Teoria da Computação

Autómatos com pilha

Simão Melo de Sousa

Computer Science Department
University of Beira Interior, Portugal

Plano

Plano

Aviso Prévio

- **A redacção dos apontamentos da disciplina documento baseou-se fortemente na bibliografia indicada. Parece-nos então óbvio que a leitura e a aprendizagem directa pelas obras originais é recomendada, e mesmo essencial à compreensão profunda das noções aqui apresentadas;**
- **O português não é a língua materna do autor e o presente documento encontra-se em fase (constante) de elaboração/melhoramento pelo que se agradece e até se incentiva qualquer sugestão ou correcção;**

Referencias bibliográficas

- (Principal) C. H. Papadimitriou, H. R. Lewis. emphElements of the Theory of Computation por Prentice Hall, 1997. Tradução brasileira: *Elementos de Teoria da Computação*, 2a Edição. Bookman, Porto Alegre, 2000.
- (Introdutório, em francês - embora deva existir algo em inglês algures) P. Wolper. *Introduction à la calculabilité*, 3ª edição, Dunod, 2006.
- (introdutório e de leitura agradável) P. Linz. *An introduction to formal languages and automata*. Jones and Bartlett Publisher, 2006.
- (Uma obra de referência e muito completo... um "must") John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation* (3nd Edition). Addison Wesley, 2006 (existe em português do Brasil).
- (Completo e também um "must") M. Sipser. *Introducton to the Theory of Computation*. PWS Publishing, 2006.

Contexto

- Já sabemos que os autómatos finitos quer sejam deterministas ou não deterministas, não cobrem todas as linguagens.
- logo também não conseguem cobrir convenientemente a noção de algoritmo.
- Tentemos agora diagnosticar porquê e propor soluções.
- Ou seja, como extender os autómatos de forma a que sejam ultrapassadas as limitações diagnosticadas.
- vejamos um exemplo: porquê autómatos finitos não conseguem reconhecer a linguagem $\{w\tilde{w} \mid w \in \{a, b\}^*\}$?
- Claramente a gramática cujas produções são $\{ S \rightarrow \epsilon;\ S \rightarrow a\ S\ a;\ S \rightarrow b\ S\ b \}$ reconhece esta linguagem.

Contexto

- O processo de reconhecimento via um autómato vai explorar a palavra por reconhecer da esquerda para a direita. Por isso, para reconhecer uma palavra de tal linguagem é necessário ser capaz de memorizar a primeira metade da palavra para poder compara-la coma segunda metade.
- Os autómatos finitos não tem esta capacidade de memorização.

Contexto

- No entanto basta
  - dispor dum mecanismo de acumulação de caracteres que vai sendo alimentado com os caracteres lidos da entrada;
  - e de utilizar o não determinismo para adivinhar o caracter central: quando lemos um caracter, ou este faz parte da primeira metade, ou este é o primeiro caracter da segunda parte. No primeiro caso, o processo acumula o caracter lido e vai processar o seguinte, no segundo caso o caracter lido é comparado com o ultimo acumulado, em caso de igualdade o caracter acumulado é descartado e o processo continua. No caso de desigualdade, o processo falha.
- É este mecanismo de memória que faz falta aos autómatos finitos.

Contexto

- Outro exemplo: como reconhecer a linguagem gerada pela gramática cujas produções são: $\{ S \rightarrow aSb;\ S \rightarrow \epsilon\}$?
- É fácil ver que a linguagem gerada é $\{a^n b^n \mid n \geq 0\}$.
- O autómato reconhecedor tem de ser capaz, ao consumir um a, de memorizar que vai ter de reconhecer,mais tarde, um b.
- O reconhecimento com base numa máquina de estado pode ser feito de forma não determinística com a ajuda duma memória que vai acumulando os a lidos.
- O processo deve ler tantos b como os a que estão acumulados na memória. Caso contrário o reconhecimento falha.

Contexto

- É de realçar que se o numero de a de de b é limitado (por exemplo $\{a^n b^n \mid 0 \leq n \leq k\}$ para um determinado k) então deixamos de necessitar da tal memória adicional (um autómato finito, embora volumoso, consegue reconhecer a linguagem). O problema advém da necessidade de reconhecer $a^n c b^n$ **qualquer que seja o** n.
- Mais uma vez, este exemplo só exige que a memória seja algo de semelhante a uma **pilha** (*stack* ou *pushdown store* em inglês).
- É essa a ideia subjacente dos autómatos com pilha.

Plano

Descrição informal

- Informalmente um autómato com pilha é composto dos mesmos elementos constituintes dos autómatos finitos:
  - uma fita de dados de entrada,
  - um conjunto de estados (alguns deles iniciais e outros finais)
  - uma relação de transição.
- Em relação aos autómatos de estados finitos, acrescentamos uma pilha.
- A execução do autómato funciona nos seguintes moldes: Em cada passo de execução,
  - o autómato consulta a pilha, a letra por consumir e o estado em que se encontra e
  - avança para o estado seguinte consoante estes valores e a relação de transição. A transição poderá igualmente originar mudanças na pilha.

Descrição formal

Mais formalmente: a noção de autómato com pilha é formalizado por um 6-tuplo $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ onde

- Q é o conjunto finito dos estados do autómato
- Σ é o alfabeto de entrada
- Γ é o alfabeto da pilha (não é requerido que $\Gamma \cap \Sigma\neg = \emptyset$)
- $Z \in \Gamma$ é o símbolo inicial da pilha (único elemento da pilha no momento inicial – iremos ver que este elemento é facultativo)
- $s \in Q$ é o estado inicial do autómato
- $F \subseteq Q$ é o conjunto dos estados finais
- $\Delta \subseteq ((Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*))$ é a relação (finita) de transição.

(Quando se isenta a utilização do símbolo inicial de pilha a definição dum autómato restringe-se a um 6-tuplo – $Z = \epsilon$)

Transições

$$((p, u, \beta), (q, \gamma)) \in \Delta$$

Significa que o autómato

- pode passar do estado p para o estado q
- na condição que:
  - a entrada tenha por prefixo u
  - o conteúdo da pilha tenha por prefixo a palavra β
- neste caso a execução da transição leva a que o autómato:
  - consuma o prefixo u da entrada
  - consuma o prefixo β da pilha
  - produza γ no topo da pilha
  - passe do estado p para o estado q

Transições

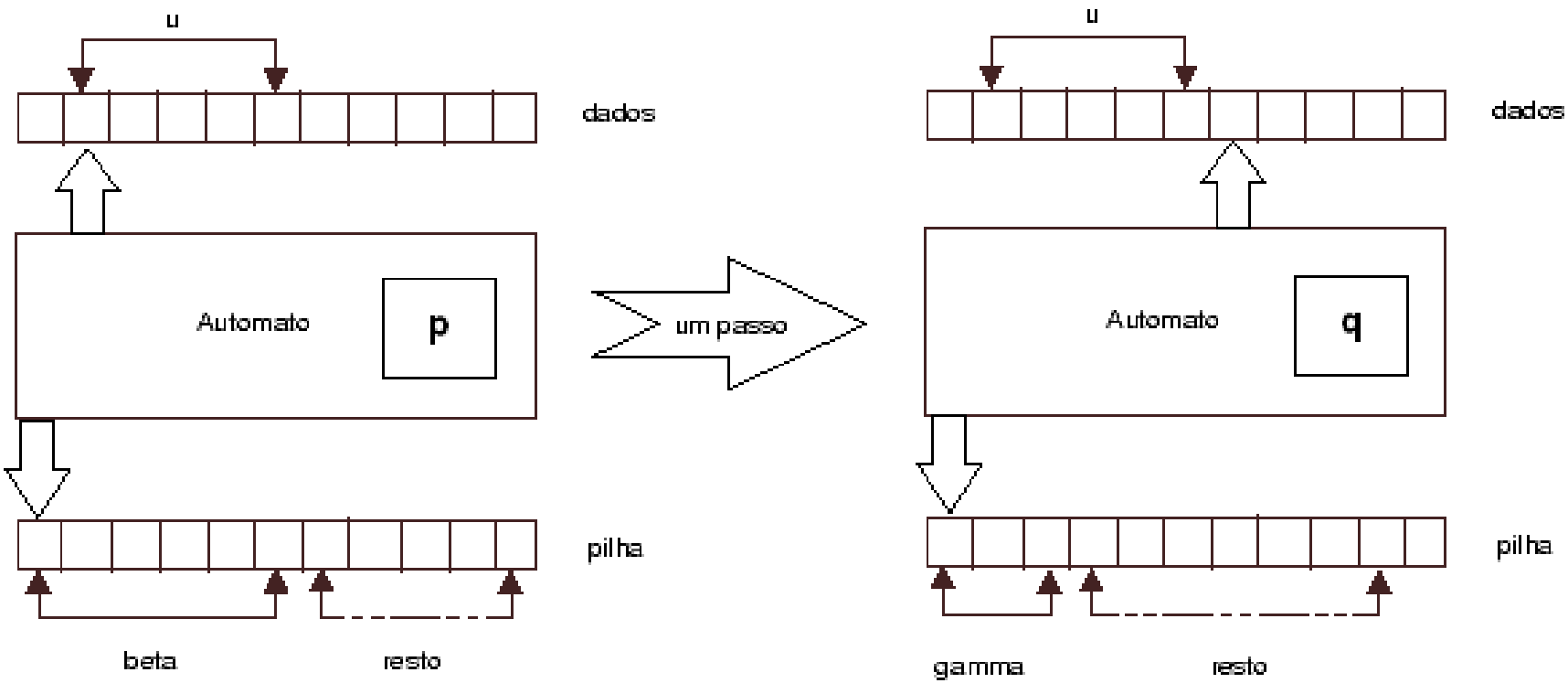


Figura: Transição $((p, u, \beta), (q, \gamma)) \in \Delta$ graficamente

Configuração

De forma semelhante ao casos dos autómatos finitos, para definir formalmente a noção de execução é necessário definir a noção de estado interno dum autómato com pilha num momento particular: uma configuração.

Definition (Configuração)

Uma configuração é um triplo $(q, u, \beta) \in Q \times \Sigma^* \times \Gamma^*$ em que

- q é o estado em que o autómato se encontra
- u é a entrada que resta por analisar (uma palavra)
- β é o conteúdo da pilha actualmente considerado. É também visto como uma palavra. O topo da pilha é o primeiro caracter da palavra.

Derivação

A derivação formaliza a noção de (passos de) execução.

Definition (Derivação num passo)

A configuração (q', w', α') **é derivável num passo** da configuração (q, w, α) pelo autómato $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ (notação: $(q, w, \alpha) \vdash_M (q', w', \alpha')$), se

- $w = uw'$ (a palavra da entrada w começa pelo prefixo $u \in \Sigma^*$)
- $\alpha = \beta\delta$ (antes da transição, $\beta \in \Gamma^*$ é um prefixo da pilha, ou seja os primeiros elementos da pilha formam β)
- $\alpha' = \gamma\delta$ (após a transição, β foi consumido - retirado da pilha - e juntamos γ à pilha. Assim o primeiro caracter de γ está agora no topo da pila)
- $((q, u, \beta), (q', \gamma)) \in \Delta$

Derivação

Uma derivação (em vários passos), denotada por $\vdash_M^*$ é o fecho reflexivo e transitivo de $\vdash_M$.

Definition (Derivação – em passos múltiplos)

Uma configuração C' é derivável (em 0 ou mais passos) da derivação C e pela máquina M se existe um $k \in \mathbb{N}, k \geq 0$ e configurações $C_0, C_1, \ldots, C_k$ tais que $C = C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_{k-1} \vdash_M C_k = C'$

Execução

Definition (Execução)

Uma **execução** dum autómato com pilha M sobre uma palavra w é uma sequência *máxima* de configurações da forma

$$C_{ini} \vdash_M (q_1, w_1, \alpha_1) \vdash_M (q_2, w_2, \alpha_2) \vdash_M \cdots \vdash (q_n, w_n, \alpha_n) \vdash_M \cdots$$

onde C_{ini} é a configuração (s, w, Z), designada de **configuração inicial** em que s é o estado inicial.
Este sequência é *máxima* no sentido que:

- ou termina numa configuração (p, ϵ, γ) com $\gamma \in F$, isto é, onde a entrada foi integralmente consumida e que o estado resultante é final.
- ou termina numa configuração (p, α, γ) a partir da qual não é possível derivar mais nenhuma outra configuração.
- ou é infinita (possível devido as transições ϵ)

Relembra-se que os autómatos aqui descritos são não-determinísticos, pelo que é possível existirem várias execuções distintas possíveis a partir da mesma palavra.

Palavra e Linguagem Aceite

Definition (Palavra Aceite)

Uma palavra w é aceite pelo autómato $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ se $(s, w, Z) \vdash_M^* (p, \epsilon, \epsilon)$ com $p \in F$.

Informalmente diz-se que o autómato aceita a palavra w **sobre estado final e pilha vazia**.

Definition (Linguagem Aceite)

A linguagem aceite por um autómato com pilha $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$, designada de $L(M)$, é o conjunto das palavras aceites pelo autómato.

Variantes

Existem definições alternativas as noções aqui apresentadas. Todas elas acabam por serem equivalentes em termos de cobertura expressiva (podem é ser mais convenientes ou simplificar).
Por exemplo:

- Em muitas situações é pratico poder marcar o fim da entrada ou marcar o estado inicial da pilha.
- Assim pode convencionar-se que todas as entradas acabam com um caracter especial (como o caracter \$ ou *EOF* que contemplaremos na disciplinas de compiladores).
- Ou considerar que a pilha na configuração inicial contém um símbolo, designado de **símbolo de pilha inicial**, habitualmente Z (Relembra-se que é o que foi convencionado nas definições até agora introduzidas). A configuração inicial por considerar é então (s, w, Z). Convenciona-se que este símbolo é introduzido uma única vez no processo de execução via a configuração inicial.
- Quando não se convenciona a existência inicial dum simbolo especial na pilha, basta considerar que $Z = \epsilon$ e todas as definições até agora introduzidas (configuração inical, etc..) se mantém assim inalteradas.

Variantes

- Mais:
  - Definir que uma palavra w é aceite se $(s, w, \epsilon) \vdash_M^* (p, \epsilon, \epsilon)$ qualquer que seja o estado p. Neste caso diz-se que o autómato aceita a palavra w **sobre pilha vazia**.
  - Definir que uma palavra w é aceite se $(s, w, \epsilon) \vdash_M^* (p, \epsilon, \gamma)$ com $p \in F$. Neste caso diz-se que o autómato aceita a palavra w **sobre estado final** (γ pode não ser ϵ).

Variantes

Exercício

- Mostrar (definir algoritmos) que as três definições de palavra aceite são equivalentes.
- Mostrar que se pode simular o símbolo inicial da pilha com um autómato onde este não é requerido.

Considerações

Considerações

Doravante, iremos considerar, excepto menção explícita,

- a configuração inicial (s, w, Z)
- autómatos que aceitam **sobre estado final e pilha vazia**

Exemplos

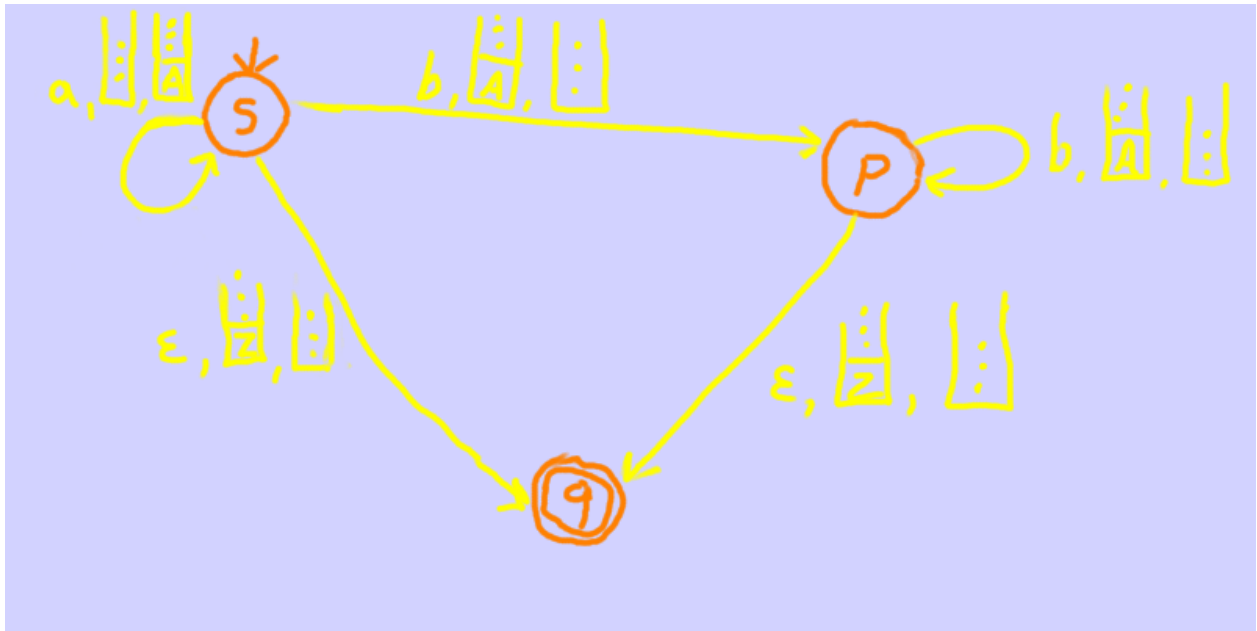
$L(M) = \{a^n b^n \mid n \geq 0\}$

O autómato $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ tal que

- Q ={s,p,q}
- $\Sigma = \{a, b\}$
- $\Gamma = \{A\}$
- $F = \{q\}$
- $\Delta =$

$$\begin{array}{lcl} (s, a, \epsilon) & \rightarrow & (s, A) \\ (s, \epsilon, Z) & \rightarrow & (q, \epsilon) \\ (s, b, A) & \rightarrow & (p, \epsilon) \\ (p, b, A) & \rightarrow & (p, \epsilon) \\ (p, \epsilon, Z) & \rightarrow & (q, \epsilon) \end{array}$$

aceita sobre estado final e pilha vazia a linguagem $\{a^n b^n \mid n \geq 0\}$.



Exemplos

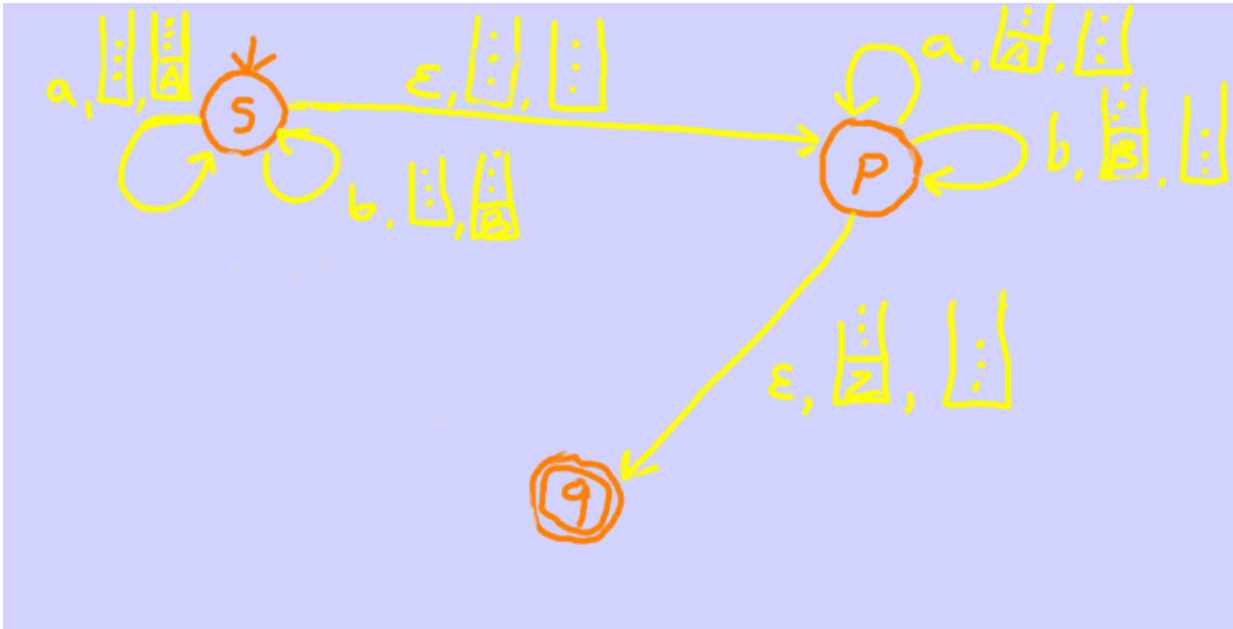
$L(M) = \{w\tilde{w} \mid w \in \Sigma^*\}$

O autómato
$M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ tal que

- Q ={s,p,q}
- $\Sigma = \{a, b\}$
- $\Gamma = \{A, B\}$
- $F = \{q\}$
- $\Delta =$

$$
\begin{array}{lcl}
(s, a, \epsilon) & \rightarrow & (s, A) \\
(s, b, \epsilon) & \rightarrow & (s, B) \\
(s, \epsilon, \epsilon) & \rightarrow & (p, \epsilon) \\
(p, a, A) & \rightarrow & (p, \epsilon) \\
(p, b, B) & \rightarrow & (p, \epsilon) \\
(p, \epsilon, Z) & \rightarrow & (q, \epsilon)
\end{array}
$$

aceita sobre estado final e pilha vazia a linguagem $\{w\tilde{w} \mid w \in \Sigma^*\}$.



Exemplos

$\{w\tilde{w} \mid w \in \Sigma^*\}$

O autómato $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ tal que

- $Q = \{s,p\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A, B\}$
- $F = Q$
- $\Delta = \begin{array}{lcl} (s, a, \epsilon) & \rightarrow & (s, A) \\ (s, b, \epsilon) & \rightarrow & (s, B) \\ (s, \epsilon, \epsilon) & \rightarrow & (p, \epsilon) \\ (p, a, A) & \rightarrow & (p, \epsilon) \\ (p, b, B) & \rightarrow & (p, \epsilon) \\ (p, \epsilon, Z) & \rightarrow & (p, \epsilon) \end{array}$

aceita sobre pilha vazia a linguagem $\{w\tilde{w} \mid w \in \Sigma^*\}$. Daí F ser pouco relevante (por questões de coerência fixou-se $F = Q$).

Exemplos

O autómato $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ tal que

- Q ={s,q,f}
- $\Sigma = \{a, b\}$
- $\Gamma = \{a, b, Z\}$
- $F = \{f\}$
- $\Delta =$

1	(s, ϵ, ϵ)	$\rightarrow$	(q, Z)
2	(q, a, Z)	$\rightarrow$	(q, aZ)
3	(q, a, a)	$\rightarrow$	(q, aa)
4	(q, a, b)	$\rightarrow$	(q, ϵ)
5	(q, b, Z)	$\rightarrow$	(q, bZ)
6	(q, b, b)	$\rightarrow$	(q, bb)
7	(q, b, a)	$\rightarrow$	(q, ϵ)
8	(q, ϵ, Z)	$\rightarrow$	(f, ϵ)

Não coloca elemento inicial na pilha e aceita sobre estado final e pilha vazia a linguagem
$\{w \mid w$ contém tantos $a's$ como $b's\}$.

Estado	*Entrada*	*Pilha*	*Trans.*	*Coment.*
s	*abbbabaa*	ϵ	–	*conf .inicial*
q	*abbbabaa*	Z	1	*Marca fin.*
q	*bbbabaa*	aZ	2	*push a's*
q	*bbabaa*	Z	7	*pop a*
q	*babaa*	bZ	5	*push b's*
q	*abaa*	bbZ	6	...
q	*baa*	bZ	4	
q	*aa*	bbZ	6	
q	*a*	bZ	4	
q	ϵ	c	4	
f	ϵ	ϵ	8	*aceite*

Exercício

Exercício: *NDFA* $\longrightarrow$ *pushdown*

Defina um algoritmo que transforme um autómato finito não determinista M num autómato com pilha M_p tal que $L(M) = L(M_p)$.

Plano

Resultado Fundamental

- Sabemos que uma linguagem algébrica é uma linguagem que pode ser gerada por gramáticas algébricas (de tipo 2).
- Vamos agora ver que os autómatos com pilha são mecanismos reconhecedores.

Theorem

A classe das linguagens aceites por autómatos com pilha é exactamente a classe das linguagens geradas por gramáticas algébricas: as linguagens algébricas. Formalmente:

$$\forall L \textit{ linguagem}, \quad \exists G \textbf{ gramática algébrica}, L = L(G)$$
$$\iff$$
$$\exists M \textbf{ autómato com pilha}, L = L(M)$$

Uma Demonstração Construtiva - Parte 1

Theorem

Existe para cada linguagem algébricas um autómato com pilha que a aceite. Formalmente,

$$\forall G \textbf{ gramática algébrica}, \exists M \textbf{ autómato com pilha}, L(G) = L(M)$$

Demonstração.

- Seja $G = (N, \Sigma, P, S)$ uma gramática livre de contexto. A ideia é construir um autómato com pilha M tal que $L(M) = L(G)$.
- Seja $M = (\{p, q\}, \Sigma, \Sigma \cup N, \Delta, p, \{q\})$ onde Δ é:

1	(p, ϵ, ϵ)	$\rightarrow$	(q, S)	
2	(q, ϵ, A)	$\rightarrow$	(q, α)	$\forall$*regra* $(A \rightarrow \alpha)$ *de* P
3	(q, a, a)	$\rightarrow$	(q, ϵ)	$\forall a \in \Sigma$

Uma Demonstração Construtiva - Parte 1

Resta-nos demonstrar que o autómato M assim construído gera exactamente o que pretendemos. Ou seja que o algoritmo proposto é correcto. Para isso basta demonstrar que

Theorem

Seja $w \in \Sigma^a st$ *e* $\alpha \in (N)(N \cup \Sigma)^* \cup \{\epsilon\}$ *então*

$$S \stackrel{*}{\Longrightarrow} w\alpha \iff (q, w, S) \vdash_M^* (q, \epsilon, \alpha)$$

Para o nosso propósito interessa-nos o caso $\alpha = \epsilon$.

Demonstração.

$\Longrightarrow$: Admitimos $S \stackrel{*}{\Longrightarrow} w\alpha$. Demonstra-se $(q, w, S) \vdash_M^* (q, \epsilon, \alpha)$ por indução sobre o comprimento da derivação esquerda de w a partir de S.

$\Longleftarrow$: Admitimos $(q, w, S) \vdash_M^* (q, \epsilon, \alpha)$. Demonstra-se $S \stackrel{*}{\Longrightarrow} w\alpha$ por indução sobre o número de aplicação das transições de tipo 2 na execução.

(detalhes, ver livro de Papadimitriou p. 138-139).

Exemplo

Considere a gramática cujas produções são:

$$\begin{array}{lcl} S & \rightarrow & aSa \\ S & \rightarrow & bSb \\ S & \rightarrow & c \end{array}$$

O algoritmo descrito devolve o seguinte autómato $M = (Q, \Sigma, \Gamma, \Delta, p, F)$ tal que

- Q ={p,q}
- $\Sigma = \{a, b, c\}$
- $\Gamma = \{S, a, b, c\}$
- $F = \{q\}$
- $\Delta =$

$$\begin{array}{llcl} 1 & (p, \epsilon, \epsilon) & \rightarrow & (q, S) \\ 2 & (q, \epsilon, S) & \rightarrow & (q, aSa) \\ 3 & (q, \epsilon, S) & \rightarrow & (q, bSb) \\ 4 & (q, \epsilon, S) & \rightarrow & (q, c) \\ 5 & (q, a, a) & \rightarrow & (q, \epsilon) \\ 6 & (q, b, b) & \rightarrow & (q, \epsilon) \\ 7 & (q, c, c) & \rightarrow & (q, \epsilon) \end{array}$$

Uma Demonstração Construtiva - Parte 2

Theorem

Se uma linguagem é aceite por um autómato com pilha M então existe uma gramática livre de contexto G que a gere. Formalmente,

$$\forall M \textbf{ autómato com pilha}, \exists G \textbf{ gramática algébrica}, L(G) = L(M)$$

Demonstração.
Complexa.... Passa por definir (e demonstrar correcto) um algoritmo que constrói uma gramática livre de contexto a partir do autómato. Detalhes, ver livro de Papadimitriou p. 139-142.

Plano

Contexto

- Existe mais linguagens do que as linguagens algébricas.
- Como existe linguagens que os autómatos com pilha não conseguem reconhecer.
- Estes dois resultados equivalentes vão ser demonstrados por um **teorema do bombeamento** adaptado ao caso dos autómatos com pilha.

Definições Preliminares

- O **resultado** duma árvore de derivação a (notação $\rho(a)$) é a palavra gerada pela árvore (concatenação das folhas - terminais, da esquerda para a direita).
- O **leque** duma gramática G (notação $\phi(G)$) é o número de símbolos do maior *rhs* presente nas regras da gramática G
- Um **caminho** numa árvore de derivação é a sequência de nodos distintos em conexão directa, sendo que esta sequência começa a partir da raíz e acaba numa folha.
- O **comprimento** dum caminho é o número de nodos que o constituí.
- A **altura** duma árvore de derivação a (notação $\varphi(a)$) é o comprimento do maior caminho desta árvore.

Resultado Preliminar

Lemma

Seja G uma gramática livre de contexto. Qualquer que seja o resultado w duma arvore de derivação a de G de altura n, $|w| \leq \phi(G)^n$, onde $w = \rho(a)$ e $n = \varphi(a)$.

Demonstração:
Por indução sobre n.

- Será a propriedade válida para $n = 1$?

  Caso trivial. Neste caso a árvore representa a aplicação duma única regra em que S é o *lhs*. De forma óbvia, o resultado tem no máximo $\phi(G)$ símbolos

- Admitimos que a propriedade é válida para n. Será ela válida para $n+1$?.

  Uma arvore de derivação de altura $n+1$ é constituída por uma raíz conectada a, no máximo, $\phi(G)$ árvores de altura máxima n. Por hipótese de indução todas estas árvores tem um resultado de comprimento máximo $\phi(G)^n$. Logo o comprimento total é no máximo $\phi(G) \times \phi(G)^n = \phi(G)^{n+1}$.

Lemma do Bombeamento para Autómatos com Pilha

Theorem

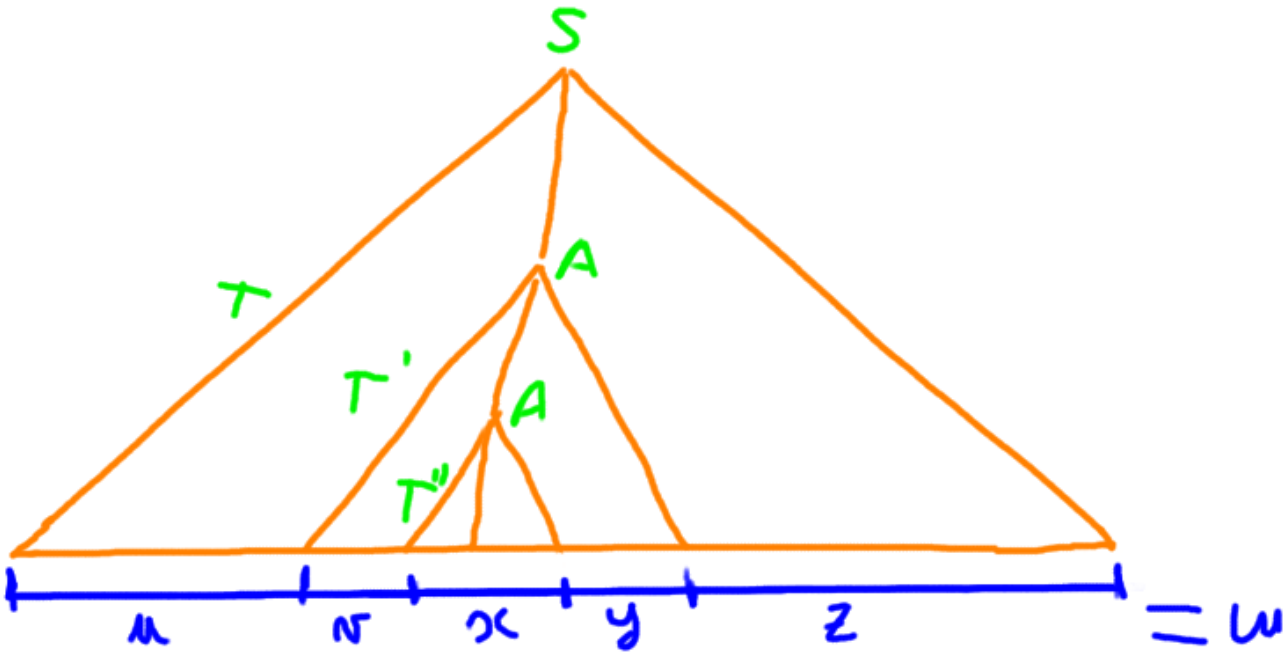
Seja $G = (N, \Sigma, P, S)$ *uma gramática livre de contexto. Qualquer que seja a palavra* $w \in L(G)$ *de comprimento máximo do que* $\phi(G)^N$ *pode ser reescrita em* $w = uvxyz$ *de tal forma que* $(v \neq \epsilon \vee y \neq \epsilon)$ *e* $\forall n \in \mathbb{N}, uv^n xy^n z \in L(G)$.

Demonstração

- Seja w uma tal palavra. e T a árvore de derivação que produz w com o menor número de folhas possível.
- Visto que $|w| \leq \phi(G)^{|N|}$, então T tem um caminho de comprimento de pelo menos $|N| + 1$ com pelo menos $|N| + 2$ nodos. Só um desses, a folha, contém um terminal.
- Logo há mais nodos com não-terminais do que não-terminais em N. Logo há pelo menos uma repetição neste caminho. Seja A um não-terminal repetido no caminho considerado.

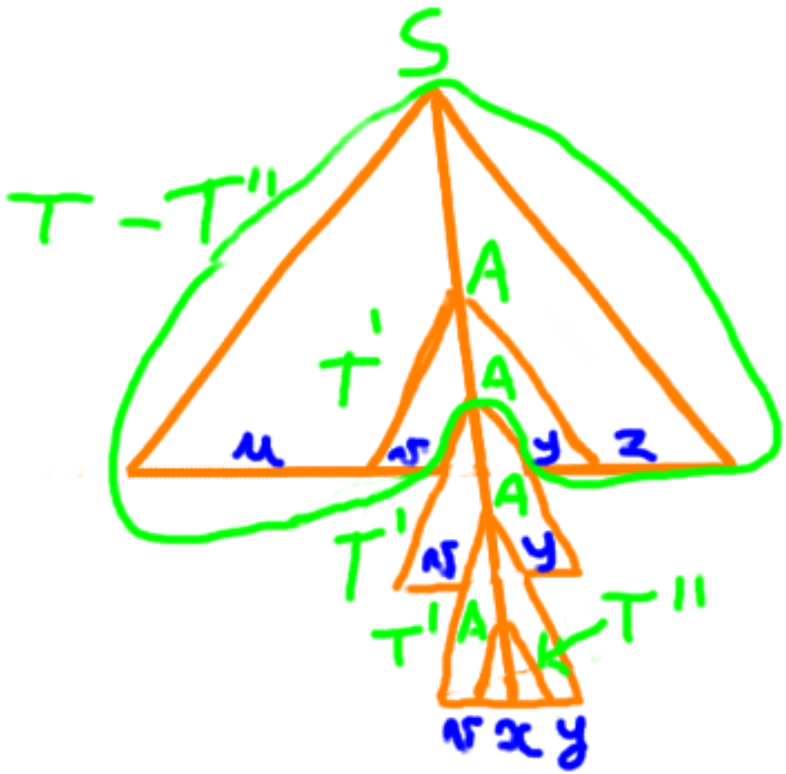
Demonstração

- A situação pode ser graficamente representada pela figura seguinte:



Demonstração

- Destaca-se a subárvore T' da qual se extraí T''. Esta subárvore pode ser repetida em número arbitrário (incluindo 0) no lugar de T''. como o mostra a figura



Demonstração

- Neste caso temos de facto como resultado da árvore palavras da forma uv^nxy^nz com $n \geq 0$.
- a condição $vy \neq \epsilon$ é garantia pelo facto de T ser a minimal (ver papadimitriou p.146)
- Q.E.D.

O princípio

- O Lemma de bombeamento garante que qualquer que seja a gramática livre de contexto, existem palavras geradas suficientemente grandes para necessitar que seja utilizada um não-terminal mais do que uma vez.
- Essa repetição gera padrões particulares.
- Demonstrar que uma gramática não é algébrica pode ser feito com base na ausência desses padrões.
- Basicamente: apresentar uma palavra gerada $w = uvxyz$ suficientemente grande para necessitar a utilização repetida de um mesmo não terminal e demonstrar que uv^nxy^nz não pode pertencer a linguagem gerada.
- De forma prática, procede-se a esta demonstração por contradição: admite-se que a gramática é algébrica e, logo, que uv^nxy^nz pode ser gerado.

Exemplos

$$L = \{a^n b^n c^n |\ n \in \mathbb{N}\}$$

- Demonstração por contradição de que L não é livre de contexto.
- Supomos que L é algébrica. Seja $G = (N, \Sigma, P, S)$ a gramática que gere L.
- Seja $n > \frac{\phi(G)^N}{3}$. Então $w = a^n b^n c^n \in L$ e existem u, v, x, y, z tal que $w = uvxyz$ (com $vy \neq \epsilon$).
- Neste caso $\forall m \in \mathbb{N},\ uv^m xy^m z \in L$. Olhemos para v e y. Dois casos se apresentam.
  - vy contém ocorrências de a de b e de c. Neste caso, pelo menos dois deles ocorrem ou em v ou em y. Então uv^2xy^2z não contém as ocorrências de a de b e de c na ordem certa.
  - vy não contém ocorrências dos três símbolos juntos. Então uv^2xy^2z não contém o mesmo números de ocorrências de a, b e de c.
  - Contradição. QED.

Exemplos

$$L = \{a^n |\ n \in \mathbb{N} \wedge n\ primo\}$$

- Demonstração, muito identica, por contradição de que L não é livre de contexto.
- Considera-se um primo p maior do que $\phi^{|N|}$
- $w = a^p$ e pode se descompor em $w = uvxyz$
- Supomos que $vy = a^q$ e $uxz = a^r$ para $q, r \in \mathbb{N}$
- Assim, dizer $\forall n \in \mathbb{N}, uv^n xy^n z \in L$ equivale a dizer que $\forall n \in \mathbb{N},\ r + nq$ é primo. O que é falso. QED.

Exemplos

Theorem

As linguagens livres de contexto não são fechadas por intersecção e complementação. Ou seja:
Sejam L_1 e L_2 duas linguagens.

- L_1 e L_2 **algébricas** $\not\Longrightarrow$ $L_1 \cap L_2$ **algébrica**
- L_1 **algébrica** $\not\Longrightarrow$ $\bar{L_1}$ **algébrica**

Plano

PDA vs FSM

- Em oposição aos autómatos finitos, não existe correspondência entre autómatos com pilha não determinista e autómato com pilha deterministas. Assim, os autómatos deterministas são estritamente menos expressivos do que os autómatos não deterministas.
- Os autómatos com pilha determinista, as linguagens que eles geram e a algoritmia associada são fundamentais aos processos de analise sintáctica que iremos abordar na disciplina de LFC.
- As limitações dos autómatos com pilha têm origem nas propriedades das pilhas. A consulta do topo da pilha faz-se pela sua remoção. As pilhas só permitam uma utilização única do seu conteúdo.
- Por exemplo, para reconhecer $\{a^n b^n c^n \mid n \geq 0\}$ é preciso utilizar n duas vezes logo é preciso memorisar n. O que não o occore com a linguagem $\{a^n b^n \mid n \geq 0\}$
- Modelos computacionais mais expressivos removem este inconveniente (Máquinas de Turing, λ-calculo, autómatos com 2 pilhas).