

# Analisadores

## léxicos e sintáticos

Prof. Leônidas de Oliveira Brandão  
Departamento de Ciência da Computação - IME - USP  
<http://www.ime.usp.br/~leo>

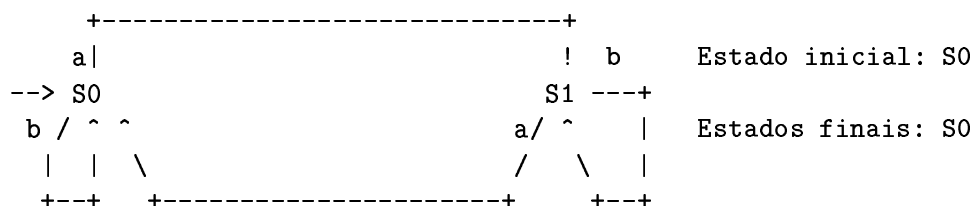
### 1.1 Analisadores léxicos e sintáticos

#### 1.1.1 Analisador léxico

O objetivo da análise léxica é reconhecer, dentre as cadeias de caracteres de um alfabeto, aquelas que são válidas dentro de uma linguagem. Em uma linguagem computacional, os caracteres são agrupados em **itens léxicos**, que devem ser reconhecidos/recuperados durante a análise léxica.

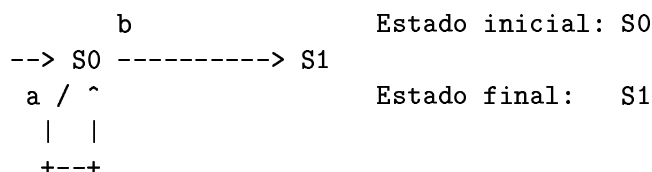
Uma maneira de fazer a análise léxica é construir uma **autômato de estados finitos** para reconhecer as cadeias aceitas pela linguagem.

**Exemplo 1.1** Reconhecer a linguagem  $L$ , formada por 'a' e 'b', com um número par de a's.



Em um autômato, existe o alfabeto  $A$ , o conjunto  $E$  dos estados do autômato (S0 e S1 no exemplo acima) e a função transição ( $\phi(E, c) : E \times A \rightarrow E$ ). No exemplo anterior, no estado S0, inicial e final, ao ler um caractere 'a' deve-se transitar para o estado S1, portanto  $\phi(S0, 'a') = S1$ .

**Exemplo 1.2** Um autômato que reconhecer a linguagem  $L = \{b, ab, aab, aaab, \dots\} = \{a^*b\}$ .



#### 1.1.2 Analisador sintático

A função da análise sintática é verificar a sintaxe de um código fonte escrito em determinada linguagem. Por exemplo, em **Java**, deve ser anotado como errado um código que apareça "if a==b ...".

A sintaxe pode ser formalmente descrita através de uma gramática, e suas **regras de produção**. Uma gramática contém símbolos terminais ( $T$ ) e não terminais ( $N$ ), sendo  $T \cup N$  o alfabeto da gramática. As cadeias de caracteres são formadas apenas por terminais.

Um regra de produção é do tipo:  $X \rightarrow \alpha$ ,  $X \in N$  e  $\alpha \in (T \cup N)^*$ . Os terminais são representados em letras minúsculas e os não terminais em maiúsculas.

**Exemplo 1.3** A gramática com alfabeto  $\{S, A, B, a, b, c, d\}$ , e regra de produção inicial  $S$ , é definida pelas regras dadas abaixo.

Regras de Produção	Diagramas sintáticos
$S \rightarrow aB$	$S \rightarrow a \rightarrow B \rightarrow  $
$A \rightarrow c$	
$A \rightarrow d$	$A \rightarrow c \rightarrow  $
$B \rightarrow bA$	$  \quad \wedge$ $\rightarrow d \rightarrow$
	$B \rightarrow b \rightarrow A \rightarrow  $

A gramática acima gera/reconhece a linguagem  $\{abc, abd\}$ .

Para usarmos na prática uma gramática é necessário que não existam **ambiguidades**. Para decidirmos se uma gramática é ou não ambigua precisamos definir árvores sintáticas. Vejamos um exemplo,

**Exemplo 1.4** A gramática definida pelas regras  $E \rightarrow E + E | E * E | N$ , com  $N$  um não terminal para reconhecer constantes e identificadores, é ambígua, pois a cadeia válida  $4 + 5 * 2$  admite duas árvores distintas (uma produzindo como resultado 18 e a outra 14).

--E	E--
/   \	/   \
E   E	E   E
/   \	/   \
E   E	E   E
N   N   N	N   N   N
4 + 5 * 2 = 18	4 + 5 * 2 = 14

Um gramática é não ambigua se, e somente se, não existirem duas ou mais cadeias válidas da linguagem com duas ou mais **árvores sintáticas**.

Uma gramática é **regular** se suas regras de produção forem do tipo  $N \rightarrow tN$ . Isso implica que as árvores sintáticas de uma gramática regular são do tipo binária.

**Exemplo 1.5** Uma gramática regular para produzir expressões aritméticas pode ser:  $E \rightarrow tA|t$ ;  $A \rightarrow +E | -E | *E | /E | E$ .

Entretanto esta gramática não consegue representar a expressão  $4 + 5 * 2$  resultando 18. A gramática abaixo resolve isso e não é ambigua.

**Exemplo 1.6** Uma gramática regular para produzir expressões aritméticas pode ser:

$E \rightarrow A \mid E + A \mid E - A$   
 $A \rightarrow B \mid A * B \mid A / B$   
 $B \rightarrow t \mid B ^ t$

### 1.1.3 Referências

1. Valdemar W. Setzer e Inês S. Homen de Melo, *A construção de um compilador*, ed. Campus Ltda, RJ, 1983.
2. Tomasz Kowaltowski, *Implementação de linguagens de programação*, editora Guanabara Dois S.A., RJ, 1983.