CAPÍTULO 7

AUTÓMATOS DE PILHA

7.1. Introdução

As gramáticas livres de contexto são a base da construção das linguagens de programação e são por isso muito importantes. Existirá uma classe de autómatos que se possa associar a todas as linguagens livres de contexto?

No Cap. 4 vimos que para linguagens regulares, uma subclasse das livres de contexto, temos os autómatos finitos, determinísticos e não determinísticos, que são uma forma de representação e especificação de linguagens. No entanto como sabemos os autómatos finitos não conseguem representar todas as linguagens.

Sempre que uma linguagem é infinita e que as suas cadeias têm uma estrutura tal que é necessário guardar memória da parte da cadeia já lida, é impossível encontrar um autómato finito para a linguagem. Por exemplo, nenhum autómato finito é capaz de aceitar a linguagem $L = \{a^n b^n :, n \geq 0\}$ porque para isso é necessário uma capacidade infinita de contagem.

Também no caso de cadeias simétricas em relação a um ponto central, L = $\{v \in \Sigma^*$ tal que $v=ww^R$, $w \in \Sigma^*\}$, se verifica o mesmo, sendo até necessário aqui a memorização de uma cadeia de símbolos (e não apenas o seu número) por ordem inversa. Este facto sugere a utilização de uma pilha como memória, pilha do tipo LIFO (*last in first out*) resultando numa classe de máquinas chamadas **autómatos de pilha** (***pushdown automata***, **PDA**) ou **aceitadores de pilha** (***pushdown accepters***, **PDA**). A pilha pode ter uma dimensão arbitrariamente grande, até ∞.

7.2. O autómato de pilha, PDA – *Push Down Automata*

Retomando aqui o esquema geral de um autómato, que vimos no Cap. 2, o PDA tem uma pilha adicional em relação ao autómato finito, como ilustrado na Fig. 7.2.1. Tal como ele tem um registo de entrada, onde se escrevem cadeias, e uma unidade de controlo com um certo número finito de estados internos. Pode ler e escrever no topo da pilha em cada instante.

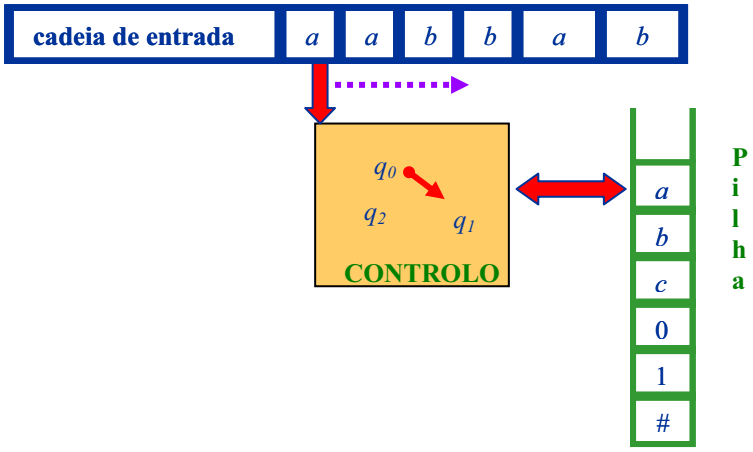


Figura 7.2.1 Esquema do autómato de pilha.

O PDA lê caracteres de entrada, estando num certo estado interno; lê o símbolo que está no momento presente no topo da pilha; em função de tudo o que lê, e em função do seu estado interno actual, executa uma movida do autómato para um outro estado interno, e eventualmente escreve (ou apaga) um ou vários símbolos no topo da pilha. Isto é, a função de transição δ tem três argumentos de entrada e dois argumentos de saída, como na Fig. 7.2.2.

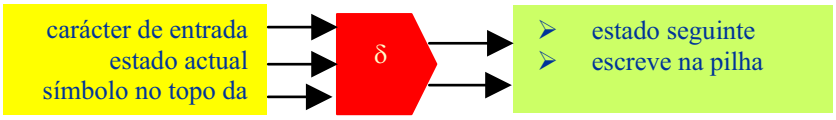


Figura 7.2.2 Função de transição do PDA.

Como veremos posteriormente, se se permitir que este autómato seja não determinístico, obtém-se uma classe de autómatos que aceitam exactamente as linguagens livres de contexto. Neste caso, contrariamente ao que acontece para os autómatos finitos, e infelizmente, não há equivalência entre os determinísticos e os não-determinísticos.

A classe dos autómatos de pilha determinísticos define uma nova família de linguagens, as linguagens determinísticas livres de contexto, muito importantes para as linguagens de programação e que são um subconjunto das linguagens livres de contexto maior do que as linguagens regulares (Fig. 7.2.3).

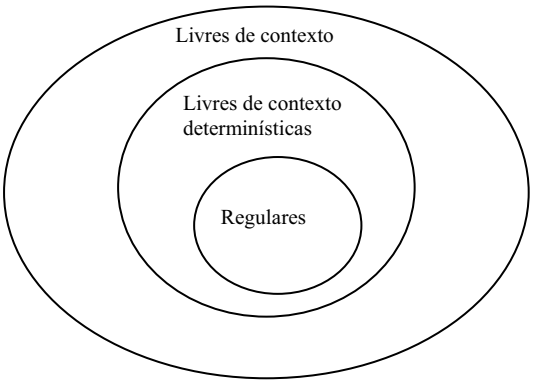


Figura 7.2.3. Relação entre as linguagens regulares, livres de contexto determinísticas e livres de contexto.

7.3. Autómatos de pilha não-determinísticos (NPDA)

Num autómato de pilha não determinístico, cujo acrónimo é NPDA (do inglês), cada movida da unidade de controlo

- lê um símbolo na cadeia de entrada,
- altera o conteúdo da pilha através das operações usuais em pilhas.

Cada movida da unidade de controlo é determinada por

- símbolo de entrada actual
- símbolo no topo da pilha no momento presente.

O resultado de uma movida da unidade de controlo é

- um novo estado da unidade de controlo
- uma mudança no topo da pilha.

Poderemos conceber também PDA's transdutores, com um ficheiro de saída onde ele escreve. No entanto só nos interessaremos aqui pelos PDA aceitadores.

7.3.1. Definição formal de autómato de pilha não-determinístico

Definição 7.3.1.

Um **aceitador não determinístico de pilha (NPDA- *nondeterministic pushdown accepter*)** é definito por um septeto

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

em que

Q : conjunto finito de estados internos da unidade de controlo

Σ: (sigma) o alfabeto de entrada

Γ: (gama) conjunto finito de símbolos chamado **alfabeto da pilha**

δ: $Q\times(\Sigma\cup\{\lambda\})\times\Gamma\rightarrow$ subconjuntos finitos de $Q\times\Gamma^*$, é a função de transição, ver a figura 7.2.2.

$q_0 \in Q$: estado inicial da unidade de controlo

$z_0 \in \Gamma$: símbolo de inicialização da pilha, geralmente #

$F \subseteq Q$: conjunto de estados finais (aceitadores).

Note-se que, pela definição, uma transição pode ter como carácter de entrada λ, sendo neste caso uma transição-λ. Existem dois alfabetos num NPDA: o de entrada, Σ (sigma), e o da pilha, Γ (gama), que são em geral diferentes (mas não obrigatoriamente).

Por outro lado a definição de transição exige um símbolo no topo da pilha; se a pilha estiver vazia, as movidas serão inibidas, o aceitador pára.

O contradomínio de δ deve ser um conjunto finito porque $Q\times\Gamma^*$ é um conjunto infinito e por isso tem subconjuntos finitos e infinitos. Embora o NPDA possa ter várias escolhas possíveis para as suas movidas, elas devem ser em número finito de possibilidades.

De uma movida resulta a escrita de um carácter no topo da pilha. Esta escrita é feita carácter a carácter, da direita da cadeia para a esquerda (Fig.7.3.1).

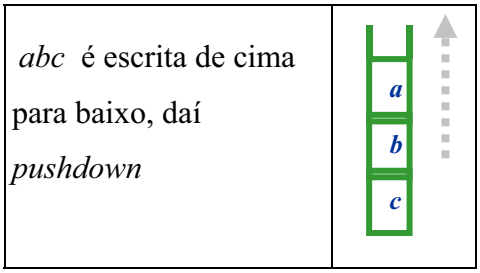


Figura 7.3.1 Escrita na pilha do NPDA

As operações sobre a pilha são do tipo "***push***" – introduzir no topo um carácter adicional, ou do tipo "***pop***"- apagar um carácter, ou do tipo **substituição** de um carácter por outro (mantendo-se o tamanho da pilha). Uma substituição é feita por uma sequência *pop-push*.

A **inicialização** da pilha pode ser feita com qualquer símbolo de Γ. Normalmente usa-se para símbolo inicial da pilha o carácter especial # (cardinal).

Uma movida do autómato tem a seguinte sintaxe, sendo δ a função de transição.:

$$\delta(q_0, a, 0) = \{ (q_1, 10) \}$$

Sendo

q_0 : estado actual

a : carácter lido à entrada

0 : símbolo no topo da pilha

q_1 : estado seguinte

10: escrever 1 no topo, por cima do zero; i.e, "push 1".

A tabela 7.3.1 ilustra outros tipos de transições possíveis.

Tabela 7.3.1 Operações sobre a pilha no NPDA

Transições	Operações sobre a a pilha	Significado
1. $\delta(q_0, a, \#) = \{ (q_1, 0\#) \}$	*push*	acrescenta 0
2. $\delta(q_0, b, 1) = \{ (q_1, \lambda) \}$	*pop*	apaga 1
3. $\delta(q_1, b, 0) = \{ (q_1, 1) \}$	substituição	substitui 0 por 1
4. $\delta(q_0, b, 1) = \{ (q_1, 1) \}$	nenhuma	não altera
5. $\delta(q_1, \lambda, 0) = \{ (q_2, \lambda) \}$	*pop*	apaga 0
6. $\delta(q_1, \lambda, 0) = \{ (q_2, 10) \}$	*push*	acrescenta 1

Exemplo 7.3.1.

Considere-se a transição num NPDA definida por

$$\delta(q_1, a, b) = \{(q_2, cd), (q_3, \lambda)\}$$

Quando o autómato está no estado q_1, se aparecer um a na cadeia de entrada, então acontecerá uma de duas coisas:

i) - a unidade de controlo passa ao estado q_2

- substitui b pela cadeia cd no topo da pilha (que fica assim com mais um carácter) , ou seja, *pop b* e *push cd*.

ii) - a unidade de controlo passa ao estado q_3

- na pilha substitui b por λ, isto é, apaga b do topo da pilha (que fica assim com menos um carácter), ou seja, *pop b*.

Exemplo 7.3.2.

Seja o NPDA com

$Q = \{q_0, q_1, q_2, q_3,\}$
$\Sigma = \{a, b\}$
$\Gamma = \{\#, 0, 1\}$
$z_0 = \#$
$F = \{q_3\}$

e

$\delta(q_0, a, \#) = \{(q_1, 1\#), (q_3, \lambda)\}$,
$\delta(q_0, \lambda, \#) = \{(q_3, \lambda)\}$,
$\delta(q_1, a, 1) = \{(q_1, 11)\}$,
$\delta(q_1, b, 1) = \{(q_2, \lambda)\}$,
$\delta(q_2, b, 1) = \{(q_2, \lambda)\}$,
$\delta(q_2, \lambda, \#) = \{(q_3, \lambda)\}$.

Note-se que δ não é uma função total. Para os pontos do seu domínio para os quais não se encontra definida, assume-se que transita para o conjunto nulo representando uma **configuração morta** do NPDA, isto é, se o autómato cair num estado para o qual δ não está definido, nunca mais de lá sai, estando por isso morto.

Há duas transições neste exemplo que merecem uma análise mais detalhada:

1ª : $\delta(q_1, a, 1) = \{(q_1, 11)\}$

- se no estado q_1 lê um a, tendo um 1 no topo da pilha, mantém-se em q_1 e acrescenta um 1 à pilha, ou seja, *push* 1.

- por cada a que aparece acrescenta um 1 à pilha, enquanto existirem 1's na pilha.

2ª : $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$

- se no estado q_2 lê um b, tendo 1 no topo da pilha, passa ao estado q_2 e apaga o 1 do topo da pilha, ou seja *pop* 1.

- por cada b que aparece nestas circunstâncias, apaga um 1 da pilha, até que fique apenas o símbolo inicial #.

A passagem do estado q_1 ao estado q_2 acontece quando aparece o primeiro b.

Que faz então o autómato? Quando atinge q_3 ?

Ele conta os a's que vai lendo através dos 1's que escreve na pilha. Num certo instante tem tantos 1's quantos os a's que leu. Depois aparece um b, muda de estado e apaga um 1. Por cada b que lê depois, apaga um 1. Se o número de b's que leu até um certo instante igualar o número de a's que leu na primeira parte, apaga todos os 1's e chega ao símbolo inicial # que está no fundo da pilha. E agora se estiver no fim da cadeia, não havendo na leitura mais do que o λ, passa ao estado aceitador q_3. O estado q_3 é alcançado, por este modo, quando o NPDA lê uma cadeia de $L = \{a^n b^n : n \geq 0\}$.

Note-se que após a leitura do primeiro a o autómato pode passar directamente ao estado final q_3, devido a

$$\delta(q_0, a, \#) = \{(q_1, 1\#), (q_3, \lambda)\}$$

e por aqui aceita a linguagem $L=\{a\}$.

O estado q_3 é assim alcançado quando o NPDA lê uma cadeia de $L = \{a^n\, b^n : n \geq 0\} \cup \{a\}$. Pode-se afirmar, por analogia com o que acontece nos DFA, que o NPDA aceita esta linguagem. Se na primeira transição eliminarmos a possibilidade (q_3, λ), a linguagem aceite pelo autómato será simplesmente $L = \{a^n\, b^n : n \geq 0\}$.

Note-se que o NPDA muda de estado após o aparecimento do primeiro *b*. Se aparecer depois um *a*, o autómato passa ao estado morto porque de q_2 não há uma transição com *a*. Assim, deste modo simples, impõe a restrição de que os *b'*s apareçam depois dos *a'*s. Se não houvesse mudança de estado naquela altura então os *b'*s poderiam ser seguidos de *a'*s, resultando numa linguagem bem diferente.

O que significa aceitar uma linguagem, num NPDA? Vejamos a noção de descrição instantânea do aceitador.

Descrição instantânea de um NPDA

A descrição instantânea do NPDA é completamente definida pelo tripleto

$$(q, w, u)$$

em que

q : é o estado actual do autómato
w : é a parte ainda não lida da cadeia de entrada
u : é o conteúdo da pilha, sendo o topo o símbolo mais à esquerda.

Uma movida de uma descrição para outra descrição é denotada pelo símbolo ⊢—, já nosso conhecido dos capítulos anteriores.

Por exemplo

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

só é possível se existir a transição

$$\delta(q_1, a, b) = \{(q_2, y)\}$$

As movidas com um número arbitrário de passos são denotadas por $\overset{*}{\vdash}$. Por outro lado a notação $\vdash_M$ ou $\overset{*}{\vdash}_M$ indica que as movidas se referem ao autómato específico *M*.

Linguagem aceita por um NPDA

Estamos agora em condições de definir formalmente a condição de aceitação de um NPDA.

Definição 7.2. Aceitação por estado final

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ um autómato de pilha não determinístico.

A linguagem aceite por M é

$$L(M) = \left\{ w \in \Sigma^* : (q_0, w, z_0) \overset{*}{\vdash}_M (p, \lambda, u), p \in F, u \in \Gamma^* \right\}$$

Ou seja, a linguagem aceite por *M* é o conjunto de todas as cadeias capazes de colocarem o autómato num estado aceitador (*F* é o conjunto dos estados aceitadores) no final da leitura cadeia. O conteúdo da pilha, *u*, é irrelevante para esta definição de aceitação, isto é, nesta definição a pilha pode ficar com um conteúdo arbitrário desde que termine num estado aceitador, contrariamente ao que ao que acontece na definição seguinte.

Definição 7.3. Aceitação por pilha vazia

Diz-se que um NPDA aceita uma cadeia por pilha vazia se ele, depois de inicializado e depois da leitura da cadeia, termina com a pilha vazia e num estado qualquer.

Formalmente, o NPDA

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

aceita a linguagem $L(M)$ por pilha vazia se

$$L(M) = \{ w \in \Sigma^* : (q_0, w, z_0) \overset{*}{\vdash}_M (q, \lambda, \lambda), \text{ para algum } q \in Q \}$$

Pode-se demonstrar que as duas definições são equivalentes: se existe um NPDA que aceita uma linguagem *L* por um estado final, então existe um NPDA que aceita a mesma *L* pela pilha

vazia, e vice-versa. Ver em Hopcroft e Coll. (p. 230-236).

Exemplo 7.3.2.1.

Qual o NPDA que aceita

$$L = \{w \in \{a, b\}^* : n_a(w)=n_b(w)\} \ ?$$

É necessário contar o número de *a'*s e *b*'s, o que se pode fazer facilmente usando a pilha, tal como no exemplo anterior. Mas agora a ordem dos *a'*s e *b*'s é irrelevante, e por isso é necessário uma nova estratégia construtiva.

Uma cadeia pode iniciar-se por *a* ou por *b*. Como o número de *a*'s que surgiu até agora pode ser maior ou menor do que o número de *b*'s que já foram lidos, encarregando-se o futuro de igualar os dois, é necessário contar os *a*'s e os *b'*s com símbolos diferentes, dado que não podemos ter aqui a representação de números negativos.

Define-se um símbolo para contar o número de *a*'s, por exemplo o 0.

Suponhamos que a cadeia se inicia por *a*'s: sempre que aparece um *a* acrescenta-se um 0 à pilha (*push*). Depois se aparece um *b*, apaga-se um zero da pilha (*pop*). Mas se vierem seguidos tantos *b*'s que esgotem os 0's, então introduz-se na pilha um 1 por cada *b* em excesso, isto é quando não houver mais zeros, restando apenas o símbolo de inicialização da pilha, e aparecer um *b*, acrescenta-se um 1 à pilha (*push*). Agora quando o topo da pilha for um 1 e aparecer um *a*, apaga-se esse 1 (*pop*). Há aqui uma luta entre os *a*'s e os *b*'s para ver quem ganha. Um *a* ou escreve 0 ou apaga 1, um *b* ou escreve 1 ou apaga 0.

Teremos assim as seguintes transições:

$\delta(q_0, \lambda, \#) = \{(q_f, \#)\}$,	aceita a cadeia vazia, $n_a(w)=n_b(w)=0$
$\delta(q_0, a, \#) = \{(q_0, 0\#)\}$,	um *a* inicial acrescenta 0
$\delta(q_0, b, \#) = \{(q_0, 1\#)\}$,	um *b* inicial acrescenta 1
$\delta(q_0, a, 0) = \{(q_0, 00)\}$,	um *a* acrescenta 0
$\delta(q_0, b, 0) = \{(q_0, \lambda)\}$,	um *b* apaga 0
$\delta(q_0, a, 1) = \{(q_0, \lambda)\}$,	um *a* apaga 1
$\delta(q_0, b, 1) = \{(q_0, 11)\}$.	um *b* acrescenta 1.

A transição-λ introduz indeterminismo no autómato. Quando aparece um a à entrada, pode acontecer uma de duas coisas:

- o autómato transita para o estado q_f sem consumir o a da entrada (transição - λ)

- o autómato consome o carácter de entrada, a, escreve 0 na pilha e mantém-se em q_0

O NPDA é formalmente definido pelo septeto

$$M = (\{q_0, q_f\}, \{a,b\}, \{\#, 0, 1\}, \delta, q_0, \#, \{q_f\}).$$

Por exemplo a cadeia *abbabbaa* produz as seguintes movidas:

$(q_0, abbabbaa, \#) \vdash (q_0, abbabba, 0\#) \vdash (q_0, abbabb, 00\#) \vdash (q_0, abbab, 0\#)$

$\vdash (q_0, abba, \#) \vdash (q_0, abb, 0\#) \vdash (q_0, ab, \#) \vdash (q_0, a, 1\#) \vdash$

$\vdash (q_0, \lambda, \#) \vdash (q_f, \#)$

e portanto é aceite, por estado final.

7.3.2 Grafo de um autómato de pilha

O grafo de um NPDA tem algumas semelhanças com o de um NFA. Os nós são os estados e as arestas representam as transições. As etiquetas das arestas têm que conter toda a informação necessária. Não há uma convenção única de etiquetagem. Conhecem-se duas principais:

- a notação de Linz e do JFLAP

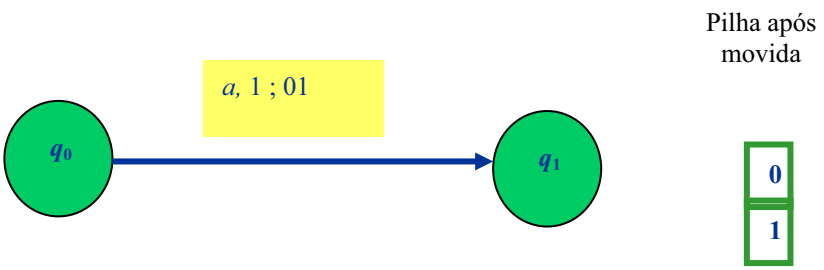


Figura 7.3.2.1. Notação JFLAP.

Com o significado seguinte:

Estando o autómato no estado q_0, se lê a e se a pilha estiver com 1 no topo, o autómato

- passa ao estado q_1,
- faz o "*pop*" da pilha (apaga 1)
- e depois o "*push*" de 01 (escreve 1 seguido de 0, topo à esquerda, 0)

- a notação de Taylor e DEM

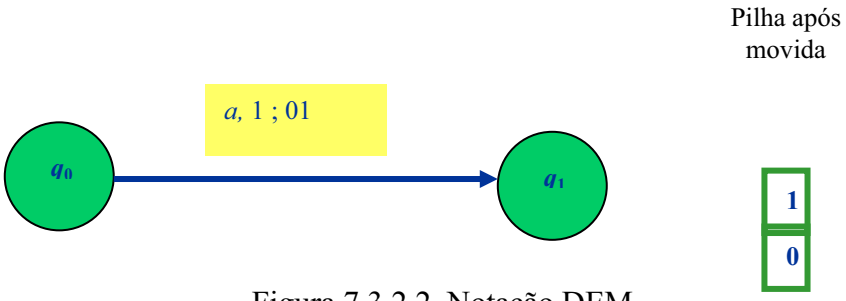


Figura 7.3.2.2. Notação DEM

Com o significado: estando o autómato no estado q_0, se lê a e se a pilha estiver com 1 no topo, o autómato

- passa ao estado q_1,
- faz o "*pop*" da pilha (apaga 1)
- e depois o "*push*" de 0 e 1 (escreve 0 seguido de 1, topo à direita, 1)

A diferença entre as duas está apenas na ordem porque se anota a inserção dos símbolos na pilha: da direita para a esquerda, 01# (JFLAP) ou da esquerda para a direita #01 (DEM).

O grafo do NPDA, com $L = \{w \in \{a, b\}^* : n_a(w)=n_b(w)\}$, com notação do JFALP, será

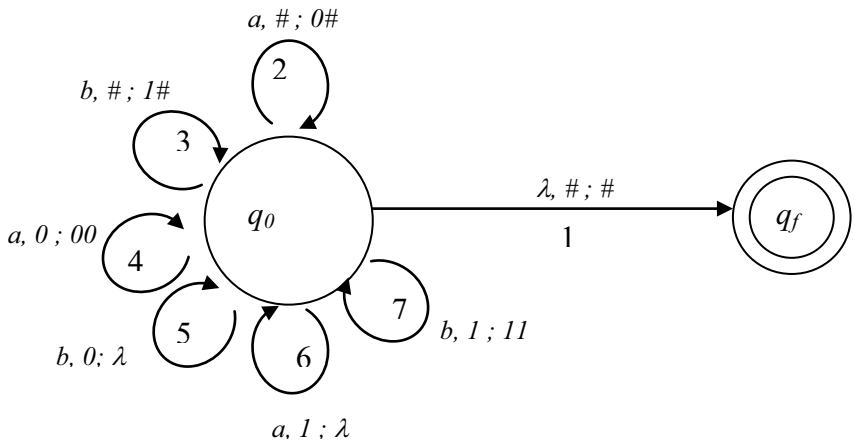


Figura 7.3.2.3 Grafo do NPDA do exemplo 7.3.2.1

O estado aceitador desenha-se com dupla circunferência. (Ver o mesmo exemplo em Taylor 538, com o DEM).

Exemplo 7.3.2.2

Construir um NPDA que reconheça a linguagem dos palíndromos pares

$$L = \{ ww^R : w \in \{a, b\}^+ \}$$

O símbolos são inseridos na pilha pela ordem inversa por que são apagados.

Descrição informal (Hopcroft, 221):

1. Inicia-se no estado q_0 que representa o "palpite" de que ainda não chegámos ao meio da cadeia, i.e., ainda não chegámos ao fim da cadeia w que deve ser seguida pela sua reversa.

   Enquanto se estiver em q_0, lêem-se os símbolos e vão-se colocando na pilha, fazendo o "push" de cada carácter, um de cada vez.

2. Em qualquer altura pode-se fazer o palpite de que se atingiu o meio. Nessa altura a pilha contém w, com o carácter mais à direita no topo, e o primeiro carácter no fundo.

   Transita-se então para o estado q_1.

   Como o autómato é não determinístico, podemos fazer os dois palpites: supõe-se que se atingiu o meio de w e supõe-se também que não, mantendo o autómato no estado q_0 continuando a ler entradas e a colocá-las na pilha.

3. Depois de transitar para o estado q_1, compara-se o símbolo da entrada com o símbolo no topo da pilha. Se forem iguais, consome-se o símbolo de entrada e apaga-se o topo da pilha (pop). Se forem diferentes, fizemos um palpite errado: w não é seguido de w^R. Este ramo morre, embora outros ramos do autómato não determinístico possam sobreviver e eventualmente atingir o estado de aceitação.

4. Se se esvaziar a pilha, leu-se de facto uma entrada w seguida pelo seu reverso w^R. Aceita-se a entrada que foi lida até este ponto, criando para isso um estado q_2.

Em termos das transições, poderemos dividi-las em várias partes:

1ª parte para fazer o "push" de w

$$\delta(q_0, a, a) = \{(q_0, aa)\},$$
$$\delta(q_0, b, a) = \{(q_0, ba)\},$$
$$\delta(q_0, a, b) = \{(q_0, ab)\},$$
$$\delta(q_0, b, b) = \{(q_0, bb)\},$$
$$\delta(q_0, a, \#) = \{(q_0, a\#)\},$$
$$\delta(q_0, b, \#) = \{(q_0, b)\}.$$

2ª parte palpitar o meio da cadeia

$$\delta(q_0, \lambda, a) = \{(q_1, a)\},$$
$$\delta(q_0, \lambda, b) = \{(q_1, b)\}.$$

3ª parte para comparar w^R com o conteúdo da pilha

$$\delta(q_1, a, a) = \{(q_1, \lambda)\},$$
$$\delta(q_1, b, b) = \{(q_1, \lambda)\},$$

4ª parte para fazer a aceitação

$$\delta(q_1, \lambda, \#) = \{(q_2, \#)\},$$

Por exemplo a cadeia *ababbaba*, em que *w=abab*, pode ser aceite pelas seguintes transições

$(q_0, ababbaba, \#) \vdash (q_0, ababbab, a\#) \vdash (q_0, ababba, ba\#) \vdash (q_0, ababb, aba\#)$
$\vdash (q_0, abab, baba\#) \vdash (q_1, abab, baba\#) \vdash (q_1, abab, baba\#)$
$\vdash (q_1, aba, aba\#) \vdash (q_1, ab, ba\#) \vdash (q_1, a, a\#) \vdash (q_1, \lambda, \#)$
$\vdash (q_2, \#)$

e portanto é aceite. O grafo do autómato será o da Fig. 7.3.2.4.

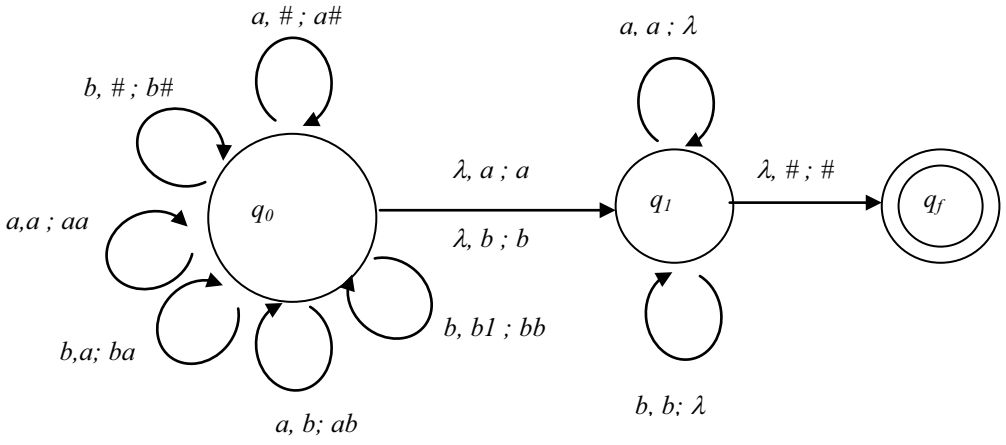


Figura 7.3.2.4. NPDA do exemplo 7.3.2.2.

7.4. Autómatos de pilha e linguagens livres de contexto

A relação entre linguagens livres de contexto e NPDA é análoga à relação entre autómatos finitos e linguagens regulares: para toda a linguagem livre de contexto existe um NPDA que a aceita; se um NPDA aceita uma linguagem então ela é livre de contexto.

A prova é construtiva: vamos construir um NPDA para uma CFL qualquer. Para isso usamos a forma normalizada da Greibach , em que todas as produções são da forma

$$A \rightarrow ax,\ a \in T,\ x \in V^*$$

por ser mais fácil e intuitivo. Repare-se que esta escolha não tira generalidade à prova porque para qualquer linguagem livre de contexto existe uma FN de Greibach, como vimos no Cp. 6.

7.4.1. NPDA's para gramáticas livres de contexto

Suponhamos que a CFG está na FN de Greibach. A construção de um NPDA para esta gramática faz-se da seguinte forma:

- o NPDA implementa a derivação pela extrema-esquerda de qualquer cadeia.
- as variáveis da parte direita da forma sentencial colocam-se na pilha.
- a parte esquerda da forma sentencial, composta pelos símbolos terminais, é idêntica à entrada lida.
- começa-se colocando o símbolo inicial na pilha.
- para simular $A \rightarrow ax$:
    - coloca-se A no topo da pilha
    - introduz-se a como símbolo de entrada
    - a variável na pilha é removida e substituída por x

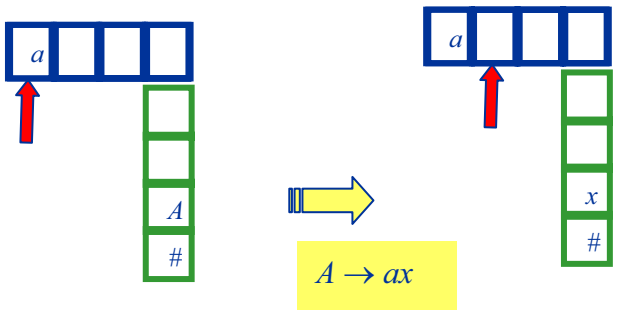


Figura 7.4.1 Simulação de uma produção no NPDA

Exemplo 7.4.1 (Taylor 550)

Seja a CGF em forma normal de Greibach

Produção	1.1-1.2	$S \rightarrow aA \mid aB$
	2.1.-2.2	$A \rightarrow aAB \mid aBB$
	3	$B \rightarrow b$

que gera a linguagem

$$L(G) = \{ a^n b^n : n \geq 1 \}$$

Vejamos a derivação pela extrema esquerda de *aaaabbbb* :

$S \Rightarrow aA$	(de 1.1)
$\Rightarrow aaAB$	(de 2.1)
$\Rightarrow aaaABB$	(de 2.1)
$\Rightarrow aaaaBBBB$	(de 2.2)
$\Rightarrow aaaabBBB$	(de 3.1)
$\Rightarrow aaaabbBB$	(de 3.1)
$\Rightarrow aaaabbbB$	(de 3.1)
$\Rightarrow aaaabbbb$	(de 3.1)

Procuremos agona um NPDA que modelize esta gramática. Vejamos se chegam três estados, o inicial q_0 , o de derivações, q_1 e o aceitador q_f.

O alfabeto da pilha é $\Gamma = \{\#, S, A, B\}$ e $\Sigma = \{a, b\}$.

1º Introduzir o símbolo inicial na pilha:

$$\delta(q_0, \lambda, \#) = \{ (q_1, S\#)\}$$

2º Simular

$$S \rightarrow aA \qquad \delta(q_1, a, S) = \{ (q_1, A)\}$$

3º Simular

$$S \rightarrow aB \qquad \delta(q_1, a, S) = \{ (q_1, B)\}$$

4º Simular

$$A \rightarrow aAB \qquad \delta(q_1, a, A) = \{ (q_1, AB)\}$$

5º Simular

$A \rightarrow aBB$ $\delta(q_1, a, A) = \{ (q_1, BB)\}$

6º Simular

$B \rightarrow b$ $\delta(q_1, b, B) = \{ (q_1, \lambda)\}$

7º Aceitar a cadeia quando aparece # no topo da pilha: muda para o estado aceitador

$\delta(q_1, \lambda, \#) = \{ (q_f, \#)\}$

Note-se o carácter não determinístico no autómato (várias escolhas de movidas possíveis para uma da configuração).

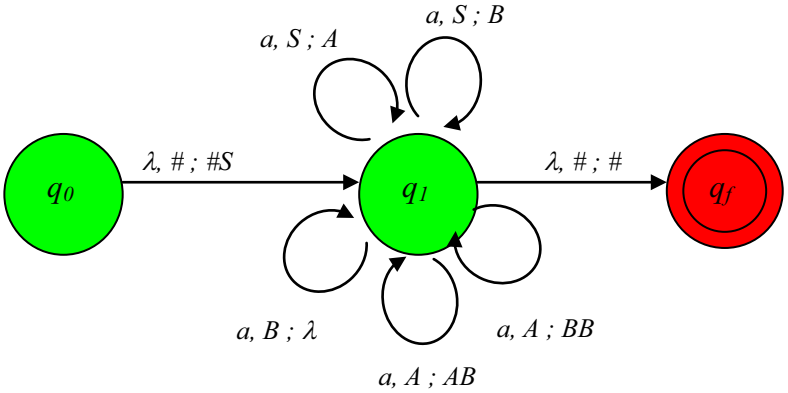
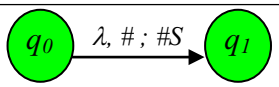
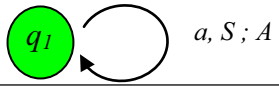
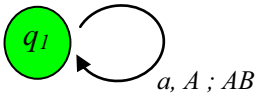
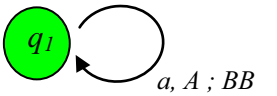
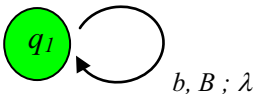
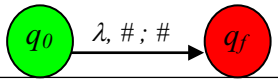


Figura 7.4.2 PDA do exemplo 7.4.1 (DEM).

Tabela 7.4.1 Resumo da simulação das derivações da gramática do Ex. 7.4.1

Produção	**Movida**	**Grafo**
Carregar S	$\delta(q_0, \lambda, \#) = \{ (q_1, S\#)\}$	
$S \rightarrow aA$	$\delta(q_1, a, S) = \{ (q_1, A)\}$	
$S \rightarrow aB$	$\delta(q_1, a, S) = \{ (q_1, B)\}$	
$A \rightarrow aAB$	$\delta(q_1, a, A) = \{ (q_1, AB)\}$	
$A \rightarrow aBB$	$\delta(q_1, a, A) = \{ (q_1, BB)\}$	
$B \rightarrow b$	$\delta(q_1, b, B) = \{ (q_1, \lambda)\}$	
aceitar a cadeia	$\delta(q_1, \lambda, \#) = \{ (q_f, \#)\}$	

Exemplo 7.4.2

Construir um NPDA que aceite a linguagem gerada pela gramática

$$S \rightarrow aSbb \mid a$$

Antes de mais, deve transformar-se a gramática na forma de Greibach, mudando as produções para (neste caso é simples)

$$S \rightarrow aSA \mid a$$
$$A \rightarrow bB,$$
$$B \rightarrow b.$$

O autómato correspondente terá 3 estados $\{q_0, q_1, q_2\}$ sendo $q_2=q_f$.

1º colocar o símbolo inicial S na pilha:

$$\delta(q_0, \lambda, \#) = \{ (q_1, S\#)\}.$$

2º simular a produção $S \rightarrow aSA$,

removendo S da pilha e colocando lá AS

lendo a à entrada

$$\delta(q_1, a, S) = \{ (q_1, SA), (q_1, \lambda)\}$$

simular a produção $S \rightarrow a$, de modo análogo, substituindo S por λ, e daí (q_1, λ).

3º simular a produção $A \rightarrow bB : \delta(q_1, b, A) = \{ (q_1, B)\}$.

4º simular a produção $B \rightarrow b : \delta(q_1, b, B) = \{ (q_1, \lambda)\}$.

5º completar a derivação quando aparece # no topo da pilha: $\delta(q_1, \lambda, \#) = \{ (q_f, \#)\}$.

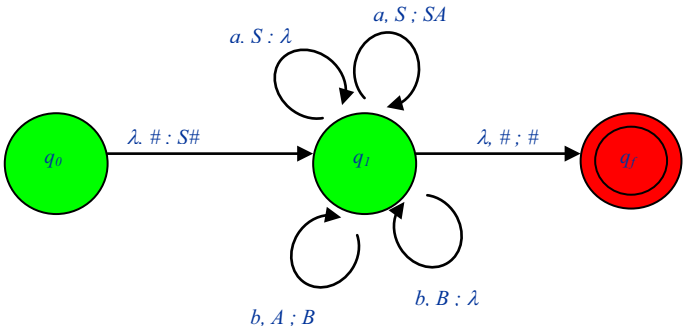


Figura 7.4.3. O NPDA do exemplo 7.4.1 (notação JFLAP).

Qual a linguagem do NPDA ? $L(G) = \{ a^n b^{2(n-1)} : n \geq 1 \}$.

Qual o número mínimo de estados de um NPDA, para uma linguagem definida por uma gramática de Greibach ?

- no exemplo obtiveram-se três ;
  - o estado q_0 pode ser eliminado, ficando o autómato não-determinístico.
  - λ passa a ser aceite

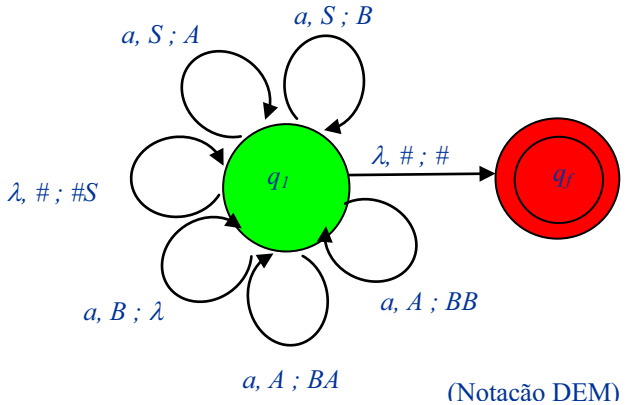


Figura 7.4.4. NPDA com dois estados equivalente ao da Fig. 7.4.2

Este resultado é geral: para qualquer CFL com λ existe um NPDA com dois estados, sem λ, com três estados.

Se quisermos evitar a introdução do λ, pode-se utilizar um símbolo especial na pilha, $\$$, para inicializar a gramática:

$$\delta(q_0, \lambda, \#) = \{ (q_0, S\$) \}$$

E agora começa-se com a gramática de Greibach.

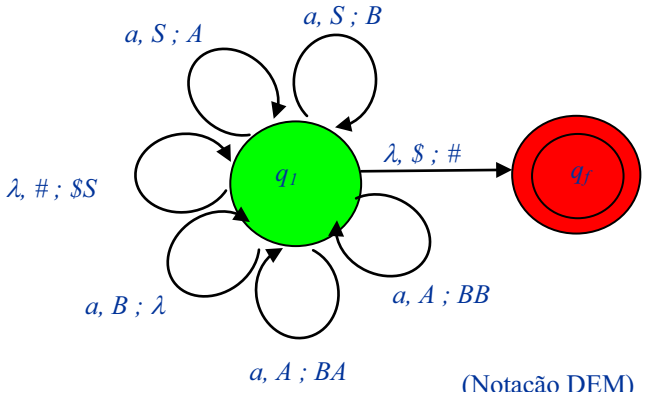


Figura 7.4.5. NPDA com dois estados para uma linguagem sem λ.

Finalmente concluímos que para qualquer CFL sem λ existe um NPDA com dois estados.

Teorema 7.4.1.

Para qualquer linguagem L livre de contexto, existe um NPDA M tal que $L = L(M)$

A demonstração é feita pela construção anterior:

- Se L é uma linguagem livre de contexto λ, então existe uma gramática livre de contexto na forma normal de Greibach para L.

-constrói-se um NPDA para simular as derivações pela extrema-esquerda dessa gramática.

Note-se que a exigência de que a gramática esteja na forma de Greibach é apenas para facilitar o desenvolvimento. O teorema aplica-se a toda a gramática livre de contexto.Ver uma demonstração mais detalhada em Linz, 185.

7.4.2. Gramáticas livres de contexto para NPDA's

O inverso do teorema anterior também é verdadeiro: dado um NPDA, existe para ele uma gramática livre de contexto.

Para o provar baste reverter o processo construtivo, de modo que a gramática simule as movidas do NPDA. Ver em Linz 189.

Teorema 7.4.2.

Se $L = L(M)$ para algum NPDA M, então L é uma linguagem livre de contexto.

7.5. Autómatos de pilha determinísticos e linguagens livres de contexto determinísticas

Se concebermos um autómato de pilha que não tenha escolhas possíveis em qualquer estado, ele será determinístico. Chama-se, em inglês, DPDA – *Deterministic Push Down Accepter*.

Definição 7.5.1. Autómato de pilha determinístico.

Um autómato de pilha $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ diz-se determinístico se for um autómato obedecendo a definição 7.3.1. e além disso se se submeter às restrições seguintes, para todo o $q \in Q, a \in \Sigma \cup \{\lambda\}$ a $b \in \Gamma$,

1. $\delta(q, a, b)$ contém no máximo um elemento (uma só movida possível):
   dado um símbolo de entrada e um símbolo no topo da pilha, só é possível uma movida, no máximo.

2. se $\delta(q, \lambda, b)$ não é vazia, então $\delta(q, c, b)$ deve ser vazio para todo o $c \in \Sigma$:
   quando é possível uma transição-λ para alguma configuração, não existe nenhuma alternativa que consuma caracteres de entrada para essa mesma configuração.

Definição 7.5.2. Linguagem livre de contexto determinística.

Uma linguagem L diz-se **livre de contexto determinística** se e só se existir um DPDA M tal que $L = L(M)$.

Exemplo 7.5.1

Consideremos a linguagem

$$L = \{ a_n b_n : n \geq 1 \}$$

Vejamos se ela é livre de contexto determinística.

O PDA $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{\#,1\}, \delta, q_0, \#, \{q_f\})$ com

$\delta(q_0, a, \#) = \{(q_1, 1\#)\}$ (inicia a contagem dos *a*'s)
$\delta(q_1, a, 1) = \{(q_1, 11)\}$, (conta os *a*'s)
$\delta(q_1, b, 1) = \{(q_2, \lambda)\}$, (detecta o primeiro *b*)
$\delta(q_2, b, 1) = \{(q_2, \lambda)\}$, (conta os *b*'s)
$\delta(q_2, \lambda, \#) = \{(q_f, \lambda)\}$ (aceita se nº *a*'s = nº *b*'s)

aceita aquela linguagem. Logo ela é determinística (e livre de contexto, como já se sabe).

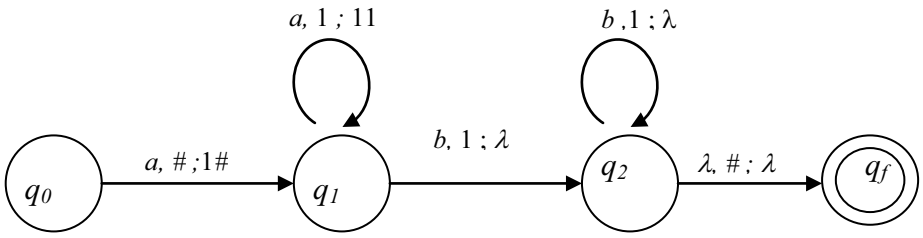


Figura 7.5.1 O PDA determinístico do Exemplo 7.5.1 (JFLAP)

Se quisermos aceitar a cadeia vazia, $L = \{ a^n b^n : n \geq 0 \}$, teremos a Fig. 7.5.2.

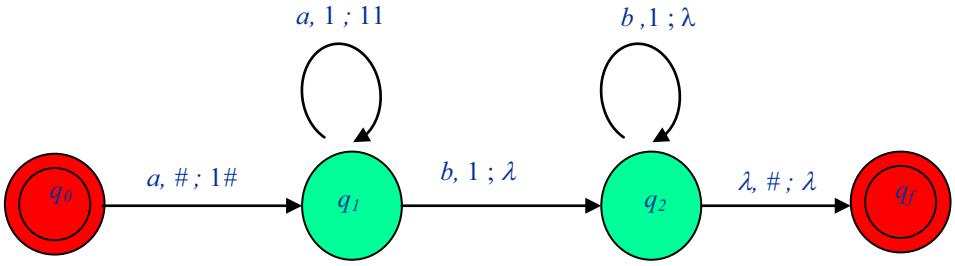


Figura 7.5.2. PDA que aceita a cadeia vazia.

ou ainda a Figura 7.5.3.

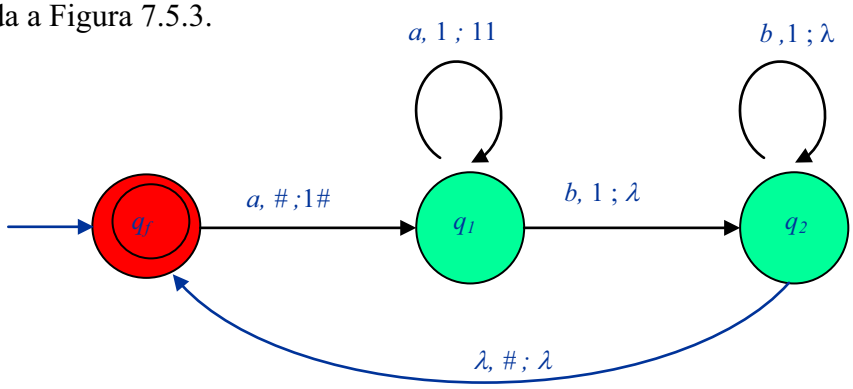


Figura 7.5.3. PDA equivalente ao da Figura anterior, com um só estado aceitador: chegando a q_2 e lendo # na pilha, passa ao estado aceitador.

Contrariamente ao que acontece nos autómatos finitos, nos PDA não há equivalência entre os determinísticos e os não determinísticos.

Diferenças entre os DFA e os DPDA

Também os DFA não têm escolhas possíveis. Há no entanto algumas diferenças entre estes e os DPDA:

- aqui admitem-se transições-λ sem quebra de determinismo. Como o topo da pilha também desempenha um papel importante na determinação da movida seguinte, o carácter no topo da pilha levanta o indeterminismo da entrada λ.

- a função δ não é aqui necessariamente total, podendo ser parcial, isto é, pode haver transições para o conjunto vazio, transições não definidas, e portanto configurações mortas.

- nos DPDA a única propriedade característica do determinismo é que em cada configuração só há no máximo uma movida possível.

A tabela 7.5.1 resume estas diferenças.

Tabela 7.5.1 Diferenças entre os DFA e os DPDA

DFA	DPDA
não admite transições - λ	admite transições -λ
δ é uma função total	δ não é necessariamente uma função total
há equivalência entre os determinísticos e os não determinísticos	**não há** equivalência entre os determinísticos e os não determinísticos

Semelhanças entre DFA e DPDA para linguagens regulares

As linguagens regulares são um subconjunto das linguagens livres de contexto.Toda a linguagem livre de contexto determinística tem um DPDA. As linguagens regulares são determinísticas, e portanto as linguagens regulares são linguagens livres de contexto determinísticas. Podemos assim concluir que para qualquer linguagem regular existe um DPDA que a aceita. Quais as diferenças e semelhanças entre o DFA e o DPDA de uma linguagem regular?

Vejamos um exemplo.

Exemplo 7.5.2 (Taylor, 556**)**

O DFA da Figura 7.5.4 aceita a linguagem definida pela expressão regular r = *a*b.*

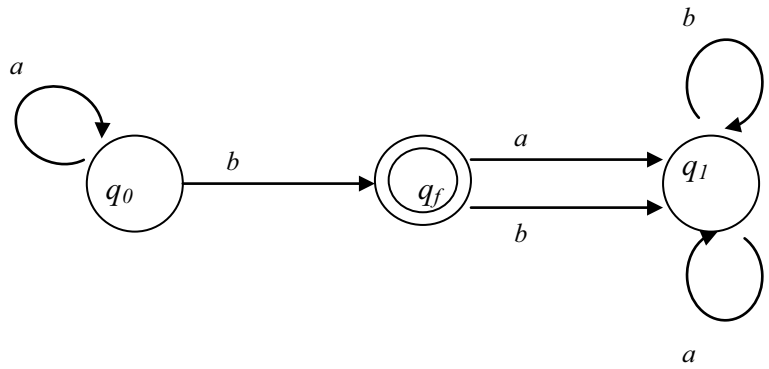


Figura 7.5.4. DFA do Exemplo 7.5.2.

Para desenhar um DPDA equivalente, é necessário alterar as etiquetas das transições a fim de introduzir as operações sobre a pilha. Como a pilha não é de facto necessária, em todas as transições não se faz nada na pilha, que se inicia com # e assim se mantém até ao fim.

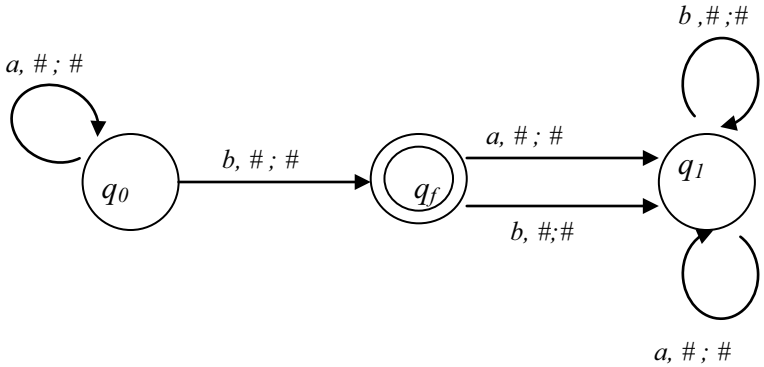


Figura 7.5.5. DPDA equivalente ao DFA da figura anterior. A pilha não é usada.

Podemos assim dizer que um DFA é um DPDA sem pilha, ou com a pilha desactivada.

7.6. Gramáticas para linguagens livres de contexto determinísticas

As linguagens livres de contexto determinísticas incluem todas as linguagens de programação. Elas têm características relevantes relacionadas com a construção de compiladores eficientes, pois pode-se fazer o seu *parsing* facilmente (gramáticas-*s,* mas não só). Têm importância particular as gramáticas *LL (Left scan, Leftmost derivations*) e as gramáticas *LR (Left scan, Rightmost derivations)*, que serão objecto de estudo na disciplina de Compiladores.

Exemplo 7.6.1.

Desenhar o grafo do NPDA da linguagem $L = \{a^n b^m : 0 \leq n \leq m\ ,\ m \geq 2\ \}$ (Taylor, 541). Este problema é um exemplo implementado no Deus Ex Máquina – ficheiro **Example 10.3.2** na directoria dos "*Pushdown Automata*".

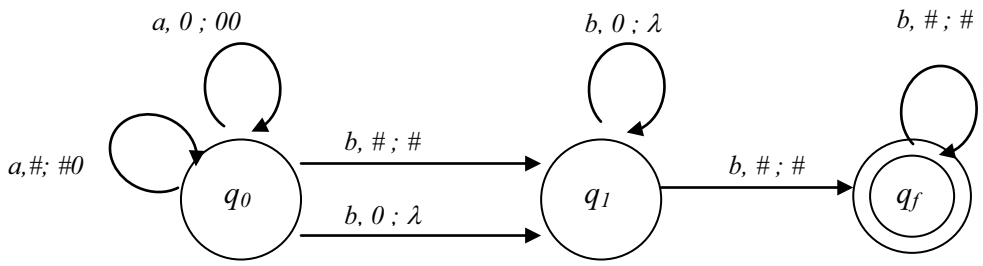


Figura 7.6.1. NPDA do Exemplo 7.6.1 (notação DEM).

Bibliografia.

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001

Models of Computation and Formal Languages, R. Gregory Taylor, Oxford University Press, 1998.

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

Introduction th the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.