E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Linguagens e Computação

CÓDIGO: 21078

DOCENTE: Jorge Morais

NOME: Hélio Emanuel Soares de Sousa

N.º DE ESTUDANTE: 2000027

CURSO: Engenharia Informática

DATA DE ENTREGA: 2020-12-22

Considere o alfabeto ∑ = {0,1}.

1. Escreva uma gramática independente de contexto que reconheça todas as sequências da forma $0^{2n}1^k0^{n-k}$, onde n e k são números inteiros positivos e k ≤ n

Pelo enunciado, sabemos que n>0, k>0 e (k ≤ n) ≥ 0, ou seja, os expoentes serão sempre >= 0. Em baixo, apresento exemplos da gramática até n=5:

	K=1	K=2	K=3	K=4	K=5
n=1	$0^2 1^1 0^0$ 001				
n=2	$0^4 1^1 0^1$ 00010	$0^4 1^2 0^0$ 00011			
n=3	$0^6 1^1 0^2$ 0000100	$0^6 1^2 0^1$ 0000010	$0^6 1^3 0^0$ 0000011		
n=4	$0^8 1^1 0^3$ 000001000	$0^8 1^2 0^2$ 000001100	$0^8 1^3 0^1$ 000001110	$0^8 1^4 0^0$ 000001111	
n=5	$0^{10} 1^1 0^4$ 0000000010000	$0^{10} 1^2 0^3$ 0000000011000	$0^{10} 1^3 0^2$ 0000000011100	$0^{10} 1^4 0^1$ 0000000011110	$0^{10} 1^5 0^0$ 0000000011111

A minha primeira abordagem foi:

S -> 00S | 00KN

K -> 1 | 1K

N -> ε |0 |0N

Explicação desta abordagem inicial: Criei 3 variáveis, S para o início, K para o meio e N para o final. No início ou temos sequências pares de zero (**S -> 00S**) ou apenas uma primeira sequência de 00 que depois pode ser seguida por apenas um 1 ou mais 1´s e ou um ou mais 0´s, essa produção é agregada em (**S -> 00KN)**.

A variável K pode ser 1 ou mais 1´s que pode ser derivada a partir da produção S que contém o K (**K -> 1 | 1K)**.

E no fim da palavra podemos ter o N, que quando n=k é vazio (**ε**) ou quando n≠k terá sempre um ou mais zeros. (**N -> ε |0 |0N).**

SIMPLIFICAÇÃO DO CFG:

As produções apresentadas são quase uma conversão direta da gramática dada no enunciado, mas pareceu que seria possível simplificar o número de produções e variáveis, sobretudo porque ter uma **ε-produção** é redundante num **CFG**.

No capítulo 7 de (Hopcroft, Motwani, & Ullmann, 2006) são dados conceitos que permitem realizar a simplificação de um **CFG** e em (NesoAcademy, 2020) uma explicação em vídeo demonstrando a aplicação dos mesmos. Com base nesse conhecimento, comecei a simplificação das produções iniciais.

Num CFG, um símbolo não terminal A é uma variável nula se existir uma produção A -> **ε** ou se existir uma derivação que começa em A e pode ser derivada até **ε**, isto é, a variável A é anulável se $A \overset{*}{\Rightarrow} \varepsilon$. É definido um procedimento para eliminar as **ε-produções** em 7.1.3 de (Hopcroft, Motwani, & Ullmann, 2006).

- **Procedimento para a remoção de ε**

Passo 1: Para remover A -> **ε**, procurar por todas as produções cujo lado direito contenham A.
Passo 2: Substituir todas as ocorrências de A em cada uma das produções com **ε**.
Passo 3: Adicionar as produções resultantes à gramática.

Passo 1: Procurar N -> ε
Neste caso, temos apenas uma produção que deriva **ε**: **N -> ε**
Constata-se que temos uma produção contendo N do lado direito: **S -> 00KN**
Nota: **S -> 00S | 00KN** apesar de ter N não deriva **ε** de acordo com o teorema 7.7 pois todas as variáveis da produção têm de ser anuláveis, e K é não anulável porque não tem a produção de **ε.**

Passo 2: Substituir N -> ε
Substituir N por **ε**
S -> 00KN -> 00Kε -> 00K
Obtemos assim novas produções para S, que se acrescentam às existentes:
S -> 00S | 00KN | 00K

Passo 3: Eliminar N -> ε
Neste passo apresento a simplificação obtida em que **ε** é eliminado do CFG.
S -> 00S | 00K |00KN
K -> 1 | 1K
N -> 0 |0N

Na realidade, ficamos com o mesmo número de produções e variáveis, mas a remoção do **ε** foi um passo essencial para a etapa seguinte.

- **Remoção de produções, variáveis e gramática**

Nesta fase, os métodos dados no capítulo 7 de (Hopcroft, Motwani, & Ullmann, 2006) e (ReductionCFGNesoAcademy, 2020) são bastante uteis para compreender o que pode ser feito, isto em conjunto com os conhecimentos obtidos até agora em todos os outros capítulos.
Analisando em detalhe as produções de S temos três produções contendo 00 como prefixo, o que é redundante, e também se verifica que para n≠k sempre que cada dois 0´s são gerados no início é gerado um 0 no fim, assim estas três produções são na essência uma só, isto é: **S -> 00S0.**
Colocar a variável S entre 00 e 0 permite chamadas recursivas á medida que n é incrementado, produz as sequências **000, 000000, 000000000, …, 00S0.**
Falta agora verificar a produção de **001´s**.
Pela análise da tabela, constata-se que existe a repetição de um padrão de 001´s sucessivos, isto é, por cada 1 gerado, são gerados dois 0´s imediatamente à sua esquerda, e no caso de ter sido já adicionado um 001 o seguinte é adicionado entre o 0 mais á direita e o 1 mais á esquerda, ou seja, adiciona no meio, assim sendo, simplifico/agrego a última produção em **S -> 00S1**.

A colocação do S entre o 00 e o 1 permite a chamada recursiva da variável **S**, produzindo as sequências de **000011, 000000111,…, 00S1.**
Falta-nos agora a produção mais importante deste CFG**,** isto é a sequência que está presente em todas as palavras da linguagem, e que será usada para fazer o fecho da palavra. Como facilmente se constata na tabela criada, corresponde à palavra inicial da linguagem, 001 simplesmente a adiciono como uma produção: **S -> 001**

Executei uma revisão massiva de palavras no software JFLAP 7.1 (Rodger H. S., 2020) e passou todas as strings até n=10, e tendo em conta que temos um padrão com uma regularidade bem definida de produção da linguagem posso afirmar que para n=∞ também irá produzir todas as palavras pertencentes à linguagem.
No entanto, apareceram palavras aceites tais como 000000101, 000000001001 , 000000001101 , 000000001011 que pela analise da gramática estava incorreto. Algo de errado estava nas produções, verifiquei de novo e detetei que a variável S estava a ser novamente chamada após a ocorrência de uma subsequência 001. Pelo que fui forçado a criar uma variável auxiliar para que ou inicia em 00S0 e depois passa para as sequencias 001 ou vai direto para as sequencias 001, assim garanto que o 00S0 apenas e só ocorre no início. Designei essa variável auxiliar por X. Passei todas as produções de 001 para a variável X e acrescentei uma produção para a variável S que produza X.
Assim o resultado final são as seguintes produções de **S** e **X**:
S -> 00S0 | X
X -> 001 | 00X1

Agora, a explicação do resultado final produção a produção:
001 – Caso especial, para n=k=1, é a primeira palavra da linguagem, todas as palavras aceites têm uma ou mais sequências de 001.
00S0 – Para n≠k as palavras aceites começarão sempre por um ou mais pares de 0´s e terminam sempre em um ou mais zeros, na proporção de 1:2 em relação aos pares. Podendo ter no meio sequências de 001, que a chamada recursiva da produção 00X1 permite que ocorra.
00X1 – Repetições de 001 para todos os valores em que k <= n e para todos os valores de n >= 2, inferido a partir da análise da tabela e da sua expetável evolução, ou seja, ficarão sempre no lado mais à direita da tabela.
Esta repetição ocorre no meio da palavra.
Derivando o X pela produção 00X1 garantimos que as sequencias 001 são colocadas na palavra na ordem correta.
X – Variável auxiliar necessária para que a produção 00S0 não volte a ser chamada de forma recursiva, e garantir que após ocorrer a primeira produção 001, teremos apenas e só sequências de produções 001.

Definição formal da linguagem obtida com esta gramática segundo (Hopcroft, Motwani, & Ullmann, 2006) será:

G = ({S,X}, {0,1}, {S -> X | S -> 00S0 | X -> 001 | X -> 00X1 }, S)

Um teste formal da linguagem será feito utilizando o conceito de derivação definido na seção 5.1 de (Hopcroft, Motwani, & Ullmann, 2006).

Palavra: 001
Derivação: S $\Longrightarrow$ X $\Longrightarrow$ 001

Palavra: 000010
Derivação: S $\Longrightarrow$ 00S0 $\Longrightarrow$ 00X0 $\Longrightarrow$ 000010

Palavra: 000000110
Derivação: S $\Longrightarrow$ 00S0 $\Longrightarrow$ 00X0 $\Longrightarrow$ 0000X10$\Longrightarrow$ 000000110

Palavra: 000000001000
Derivação: S $\Longrightarrow$ 00S0 $\Longrightarrow$ 0000S00 $\Longrightarrow$ 000000S000 $\Longrightarrow$ 000000X000 $\Longrightarrow$ 000000001000

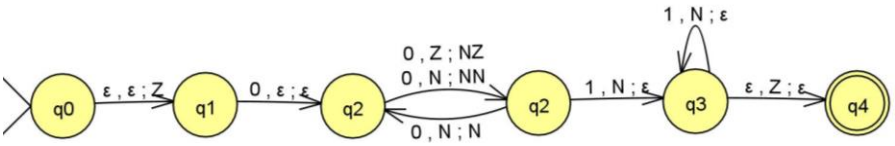
Palavra: 000000000011100
Derivação: S $\Longrightarrow$ 00S0 $\Longrightarrow$ 0000S00 $\Longrightarrow$ 0000X000 $\Longrightarrow$ 000000X100 $\Longrightarrow$ 00000000X1100 $\Longrightarrow$ 000000000011100

2. Construa um autómato de pilha que reconheça a mesma linguagem (sem recorrer à gramática da questão anterior).

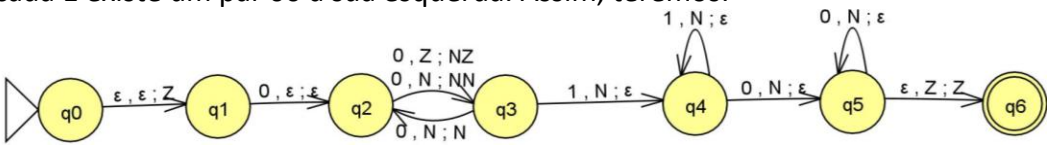
A vantagem de um **PDA** sobre um **ε-NFA** é a pilha e a sua memória ilimitada, com base nisso, e tendo por base os conceitos dados no capítulo 6 de (Hopcroft, Motwani, & Ullmann, 2006), as metodologias de (NesoAcademy, 2020) e o *software* disponibilizado por (Rodger H. S., 2020) e o livro de apoio do mesmo (Rodger & Finley, 2006) irei apresentar a minha sugestão de autómato de pilha para a linguagem solicitada.

Para esta gramática em particular, começarei por um autómato de pilha com aceitação por estado final, isto é, a palavra pertence à linguagem se todo o input for consumido e o PDA entrar em estado de aceitação.

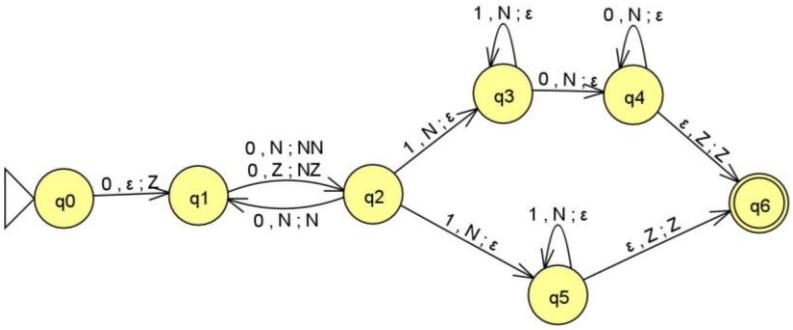
A gramática do enunciado tem duas situações distintas, n=k e n≠k.
Começo pelo caso de n = k. Utilizei um contador designado por N, sempre que o PDA vê 00 é adicionado um N ao topo da pilha. Quando vê um 1, faz transição para outro estado e remove um N da pilha, caso existam múltiplos N´s, ou seja, caso existam vários pares de 00´s no início no estado seguinte é feita a remoção de N´s sempre que um 1 é visto, e apenas quando chega ao Z é que pode ocorrer a transição. Esta ideia baseia-se na constatação que para n=k, existem sempre dois zeros imediatamente à sua esquerda e um 1 imediatamente à sua direita.



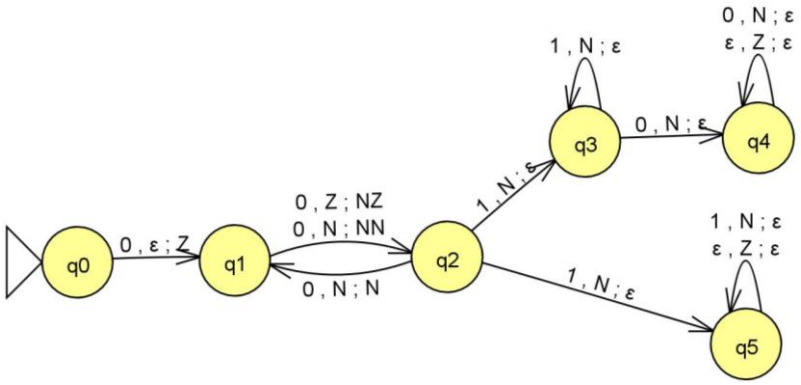
De seguida, para n≠k, por cada, um 0 à direita de 1 teremos um 00 à esquerda de 1. E por cada 1 existe um par 00 à sua esquerda. Assim, teremos:



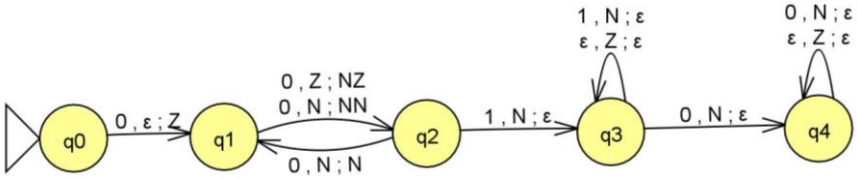
Em baixo, procedo à junção dos dois PDA´s, em que de q_0 até q_2, são iguais depois separam-se cada um dos ramos n=k ou n≠k n e no fim ambos vão ter ao mesmo estado final de aceitação. Aproveito e coloco em apenas uma transição a leitura de um terminal zero e a colocação do Z na pilha.



Após a junção, vou converter o PDA anterior para um autómato de pilha vazia eliminando assim 2 transições e um estado.



Analisando o PDA, ainda é possível fazer mais uma simplificação, a transição q2 -> q3 e q2 -> q5 são iguais, o arco que está em q3 e q5 é o mesmo. Pelo que apenas, diferencia o teste de pilha vazia.
Assim, coloquei o teste de pilha vazia em q3, e eliminei o estado q5.
Obtendo como resultado final o seguinte PDA simplificado de 5 estados e 10 transições.



Em baixo, apresento os testes realizados na aplicação JFLAP versão 7.1:

Input	
001	Accept
000010	Accept
000011	Accept
000000100	Accept
000000110	Accept
000000111	Accept
000000001000	Accept
000000001100	Accept
000000001110	Accept
000000001111	Accept
000000000010000	Accept
000000000011000	Accept
000000000011100	Accept
000000000011110	Accept
000000000011111	Accept
000000000000100000	Accept

Input	
000000000000100000	Accept
000000000000110000	Accept
000000000000111000	Accept
000000000000111100	Accept
000000000000111110	Accept
000000000000111111	Accept
00000000000000001000000	Accept
00000000000000001100000	Accept
00000000000000001110000	Accept
00000000000000001111000	Accept
00000000000000001111100	Accept
00000000000000001111110	Accept
00000000000000001111111	Accept
000	Reject
000000101	Reject
000000001001	Reject

Input	
00000000000000001111111	Accept
000	Reject
000000101	Reject
000000001001	Reject
000000001101	Reject
000000001011	Reject
001	Accept
1	Reject
00	Reject
01	Reject
10	Reject
11	Reject
000	Reject
010	Reject
011	Reject
100	Reject

O PDA de pilha vazia obtido pode ser dado por apenas 6 tuplos, em vez dos habituais 7, pois não temos conjunto de estados de aceitação, aliás nem os poderíamos ter porque em q3 ou q4 pode ocorrer a aceitação da palavra mas também pode não ocorrer, pois para as palavras rejeitadas nunca poderemos chegar ao fim da pilha e eliminar o Z e a palavra pode originar ID´s que fiquem em q3 e q4.

Definição formal de um dado PDA P:

$$P = (Q,\sum,\Gamma,\delta,q_0,Z_0,F)$$

$Q = \{q_0,q_1,q_2,q_3,q_4\}$
$\sum = \{0,1\}$
$\Gamma = \{Z,N\}$
δ = produções dadas pela notação gráfica apresentada do resultado final do PDA e em baixo de acordo com a seção 6.1.2 de (Hopcroft, Motwani, & Ullmann, 2006).
$\delta\ (q_0, 0, \varepsilon) = \{q_1, Z\}$
$\delta\ (q_1, 0, Z) = \{q_2, NZ\}$
$\delta\ (q_1, 0, N) = \{q_2, NN\}$
$\delta\ (q_2, 0, N) = \{q_1, N\}$
$\delta\ (q_2, 1, N) = \{q_3, \varepsilon\}$
$\delta\ (q_3, 1, N) = \{q_3, \varepsilon\}$
$\delta\ (q_3, \varepsilon, Z) = \{q_3, \varepsilon\}$
$\delta\ (q_3, 0, N) = \{q_4, \varepsilon\}$
$\delta\ (q_4, 0, N) = \{q_4, \varepsilon\}$
$\delta\ (q_4, \varepsilon, Z) = \{q_4, \varepsilon\}$

$q_0 = q_0$
$Z_0 = Z$
F – autómato de pilha vazia, não tem estados de aceitação, pelo que esta tupla pode ser removida do PDA P.
Definição formal do PDA P apresentado como possível solução para o enunciado:

$$P = (\{q_0,q_1,q_2,q_3,q_4\},\{0,1\},\{Z,N\},\delta,q_0,Z)$$

E a linguagem por aceitação de pilha vazia é dada por:

$$N(P) = \{w \mid (q_0,w,Z_0) \vdash^* (q,\varepsilon,\varepsilon)\}$$

$q_0 = q_0$
w – é a palavra a derivar para testar pertença à linguagem
$Z_0 = Z$
q – qualquer estado onde a palavra pode ser aceite por pilha vazia, neste caso: q_3 e q_4

$$N(P) = \{ w \mid (q_0,w,Z) \vdash^* ((q_3,\varepsilon,\varepsilon) \mid (q_4,\varepsilon,\varepsilon)) \ \}$$

Isto é, N(P) é o conjunto de entradas (terminais) w que P pode consumir na totalidade e ao mesmo tempo esvaziar a pilha.

3. Usando uma das respostas anteriores, mostre que 000000000011000 pertence à respetiva linguagem.

Informalmente, podemos verificar que a palavra pertence á gramática $\mathbf{0^{2n}1^k0^{n-k}}$ da seguinte forma: Temos 10 zeros, que é número par e é uma sequência de zeros, logo o valor de n tem de ser 5, temos uma sequencia seguida de 2 números 1´s, logo k terá de ser 2. Por fim, temos três 0´s, logo n-k tem de ser igual a 3, pelo que 5-2=3. Cumpre todas as regras da gramática.

Logo, obtemos $\mathbf{0^{2x5}1^20^{5-2} = 0^{10}1^20^3 = 000000000011000}$

Assim determinamos, informalmente, que esta palavra pertence à linguagem.

Com este ponto de partida, irei formalizar a verificação de pertença:

Dado que G é um CFG com a seguinte definição:

G = ({S,X}, {0,1}, {S -> X | S -> 00S0 | X -> 001 | X -> 00X1 }, S)

E, de acordo com (Hopcroft, Motwani, & Ullmann, 2006) , sabemos que se G é um CFG então L(G), ou CFL G, é o conjunto de palavras compostas por terminais que possuem derivações desde o símbolo de início, isto é:

$$L(G) = \{w \text{ está em } \{0,1\}^* \text{ tal que } S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

Aplicando então a metodologia da derivação pode-se provar que a palavra pertence à linguagem L(G):

Palavra: 000000000011000

Derivação:

Produção	Derivação	Palavra
		S
S -> 00S0	$\Longrightarrow$	00S0
S -> 00S0	$\Longrightarrow$	0000S00
S -> 00S0	$\Longrightarrow$	000000S000
S -> X	$\Longrightarrow$	000000X000
X -> 00X1	$\Longrightarrow$	00000000X1000
X -> 001	$\Longrightarrow$	000000000011000

De acordo com (Hopcroft, Motwani, & Ullmann, 2006) se $S \overset{*}{\underset{G}{\Rightarrow}} w$, então w pertence à linguagem, então dado que $S \overset{*}{\underset{G}{\Rightarrow}}$ 000000000011000 então a palavra pertence a CFL G.

BIBLIOGRAFIA

Hopcroft, J., Motwani, R., & Ullmann, J. (2006). *Introduction to Automata Theory, Languages and Computation* (3 ed.). Boston: Pearson Education, Inc.

NesoAcademy. (20 de 12 de 2020). *http://www.nesoacademy.org/*. Obtido de Theory of Computation: http://www.nesoacademy.org/computer-science/toc-and-automata-theory/theory-of-computation

Rodger, H. S. (19 de 12 de 2020). *JFLAP Version 7.1 RELEASED July 27, 2018*. Obtido de JFLAP: http://www.jflap.org/

Rodger, S. H., & Finley, T. W. (2006). *JFLAP: An Interactive Formal Languages and Automata Package.* Sudbury, California, Estados Unidos: Jones & Bartlett Publishers. Obtido em 19 de 12 de 2020, de http://www.jflap.org/