Capítulo 5

Gramáticas de Atributos

Este capítulo é uma primeira introdução ao estudo da semântica das linguagens formais.

5.1 O método dos atributos

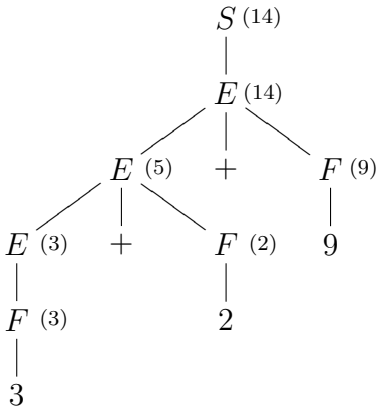
Vimos anteriormente, para gramáticas de expressões aritméticas, como se podiam usar as árvores de derivação para calcular o valor das expressões. O ponto de vista que vamos aqui adoptar é que o valor de uma expressão é apenas um dos seus possíveis *atributos*. Vamos ver como se podem associar atributos aos símbolos não terminais de uma gramática, e como se pode especificar o cálculo dos valores desses atributos.

Consideremos a seguinte gramática muito simplificada de expressões aritméticas, que contempla apenas adições de algarismos decimais:

$$\begin{aligned} S &\longrightarrow E \\ E &\longrightarrow E + F \mid F \\ F &\longrightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9 \end{aligned}$$

Estamos interessados em especificar o cálculo do valor de uma destas expressões, usando árvores de derivação como no capítulo anterior. Por exemplo, a palavra $3 + 2 + 9$ tem a seguinte árvore de derivação, com o valor de cada nó interior (etiquetado por um símbolo não terminal) indicado entre

parênteses:



O valor de cada nó interior é calculado em função do valor dos filhos. Como um nó e os seus filhos correspondem à cabeça e ao corpo de uma produção, respectivamente, também se pode dizer que o valor da cabeça é calculado em função do valor dos símbolos do corpo.

Consideremos primeiro a produção $S \longrightarrow E$. Representando por $v(S)$ o valor de S e por $v(E)$ o valor de E, o cálculo de $v(S)$ em função de $v(E)$ pode exprimir-se pela igualdade $v(S) = v(E)$, que afirma que o valor de S é igual ao valor de E (conhecido este, determina-se aquele).

Saltemos de momento a produção $E \longrightarrow E + F$. A produção $E \longrightarrow F$ trata-se de uma forma análoga à anterior pondo $v(E) = v(F)$, onde $v(F)$ representa o valor de F.

As produções $F \longrightarrow 0, \ldots, F \longrightarrow 9$ não trazem surpresas, vindo $v(F) = 0, \ldots, v(F) = 9$.

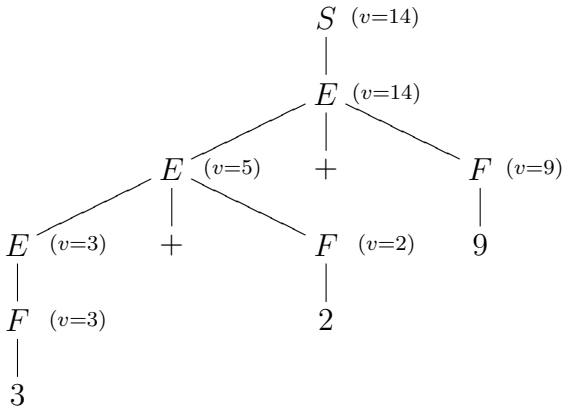
Regressemos à produção $E \longrightarrow E + F$. Para indicar que o valor da cabeça é a soma dos valores dos símbolos não terminais do corpo podíamos escrever $v(E) = v(E) + v(F)$, mas esta equação deixa $v(E)$ indeterminado e $v(F) = 0$. Há pois necessidade de distinguir as duas ocorrências de E na produção. Ela passa a ser escrita (exclusivamente para fins de definição dos atributos) na forma $E_1 \longrightarrow E_2 + F$, onde se acrescentaram índices diferentes às duas ocorrências de E. É importante notar que E_1 e E_2 não são dois "novos" símbolos não terminais, mas sim duas ocorrências distintas do mesmo símbolo. Pode então escrever-se $v(E_1) = v(E_2) + v(F)$.

A gramática, juntamente com as equações que definem os atributos,

mostra-se a seguir:

$$\begin{array}{lll} S & \longrightarrow E & v(S) = v(E) \\ E_1 & \longrightarrow E_2 + F & v(E_1) = v(E_2) + v(F) \\ E & \longrightarrow F & v(E) = v(F) \\ F & \longrightarrow 0 & v(F) = 0 \\ \vdots & & \vdots \\ F & \longrightarrow 9 & v(F) = 9 \end{array}$$

A árvore de derivação anterior pode agora redesenhar-se indicando explicitamente o atributo a que os valores se referem (isto é particularmente necessário quando um símbolo tiver mais de um atributo).



Os atributos são usados para atribuir significado às palavras de uma linguagem. Diz-se então que se está a definir a *semântica* da linguagem. Mas o significado atribuído depende do objectivo que se tem em mente. Por exemplo, para a gramática anterior definiu-se o valor de cada expressão, mas poderiam ter-se definido muitos outros atributos, como o código numa linguagem máquina que avalia a expressão, ou a expressão escrita na forma pós-fixa. Vejamos em pormenor este último caso.

A *forma pós-fixa* de uma expressão consiste em escrever sistematicamente os argumentos em primeiro lugar e só depois o símbolo da operação a efectuar. Assim, a forma pós-fixa de 3+2 é 3 2+, mas já a de 3+2+9 depende da ordem de execução das operações. Se a ordem requerida for $(3 + 2) + 9$, a forma pós-fixa é $3\,2 + 9+$, se se pretender $3 + (2 + 9)$, deve ter-se $3\,2\,9 + +$ (para ver que assim é, percorra-se a forma da esquerda para a direita e, sempre que se encontrar o '+', somem-se os dois últimos números encontrados ou resultantes de avaliações anteriores). Na gramática anterior, é dada precedência aos operadores mais à esquerda (observe-se que as árvores de derivação tendem a crescer para a esquerda), logo é a primeira das formas anteriores que se

pretende. A forma pós-fixa de uma expressão reduzida a um algarismo é o próprio algarismo.

Seja p o atributo de S, E e F cujo valor é a forma pós-fixa da expressão representada pelo símbolo em questão. Para qualquer atributo que se defina, seja ele a, é preciso definir o conjunto $\mathcal{V}_a$ dos valores que ele pode tomar. No exemplo anterior tinha-se

$$\mathcal{V}_v = \text{Nat},$$

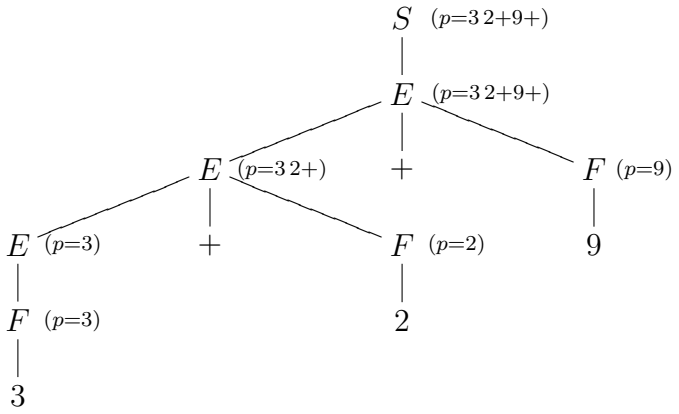
em que $\text{Nat} = \{0, 1, 2, \ldots\}$ é o conjunto dos números naturais. Para p tem-se

$$\mathcal{V}_p = \{+, 0, 1, \ldots, 9\}^*.$$

Não é difícil de ver que p se pode definir como segue:

$$\begin{array}{lll}
S & \longrightarrow E & p(S) = v(E) \\
E_1 & \longrightarrow E_2 + F & p(E_1) = p(E_2)p(F)+ \\
E & \longrightarrow F & p(E) = p(F) \\
F & \longrightarrow 0 & p(F) = 0 \\
\vdots & & \vdots \\
F & \longrightarrow 9 & p(F) = 9
\end{array}$$

A árvore de derivação para $3 + 2 + 9$ fica:



A título de exemplo considere-se a primeira ocorrência de E, mesmo abaixo de S. A equação apropriada é $p(E_1) = p(E_2)p(F)+$, onde E_1 representa a ocorrência em questão e E_2 o seu filho esquerdo. Tem-se então $p(E_2) = 3\,2+$ e $p(F) = 9$, donde $p(E_1) = 3\,2 + 9+$.

5.2 Atributos sintetizados e atributos herdados

Um atributo diz-se *sintetizado* se o seu valor num nó de uma árvore de derivação depender dos valores dos atributos dos filhos e, possivelmente, de outros atributos do próprio nó. Assim, para uma produção define-se um atributo sintetizado da sua cabeça em função de atributos do corpo e de outros atributos da cabeça. Nos exemplos anteriores, ambos os atributos eram sintetizados.

Mas existem atributos cujo valor num nó depende dos valores de atributos dos irmãos e do pai. Chamam-se *herdados* a esses atributos. Numa produção, definem-se atributos herdados de símbolos não terminais do corpo, em função de outros atributos da produção.

Em resumo, numa produção podem definir-se:

- atributos sintetizados da cabeça;
- atributos herdados de símbolos não terminais do corpo.

Qualquer destes atributos pode depender, em princípio, de qualquer outro atributo da produção. Veremos adiante uma restrição que se pode impor a estas dependências que permite o cálculo eficiente do valor dos atributos.

Para ilustrar o uso de atributos sintetizados e herdados, consideremos a seguinte gramática:

$$S \longrightarrow E$$

$$E \longrightarrow FX$$

$$X \longrightarrow +FX \mid \lambda$$

$$F \longrightarrow 0 \mid 1 \mid \cdots \mid 9$$

Trata-se de uma gramática de expressões aritméticas que define a mesma linguagem da gramática apresentada na secção anterior. Mas há uma diferença importante, do ponto de vista do cálculo de atributos: as palavras geradas pelo símbolo X não são expressões aritméticas, por isso não se pode, por exemplo, atribuir-lhes um valor numérico.

Suponhamos que pretendemos definir atributos que permitam determinar o valor numérico das expressões geradas pela gramática. Por analogia com o exemplo anterior, é de esperar que se defina para S, E e F um atributo v de valores em $\mathcal{V}_v = \text{Nat}$. É mesmo fácil ver como se calculam $v(S)$ e $v(F)$ com as produções $S \longrightarrow E$ e $F \longrightarrow 0, \ldots, F \longrightarrow 9$: as equações são como no exemplo referido, e v é um atributo sintetizado de S e F.

Já não é tão imediato ver como se calcula $v(E)$ para a produção $E \longrightarrow FX$, visto que esse valor deve depender dos atributos de X, os quais ainda nem sequer se conhecem. Comecemos então por ver que atributos X deve ter.

Numa derivação a partir de S, se num dado ponto o símbolo não terminal mais à esquerda for X, ele deve estar precedido por uma palavra $a_1 + \cdots + a_n$ onde $a_1, \ldots, a_n$ são dígitos, como se observa na derivação seguinte:

$$S \Rightarrow E \Rightarrow FX \Rightarrow a_1X \Rightarrow a_1 + FX \Rightarrow a_1 + a_2X$$
$$\Rightarrow a_1 + a_2 + FX \Rightarrow \cdots \Rightarrow a_1 + \cdots + a_nX \ .$$

A palavra que precede X tem um valor associado que é a soma dos dígitos $a_1, \ldots, a_n$. Este valor, que é conhecido ou *dado* quando se inicia a derivação de X numa derivação esquerda, constitui o atributo d de X. Este atributo, que não depende de nenhuma produção para X, não pode ser calculado para a cabeça X de uma produção em função de atributos do corpo, logo não é sintetizado. Assim, só poderá ser herdado.

As palavras sobre o alfabeto terminal que X deriva têm a forma $a_{n+1} + \cdots + a_{n+k}$ ou λ. Se ao dado de X, $d(X)$, se forem sucessivamente adicionando os algarismos $a_{n+1}, \ldots, a_{n+k}$ (ou nada, no caso de λ), obtem-se um valor numérico como *resultado* da derivação de X. Este valor constitui o atributo r de X, o qual é sintetizado.

Em resumo, os atributos da gramática, os seus tipos e os seus domínios de valores são os seguintes:

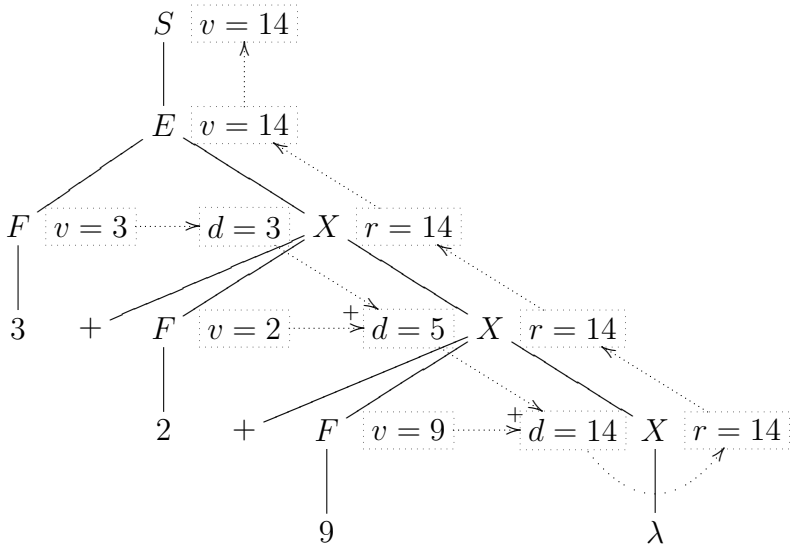
	Sint.	Herd.
S	v	
E	v	
X	r	d
F	v	

$$\mathcal{V}_v = \mathcal{V}_r = \mathcal{V}_d = \text{Nat}.$$

Para definir os atributos, procedemos do seguinte modo: para cada produção, definimos os atributos sintetizados da cabeça e os atributos herdados dos símbolos não terminais do corpo. Dado o significado pretendido para os atributos anteriormente explicado, não é difícil ver que se devem ter as seguintes definições:

	Sintetizados	Herdados
$S \longrightarrow E$	$v(S) = v(E)$	
$E \longrightarrow FX$	$v(E) = r(X)$	$d(X) = v(F)$
$X_1 \longrightarrow +FX_2$	$r(X_1) = r(X_2)$	$d(X_2) = d(X_1) + v(F)$
$X \longrightarrow \lambda$	$r(X) = d(X)$	
$F \longrightarrow 0$	$v(F) = 0$	
$\vdots$	$\vdots$	
$F \longrightarrow 9$	$v(F) = 9$	

A seguinte árvore de derivação de $3 + 2 + 9$ mostra a sequência do cálculo dos atributos:



5.3 Gramáticas de atributos

Uma gramática com definição de atributos chama-se uma *gramática de atributos*. Mais formalmente, uma gramática de atributos consiste nos seguintes elementos:

I. Gramática

Uma gramática independente do contexto $G = (T, N, S, P)$, onde

- G está na forma reduzida.
- S não ocorre no corpo de nenhuma produção.

II. Atributos

- Um conjunto $\mathcal{A}$ de elementos chamados *atributos*.
- Para cada $a \in \mathcal{A}$, um conjunto $\mathcal{V}_a$ dos *valores* de a.
- Para cada $A \in N$, um conjunto

$$\mathcal{A}(A) \subseteq \mathcal{A}$$

dos *atributos de* A.

  - O conjunto $\mathcal{A}(A)$ é uma união

$$\mathcal{A}(A) = \mathcal{H}(A) \cup \mathcal{S}(A)$$

    de um conjunto $\mathcal{H}(A)$ de atributos ditos *herdados* e de um conjunto $\mathcal{S}(A)$ de atributos ditos *sintetizados*, com $\mathcal{H}(A) \cap \mathcal{S}(A) = \emptyset$.
  - O símbolo inicial não tem atributos herdados, isto é, $\mathcal{H}(S) = \emptyset$ e $\mathcal{A}(S) = \mathcal{S}(S)$.

III. Cálculo dos atributos

Para cada produção

$$X_0 \longrightarrow X_1 \cdots X_n$$

define-se, para cada atributo sintetizado da cabeça e cada atributo herdado do corpo, uma função que indica como calcular os valores desse atributo em termos dos restantes atributos. Mais precisamente:

- Seja $a \in \mathcal{A}(X_k)$, onde $a \in \mathcal{S}(X_k)$ se $k = 0$ e $a \in \mathcal{H}(X_k)$ se $k > 0$.
- Então, para certos

$$a_1 \in \mathcal{A}(X_{k_1}), \ldots, a_m \in \mathcal{A}(X_{k_m})$$

com $0 \leq k_1 \leq \cdots \leq k_m \leq n$, é dada uma função

$$f : \mathcal{V}_{a_1} \times \cdots \times \mathcal{V}_{a_m} \to \mathcal{V}_a.$$

É costume escrever $a(X_k)$ para indicar o valor do atributo a de X_k, e então o seu cálculo é representado pela equação

$$a(X_k) = f(a_1(X_{k_1}), \ldots, a_m(X_{k_m})).$$

5.4 Outro exemplo

A seguinte gramática descreve, de forma muito simplificada, a parte de declarações de uma linguagem de programação:

$$\begin{array}{l} S \longrightarrow L \\ L \longrightarrow \lambda \mid D; L \\ D \longrightarrow var\, V : T \\ V \longrightarrow a \mid \cdots \mid z \\ T \longrightarrow int \mid real \end{array}$$

Pretende-se definir atributos que permitam construir uma *tabela de identificadores* associada a cada palavra. A tabela será representada por um conjunto

$$d(S) = \{(x_1, t_1), \ldots, (x_n, t_n)\}$$

de todos os pares (x_i, t_i) formados por uma variável $x_i \in \{a, \ldots, z\}$ e um tipo $t_i \in \{int, real\}$ correspondentes às declaraçõs contidas na palavra.

Na gramática não há nada que impeça que uma variável seja declarada mais de uma vez, mas isso é considerado um erro, mesmo que na declaração repetida a variável tenha o mesmo tipo. Assim, também se pretende determinar o conjunto

$$e(S) = \{(y_1, u_1), \ldots, (y_m, u_m)\}$$

dos erros resultantes de declarações repetidas.

Note-se que a linguagem formada pelas palavras sem declarações repetidas (isto é, com uma árvore de derivação para a qual $e(S) = \emptyset$) é uma sub-linguagem própria da linguagem definida pela gramática. Essa sub-linguagem poderia ser directamente definida por uma gramática, mas a gramática não seria independente do contexto, e é de esperar que fosse complicada e difícil de compreender. Por isso, na prática é preferível definir uma linguagem mais geral por meio de uma gramática independente do contexto (que é mais fácil de conceber e compreender) e restringi-la posteriormente com métodos semânticos, como estamos aqui a fazer com o método dos atributos. É essa a atitude adoptada, em particular, na definição das linguagens de programação.

Os atributos que irão ser usados estão definidos na seguinte tabela:

	Sint.	Herd.
S	d, e	
L	d_S, e_S	d_A, e_A
D	v, t	
V	v	
T	t	

Sobre o significado de d e e já falámos. Os valores destes atributos são conjuntos de pares (variável, tipo), logo

$$\mathcal{V}_d = \mathcal{V}_e = \mathcal{P}(\mathcal{V}_v \times \mathcal{V}_t)$$

onde $\mathcal{V}_v = \{a, \ldots, z\}$ e $\mathcal{V}_t = \{int, real\}$. O símbolo L tem atributos herdados d_A e e_A, e atributos sintetizados d_S e e_S. O significado destes atributos é semelhante ao dos atributos de X no exemplo das expressões aritméticas anteriormente apresentado. Numa derivação

$$S \Rightarrow^* var\, x_1 : t_1; \ldots; var\, x_n : t_n; L$$

$d_A(L)$ é o conjunto das declarações que ocorrem *antes* de L, isto é, no prefixo

$$var\, x_1 : t_1; \ldots; var\, x_n : t_n$$

omitindo as repetições, e $e_A(L)$ são as declarações repetidas de variáveis (isto é, os erros) *antes* de L. Mas L deriva uma palavra que tem a mesma forma do prefixo acima, e então $d_S(L)$ contém as declarações *seguintes* à derivação de L, isto é, as declarações em $d_A(L)$ às quais se vieram juntar as existentes na palavra derivada de L. O conjunto $e_S(L)$ define-se de forma semelhante. Os conjuntos de valores de d_A, e_A, d_S e e_S são, bem entendido, iguais aos de d e e:

$$\mathcal{V}_{d_A} = \mathcal{V}_{e_A} = \mathcal{V}_{d_S} = \mathcal{V}_{e_S} = \mathcal{P}(\mathcal{V}_v \times \mathcal{V}_t).$$

Os atributos v e t são fáceis de caracterizar: são respectivamente a variável e o tipo do símbolo a que se aplicam. Os seus conjuntos de valores $\mathcal{V}_v$ e $\mathcal{V}_t$ já foram definidos.

Para definir os atributos temos necessidade de uma função auxiliar

$$vars : \mathcal{P}(\mathcal{V}_v \times \mathcal{V}_t) \to \mathcal{P}(\mathcal{V}_v)$$

que dado um conjunto $X \subseteq \mathcal{V}_v \times \mathcal{V}_t$ de pares (variável, tipo) forma o conjunto

$$vars(X) = \{x : (\exists\, t)\, (x, t) \in X\}$$

das variáveis que figuram como primeiras componentes desses pares. Por outras palavras, se X for um conjunto de declarações, $vars(X)$ é o conjunto de variáveis declaradas.

A definição de atributos é a seguinte:

	Sintetizados	Herdados
$S \longrightarrow L$	$d(S) = d_S(L)$ $e(S) = e_S(L)$	$d_A(L) = \emptyset$ $e_A(L) = \emptyset$
$L \longrightarrow \lambda$	$d_S(L) = d_A(L)$ $e_S(L) = e_A(L)$	
$L_1 \longrightarrow D; L_2$	$d_S(L_1) = d_S(L_2)$	$d_A(L_2) = d_A(L_1) \cup$ IF $v(D) \in vars(d_A(L_1))$ THEN $\emptyset$ ELSE $\{(v(D), t(D))\}$ $e_A(L_2) = e_A(L_1) \cup$ IF $v(D) \in vars(d_A(L_1))$ THEN $\{(v(D), t(D))\}$ ELSE $\emptyset$
$D \longrightarrow var\, V : T$	$v(D) = v(V)$ $t(D) = t(T)$	
$V \longrightarrow a$ $\vdots$	$v(V) = a$	
$T \longrightarrow int$ $\vdots$	$t(T) = int$	

A equação para $d_A(L_2)$ podia escrever-se

$$d_A(L_2) = \begin{cases} d_A(L_1) & \text{se } v(D) \in vars(d_A(L_1)), \\ d_A(L_1) \cup \{(v(D), t(D))\} & \text{caso contrário.} \end{cases}$$

O mesmo se diz, evidentemente, de $e_A(L_2)$

5.5 Ordem de avaliação dos atributos – circularidade

Até aqui não referimos a ordem pela qual devem ser calculados os atributos, nem levantámos sequer o problema de saber se eles podem sempre ser calculados. Vamos aqui considerar uma situação em que os atributos dependem uns dos outros de forma circular. O seu cálculo obrigaria à resolução de um sistema de equações, o que nem sempre é possível. A apresentação deste

problema será feita por meio de um exemplo. Não procuraremos dar uma definição formal de "circularidade" ou apresentar algoritmos que permitam a sua detecção.

Consideremos então a gramática

$$\begin{aligned} S &\longrightarrow BA \\ B &\longrightarrow BA \mid b \\ A &\longrightarrow a \end{aligned}$$

onde estão definidos os seguintes atributos:

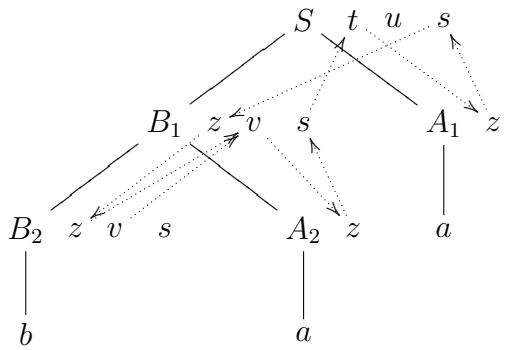
	Sint.	Herd.
S	s, t, u	
B	s, v	z
A		z

$$\begin{aligned} \mathcal{V}_s = \mathcal{V}_t = \mathcal{V}_u \\ = \mathcal{V}_v = \mathcal{V}_z = \text{Nat.} \end{aligned}$$

Consideremos a definição dos atributos:

	Sintetizados	Herdados
$S \longrightarrow BA$	$s(S) = z(A) + 1$	$z(B) = 2s(S) + 1$
	$t(S) = s(B) + 2$	$z(A) = t(S) + 1$
	$u(S) = 0$	
$B_1 \longrightarrow B_2A$	$s(B_1) = z(A) + 1$	$z(B_2) = z(B_1) + 1$
	$v(B_1) = v(B_2) + z(B_2)$	$z(A) = 2v(B_1) + 3$
$B \longrightarrow b$	$s(B) = 0$	
	$v(B) = 0$	
$A \longrightarrow a$		

Seja ainda a árvore de derivação da palavra *baa*:



As duas ocorrências de A e B foram distinguidas por meio de índices. Indicaram-se os atributos de cada nó mas não os seus valores. As setas a ponteado indicam as dependências entre atributos, tal como se deduz das equações que os definem. O percurso fechado no sentido indicado pelas setas corresponde à circularidade na definição dos atributos. Para ver como a dependência circular se traduz em termos do cálculo dos atributos, escrevemos as equações dos atributos que intervêm no percurso fechado:

$$\begin{aligned}
s(S) &= z(A_1) + 1 \\
z(A_1) &= t(S) + 1 \\
t(S) &= s(B_1) + 2 \\
s(B_1) &= z(A_2) + 1 \\
z(A_2) &= 2v(B_1) + 3 \\
v(B_1) &= v(B_2) + z(B_2) \\
v(B_2) &= 0 \\
z(B_2) &= z(B_1) + 1 \\
z(B_1) &= 2s(S) + 1.
\end{aligned}$$

Efectuando as substituições sucessivas chegava-se à equação

$$s(S) = 4s(S) + 12.$$

Esta equação traduz a dependência de $s(S)$ de si próprio. Esta independência é indesejável porque na maioria dos casos não tem solução, e mesmo quando tem, o cálculo dos atributos é ineficiente. No nosso caso, a equação não tem solução porque estamos a considerar como domínio de valores dos atributos o conjunto $\text{Nat} = \{0, 1, 2, \ldots\}$, mas se estivéssemos a considerar $\text{Int} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ já teria a solução $s(S) = -4$.

5.6 Avaliação da esquerda para a direita

Os atributos podem ser facilmente avaliados na chamada "avaliação da esquerda para a direita" se a sua definição satisfizer a seguinte propriedade:

Para toda a produção $X_0 \longrightarrow X_1 X_2 \cdots X_n$:

- Os atributos *sintetizados* de X_0 só dependem de:
  - atributos herdados de X_0;
  - atributos quaisquer de $X_1, \ldots, X_n$.
- Os atributos *herdados* de X_k com $1 \leq k \leq n$ só dependem de:
  - atributos herdados de X_0;

– atributos quaisquer de $X_1, \ldots, X_{k-1}$.

Todos os exemplos vistos até aqui, com excepção do que ilustra o fenómeno da circularidade, satisfazem estas condições. Nesse caso, a avaliação dos atributos pode ser feita de acordo com o seguinte esquema recursivo, aplicado a um nó N de uma árvore de derivação:

```
AVALIA(N):
BEGIN
  Seja X_0 ⟶ X_1 ··· X_n a produção usada no nó N;
  FOR k := 1 TO n DO
    IF X_k é um não-terminal THEN
      BEGIN
        V := k-ésimo filho de N;
        Avaliar os atributos herdados de V;
        AVALIA(V)
      END;
  Avaliar os atributos sintetizados de N
END
```

Quando AVALIA(N) é chamado, conhecem-se os atributos herdados de N e calculam-se primeiro os atributos dos filhos e depois os seus atributos sintetizados.

Capítulo 6

Autómatos de Pilha

[VERSÃO PRELIMINAR.]

Um autómato de pilha (AP) é essencialmente um autómato finito ao qual se acrescentou uma pilha, que pode conter símbolos de um alfabeto especial chamado "alfabeto da pilha". Lendo a pilha do topo até à base, símbolo a símbolo, forma-se uma palavra que descreve completamente o seu conteúdo. Uma característica importante dos AP, que os distingue dos autómatos finitos, é que têm memória infinita, visto que se podem guardar na pilha todas as palavras sobre o alfabeto correspondente.

Uma transição de um AP depende não só do estado em que ele se encontra, como do símbolo que está no topo da pilha. No fim da transição, tanto o estado como o conteúdo da pilha se alteraram. A modificação do estado consiste em substituí-lo por outro, eventualmente o mesmo. A modificação da pilha consiste em desempilhar o símbolo do topo e em seu lugar empilhar uma palavra arbitrária.

As transições são especificadas por uma função δ com três argumentos. Ao escrever $\delta(q, z, a)$, q é um estado, z é o símbolo do topo da pilha, e a pertence ao alfabeto de entrada ou é λ. Em geral, $\delta(q, z, a)$ é um conjunto finito de pares da forma (p, γ), que permitem seleccionar de forma não determinista o estado seguinte p e a palavra γ que substitui z no topo da pilha.

Definição 6.1 [Autómato de pilha] Um *autómato de pilha* (AP) é um séptuplo

$$A = (T, Q, Z, q_I, z_I, \delta, F),$$

em que:

- T é o alfabeto de *entrada*;
- Q é um conjunto finito de *estados*;
- Z é o alfabeto da *pilha*;

- $q_I \in Q$ é o estado *inicial*;
- $z_I \in Z$ é o símbolo *inicial da pilha*;
- δ é a função de *transição de configurações*;
- $F \subseteq Q$ é o conjunto de estados de *aceitação*.

A função δ tem por domínio o conjunto

$$Q \times Z \times (T \cup \{\lambda\}),$$

o que quer dizer que se calcula $\delta(q, z, a)$ para $q \in Q$, $z \in Z$ e $a \in T \cup \{\lambda\}$, como explicado mais acima. O codomínio de δ é o conjunto

$$\mathcal{P}_{\text{fin}}(Q \times Z^*)$$

de todos os subconjuntos finitos de $Q \times Z^*$. Assim, $\delta(q, z, a)$ é um conjunto finito de pares da forma (p, γ) com $p \in Q$ e $\gamma \in Z^*$, como também referido anteriormente. Escreve-se então

$$\delta : Q \times Z \times (T \cup \{\lambda\}) \to \mathcal{P}_{\text{fin}}(Q \times Z^*).$$

Definição 6.2 [Configuração] Uma *configuração* de um AP é um par (q, γ) descrevendo o estado em que o AP se encontra e o conteúdo da pilha. A configuração *inicial* é (q_I, z_I). O conjunto de todas as configurações é $Q \times Z^*$.

Definição 6.3 [Transição] Escreve-se

$$(q, z\gamma) \overset{a}{\vdash} (q', \gamma'\gamma)$$

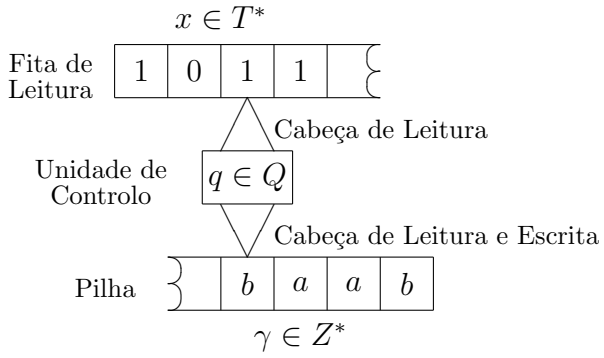
se

$$(q', \gamma') \in \delta(q, z, a),$$

onde $q, q' \in Q$, $z \in Z$, $\gamma, \gamma' \in Z^*$ e $a \in T \cup \{\lambda\}$. Diz-se então que há uma *transição* da configuração $(q, z\gamma)$ para a configuração $(q', \gamma'\gamma)$ etiquetada por a, o que justifica o nome dado a δ.

Note-se que, ao representar o conteúdo da pilha por uma palavra, o símbolo do topo é o primeiro da esquerda. Assim, em $z\gamma$, o topo é z, o qual, depois de desempilhado e substituído por γ', dá o novo conteúdo $\gamma'\gamma$.

É costume representar graficamente um AP na seguinte forma:



Vejamos com mais pormenor a interpretação do seu funcionamento.
Inicialmente:

- Uma palavra $x \in T^*$ está inscrita na fita de leitura, e a cabeça de leitura está colocada sobre o primeiro símbolo de x (o mais à esquerda).
- A unidade de controlo está no estado inicial q_I.
- A pilha contém unicamente o símbolo z_I e a cabeça de leitura e escrita está colocada sobre z_I.

Em resumo, inicialmente o AP está na configuração inicial (q_I, z_I) e tem a palavra x para ler.

Em certa etapa do funcionamento:

- A cabeça de leitura está sobre uma célula da fita de leitura contendo um símbolo $a \in T$ que ocorre na palavra x; os símbolos à esquerda de a (mas não o próprio a) já foram lidos.
- Na pilha está inscrita uma palavra de Z^*, podendo dar-se dois casos:
  - Essa palavra é a palavra vazia λ.
  - A palavra é não vazia e o primeiro símbolo (z, digamos) está a ser inspeccionado pela cabeça de leitura e escrita. Podemos designar essa palavra por $z\gamma$, com $z \in Z$ e $\gamma \in Z^*$. (Inicialmente, $z = z_I$ e $\gamma = \lambda$.)
- A unidade de controlo está num estado $q \in Q$.

Assim, o AP está numa configuração (q, λ) ou $(q, z\gamma)$, está a examinar o símbolo $a \in T$, e prepara-se para efectuar uma transição para uma nova configuração.

Transição de configuração:

- Consideremos a configuração anterior. Se a palavra inscrita na pilha for λ, isto é, se a configuração for (q, λ), não há transição possível.

- Caso contrário, a transição vai depender dos símbolos q, z, podendo o AP escolher entre ignorar ou não o símbolo a. Esta possibilidade de escolha, que traduz o primeiro aspecto não determinista do AP, existe em princípio visto que tanto (q, z, λ) como (q, z, a) (ignorar e não ignorar a, respectivamente) pertencem ao domínio de δ.
  - Ignorando a, suponhamos que

    $$\delta(q, z, \lambda) = \{(q_1, \gamma_1), \ldots, (q_r, \gamma_r)\}.$$

    Então o AP pode escolher qualquer um dos pares acima enumerados, que é o segundo aspecto do não determinismo de um AP. Se escolheu (q_i, γ_i), transita para a configuração $(q_i, \gamma_i\gamma)$ e *mantém* a cabeça de leitura sobre o símbolo a. O novo estado da unidade de controlo é q_i, e a cabeça de leitura e escrita apagou (desempilhou) z e escreveu (empilhou) γ_i, ficando a inspeccionar o primeiro símbolo de $\gamma_i\gamma$. Se $\gamma_i = \lambda$, esta acção reduz-se a desempilhar z. Se $\gamma_i = z$, a pilha fica inalterada. Se $\gamma_i = \rho z$, a acção consiste em empilhar ρ em cima de z. Note-se que se $\delta(q, z, \lambda) = \emptyset$, não há transição possível ignorando a.
  - Tendo a em consideração, se

    $$\delta(q, z, a) = \{(p_1, \rho_1), \ldots, (p_s, \rho_s)\},$$

    tudo se passa como anteriormente, com a excepção de que o símbolo a foi lido, e portanto a cabeça de leitura *avança* para o próximo símbolo de x (se existir; se não, para a próxima célula da fita, que está vazia). Se $\delta(q, z, a) = \emptyset$, não há transição possível lendo a.

O comportamento do AP consiste numa sucessão de transições a partir da configuração inicial. Contrariamente ao caso dos autómatos finitos, é hábito considerar dois critérios de reconhecimento de linguagens.

Critérios de reconhecimento:

- Após ler uma palavra, o AP atinge uma configuração em que a pilha está vazia. A linguagem reconhecida por este critério chama-se "linguagem nula" e denota-se $N(A)$ para um AP A.
- Após ler uma palavra, o AP atinge uma configuração em que o estado é de aceitação. A linguagem reconhecida por este critério denota-se $L(A)$.
- Uma situação de erro é detectada em qualquer dos casos quando a palavra não tiver sido toda lida e $\delta(q, z, \lambda) = \emptyset = \delta(q, z, a)$ para o estado corrente q, topo da pilha z e símbolo de entrada a.

Para formalizar as noções de funcionamento e reconhecimento começamos por generalizar a relação de transição de configurações a palavras arbitrárias de T^*.

Definição 6.4 [Transição iterada] Dadas configurações c e c' e $x \in T^*$, põe-se

$$c \overset{x}{\vdash} c'$$

se e só se existirem

- configurações $c_0, c_1, \ldots, c_n$ e
- elementos $a_1, a_2, \ldots, a_n \in T \cup \{\lambda\}$

tais que

- $c = c_0 \overset{a_1}{\vdash} c_1 \overset{a_2}{\vdash} \cdots \overset{a_n}{\vdash} c_n = c'$, e
- $x = a_1 a_2 \cdots a_n$.

A relação $c \overset{x}{\vdash} c'$ significa que o AP pode atingir a configuração c' partindo de c após ter lido a palavra x. Convenciona-se que

$$c \overset{\lambda}{\vdash} c$$

para toda a configuração c, o que está de acordo com a interpretação anterior.

Definição 6.5 [Linguagem reconhecida] Seja A um AP. A *linguagem reconhecida por A pelo critério da pilha vazia*, ou linguagem *nula* de A, é a linguagem

$$N(A) = \{x \in T^* : (\exists q \in Q)\,(q_I, z_I) \overset{x}{\vdash} (q, \lambda)\}.$$

A *linguagem reconhecida por A pelo critério dos estados de aceitação* é

$$L(A) = \{x \in T^* : (\exists q \in F, \gamma \in Z^*)\,(q_I, z_I) \overset{x}{\vdash} (q, \gamma)\}.$$

Veremos adiante que estes dois critérios são equivalentes, no sentido em que definem a mesma classe de linguagens.

Exemplo 6.6 Para toda a palavra $x = a_1 a_2 \cdots a_n$ sobre um determinado alfabeto, seja

$$x^I = a_n \cdots a_2 a_1$$

a palavra "inversa" de x, também chamada "imagem ao espelho" de x. Seja agora $T = \{a, b\}$ e consideremos a linguagem

$$L = \{x x^I : x \in T^*\}$$

formada por palavras arbitrárias seguidas da sua imagem ao espelho. Um AP A para reconhecer esta linguagem pelo critério da pilha vazia pode funcionar do seguinte modo.

O autómato vai lendo sucessivamente os símbolos inscritos na fita de leitura e empilha-os. Num dado momento, "adivinha" (não deterministicamente) que leu a metade x da palavra xx^I inicialmente na fita de leitura, e altera o seu modo de funcionamento. Se o seu "palpite" estiver correcto, o conteúdo da pilha é x^I, como é fácil de ver. Assim, o autómato passa a comparar o símbolo lido com o símbolo do topo da pilha e, caso sejam iguais (como é de esperar, se a palavra inicialmente na fita de leitura pertencer a L e se o palpite estiver correcto), desempilha, avança na fita de leitura, e repete a operação até a pilha ficar vazia.

O AP vai ter dois estados, $Q = \{p, q\}$, tal que no estado p empilha e no estado q desempilha. Como a operação inicial é desempilhar, deve ter-se $q_I = p$. O alfabeto da pilha é $Z = \{a, b, z\}$, onde z é um símbolo que inicialmente tem de estar na pilha para ela não estar vazia. Por essa mesma razão, $z_I = z$. Como o AP vai reconhecer a linguagem pelo critério da pilha vazia, o conjunto F é irrelevante e podemos pôr $F = \emptyset$. Para terminar a especificação do autómato falta definir a função δ, o que é feito mediante a seguinte tabela:

δ	a	b	λ
p, z	p, az	p, bz	q, z
p, a	p, aa	p, ba	q, a
p, b	p, ab	p, bb	q, b
q, z			q, λ
q, a	q, λ		
q, b		q, λ	

As linhas da tabela estão indexadas por pares em $Q \times Z$ que indicam o estado corrente e o topo da pilha (omitiram-se os parênteses da notação usual dos pares para simplificar a leitura). As colunas estão indexadas por $T \cup \{\lambda\}$, e as casas da tabela contêm os valores de δ. Por exemplo, no cruzamento da linha p, z com a coluna a encontra-se p, az, o que indica que

$$\delta(p, z, a) = \{(p, az)\}.$$

Evidentemente, vários pares podem ocupar a mesma casa, visto que os valores de δ podem ser conjuntos com mais de um elemento. Algumas casas podem estar vazias, e nesse caso o valor de δ é o conjunto vazio, como em $\delta(q, z, a) = \emptyset$ na tabela acima.

O comportamento do AP no estado p é independente de o símbolo que está no topo da pilha ser z, a ou b, e consiste ou em empilhar o símbolo lido ou em mudar para o estado q sem ler. Neste estado desempilham-se todos os símbolos da pilha que coincidam com os símbolos lidos ou, no caso de no topo da pilha estar z, este é desempilhado sem atender à fita de leitura. O comportamento em p é não determinista, visto que há a opcção entre ler o próximo símbolo e empilhá-lo, e ignorá-lo mudando o estado para q.

O comportamento do AP pode ser observado passo a passo indicando as sucessivas configurações pelas quais vai passando juntamente com a porção da fita que ainda falta ler. Um comportamento possível em que a palavra *abba* é reconhecida é o seguinte:

ESTADO	PILHA	FITA
p	z	$abba$
p	az	bba
p	baz	ba
q	baz	ba
q	az	a
q	z	λ
q	λ	λ

A palavra ab não é reconhecida, visto que *nenhum* comportamento conduz á pilha vazia com a palavra toda lida. Eis um comportamento possível:

ESTADO	PILHA	FITA
p	z	ab
p	az	b
q	az	b

A partir daqui não existem transições, visto que o topo da pilha não coincide com o primeiro símbolo da fita de leitura. Em alternativa, a partir da segunda linha podia prosseguir-se para

p	baz	λ
q	baz	λ

mas de novo não haveria transição. Aqui a palavra foi toda lida, mas como não se consegue esvaziar a pilha, a palavra não é reconhecida.

No próximo exemplo introduzimos o conceito de AP determinista, que não é exactamente o que se espera por analogia com os autómatos finitos.

Exemplo 6.7 O comportamento não determinista do AP do Exemplo 6.6 era devido à necessidade de adivinhar o meio da palavra a reconhecer. Assim, é de esperar que se modificarmos a linguagem inserindo a meio de cada palavra um símbolo separador especial, o autómato passe a ter um comportamento determinista. A razão é que, ao detectar esse símbolo, o autómato fica a "saber" que é a altura de passar da fase de empilhar à de desempilhar.

Assim, suponhamos que $T = \{a, b, c\}$ onde c é o tal símbolo especial e redefinamos

$$L = \{xcx^I : x \in \{a, b\}^*\}.$$

O AP correspondente é definido por $Q = \{p, q\}$, $q_I = p$, $Z = \{a, b, z\}$, $z_I = z$ e $F = \emptyset$ tal como no autómato do Exemplo 6.6. Por comparação com esse autómato, não é difícil reconhecer que δ pode ser definida pela tabela seguinte:

δ	a	b	c	λ
p, z	p, az	p, bz	q, z	
p, a	p, aa	p, ba	q, a	
p, b	p, ab	p, bb	q, b	
q, z				q, λ
q, a	q, λ			
q, b		q, λ		

Note-se que se considera que este AP é determinista, mesmo tendo transições etiquetadas por λ, ao contrário do que se passa com os autómatos finitos. Isto é razoável, porque em nenhumas circunstâncias é necessário optar entre duas ou mais transições possíveis. Isto pode ser constatado da seguinte maneira:

- Cada casa da tabela tem no máximo um par, logo não há escolhas a fazer *dentro de cada casa*.
- Em cada linha, quando a coluna de λ está preenchida, as de T estão vazias, e quando pelo menos uma coluna de T está preenchida, é a de λ que está vazia. Assim, não há escolhas a fazer *entre uma ou outra casa*.

O determinismo do AP pode ser observado com o reconhecimento da palavra

abcba:

ESTADO	PILHA	FITA
p	z	$abcba$
p	az	$bcba$
p	baz	cba
q	baz	ba
q	az	a
q	z	λ
q	λ	λ

Este comportamento deve ser comparado com o do Exemplo 6.6 para a palavra *abba*.

Vamos agora ver um exemplo de um AP que reconhece uma linguagem pelo critério dos estados de aceitação.

Exemplo 6.8 Consideremos a linguagem

$$L = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}\cdots a^{n_k}b^{n_k} : k \geq 1, n_1 \geq 1, n_2 \geq 1, \ldots, n_k \geq 1\},$$

sobre o alfabeto $T = \{a, b\}$. Uma palavra de L consiste numa secção da forma $a^{n_1}b^{n_1}$, com igual número de a e b, que se repete algumas vezes, possivelmente com outros números de a e b.

O autómato que vamos definir tem estados $Q = \{q_I, q, q_F\}$, onde q_I é evidentemente o estado inicial e q_F é o único estado de aceitação, isto é, $F = \{q_F\}$. O alfabeto da pilha é $Z = \{a, z_I\}$, sendo z_I o símbolo inicial. Vejamos primeiro informalmente como se define δ, em termos do comportamento desejado para o autómato.

O reconhecimento de

$$a^{n_1}b^{n_1}a^{n_2}b^{n_2}\cdots a^{n_k}b^{n_k}$$

processa-se do seguinte modo:

- O estado q_I empilha os n_1 símbolos a em cima de z_I. Ao encontrar o primeiro b, desempilha um a e passa para o estado q.
- O estado q lê os restantes $n_1 - 1$ símbolos b e por cada um desempilha um a. Quando no topo da pilha estiver z_I, o estado passa a q_F deixando a pilha inalterada.
- Se $k = 1$, a palavra foi toda lida e é reconhecida por q_F ser de aceitação. Se $k > 1$, o autómato lê o próximo a, empilha-o sobre z_I, e regressa ao estado q_I, onde o ciclo que acabou de ser descrito se repete.

Desta descrição informal apercebemo-nos facilmente que o AP é determinista. Isso também pode ser observado na tabela mais abaixo como se indicou a propósito do Exemplo 6.7.

δ	a	b	λ
q_I, z_I	q_I, az_I		
q_I, a	q_I, aa	q, λ	
q, z_I			q_F, z_I
q, a		q, λ	
q_F, z_I	q_I, az_I		

Note-se que a linha de q_F, a foi omitida, por ter todas as casas vazias.

Uma diferença importante entre os critérios de aceitação por pilha vazia e por estados de aceitação pode observar-se desde já: quando a pilha fica vazia, o autómato pára; porém, quando atinge um estado de aceitação, pode prosseguir com a computação, passando mesmo por estados que não são de aceitação antes de regressar a um estado de aceitação.

Definição 6.9 [AP determinista] Para um conjunto finito X, seja $|X|$ o número dos seus elementos. Um AP A diz-se *determinista* se, para todo o $q \in Q$ e $z \in Z$:

- ou $(\forall a \in T)\, |\delta(q, z, a)| \leq 1$ e $\delta(q, z, \lambda) = \emptyset$;
- ou $(\forall a \in T)\, \delta(q, z, a) = \emptyset$ e $|\delta(q, z, \lambda)| \leq 1$.

A seguinte propriedade é a que se espera.

Proposição 6.10 *Seja $c \in Q \times Z^*$ uma configuração de um AP determinista e $a \in T$. Se existirem transições $c \overset{a_1}{\vdash} c_1$ e $c \overset{a_2}{\vdash} c_2$ com $a_1, a_2 \in \{a, \lambda\}$ e $c_1, c_2 \in Q \times Z^*$ então $a_1 = a_2$ e $c_1 = c_2$.*

Dem. Pela Definição 6.3, se $c \overset{a_i}{\vdash} c_i$ $(i = 1, 2)$ então c tem a forma $(q, z\gamma)$ e existe $(q_i, \gamma_i) \in \delta(q, z, a_i)$ tal que $c_i = (q_i, \gamma_i\gamma)$. Não se pode ter $a_1 \neq a_2$ porque então um dos dois seria igual a a e o outro igual a λ, vindo $|\delta(q, z, a)| > 0$ e $|\delta(q, z, \lambda)| > 0$, o que não é permitido pela definição de AP determinista. Tem-se então $a_1 = a_2$. Se $c_1 \neq c_2$, isto é, $(q_1, \gamma_1\gamma) \neq (q_2, \gamma_2\gamma)$, teria de ser $(q_1, \gamma_1) \neq (q_2, \gamma_2)$. Como $a_1 = a_2$, isto implicaria $|\delta(q, z, a_1)| \geq 2$, o que também contraria a definição de AP determinista. Conclui-se que tem de ser $c_1 = c_2$. ∎

Convenção 6.11 No que se segue é cómodo considerar que a função δ é definida por um conjunto de quíntuplos (q, z, a, p, γ) tais que $(p, \gamma) \in \delta(q, z, a)$. Por comodidade de leitura, escreveremos tais quíntuplos na forma $(q, z) \xrightarrow{a} (p, \gamma)$, e chamar-lhes-emos "regras de transição". As regras de transição podem ser interpretadas em termos da tabela que caracteriza δ, como indicando que a casa indexada por (q, z) na linha e a na coluna contém o elemento (p, γ), possivelmente entre outros elementos. ∎

Vamos agora ver que os critérios de reconhecimento por pilha vazia e por estados de aceitação são equivalentes, no sentido em que reconhecem a mesma classe de linguagens.

Teorema 6.12 *Seja A um AP e $N(A)$ a linguagem por ele reconhecida pelo critério da pilha vazia. Existe um AP A' que reconhece a mesma linguagem pelo critério dos estados de aceitação, isto é, tal que $L(A') = N(A)$. Além disso, se A for determinista, A' é determinista.*

Dem. O autómato A' é construído como uma extensão de A. Vai ter um novo estado inicial q'_I e um novo símbolo inicial na pilha z'_I, e a sua primeira transição é $(q'_I, z'_I) \overset{\lambda}{\vdash} (q_I, z_I z'_I)$, determinada por uma regra de transição $(q'_I, z'_I) \xrightarrow{\lambda} (q_I, z_I z'_I)$. A partir daqui, A' comporta-se exactamente como A, excepto que cada configuração por que passa tem o símbolo adicional z'_I na base da pilha. Mais precisamente, as configurações de A' têm a forma $(q, \gamma z'_I)$ para toda a configuração (q, γ) de A. Quando A está numa configuração de aceitação (q, λ) em que a pilha está vazia, A' está na configuração (q, z'_I). O que é preciso fazer então é promover a transição desta configuração para uma outra com um estado de aceitação. Para tanto basta criar apenas um estado de aceitação q'_F e acrescentar a A' as regras de transição $(q, z'_I) \xrightarrow{\lambda} (q'_F, z'_I)$ para cada $q \in Q$. Tem-se a transição $(q, z'_I) \overset{\lambda}{\vdash} (q'_F, z'_I)$, como se pretende.

Vejamos com mais precisão a definição de A'. Tem-se

$$A' = (T, Q', Z', q'_I, z'_I, \delta', F'),$$

em que:

- $Q' = Q \cup \{q'_I, q'_F\}$ com $q'_I, q'_F \notin Q$;
- $Z' = Z \cup \{z'_I\}$ com $z'_I \notin Z$;
- $F' = \{q'_F\}$;
- δ' é constituída pelas regras de δ, às quais se acrescentaram:
  - $(q'_I, z'_I) \xrightarrow{\lambda} (q_I, z_I z'_I)$; e

– $(q, z'_I) \xrightarrow{\lambda} (q'_F, z'_I)$ para todo o $q \in Q$.

Atendendo à discussão anterior, vê-se facilmente que $(q_I, z_I) \overset{x}{\vdash} (q, \lambda)$ se e só se $(q'_I, z'_I) \overset{x}{\vdash} (q'_F, z'_I)$, o que mostra que $L(A') = N(A)$. Além disso, as regras de transição que foram acrescentadas a δ não introduzem não determinismo, pelo que A' é determinista se A o for. ∎

Exemplo 6.13 Consideremos o autómato A do Exemplo 6.7. Da construção do autómato A' apresentamos apenas a definição da função δ':

δ'	a	b	c	λ
q'_I, z'_I				$q_I, z_I z'_I$
p, z	p, az	p, bz	q, z	
p, a	p, aa	p, ba	q, a	
p, b	p, ab	p, bb	q, b	
q, z				q, λ
q, a	q, λ			
q, b		q, λ		
p, z'_I				q'_F, z'_I
q, z'_I				q'_F, z'_I

Note-se que esta tabela se obteve acrescentando a primeira e as duas últimas linhas à tabela de δ, seguindo a prescrição da demonstração do teorema anterior.

O resultado seguinte é o inverso do anterior. Note-se porém, que não se faz a correspondente afirmação sobre o caso determinista.

Teorema 6.14 *Para todo o AP A, é possível construir um AP A' tal que $N(A') = L(A)$.*

Dem. Também neste caso A' é uma extensão de A, executando uma primeira transição $(q'_I, z'_I) \overset{\lambda}{\vdash} (q_I, z_I z'_I)$ e passando a "simular" A. Quando A atinge uma configuração (q, γ) para um estado de aceitação $q \in F$, A' atinge a configuração $(q, \gamma z'_I)$, podendo escolher não deterministicamente continuar a simular A ou esvaziar a pilha. O esvaziamento da pilha consegue-se transitando para um estado q'_V por meio da regra de transição $(q, z) \xrightarrow{\lambda} (q'_V, z)$, o qual se encarrega de desempilhar todos os símbolos usando regras $(q'_V, z) \xrightarrow{\lambda} (q'_V, \lambda)$. A definição completa de A' é a seguinte:

$$A' = (T, Q', Z', q'_I, z'_I, \delta', F'),$$

em que:

- $Q' = Q \cup \{q'_I, q'_V\}$ com $q'_I, q'_V \notin Q$;
- $Z' = Z \cup \{z'_I\}$ com $z'_I \notin Z$;
- $F' = \emptyset$;
- δ' é constituída pelas regras de δ e pelas regras:
  - $(q'_I, z'_I) \xrightarrow{\lambda} (q_I, z_I z'_I)$;
  - $(q, z) \xrightarrow{\lambda} (q'_V, z)$ para todo o $q \in F$ e $z \in Z'$; e
  - $(q'_V, z) \xrightarrow{\lambda} (q'_V, \lambda)$ para todo o $z \in Z'$.

Para $x \in T^*$ e $q \in F$, tem-se $(q_I, z_I) \overset{x}{\vdash} (q, \gamma)$ para um certo γ se e só se $(q'_I, z'_I) \overset{x}{\vdash} (q'_V, \lambda)$, logo $N(A') = L(A)$. ∎

Exemplo 6.15 Dado o autómato A do Exemplo 6.8, a tabela da função δ' do autómato A' é a seguinte:

δ'	a	b	λ
q'_I, z'_I			$q_I, z_I z'_I$
q_I, z_I	$q_I, a z_I$		
q_I, a	q_I, aa	q, λ	
q, z_I			q_F, z_I
q, a		q, λ	
q_F, z_I	$q_I, a z_I$		q'_V, z_I
q_F, z'_I			q'_V, z'_I
q_F, a			q'_V, a
q'_V, z			q'_V, λ

(Na última linha, z é qualquer dos símbolos z'_I, z_I, a.) Note-se que, embora A seja determinista, A' não o é em virtude de se ter acrescentado a regra $(q_F, z_I) \xrightarrow{\lambda} (q'_V, z_I)$.

O facto de, no teorema anterior, se chegar a um autómato que pode ser não determinista, mesmo partindo de um autómato determinista, é indesejável, mas pode ser corrigido com um artifício simples. Trata-se de acrescentar um símbolo terminador, que representaremos por \$, a cada palavra, e permitir que o autómato A' reconheça a linguagem $N(A') = L(A)\{\$\}$.

Teorema 6.16 *Seja A um AP determinista e \$ um símbolo não pertencente a T. Existe um AP determinista A' tal que $N(A') = L(A)\{\$\}$.*

Dem. A construção de A' é semelhante à do teorema anterior. A única diferença é que a decisão de esvaziar a pilha é determinista, e ocorre quando se encontra \$ na fita de leitura. Mais precisamente, tem-se:

$$A' = (T \cup \{\$\}, Q', Z', q'_I, z'_I, \delta', F'),$$

em que Q', Z' e F' se definem como anteriormente, e δ' é constituída pelas regras de δ e pelas regras:

- $(q'_I, z'_I) \xrightarrow{\lambda} (q_I, z_I z'_I)$;
- $(q, z) \xrightarrow{\$} (q'_V, z)$ para todo o $q \in F$ e $z \in Z'$; e
- $(q'_V, z) \xrightarrow{\lambda} (q'_V, \lambda)$ para todo o $z \in Z'$.

É fácil ver que $N(A') = L(A)\{\$\}$ e A' é determinista. ∎

Exemplo 6.17 Retomemos o autómato A do Exemplo 6.8. A tabela da função δ' para o autómato A' com o símbolo terminador \$ é a seguinte:

δ'	a	b	\$	λ
q'_I, z'_I				$q_I, z_I z'_I$
q_I, z_I	$q_I, a z_I$			
q_I, a	q_I, aa	q, λ		
q, z_I				q_F, z_I
q, a		q, λ		
q_F, z_I	$q_I, a z_I$		q'_V, z_I	
q_F, z'_I			q'_V, z'_I	
q_F, a			q'_V, a	
q'_V, z				q'_V, λ

Esta tabela deve ser comparada com a do exemplo anterior.

Vamos agora ver uma generalização de autómato de pilha em que se permite que as transições sejam determinadas, não apenas pelo símbolo que está no topo da pilha, mas possivelmente também por alguns símbolos que se lhe seguem.

Definição 6.18 [AP generalizado] Um *autómato de pilha generalizado* define-se tal como um autómato de pilha, excepto que o domínio de δ é $Q \times ZZ^* \times (T \cup \{\lambda\})$. Noutros termos, as regras de transição têm a forma $(q, z\gamma) \xrightarrow{a} (q', \gamma')$ com $z \in Z$ e $\gamma \in Z^*$. Diremos que o AP generalizado é *determinista no primeiro símbolo* se o AP que se obtem substituindo cada regra $(q, z\gamma) \xrightarrow{a} (q', \gamma')$ por $(q, z) \xrightarrow{a} (q', \gamma')$ for determinista.

Assim, em cada transição de um AP generalizado, desempilha-se a palavra $z\gamma$ e empilha-se γ'. O AP generalizado é determinista no primeiro símbolo se a escolha da transição a efectuar ficar determinada pelo símbolo z do topo da pilha. O próximo resultado mostra que os AP generalizados não aumentam a classe das linguagens reconhecidas pelos AP, pelo que se resumem a ser de utilização mais cómoda em certas situações. Veremos uma tal utilização na construção de analisadores sintácticos ascendentes.

Teorema 6.19 *Toda a linguagem reconhecida por um AP generalizado é reconhecida por um AP (normal). Além disso, se o AP generalizado for determinista no primeiro símbolo, o correspondente AP é determinista.*

Dem. O AP generalizado é transformado num AP substituindo cada regra

$$(q, z_1 \cdots z_k) \xrightarrow{a} (q', \gamma') \qquad (k > 1)$$

pelas regras

$$\begin{aligned} (q, z_1) &\xrightarrow{a} (r_2, \lambda) \\ (r_2, z_2) &\xrightarrow{\lambda} (r_3, \lambda) \\ &\vdots \\ (r_k, z_k) &\xrightarrow{\lambda} (q', \gamma') \end{aligned}$$

onde $r_2, \ldots, r_k$ são novos estados, distintos dos estados previamente existentes no autómato e dos estados criados por análoga transformação de outras regras. A primeira das regras introduzidas lê a (se $a \in T$) e desempilha z_1; a segunda desempilha z_2, e assim sucessivamente, até que a última desempilha z_k, muda o estado para q' e empilha γ'. Procedendo desta forma, vê-se que os dois autómatos reconhecem a mesma linguagem. Os estados $r_2, \ldots, r_k$ não introduzem não determinismo, como também não o introduz a regra $(q, z_1) \xrightarrow{a} (r_2, \lambda)$ se o AP generalizado for determinista no primeiro símbolo. Fica concluída a demonstração. ∎

Vamos agora relacionar os autómatos de pilha com as gramáticas independentes do contexto.

Teorema 6.20 *Para toda a gramática independente do contexto G, existe um autómato de pilha A tal que $N(A) = L(G)$.*

Dem. Dada a gramática $G = (T, N, S, P)$, define-se $A = (T, Q, Z, q_I, z_I, \delta, F)$ por

- $Q = \{q\}$.

- $Z = V$.
- $q_I = q$.
- $z_I = S$.
- $F = \emptyset$.
- δ é definida pelas regras:
  - *Reconhecimento:* $(q, a) \xrightarrow{a} (q, \lambda)$ para todo o $a \in T$.
  - *Expansão:* $(q, A) \xrightarrow{\lambda} (q, \alpha)$ para toda a produção $A \to \alpha \in P$.

Se mostrarmos que

$$(q, \alpha) \overset{x}{\vdash} (q, \beta) \quad \Longleftrightarrow \quad \alpha \underset{E}{\Rightarrow}^* x\beta,$$

concluímos que

$$N(A) = \{x \in T^* : (q, S) \overset{x}{\vdash} (q, \lambda)\} = \{x \in T^* : S \underset{E}{\Rightarrow}^* x\} = L(G),$$

como se pretende. Para demonstrar a equivalência acima, comecemos por supor que $(q, \alpha) \overset{x}{\vdash} (q, \beta)$. Se esta transição tiver um só passo, ela resulta da aplicação de uma regra de reconhecimento ou de uma regra de expansão. No primeiro caso, a transição tem a forma $(q, a\alpha) \overset{a}{\vdash} (q, \alpha)$, dando de imediato $a\alpha \underset{E}{\Rightarrow}^* a\alpha$. No segundo caso tem-se uma transição $(q, A\alpha) \overset{\lambda}{\vdash} (q, \sigma\alpha)$ para uma produção $A \to \sigma$, e de novo $A\alpha \underset{E}{\Rightarrow} \sigma\alpha$. Se a transição tiver mais de um passo, a conclusão $\alpha \underset{E}{\Rightarrow}^* x\beta$ segue por transitividade.

Suponhamos agora que se tem uma derivação esquerda directa $xA\alpha \underset{E}{\Rightarrow} x\sigma\alpha$ para uma produção $A \to \sigma$. Aplicando $|x|$ regras de reconhecimento e uma regra de expansão, obtem-se $(q, xA\alpha) \overset{x}{\vdash} (q, A\alpha) \overset{\lambda}{\vdash} (q, \sigma\alpha)$. Generalizando a derivações esquerdas arbitrárias $\alpha \underset{E}{\Rightarrow}^* x\beta$, obtem-se, por transitividade, $(q, \alpha) \overset{x}{\vdash} (q, \beta)$. ∎

Exemplo 6.21 Consideremos a gramática de produções $S \to aSb \,|\, c$. O autómato construído na demonstração do teorema anterior tem a função de transição δ definida na seguinte tabela:

δ	a	b	c	λ	
q, S				q, aSb	q, c
q, a	q, λ				
q, b		q, λ			
q, c			q, λ		

Note-se que a casa da linha q, S e coluna λ tem dois elementos, logo o autómato é não determinista. À derivação $S \Rightarrow aSb \Rightarrow acb$ corresponde o seguinte comportamento no autómato:

ESTADO	PILHA	FITA
q	S	acb
q	aSb	acb
q	Sb	cb
q	cb	cb
q	b	b
q	λ	λ

O não determinismo do autómato corresponde exactamente à existência de várias produções para um mesmo símbolo não terminal. A construção do teorema anterior tem o interesse teórico de mostrar que a análise sintáctica é possível em todos os casos. Nos capítulos seguintes vamos estudar alguns casos importantes em que a análise sintáctica pode ser feita por AP deterministas.

O resultado recíproco do anterior também é válido.

Teorema 6.22 *Para todo o autómato de pilha A, existe uma gramática independente do contexto G tal que $L(G) = N(A)$.*

Dem. Para todos os $q, p \in Q$ e $\gamma \in Z^*$, consideremos a linguagem

$$N_{q,\gamma,p} = \{x \in T^* : (q, \gamma) \overset{x}{\vdash} (p, \lambda)\} \tag{6.1}$$

formada por todas as palavras que levam o autómato da configuração (q, γ) à configuração (p, λ). Dado que a linguagem $N(A)$ é o conjunto das palavras que levam da configuração inicial (q_I, z_I) a uma configuração (p, λ) para algum $p \in Q$, tem-se

$$N(A) = \bigcup_{p \in Q} N_{q_I, z_I, p}. \tag{6.2}$$

A gramática $G = (T, N, S, P)$ que gera $N(A)$ define-se do seguinte modo. O alfabeto terminal é o alfabeto de entrada do autómato. Os símbolos não terminais são o símbolo inicial S e um símbolo $A_{q,z,p}$ para cada $(q, z, p) \in Q \times Z \times Q$, que se pretende que gere a linguagem $N_{q,z,p}$. Em virtude da equação 6.2, escolhem-se para S as produções

$$S \to A_{q_I, z_I, p} \tag{6.3}$$

para todo o $p \in Q$. Pela equação 6.2, se cada $L(A_{q,z,p}) = N_{q,z,p}$ então $L(S) = N(A)$, como se pretende. Falta-nos definir as produções dos símbolos $A_{q,z,p}$.

Comecemos por mostrar que se tem a igualdade

$$N_{q,z,p} = \bigcup_{(q,z)\xrightarrow{a}(r,\gamma)\in\delta} \{a\}N_{r,\gamma,p}. \tag{6.4}$$

(A união é feita para todos os elementos na linha (q, z) da tabela de δ.) Se $x \in N_{q,z,p}$ então, pela equação 6.1 que define estes conjuntos, deve ter-se $(q, z) \overset{x}{\vdash} (p, \lambda)$. No primeiro passo da derivação utilizou-se uma certa regra $(q, z) \xrightarrow{a} (r, \gamma)$, pelo que a derivação se decompõe em $(q, z) \overset{a}{\vdash} (r, \gamma) \overset{y}{\vdash} (p, \lambda)$ com x=ay. De novo pela equação 6.1, tem-se $y \in N_{r,\gamma,p}$, logo $x = ay \in \{a\}N_{r,\gamma,p}$. Isto mostra que o membro esquerdo da equação 6.4 está contido no seu membro direito. O raciocínio pode ser facilmente invertido, concluindo-se que o membro direito também está incluído no esquerdo, donde a igualdade.

Os conjuntos $N_{r,\gamma,p}$ que aparecem na equação 6.4 não correspondem necessariamente a um símbolo não terminal, visto que se pode ter $|\gamma| \neq 1$. Assim, antes de podermos escrever as produções para os símbolos $A_{q,z,p}$ com base na equação 6.4, temos de ver com descrever os conjuntos $N_{r,\gamma,p}$.

Se $\gamma = z' \in Z$, para cada regra $(q, z) \xrightarrow{a} (r, z')$ a gramática tem a produção

$$A_{q,z,p} \to aA_{r,z',p} \tag{6.5}$$

visto que $\{a\}N_{r,z',p} \subseteq N_{q,z,p}$ pela equação 6.4.

Se $\gamma = \lambda$, a equação 6.1 mostra que $N_{r,\lambda,p} = \emptyset$ se $r \neq p$ e $N_{r,\lambda,p} = \{\lambda\}$ se $r = p$. Isto significa que na equação 6.4 podemos ignorar as regras $(q, z) \xrightarrow{a} (r, \lambda)$ com $r \neq p$. Para cada uma das regras $(q, z) \xrightarrow{a} (p, \lambda)$, a gramática tem a produção

$$A_{q,z,p} \to a \tag{6.6}$$

visto que, sendo $N_{p,\lambda,p} = \{\lambda\}$, se tem $\{a\}N_{p,\lambda,p} = \{a\} \subseteq N_{q,z,p}$.

Se $\gamma = z\rho$, tem-se a igualdade

$$N_{q,z\rho,p} = \bigcup_{r\in Q} N_{q,z,r}N_{r,\rho,p}. \tag{6.7}$$

Com efeito, se $x \in N_{q,z\rho,p}$ então, pela equação 6.1, $(q, z\rho) \overset{x}{\vdash} (p\lambda)$. Como o conteúdo inicial $z\rho$ da pilha é completamente desempilhado, há uma configuração intermédia (r, ρ) na anterior derivação em que pela primeira vez o conteúdo da pilha é ρ. Existem então palavras w e y tais que $x = wy$ e

$(q, z\rho) \overset{w}{\vdash} (r, \rho) \overset{y}{\vdash} (p, \lambda)$. As transições de $(q, z\rho) \overset{w}{\vdash} (r, \rho)$ não dependem de ρ, visto que em nenhuma situação intermédia o conteúdo da pilha é ρ. Deve então ter-se também $(q, z) \overset{w}{\vdash} (r, \lambda)$, o que mostra que $w \in N_{q,z,r}$ e $y \in N_{r,\rho,p}$. Deste modo, $x = wy \in N_{q,z,r}N_{r,\rho,p}$, logo o membro esquerdo da equação 6.7 está contido no seu membro direito. Por um raciocínio análogo se mostra que o membro direito está contido no esquerdo, o que justifica que se tenha a igualdade.

Iterando a equação 6.7, vê-se que, se $n > 1$,

$$N_{q,z_1\cdots z_n,p} = \bigcup_{p_1,\ldots,p_{n-1}\in Q} N_{q,z_1,p_1}N_{p_1,z_2,p_2}\cdots N_{p_{n-1},z_n,p}. \tag{6.8}$$

Assim, para cada regra $(q, z) \xrightarrow{a} (r, z_1 \cdots z_n)$ com $n > 1$, a gramática tem as produções

$$A_{q,z,p} \to aA_{q,z_1,p_1}A_{p_1,z_2,p_2}\cdots A_{p_{n-1},z_n,p} \tag{6.9}$$

para todos os $p_1, \ldots, p_{n-1} \in Q$, pelas equações 6.4 e 6.8. As produções 6.5, 6.6 e 6.9 são todas as produções do símbolo $A_{q,z,p}$. Juntando as produções destes símbolos às produções 6.3 para S, obtem-se o conjunto P das produções da gramática. Fica terminada a demonstração. ∎

Capítulo 7

Gramáticas LL(1)

[VERSÃO PRELIMINAR.]

Neste capítulo apresenta-se uma classe de gramáticas independentes do contexto — as gramáticas LL(1) — para as quais o problema da análise sintáctica descendente é particularmente simples. O algoritmo é determinista, como é desejável, no sentido em que em cada passo da análise existe (e é conhecida) uma única acção a executar. Além disso, a escolha dessa acção é determinada pelo primeiro símbolo da porção da fita de leitura que ainda falta analisar (um só símbolo de avanço ou "lookahead"). O algoritmo termina sempre, quer aceitando a palavra inicial quer assinalando um erro.

Este capítulo trata sucessivamente da definição das gramáticas LL(1), da apresentação de algoritmos que verificam se uma gramática dada é LL(1), e finalmente da caracterização do autómato de pilha associado a uma gramática LL(1).

7.1 Definição das gramáticas LL(1)

Em todo este capítulo, $G = (T, N, S, P)$ é uma gramática independente do contexto que se supõe estar na forma reduzida. Recordemos que isso significa que:

- todo o símbolo não terminal A é produtivo, isto é, gera uma linguagem $L(A) \neq \emptyset$;
- todo o símbolo $X \in N \cup T$, terminal ou não, é acessível a partir do símbolo inicial S, isto é, existem palavras $\alpha, \beta \in (N \cup T)^*$ tais que $S \Rightarrow^* \alpha X \beta$.

Como de costume, pomos $V = N \cup T$.

Definição do conjunto dos primeiros

Dada uma gramática em que as produções do símbolo inicial sejam, por exemplo,

$$S \to a\alpha \mid b\beta$$

com $a, b \in T$ e $\alpha, \beta \in V^*$, toda a palavra gerada por S começa quer por a quer por b. Dizemos então que o conjunto dos "primeiros" de S é $\{a, b\}$ e escrevemos

$$\mathrm{Prim}(S) = \{a, b\}.$$

Mais geralmente, podemos definir o conjunto $\mathrm{Prim}(\gamma)$ para qualquer palavra $\gamma \in V^*$. Por exemplo, para as duas expansões de S temos

$$\mathrm{Prim}(a\alpha) = \{a\}, \qquad \mathrm{Prim}(b\beta) = \{b\},$$

visto que toda a palavra gerada a partir de $a\alpha$ começa necessariamente por a e toda a palavra gerada a partir de $b\beta$ começa por b. A importância de se conhecerem estes conjuntos na análise sintáctica pode também ser ilustrada com o fragmento de gramática anterior. Se se pretendesse analisar uma palavra da forma $a\gamma$ a partir de S, teria de se utilizar a primeira produção de S, visto que $\mathrm{Prim}(a\gamma) = \mathrm{Prim}(a\alpha) = \{a\}$, e prosseguia-se analisando γ a partir de α. Se a palavra a analisar tivesse a forma $b\delta$ então utilizava-se a segunda produção de S. Se não tivesse nenhuma destas duas formas, isto é, se o primeiro símbolo da palavra a analisar não fosse a nem b, teria de ser assinalado um erro na palavra em análise.

Suponhamos agora que as produções de S tinham a forma

$$S \to \alpha \mid \beta$$

em que $\mathrm{Prim}(\alpha) = \{a, b\}$ e $\mathrm{Prim}(\beta) = \{b, c\}$. Se se pretendesse analisar uma palavra da forma $b\gamma$, o conhecimento apenas do primeiro símbolo de $b\gamma$ não permitiria decidir entre as duas produções de S, visto que $b \in \mathrm{Prim}(\alpha)$ e $b \in \mathrm{Prim}(\beta)$. Deste modo, $b\gamma$ podia em princípio ser gerada tanto a partir de α como a partir de β. Por esta razão, quando se têm produções da forma

$$A \to \alpha_1 \mid \cdots \mid \alpha_n$$

exige-se que se $i \neq j$ então $\mathrm{Prim}(\alpha_i) \cap \mathrm{Prim}(\alpha_j) = \emptyset$. Esta condição, que é obviamente necessária, não é, contudo, suficiente, e será generalizada mais adiante. Mas por agora contentemo-nos em definir rigorosamente o conjunto dos primeiros de uma palavra.

Definição 7.1 [Primeiros] O conjunto dos *primeiros* de $\alpha \in V^*$, que se denota $\mathrm{Prim}(\alpha)$, define-se por

$$\mathrm{Prim}(\alpha) = \{a \in T : (\exists \beta \in V^*)\, \alpha \Rightarrow^* a\beta\}.$$

Assim, $\mathrm{Prim}(\alpha)$ é o conjunto de todos os símbolos terminais que são os primeiros símbolos das palavras geradas por α. A determinação de $\mathrm{Prim}(\alpha)$, embora não seja conceptualmente complicada, não é tão simples como o poderá ter dado a entender o fragmento de gramática anterior. Um algoritmo será apresentado no parágrafo seguinte. Aqui, vamos extrair algumas consequências da definição, para a esclarecer melhor.

Propriedades 7.2 Sejam $a \in T$, $A \in N$ e $\alpha, \beta \in V^*$. Então:

i) $\mathrm{Prim}(a) = \{a\}$.
Imediato.

ii) *Se* $A \to \alpha_1 \mid \cdots \mid \alpha_n$ *forem todas as produções de* A *então* $\mathrm{Prim}(A) = \mathrm{Prim}(\alpha_1) \cup \cdots \cup \mathrm{Prim}(\alpha_n)$.
Com efeito, se $a \in \mathrm{Prim}(\alpha_i)$ então $\alpha_i \Rightarrow^* a\gamma$ para algum $\gamma \in V^*$, donde $A \Rightarrow \alpha_i \Rightarrow^* a\gamma$ e $a \in \mathrm{Prim}(A)$. Inversamente, se $a \in \mathrm{Prim}(A)$ existe uma derivação da forma $A \Rightarrow^* a\gamma$. Se $A \to \alpha_i$ for a produção usada no primeiro passo tem-se $A \Rightarrow \alpha_i \Rightarrow^* a\gamma$, o que mostra que $a \in \mathrm{Prim}(\alpha_i)$.

iii) $\mathrm{Prim}(\lambda) = \emptyset$.
Imediato.

iv)
$$\mathrm{Prim}(\alpha\beta) = \begin{cases} \mathrm{Prim}(\alpha) & \text{se } \alpha \not\Rightarrow^* \lambda, \\ \mathrm{Prim}(\alpha) \cup \mathrm{Prim}(\beta) & \text{se } \alpha \Rightarrow^* \lambda. \end{cases}$$
Com efeito, se $\alpha \Rightarrow^* \lambda$ e se $a \in \mathrm{Prim}(\beta)$, de forma que $\beta \Rightarrow^* a\gamma$ para algum $\gamma \in V^*$, vem $\alpha\beta \Rightarrow^* \beta \Rightarrow^* a\gamma$. Nesse caso, também se tem $a \in \mathrm{Prim}(\alpha\beta)$.

Estas propriedades mostram que para calcular $\mathrm{Prim}(\alpha)$ para $\alpha \in V^*$ arbitrário basta conhecer $\mathrm{Prim}(A)$ para todo o $A \in N$. No parágrafo seguinte apresenta-se um algoritmo que determina os primeiros de qualquer símbolo não terminal. Consideremos por exemplo a gramática de produções

$$\begin{aligned} S &\to AB \\ A &\to a \mid \lambda \\ B &\to b \end{aligned}$$

Tem-se $\mathrm{Prim}(A) = \mathrm{Prim}(a) \cup \mathrm{Prim}(\lambda) = \{a\}$, $\mathrm{Prim}(B) = \{b\}$ e, como $A \Rightarrow^* \lambda$, $\mathrm{Prim}(S) = \mathrm{Prim}(AB) = \mathrm{Prim}(A) \cup \mathrm{Prim}(B) = \{a, b\}$. Isto aliás é evidente se observarmos que $L(S) = \{ab, b\}$.

Definição do conjunto dos seguintes

Consideremos a gramática com as seguintes produções:

$$\begin{array}{l} S \to AB \\ A \to a \mid \lambda \\ B \to a \end{array}$$

O único símbolo não terminal com mais de uma produção é A, e tem-se $\mathrm{Prim}(a) \cap \mathrm{Prim}(\lambda) = \{a\} \cap \emptyset = \emptyset$. Tentemos, porém analisar a palavra aa:

PILHA	FITA
S	aa
AB	aa
?	

Se o analisador tiver acesso apenas ao primeiro símbolo da palavra na fita de leitura, do seu ponto de vista ela tanto pode ser aa como a. No primeiro caso, $A \to a$ é a produção que deve ser escolhida para expandir A em AB, enquanto que a produção apropriada ao segundo caso é $A \to \lambda$. Não é pois possível decidir entre as duas acções a tomar, e o tipo de gramáticas que permite que tais situações ocorram tem de ser eliminado para que o analisador funcione como se pretende.

Para vermos o que há de errado com a gramática anterior, suponhamos que na pilha AB "congelávamos" temporariamente a expansão de A. A única expansão possível para B levaria à nova pilha Aa. Vê-se assim que o símbolo a, que é um primeiro de A, também é um "seguinte" de A, isto é, pode ocorrer imediatamente a seguir a A numa palavra que deriva do símbolo inicial S. Assim, quando o primeiro símbolo da fita é a, não se sabe em princípio se ele está ali no papel de primeiro de A ou de seguinte de A, visto que A pode gerar λ.

Definição 7.3 [Seguintes] O conjunto dos *seguintes* de $A \in N$ define-se por

$$\mathrm{Seg}(A) = \{a \in T : (\exists \alpha, \beta \in V^*)\, S \Rightarrow^* \alpha A a \beta\}.$$

Assim, como ficou dito atrás, os seguintes de A são todos os símbolos terminais que podem ocorrer imediatamente a seguir a A em palavras que derivam do símbolo inicial. Em face do exemplo anterior, não é difícil concluir que, quando se têm duas produções distintas $A \to \alpha$ e $A \to \beta$ para um mesmo símbolo não terminal A em que $\alpha \Rightarrow^* \lambda$, deve ter-se

$$\mathrm{Seg}(A) \cap \mathrm{Prim}(\beta) = \emptyset.$$

Para ver que assim é, suponhamos, pelo contrário, que $a \in \mathrm{Seg}(A) \cap \mathrm{Prim}(\beta)$. Consideremos uma situação em que a é o primeiro símbolo da fita de leitura e $A\gamma$ é o conteúdo da pilha. Para a ser reconhecido, tem de ser produzido no topo da pilha, o que em princípio pode ser feito de duas maneiras. Por um lado, a pode ser produzido por A, como em $A\gamma \Rightarrow \beta\gamma \Rightarrow^* a\eta\gamma$, em virtude de ser $a \in \mathrm{Prim}(\beta)$. Por outro lado, a pode ser produzido por γ, devido a ser $a \in \mathrm{Seg}(A)$, vindo $A\gamma \Rightarrow \alpha\gamma \Rightarrow^* \gamma \Rightarrow^* a\rho$. Não se sabe se se deve escolher a expansão α se a β, que é precisamente o não determinismo que se pretende evitar.

As duas condições que acabámos de ver, a que se refere aos primeiros e a que envolve os seguintes, podem ser reunidas numa só condição, que apresentaremos adiante.

Definição de gramática LL(1)

Para integrar as condições dos primeiros e dos seguintes definem-se os símbolos directores de uma produção.

Definição 7.4 [Símbolos directores] O conjunto dos *símbolos directores* de uma produção $A \to \alpha$ define-se do seguinte modo:

$$\mathrm{Dir}(A, \alpha) = \begin{cases} \mathrm{Prim}(\alpha) & \text{se } \alpha \not\Rightarrow^* \lambda, \\ \mathrm{Prim}(\alpha) \cup \mathrm{Seg}(A) & \text{se } \alpha \Rightarrow^* \lambda. \end{cases}$$

Podemos agora definir gramática LL(1).

Definição 7.5 [Gramática LL(1)] Uma gramática G diz-se LL(1) se satisfizer as seguintes condições:

- A gramática está na forma reduzida.
- Para todo o par de produções distintas $A \to \alpha$ e $A \to \beta$ para o mesmo símbolo não terminal,
  - $\mathrm{Dir}(A, \alpha) \cap \mathrm{Dir}(A, \beta) = \emptyset$;
  - não se pode ter $\alpha \Rightarrow^* \lambda$ e $\beta \Rightarrow^* \lambda$.

A primeira destas condições é a reunião numa só das condições dos primeiros e dos seguintes. A segunda condição merece alguns comentários, para o que necessitamos de uma definição preliminar.

Definição 7.6 [Recursividade esquerda] Uma gramática diz-se *recursiva* se existir um símbolo não terminal A tal que $A \Rightarrow^+ \alpha A\beta$ para alguns $\alpha, \beta \in V^*$, e *recursiva à esquerda* se $A \to^+ A\beta$ para algum $\beta \in V^*$.

Uma gramática recursiva não é necessariamente indesejável. Bem pelo contrário, pode demonstrar-se que uma gramática só gera uma linguagem infinita (como em geral se pretende) se for recursiva. Já uma gramática recursiva à esquerda é indesejável do ponto de vista da análise sintáctica descendente, porque pode originar computações infinitas. Suponhamos que numa dada fase da análise de uma certa palavra se chegava à situação

PILHA	FITA
$A\alpha$	x

e que a gramática dada era recursiva à esquerda em A. Se se empregassem sucessivamente expansões que levassem de A a $A\beta$ chegava-se à situação

PILHA	FITA
$A\beta\alpha$	x

e o processo repetia-se indefinidamente, não havendo pois possibilidade de o algoritmo terminar.

Ora a primeira das condições definidoras das gramáticas LL(1) não garante só por si que a gramática não seja recursiva à esquerda. Consideremos, por exemplo, a gramática com as seguintes produções:

$$\begin{aligned} S &\to A \mid B \\ A &\to B \mid \lambda \\ B &\to A \mid \lambda \end{aligned}$$

Tem-se $L(S) = L(A) = L(B) = \{\lambda\}$, por conseguinte os conjuntos directores são todos vazios, e a primeira condição verifica-se trivialmente. A gramática é, porém, recursiva à esquerda, visto que $A \Rightarrow B \Rightarrow A$. Logo, a primeira condição só por si não elimina a possibilidade de existência de recursividade à esquerda. Note-se, contudo, que a gramática em exemplo não é LL(1), visto não satisfazer a segunda condição. Pode aliás demonstrar-se o seguinte resultado.

Proposição 7.7 *Uma gramática LL(1) não é recursiva à esquerda.*

Dem. Seja G uma gramática LL(1) (na forma reduzida, recorde-se!) e comecemos por admitir que G é recursiva à esquerda, para em seguida chegar a uma contradição. Seja A recursivo à esquerda, e consideremos a árvore de uma derivação $A \Rightarrow^+ A\beta$. Todos os nós da raiz A à folha A são etiquetados por símbolos recursivos à esquerda. Com efeito, seja B a etiqueta de um nó nesse caminho, e tomemos a sub-árvore enraizada nesse nó. Se na folha

A da sub-árvore dependurarmos uma cópia da árvore inicial, obtemos um caminho da raiz B a um nó B, o que mostra que B é recursivo; como os irmãos esquerdos de B geram a palavra vazia, B é recursivo à esquerda. Se todos esses símbolos recursivos à esquerda tivessem apenas uma produção na gramática, essa produção teria outro símbolo recursivo à esquerda no corpo, logo os símbolos não seriam produtivos, contrariando a suposição de que G está na forma reduzida. Assim, há pelo menos um símbolo não terminal recursivo à esquerda com mais de uma produção, e podemos supor, sem perda de generalidade, que esse símbolo é A.

Uma vez que, por hipótese, $A \Rightarrow^+ A\beta$, existe uma produção $A \to \alpha$ tal que $A \Rightarrow \alpha \Rightarrow^* A\beta$. Como existe outra produção $A \to \sigma$ para A, podemos escrever

$$A \Rightarrow \alpha \Rightarrow^* A\beta \Rightarrow \sigma\beta.$$

Assim, vê-se que $\mathrm{Prim}(\sigma) \subseteq \mathrm{Prim}(\alpha)$. Se $\mathrm{Prim}(\sigma) \neq \emptyset$, segue-se que $\mathrm{Dir}(A,\alpha) \cap \mathrm{Dir}(A,\sigma) \neq \emptyset$, contrariamente à hipótese de G ser LL(1). Se $\mathrm{Prim}(\sigma) = \emptyset$, como G é reduzida tem-se $L(\sigma) = \{\lambda\}$, donde em particular $\sigma \Rightarrow^* \lambda$. Não se pode ter $\mathrm{Prim}(\beta) = \emptyset$, se não $\beta \Rightarrow^* \lambda$ pela mesma razão anterior e portanto $\alpha \Rightarrow^* \sigma\beta \Rightarrow^* \lambda$, contrariando a segunda condição da definição de gramática LL(1). Logo, $\mathrm{Prim}(\beta) \neq \emptyset$. A derivação anterior mostra que $\mathrm{Prim}(\beta) \subseteq \mathrm{Seg}(A) \subseteq \mathrm{Dir}(A,\sigma)$ e $\mathrm{Prim}(\beta) \subseteq \mathrm{Prim}(\alpha) \subseteq \mathrm{Dir}(A,\alpha)$, por conseguinte $\mathrm{Dir}(A,\alpha) \cap \mathrm{Dir}(A,\sigma) \neq \emptyset$. Uma vez mais se contradiz a hipótese de que a gramática dada é LL(1), e fica terminada a demonstração. ∎

Voltando à segunda condição da definição de gramática LL(1), se ela é indispensável, como se viu, frequentemente ela é uma consequência da primeira condição. Na prática, há uma maneira de garantir que isso se passa sempre assim, adicionando à gramática um símbolo "terminador" \$.

Proposição 7.8 *Seja $G = (T, N, S, P)$ e defina-se $G' = (T', N', S', P')$ pondo $T' = T \cup \{\$\}$, $N' = N \cup \{S'\}$ e $P' = P \cup \{S' \to S\$\}$, onde \$ e S' são símbolos distintos não existentes em V. Tem-se $L(G') = L(G)\{\$\}$ e, se G' satisfizer a primeira condição da definição de gramáticas LL(1), também satisfaz a segunda.*

Dem. É evidente que $L(G') = L(G)\{\$\}$, visto que, como é fácil reconhecer, $S \Rightarrow^* \alpha$ se e só se $S' \Rightarrow^* \alpha\$$. Uma consequência deste facto é que se $S' \Rightarrow^* \sigma A\theta$ então θ é terminada por \$, logo $\mathrm{Seg}(G', A)$ (os seguintes de A em G') é não vazio. Assim, se $A \to \alpha$ e $A \to \beta$ forem produções tais que $\alpha \Rightarrow^* \lambda$ e $\beta \Rightarrow^* \lambda$, os seus conjuntos directores em G' contêm ambos $\mathrm{Seg}(G', A) \neq \emptyset$. Como G' satisfaz a primeira condição das gramáticas LL(1), tem de ser $\alpha = \beta$, logo a segunda ccondição também fica satisfeita. ∎

Com as notações da proposição anterior, G é LL(1) se e só se G' for LL(1). Esta é a razão pela qual é tão útil acrescentar a cada palavra gerada por G um terminador especial \$.

7.2 Verificação das condições LL(1)

Nesta secção vamos apresentar algoritmos para a determinação dos primeiros e seguintes de símbolos não terminais. Mostramos também como se pode eliminar a recursividade à esquerda de uma gramática num caso simples.

Determinação dos primeiros

O algoritmo consiste em definir um grafo apropriado e obter os primeiros directamente a partir do grafo.

Definição 7.9 [Grafo dos primeiros] O *grafo dos primeiros* define-se do seguinte modo:

- Os vértices são os símbolos de V.
- Para cada produção $A \to X_1 X_2 \cdots X_n$ com $n > 0$, há uma arco de A para X_1; se $X_1 \Rightarrow^* \lambda$, há também um arco de A para X_2, e assim sucessivamente.

Note-se que só partem arcos de símbolos não terminais, e nunca de símbolos terminais. Havendo um arco de A para X, a propriedade crucial é que $A \Rightarrow^* X\beta$ para algum β, como é fácil de verificar directamente a partir da definição de arco. Esta propriedade mantem-se válida ao longo de um caminho. Assim, se existir um arco de X para Y, tem-se $X \Rightarrow^* Y\gamma$ para algum γ, logo $A \Rightarrow^* Y\gamma\beta$. O raciocínio pode ser repetido para caminhos de comprimento qualquer. Em particular, se existir um caminho de A para um símbolo terminal a então $A \Rightarrow^* a\alpha$ para algum α, por conseguinte $a \in \mathrm{Prim}(A)$.

Inversamente, se $a \in \mathrm{Prim}(A)$, isto é, se $A \Rightarrow^* a\alpha$ para algum α, na árvore de derivação correspondente existe um caminho da raiz A para a folha a, e todas as folhas à esquerda dessa têm etiqueta λ. Dois nós consecutivos X e Y nesse caminho na árvore estão relacionados por uma produção $X \to X_1 \cdots X_k$, em que Y é um certo X_j. Mas então os símbolos X_i com $i < j$ geram a palavra vazia, logo no grafo dos primeiros existe um arco de X para Y. Conclui-se que existe um caminho de A para a nesse grafo. Ficou assim mostrado:

Proposição 7.10 *Para todos os símbolos não terminal A e terminal a, tem-se $a \in \mathrm{Prim}(A)$ se e só se existe um caminho de A para a no grafo dos primeiros.*

Esta propriedade mostra que $\mathrm{Prim}(A)$ se calcula determinando todos os descendentes terminais de A no grafo dos primeiros. Consideremos, por exemplo, a gramática de produções

$$\begin{array}{l} S \to E\$ \\ E \to TU \\ U \to +TU \mid \lambda \\ T \to FG \\ G \to *FG \mid \lambda \\ F \to (E) \mid a \end{array}$$

Os símbolos que geram a palavra vazia são U e G. Os nós do grafo dos primeiros são $S, E, U, T, G, F, \$, +, *, (,), a$. Os arcos podem ser descritos indicando os sucessores de cada nó (necessariamente um símbolo não terminal). Na tabela seguinte mostram-se os sucessores e os primeiros dos símbolos não terminais.

Nós	Sucessores	Primeiros
S	E	$($, a
E	T	$($, a
U	$+$	$+$
T	F	$($, a
G	$*$	$*$
F	$($, a	$($, a

Determinação dos seguintes

O algoritmo para determinar os seguintes é uma consequência das duas propriedades enunciadas a seguir. Relembre-se que estamos a supor que a gramática em consideração está na forma reduzida.

Propriedades 7.11 *Seja $A \to \alpha B \beta$ uma produção. Tem-se:*

i) $\mathrm{Prim}(\beta) \subseteq \mathrm{Seg}(B)$.
Com efeito, como A é acessível tem-se $S \Rightarrow^* \sigma A \theta \Rightarrow \sigma \alpha B \beta \theta$ para alguns σ e θ. Se $a \in \mathrm{Prim}(\beta)$, tem-se $\beta \Rightarrow^* a\gamma$ para algum γ. Mas então $S \Rightarrow^* \sigma \alpha B a \gamma \theta$, donde $a \in \mathrm{Seg}(B)$.

ii) *Se* $\beta \Rightarrow^* \lambda$ *então* $\text{Seg}(A) \subseteq \text{Seg}(B)$.
Se $a \in \text{Seg}(A)$ tem-se $S \Rightarrow^* \sigma A a \theta \Rightarrow \sigma\alpha B\beta a\theta \Rightarrow^* \sigma\alpha B a\theta$ para alguns σ e θ, logo $a \in \text{Seg}(B)$.

A determinação dos seguintes também se baseia na construção de um grafo.

Definição 7.12 [Grafo dos seguintes] O *grafo dos seguintes* define-se por:

- Os vértices são os símbolos de V.
- Para cada produção $A \to \alpha B\beta$ com um símbolo não terminal B no corpo,
  - existe um arco de B para cada $b \in \text{Prim}(\beta)$;
  - se $\beta \Rightarrow^* \lambda$, existe também um arco de B para A.

Tal como no caso dos primeiros, não é difícil concluir que se tem a seguinte propriedade.

Proposição 7.13 *Para todos os símbolos não terminal A e terminal a, tem-se $a \in \text{Seg}(A)$ se e só se existe um caminho de A para a no grafo dos seguintes.*

Deste modo, $\text{Seg}(A)$ calcula-se determinando todos os descendentes terminais de A no grafo dos seguintes. Para a gramática do exemplo anterior, mostram-se na tabela seguinte os sucessores no grafo dos seguintes e os seguintes dos símbolos não terminais.

Nós	Sucessores	Seguintes
S		
E	\$,)	\$,)
U	E, U	\$,)
T	$+, E, U$	\$, +,)
G	T, G	\$, +,)
F	$*, T, G$	\$, $*$, +,)

Eliminação da recursividade á esquerda

Quando uma gramática é recursiva à esquerda, já se sabe que não é LL(1). Vamos apresentar um procedimento que permite transformar uma gramática recursiva à esquerda numa que não o é. Este procedimento não garante, evidentemente, que a gramática obtida seja LL(1), mas é aplicável num grande número de situações. Vamos limitar-nos a "remover" a recursividade à esquerda que surge de forma "directa" em produções do tipo $A \to A\alpha$.

Proposição 7.14 *Seja G uma gramática cujas produções para um símbolo não terminal A são*

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \cdots \mid \beta_k.$$

Seja G' a gramática que se obtem substituindo estas produções por

$$\begin{array}{l} A \to \beta_1 B \mid \cdots \mid \beta_k B \\ B \to \lambda \mid \alpha_1 B \mid \cdots \mid \alpha_n B \end{array}$$

em que B é um novo símbolo não terminal não existente em V. Nestas condições, tem-se $L(G') = L(G)$.

Dem. Não é difícil ver como toda a árvore de derivação em G de raiz etiquetada por A pode ser transformada numa árvore em G' com a mesma fronteira, bem como a transformação inversa. ∎

Aplicando o procedimento descrito nesta proposição até não existirem produções do tipo indicado, elimina-se a recursividade à esquerda directa. Por exemplo, a gramática de produções

$$\begin{array}{l} S \to E\$ \\ E \to E + T \mid T \\ T \to T * F \mid F \\ F \to (E) \mid a \end{array}$$

fica transformada em

$$\begin{array}{l} S \to E\$ \\ E \to TU \\ U \to +TU \mid \lambda \\ T \to FG \\ G \to *FG \mid \lambda \\ F \to (E) \mid a \end{array}$$

para os dois novos símbolos não terminais U e G. Para esta gramática já foram calculados os conjuntos de primeiros e seguintes de cada símbolo não terminal. Podemos então calcular os conjuntos directores de cada produção e verificar se a gramática é ou não LL(1).

Definição 7.15 [Tabela de análise sintáctica] Os conjuntos directores podem ser representados pela chamada *tabela de análise sintáctica* que, para uma gramática $G = (T, N, S, P)$, é uma tabela de linhas indexadas por N, colunas indexadas por T, e casas preenchidas por palavras de V^* ou vazias. A palavra α ocupa a casa da linha A e coluna a se e só se $a \in \mathrm{Dir}(A, \alpha)$.

A gramática é LL(1) se e só se uma mesma casa não for ocupada por duas ou mais palavras diferentes. As casas vazias constituem situações de erro, a serem detectadas na análise sintáctica. A tabela para a gramática anterior é a seguinte:

	$($	a	$+$	$*$	$)$	\$
S	$E\$$	$E\$$				
E	TU	TU				
U			$+TU$		λ	λ
T	FG	FG				
G			λ	$*FG$	λ	λ
F	(E)	a				

Verifica-se que a gramática dada é LL(1).

7.3 Reconhecimento de gramáticas LL(1)

Seja $G = (T, N, S, P)$ uma gramática LL(1). O autómato de pilha que reconhece $L(G)$ tem a forma

$$A = (T, Q, Z, q_I, z_I, \delta, F)$$

em que:

- $Q = \{q_L\} \cup \{q_a : a \in T\}$.
- $Z = V$.
- $q_I = q_L$.
- $z_I = S$.
- A função de transição δ é definida mais abaixo.
- $F = \emptyset$.

A função de transição de A é descrita pelas seguintes regras:
Regras de leitura

$$(q_L, z) \xrightarrow{a} (q_a, z)$$

para todo o estado $z \in Z$ e todo o $a \in T$.
Regras de reconhecimento

$$(q_a, a) \xrightarrow{\lambda} (q_L, \lambda)$$

para todo o $a \in T$.
Regras de expansão

$$(q_a, A) \xrightarrow{\lambda} (q_a, \alpha)$$

para todo o triplo (a, A, α) tal que $a \in \mathrm{Dir}(A, \alpha)$.

Este autómato é evidentemente determinista. Pretende-se a linguagem $N(A)$ reconhecida pelo critério da pilha vazia. Tem-se o seguinte resultado:

Teorema 7.16 *Com as notações anteriores, $L(G) = N(A)$.*

Dem. A COMPLETAR. ∎

Como exemplo, consideremos a gramática de produções

$$\begin{array}{l} S \to E\$ \\ E \to TU \\ U \to +TU \mid \lambda \\ T \to FG \\ G \to *FG \mid \lambda \\ F \to (E) \mid a \end{array}$$

que já sabemos ser LL(1). As regras de leitura e reconhecimento são análogas para todos os autómatos. Mostram-se a seguir as regras de expansão para este exemplo. Para simplificar a leitura, cada regra $(q_a, A) \xrightarrow{\lambda} (q_a, \alpha)$ é escrita na forma $a, A \to a, \alpha$, omitindo-se a etiqueta λ da seta bem como os parênteses nos pares à sua esquerda e à sua direita, e representando os estados q_a apenas por a.

$$\begin{array}{llllll} (, & S & \to & (, & E\$ \\ a, & S & \to & a, & E\$ \\ (, & E & \to & (, & TU \\ a, & E & \to & a, & TU \\ +, & U & \to & +, & +TU \\ ), & U & \to &), & \lambda \\ \$, & U & \to & \$, & \lambda \end{array} \qquad \begin{array}{llllll} (, & T & \to & (, & FG \\ a, & T & \to & a, & FG \\ +, & G & \to & +, & \lambda \\ *, & G & \to & *, & *FG \\ ), & G & \to &), & \lambda \\ \$, & G & \to & \$, & \lambda \\ (, & F & \to & (, & (E) \\ a, & F & \to & a, & a \end{array}$$

Como se observa, as regras obtêm-se directamente a partir da tabela da análise sintáctica, a qual contém portanto toda a informação útil para a análise sintáctica.