \wedge (q_0, w, [\#]) \to^* (q, \varepsilon, []) \text{ para algum estado } q\ \}$$

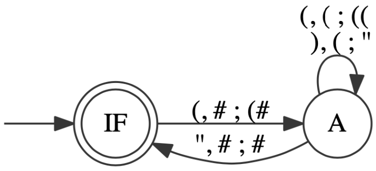
Exemplo

Um PDA para a linguagem de parêntesis pode ser desenhado como se segue:

- O alfabeto da pilha será $\{\ (\ ,\ \#\ \}$, e a pilha conterá inicialmente o símbolo $\#$
- Dois estados, IF e A
- Quando o autómato se encontra no estado IF a pilha contém apenas o simbolo inicial $\#$, e os parêntesis lidos da palavra até ao momento estão seguramente equilibrados, pelo que este estado é **final:** terminando aqui a palavra, ela deve ser aceite
- Em qualquer dos estados, sendo lido da palavra o símbolo **(**, ele será colocado na pilha depois de ser reposto o símbolo que foi *popped* do topo, qualquer que este tenha sido, e fica no estado A, uma vez que foram lidos mais **(** do que **)**

- Se no estado A for lido da palavra um símbolo **)**, sendo *popped* do topo da pilha um **(**, não é colocado nada na pilha: foi “fechado” o parêntesis que estava no topo, pelo que sai da pilha
- Encontrando-se a pilha “vazia” (só com o símbolo $\#$, o autómato transitará sem ler qualquer símbolo para o estado IF, permancendo a pilha só com $\#$, para que possa ser aceite a palavra lida até ao momento

q	x	Y		q'	S'
IF	(	#		A	[(, #]
A	(	(		A	[(, ( ]
A	)	(		A	[ ]
A	ε	#		IF	[#]



As etiquetas colocados pelo Jove sobre as transições traduzem a definição acima.

Note-se que não existe qualquer transição possível nas seguintes situações:

- é lido um **)** no estado IF
- é lido um **)** no estado A e não está um **(** no topo do pilha

Em ambas as situações temos seguramente uma palavra que não deve ser aceite, uma vez que se está a “fechar um parêntesis que não foi aberto”.

Vejamos um exemplo de execução:

(IF , (( ) ( )) , [#])
(A , ( ) ( )) , [(, #])
(A ,) ( )) , [(, (, #])
(A , ( )) , [(, #])
(A ,)) , [(, (, #])
(A ,) , [(, #])
(A , ε, [#])
(IF , ε , [#])

A noção de aceitação apropriada para este autómato é *final state acceptance,* uma vez que no final a pilha não está vazia.

Síntese Sistemática de um PDA para uma Gramática Livre de Contexto: Autómato de Reconhecimento Top-Down

A correspondência entre autómatos de pilha e gramáticas livres de contexto é estabelecida pelos dois resultados seguintes:

- É possível demonstrar que, dado um qualquer PDA, existe uma gramática livre de contexto que gera a linguagem reconhecida pelo autómato.

- Da mesma forma, dada uma gramática livre de contexto, é possível definir de forma sistemática um autómato de pilha que reconhece a linguagem especificada pela gramática.

Detalharemos aqui apenas este segundo processo, que é muito simples e ajuda a perceber a relação entre os dois formalismos.

Dada uma gramática livre de contexto com símbolo inicial S e n outros símbolos não-terminais A_i, com $i \in \{1, \ldots, n\}$

1. O autómato possui um único estado q_1, que é naturalmente o estado inicial
2. O critério de aceitação é por *empty stack* : as palavras são aceites quando a pilha se encontra vazia
3. O símbolo inicialmente colocado na pilha, que temos designado por $\#$, será agora S
4. O alfabeto Σ das palavras lidas pelo autómato será o conjunto de símbolos terminais da gramática, e o alfabeto Γ da pilha será constituído por todos os símbolos, terminais e não-terminais

5. Para cada símbolo terminal c teremos no autómato a seguinte **transição de *matching***:
   $\delta(q_1, c, c) = \{(q_1, [\,])\}$
6. Por cada símbolo não terminal $A \in \{S\} \cup \{A_i \mid 1 \leq i \leq n\}$ da gramática teremos no autómato a seguinte **transição de expansão***:*
   $\delta(q_1, \varepsilon, A) = \{(q_1, \alpha) \mid A \to \alpha \text{ é uma regra da gramática }\}$

O significado intuitivo destas regras é o seguinte:

- quando o símbolo (um terminal) lido da palavra se encontra também no topo da pilha (*matching*), isso significa que ele era esperado de acordo com uma regra, e é aceite, saindo do topo da pilha
- quando na palavra e no topo da pilha se encontram símbolos diferentes, nenhuma transição pode ser executada, e como a pilha não está vazia a execução pára sem que a palavra seja aceite — era esperado um símbolo que não aquele que foi lido
- quando no topo da pilha se encontra um símbolo não-terminal, então o autómato transita sem leitura de qualquer símbolo da palavra colocando na pilha, em vez desse símbolo, o lado direito de uma das suas regras

Este autómato faz um reconhecimento TOP-DOWN da linguagem, como o exemplo seguinte ilustra.

Exemplo
Considere-se ainda a mesma linguagem de parêntesis gerada pela gramática que vimos acima:

- $S \to (S)$
- $S \to SS$
- $S \to \varepsilon$

Construímos o autómato dado pelo método acima com os alfabetos $\Sigma = \{(,)\}$ e $\Gamma = \{(,), S\}$.
A função δ será definida pelas seguintes transições:

q	x	Y	Nome	q'	S'
q_1	ε	S	expand-1	q_1	[(, S,)]
q_1	ε	S	expand-2	q_1	[S , S]
q_1	ε	S	expand-3	q_1	[ ]
q_1	(	(	match-(	q_1	[ ]
q_1	)	)	match-)	q_1	[ ]

Para perceber o funcionamento deste autómato consideremos a palavra $(()())$.

A sua *árvore sintáctica* pode ser representada da seguinte forma:

```
          S
      /   |   \
    (     S     )
      /       \
    S           S
 /  |  \     /  |  \
(   S   )   (   S   )
    |           |
    E           E
```

Trata-se se uma representação da estrutura de uma palavra, de acordo com a gramática da linguagem.
Duas observações:

- a árvore de sintaxe abstrata testemunha a existência de uma relação muito proxima entre as gramáticas livres de contexto e os *tipos de dados indutivos*, por exemplo tal como existem na linguagem de programação Haskell. Poderíamos ter

```
data S = One ( S )
       | Two S S
       | Three
```

- as árvores de sintaxe têm um papel fundamental e omnipresente em informática. São utilizadas sistematicamente na *representação em computador de programas e dados*.

Simulemos agora a execução do autómato acima para a palavra $(()())$. Representaremos a construção por passos da árvore de sintaxe. Observa-se que em cada passo a pilha contém uma representação de parte da fronteira da árvore: a parte correspondente aos símbolos que ainda não foram lidos da palavra.

$(q_1$, (( ) ( )) , [S])
expand-1 $(q_1$, (( ) ( )) , [(, S,)])

```
            S
        /   |   \
      (     S     )
```

match-($(q_1$, ( ) ( )) , [S,)])
expand-2 $(q_1$, ( ) ( )) , [S, S,)])

```
            S
        /   |   \
      (     S     )
        /       \
      S           S
```

expand-1 $(q_1$, ( ) ( )) , [(, S,), S,)])

```
            S
        /   |   \
      (     S     )
        /       \
      S           S
   /  |  \
  (   S   )
```

match-($(q_1$,) ( )) , [S,), S,)])
expand-3 $(q_1$,) ( )) , [ ), S,)])

```
            S
        /   |   \
      (     S     )
        /       \
      S           S
   /  |  \
  (   S   )
      |
      E
```

match-) $(q_1$, ( )) , [S,)])
expand-1 $(q_1$, ( )) , [(, S,),)])

```
            S
        /   |   \
      (     S     )
        /       \
      S           S
   /  |  \     /  |  \
  (   S   )   (   S   )
      |
      E
```

match-($(q_1$,)) , [S,),)])
expand-3 $(q_1$,)) , [ ),)])

```
            S
        /   |   \
      (     S     )
        /       \
      S           S
```

```
  /  |  \      /  |  \
 (   S   )    (   S    )
     |            |
     E            E
```

match-) $(q_1\,,\,)\,,[\ )\,])$
match-) $(q_1\,,\varepsilon\,,[\ ])$

A execução do autómato corresponde à seguinte derivação na gramática:
$S \to (S) \to (SS) \to ((S)S) \to (()S) \to (()(S)) \to (()())$

Método Alternativo: Autómato de Reconhecimento Bottom-Up

É possível conceber um autómato de pilha que constrói a árvore de sintaxe de forma BOTTOM-UP, começando pelas folhas e subindo até à raíz. Temos no entanto para isso de generalizar um pouco a definição de PDA, admitindo que um autómato pode em cada transição ler uma sequência (possivelmente vazia) de símbolos a partir da pilha. Prova-se que esta definição generalizada, que não detalharemos aqui, é equivalente à que temos utilizado.

Dada uma gramática livre de contexto com símbolo inicial S e n outros símbolos não-terminais A_i, com $i \in \{1, \ldots, n\}$

1. O autómato possui um único estado q_1, que é naturalmente o estado inicial
2. O critério de aceitação é por *empty stack* : as palavras são aceites quando a pilha contiver apenas o símbolo S (inicial da gramática)
3. A pilha está vazia na configuração inicial do autómato
4. Tal como no método anterior, o alfabeto Σ das palavras lidas pelo autómato será o conjunto de símbolos terminais da gramática, e o alfabeto Γ da pilha será constituído por todos os símbolos, terminais e não-terminais
5. Para cada símbolo terminal c teremos no autómato a seguinte **transição de *shifting***, que o transfere da palavra para a pilha: $\delta(q_1, c, [\,]) = \{(q_1, [c\,])\}$
6. Por cada símbolo não terminal $A \in \{S\} \cup \{A_i \mid 1 \leq i \leq n\}$ da gramática teremos no autómato a seguinte **transição de redução***:* $\delta(q_1, \varepsilon, [\alpha_n, \ldots, \alpha_1]) = \{(q_1, [A]) \mid A \to \alpha_1 \ldots \alpha_n \text{ é uma regra da gramática }\}$

Note-se que as transições de redução retiram da pilha os n símbolos do topo, quando eles correspondem (por ordem inversa) ao lado direito de uma regra da gramática, substituindo-os pelo lado esquerdo dessa regra (um símbolo não-terminal).

O significado intuitivo destas regras é o seguinte:
- os símbolos terminais vão sendo transferidos para a pilha
- eventualmente encontramos no topo da pilha o lado direito de uma regra constituído apenas por símbolos terminais, que pode ser substituído pelo lado esquerdo — identificámos um fragmento da árvore, ainda separado
- Se no topo da pilha encontramos o lado direito de uma regra contendo símbolos não terminais, eles serão também substituídos pelo lado esquerdo: estamos agora a integrar várias árvores, construindo uma árvore maior a partir delas
- se eventualmente tivermos apenas S na pilha, estando a palavra vazia, chegámos à raíz da árvore, sendo a palavra reconhecida.

Exemplo
Retomemos a gramática da linguagem de parêntesis:
- $S \to (S)$
- $S \to SS$
- $S \to \varepsilon$

Pelo método descrito obtemos o seguinte autómato:

q	x	Y	Nome	q'	S'
q_1	ε	[ ), S, ( ]	reduce-1	q_1	[S]
q_1	ε	[S , S]	reduce-2	q_1	[S]
q_1	ε	[ ]	reduce-3	q_1	[S]
q_1	(	[]	shift-(	q_1	[( ]
q_1	)	[]	shift-)	q_1	[ )]

Simulemos a sua execução, desenhando os fragmentos da árvore de sintaxe à medida que são construídos. Depois de cada passo de redução, a pilha contém símbolos não terminais já lido, juntamente com símbolos terminais correspondentes a (sub-)palavras já reconhecidas dentro da palavra inicial.

(q_1 , (( ) ( )) , [ ])

shift-((q_1 , ( ) ( )) , [( ])

shift-((q_1 ,) ( )) , [(, ( ])

reduce-3 (q_1 ,) ( )) , [S , **(** , **(**])

```
S
|
E
```

shift-) (q_1 , ( )) , [**)**, S **,** **(** , ( ])

reduce-1 (q_1 , ( )) , [S , **(**])

```
    S
 /  |  \
(   S   )
    |
    E
```

shift-((q_1 ,)) , [(, S , ( ])

reduce-3 (q_1 ,)) , [S , (, S , ( ])

> *Nesta configuração foi lido um (, seguido de uma palavra aceite, seguida de outro (, seguido de outra palavra aceite. E falta ler dois))*

```
    S
 /  |  \
(   S   )       S
    |           |
    E           E
```

shift-) (q_1 ,) , [**)**, S **,** **(** , S , ( ])

reduce-1 (q_1 ,) , [S , S , ( ])

```
    S           S
 /  |  \     /  |  \
(   S   )   (   S   )
    |           |
    E           E
```

reduce-2 (q_1 ,) , [S , ( ])

```
          S
      /       \
    S           S
 /  |  \     /  |  \
(   S   )   (   S   )
    |           |
    E           E
```

shift-) (q_1 , ε , [**)**, S **,** **(**])

reduce-1 (q_1 , ε , [S])

```
          S
      /   |   \
    (     S     )
      /       \
    S           S
 /  |  \     /  |  \
(   S   )   (   S   )
    |           |
    E           E
```

Parsing

Atingimos neste ponto uma fronteira entre duas áreas, a da *teoria da computação*, e a do *processamento de linguagens e compilação.*

Em aplicações informáticas, de cada vez que escrevemos algo que venha a ser interpretado por um computador, terá de ser feito o reconhecimento das frases que escrevemos — para verificar se pertencem ou não à linguagem convencionada — mas além disso, caso pertençam, terá também de ser construída a respectiva árvore sintáctica.
Este processo simultâneo de reconhecimento e construção da árvore é designado por *parsing*, e os programas que o implementam por *parsers.*

Os dois métodos para a construção de autómatos apresentados acima correspondem a duas formas típicas de fazer *parsing,* no entanto o que acabamos de ver corresponde apenas a metade da história. Faltaria:

- especificar como são construídas as árvores de sintaxe — seria fácil fazê-lo, os exemplos acima já ilustram esse processo através de um exemplo

- muito mais difícil seria tornar o processo *determinístico* e *eficiente*. Claramente os autómatos construídos não são determinísticos! Nas simulações que apresentamos acima foi sempre dado o passo certo para permitir a aceitação da palavra, mas seriam possíveis outras execuções que não levariam a essa aceitação. Por exemplo, no segundo método, facilmente existe ambiguidade entre as transições *shift* e *reduce*. Tornar a execução deterministicamente obriga a efectuar *lookaheads*, “espreitando” os próximos símbolos por forma a decidir qual a transição a aplicar. É toda uma área da ciência da computação.

  Aqui fica um link para um introdução curta a este assunto:
  https://www.dickgrune.com/CS/Formal_Languages/Overview_of_Parsing_Using_Push-Down_Automata.pdf

Variante

No livro Automata and Computability: A Programmer’s Perspective é proposta a conversão de gramáticas livres de contexto em autómatos PDA com 3 estados, I, M, F, sendo I o inicial e F final. M corresponde a q_1 no método anterior. Nesta variante:

- o símbolo inicial S da gramática difere do $\#$ inicialmente colocado na pilha,
- a aceitação é por *final state* no estado F,
- acrescem as seguintes transições:
  - $\delta(I, \varepsilon, \#) = \{(M, [S, \#])\}$, ou seja o autómato transita de I para M sem ler símbolos da palavra, e colocando S na pilha
  - $\delta(M, \varepsilon, \#) = \{(F, [\#])\}$: quando a pilha fica apenas com o símbolo $\#$ o autómato transita para o estado final sem ler qualquer símbolo da palavra

Exercício

Considere a gramática livre de contexto com as seguintes regras para o alfabeto $\{a, b\}$, com símbolos não-terminais S e X, sendo S o inicial:

- $S \to XS$
- $S \to \varepsilon$
- $X \to aXb$
- $X \to Xb$
- $X \to ab$

1. Escreva uma derivação para a palavra “aabbaabbb”
2. Defina um autómato de pilha que reconheça a linguagem definida por esta gramática, escolhendo uma das variantes apresentadas em cima
3. Simule uma execução do autómato que definiu, que aceite a mesma palavra “aabbaabbb”

Pode também repetir este exercício para a gramática da linguagem $\{a^i b^j c^{i+j} \mid i, j \geq 0\}$ apresentada acima neste módulo.

Exercício
Considere a seguinte gramática livre de contexto para expressões aritméticas construídas com os operadores + e *, parêntesis (,), e variáveis id $\in \{x, y, z, ...\}$

- S → S+X
- S → X
- X → X*Y
- X → Y
- Y → (S)
- Y → id (uma regra para cada variável)

1. Desenhe a árvore de sintaxe da expressão $x + y * z$ segundo esta gramática
2. Defina os autómatos para o reconhecimento da linguagem definida por esta gramática:
    a. pelo método *top-down*
    b. pelo método *bottom-up*
3. Simule a execução do autómato e a construção da árvore de sintaxe, em ambos os casos.