Compiladores
(IF688)

Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

Estendendo para identificadores e números maiores que 9

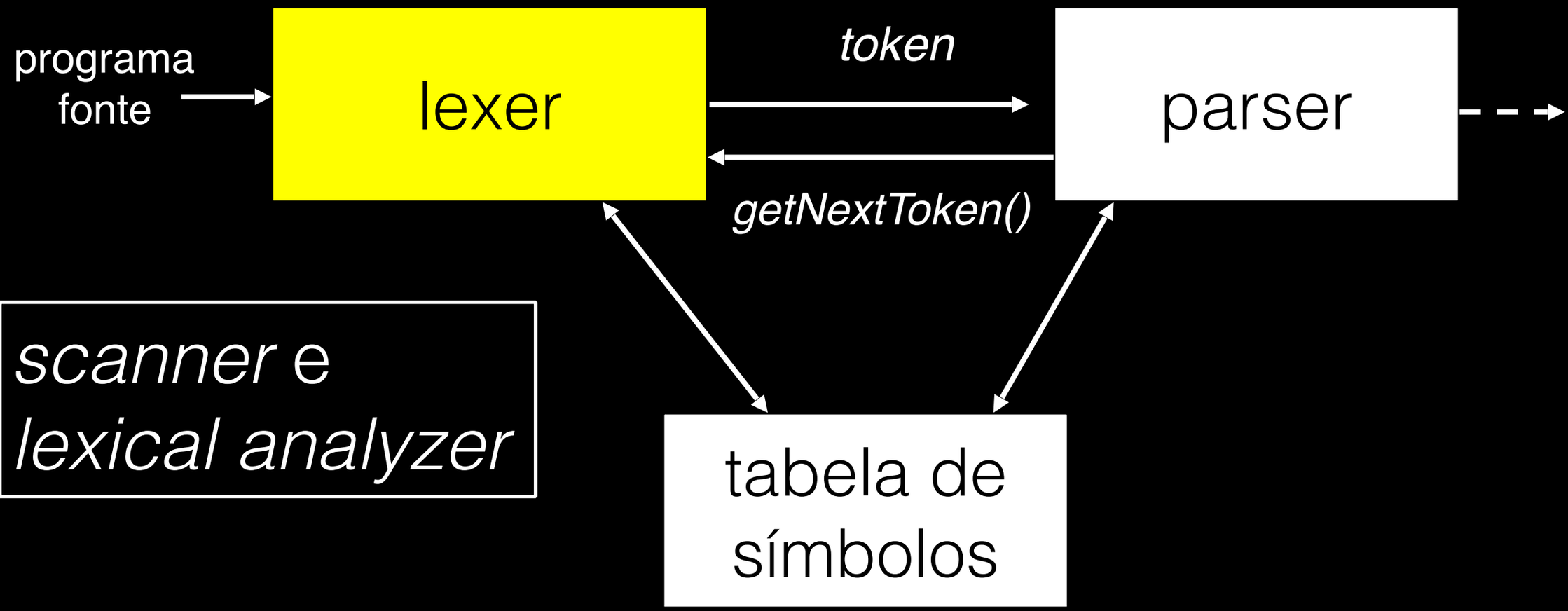
expr →	*expr* + *term*	{print (‘+’) }
expr →	*expr* – *term*	{print (‘-’) }
expr →	*term*	
term →	*term* * *factor*	{print (‘*’) }
term →	*term* / *factor*	{print (‘/’) }
term →	*factor*	
factor →	**(** *expr* **)**	
factor →	**num**	{print (**num**.*value*) }
factor →	**id**	{print (**id**.*lexeme*) }

Reescrevendo…

expr → term rest
rest → + term {print (‘+’) } rest
rest → - term {print (‘-’) } rest
rest → ε
term → factor resto
resto → * factor {print (‘*’) } resto
resto → / factor {print (‘/’) } resto
resto → ε
factor → 0 {print (‘0’) }
…
factor → 9 {print (‘9’) }

Antes de traduzir um programa, compiladores precisam entender a sua estrutura e significado.

Analisador léxico



scanner e
lexical analyzer

Construindo um analisador léxico

- Especificar tokens e lexemas da linguagem
- Implementar (ou gerar através de ferramenta) o analisador a partir da especificação

Quais as razões para separarmos análise léxica de sintática?

Lexer vs. Parser

- Design mais simples – misturar análise léxica com sintática torna o parser bem mais complicado

- Eficiência – separação possibilita construir processadores léxicos e sintáticos mais eficientes.

- Portabilidade – variações de dispositivos podem ficar restritas ao analisador léxico.

Tokens, Padrões e Lexemas

- Token: par com o nome do token e atributos opcionais
- Padrão: descrição dos possíveis lexemas associados a um tipo de token
- Lexema: sequência de caracteres que casam com o padrão de um tipo de token.

Exemplo

- Para a entrada: “var1 = 23”;
  - Primeiro **token** da entrada: <ID, “var1”>
  - Para este token
    - **Padrão**: “seq. de caracteres seguida por dígito”
    - ID (para identificador) é o tipo do token
    - **Lexema** “var1”

Observações

- Existem conjuntos de strings na entrada que geram o mesmo tipo de token
  - Exemplo: var1, xyz2 são tokens do mesmo tipo ID
- Símbolos terminais de uma gramática correspondem a tokens
  - Exemplo: Palavras reservadas, operadores, identificadores, constantes, parenteses, etc.

Exemplos de Tokens

TOKEN	Descrição Informal	Ex.: Lexemas
`if`	caracteres i, f	`if`
`else`	caracteres e,l,s,e	`else`
`comparison`	<, >, <=, >=, ==, !=	`<=, !=`
`id`	letra seguida de letras e dígitos	`pi, score, D2`
`number`	qualquer constante numérica	`3.14159, 0, 6.02e23`
`literal`	qualquer coisa exceto “, envolvida por aspas	`“exemplo"`

Atributos de um token

- Ao reconhecer um token **num** é relevante saber se seu valor é zero ou um, por exemplo.
- Geralmente associado a um token existe um atributo.
- Uso da tabela de símbolos para guardar informações auxiliares sobre tokens.

Exemplo

```
E = M * C ** 2
```

Exemplo

```
E = M * C ** 2
```

Exemplo

E	=	M	*	C	**	2

```
<id, apontador p/posição de E na tabela de símbolos>
<assign_op,>
<id, apontador p/posição de M na tabela de símbolos>
<mult_op,>
<id, apontador p/posição de C na tabela de símbolos>
<exp_op,>
<num, valor 2>
```

Capturando erros com análise léxica

- É difícil para um analisador léxico determinar, sem a ajuda de outros componentes, que há um erro no código-fonte
- Por exemplo:
  - **`fi (a == f(x))`** …
- Qual o token que deve ser retornado?

Especificação de Tokens

- Baseada em Teoria de Linguagens:
  - Alfabetos, strings e linguagens
- Operações sobre linguagens
- Seja L o conjunto {A,B,…,Z,a,b,…,z} e D o conjunto {0,1,…,9}.

Expressões Regulares

Expressões

- Na matemática, podemos usar os operadores para construir expressões a serem avaliadas, como:

(5+3)*4

Estas expressões retornam um resultado numérico.

Operações Regulares

- Em linguagens formais podemos utilizar as operações regulares para criar expressões envolvendo linguagens:

(0+1)0*

- Operações: união, concatenação, fecho reflexivo.

- São chamadas de expressões regulares.

Expressão Regular

- Formalismo denotacional
- Definida a partir de conjuntos (linguagens) básicos, concatenação e união.
- Adequadas para a comunicação
  - humano × humano
  - humano × máquina
- Importantes em diversas aplicações onde é necessário encontrar padrões em textos.

Exemplos

- $0^*10^* = ?$
- $(0+1)^*1(0+1)^* = ?$
- $(0+1)^*001(0+1)^* = ?$

Exemplos

- 0*10* =
  {w | w contém um único 1}

- (0+1)*1(0+1)* =
  {w | w tem pelo menos um 1}

- (0+1)*001(0+1)*
  {w | w contém a sub-sentença 001}

Mais Exemplos

ER	Linguagem Gerada
aa	
ba*	
(a+b)*	
(a+b)*aa(a+b)*	
a*ba*ba*	
(a+b)*(aa+bb)	
(a+ε)(b+ba)*	

Mais Exemplos

ER	Linguagem Gerada
aa	somente a palavra aa
ba*	todas as palavras que iniciam por b, seguido por zero ou mais a
(a+b)*	todas as palavras sobre { a, b }
(a+b)*aa(a+b)*	todas as palavras contendo aa como subpalavra
a*ba*ba*	todas as palavras contendo exatamente dois b
(a+b)*(aa+bb)	todas as palavras que terminam com aa ou bb
(a+ε)(b+ba)*	todas as palavras que não possuem dois a consecutivos

Definições Regulares

- Por conveniência de notação, nomeamos algumas expressões regulares
- Estes nomes são usados como símbolos em expressões subsequentes

Identificadores em C

letter → A | B | ... | Z | a | b | ...| z

digit → 0 | 1 | ... | 9

id → ***letter*** (***letter*** | ***digit***)*

Generalizando…
Em um alfabeto Σ

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\ldots$$
$$d_n \rightarrow r_n$$

onde:

1. cada d_i é um novo símbolo, que não está em Σ e é diferente dos demais d
2. cada r_i é uma expressão regular sobre $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1},\}$

Numerais

digit → 0 | 1 | … | 9

digits → ***digit digit****

optionalFraction → **.** ***digits*** | ε

optionalExponent → (E (+ | - | ε) ***digits***) | ε

number → ***digits optionalFraction optionalExponent***

Operador +

- Um outro operador regular comumente utilizado é o + que significa:

  $r^+ = rr^*$

- ou seja, enquanto r* representa toda sentença formada por 0 ou mais concatenações, r^+ representa sentenças formada por pelo menos 1 concatenação de r.

Operador ?

- Outro operador regular comumente utilizado é o ? que significa:

  r? = (r+ε)

- ou seja, a linguagem formada por 0 ou 1 concatenação sucessiva de r.

Classes de Caracteres

- Uma expressão regular $a_1|a_2|\ldots|a_n$ pode ser substituída pela abreviação $[a_1a_2a_n]$
- Se $a_1,a_2,\ldots,a_n$ formam uma sequência, podemos substituir por a_1-a_n
  - **`[abc] = a|b|c`**
  - **`[a-z] = a|b|c|…|z`**

Reescrevendo...

$$\textit{letter} \rightarrow [\text{A-Za-z\_}]$$

$$\textit{digit} \rightarrow [\text{0-9}]$$

$$\textit{id} \rightarrow \textit{letter}\ (\textit{letter} \mid \textit{digit})^*$$

Reescrevendo...

$$digit \rightarrow [0\text{-}9]$$

$$digits \rightarrow digit^{+}$$

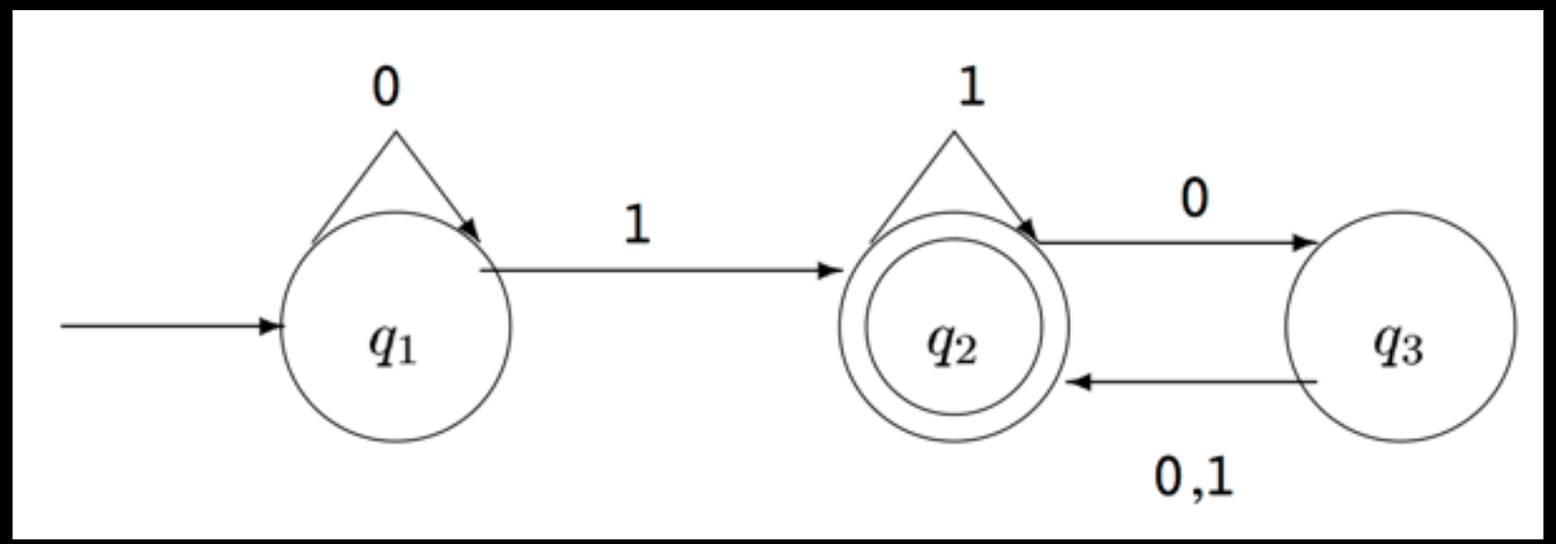
$$number \rightarrow digits\ (.\ digits)?\ (\mathbf{E}\ [+\text{-}]?\ digits)?$$

Como reconhecer os tokens?

Reconhecimento de Tokens

- Gerar diagramas de transição e depois implementar uma máquina de estados.
- Autômatos finitos determinísticos e não-determinísticos.

AFD



$M_1 = (\Sigma, Q, \delta, q_0, F)$, onde

$\Sigma = \{0, 1\}$

$Q = \{q_1, q_2, q_3\}$

$q_0 = q_1$

$F = \{q_3\}$

$\delta =$

	0	**1**
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

AFD

$M_1 = (\Sigma, Q, \delta_1, q_0, F)$

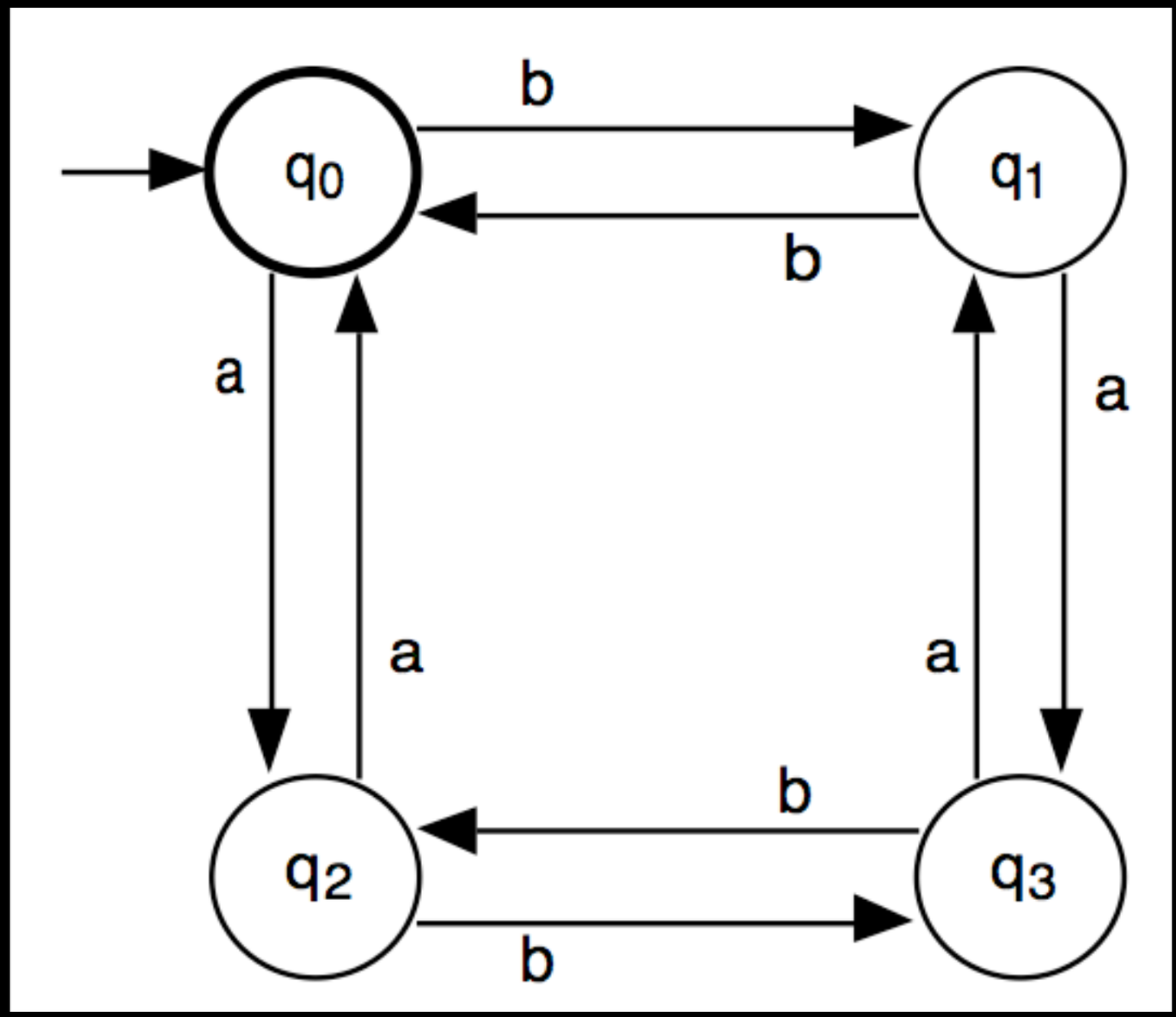
$\Sigma = \{a, b\}$

$Q = \{q_0, q_1, q_2, q_3\}$

$F = \{q_0\}$

$\delta_1 =$

	a	**b**
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

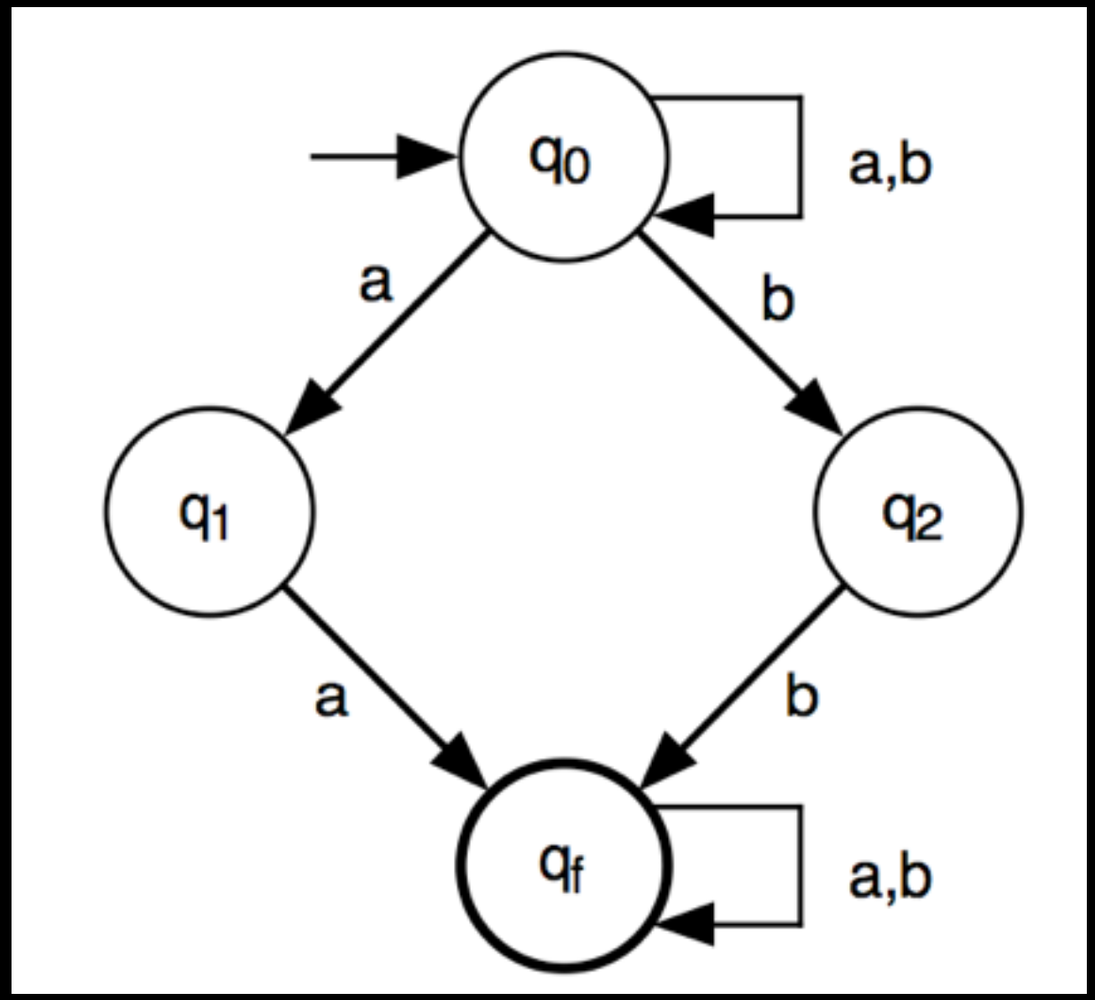


AFN

$M_2 = (\{a,b\}, \{q_0,q_1,q_2,q_f\}, \delta_2, q_0, \{q_f\})$

$\delta_2 =$

	a	**b**
q_0	$\{q_0,q_1\}$	$\{q_0,q_2\}$
q_1	$\{q_f\}$	-
q_2	-	$\{q_f\}$
q_f	$\{q_f\}$	$\{q_f\}$

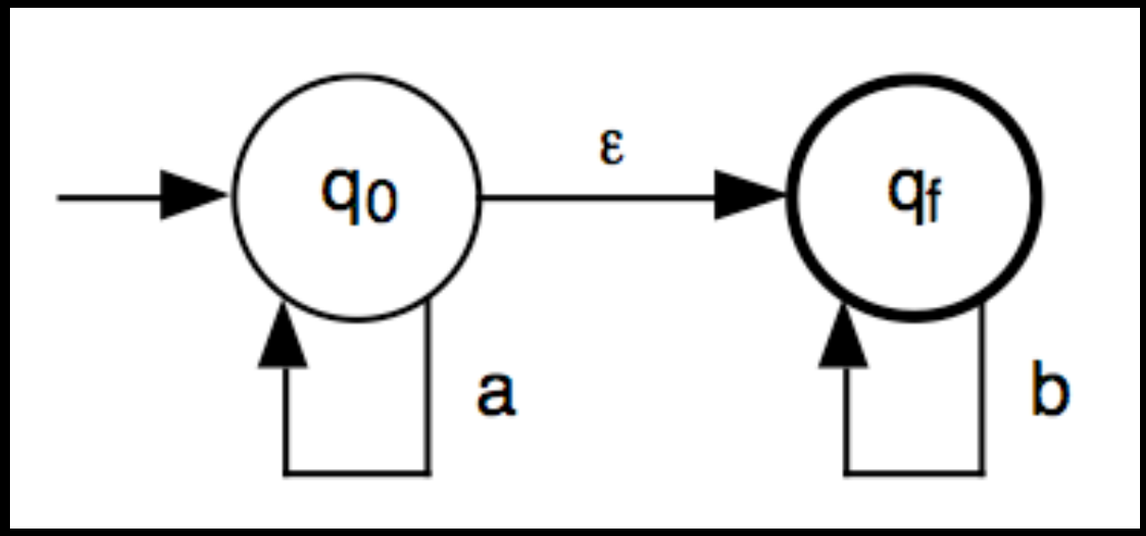


AFNε

$M_3 = (\{a,b\}, \{q_0,q_f\}, \delta_3, q_0, \{q_f\})$

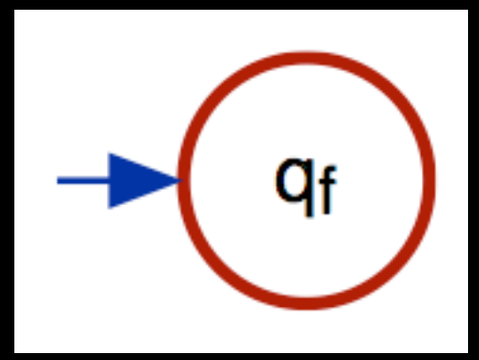
$\delta_3 =$

	a	b	ε
q_0	$\{q_0\}$	-	$\{q_f\}$
q_f	-	$\{q_f\}$	$\{q_f\}$



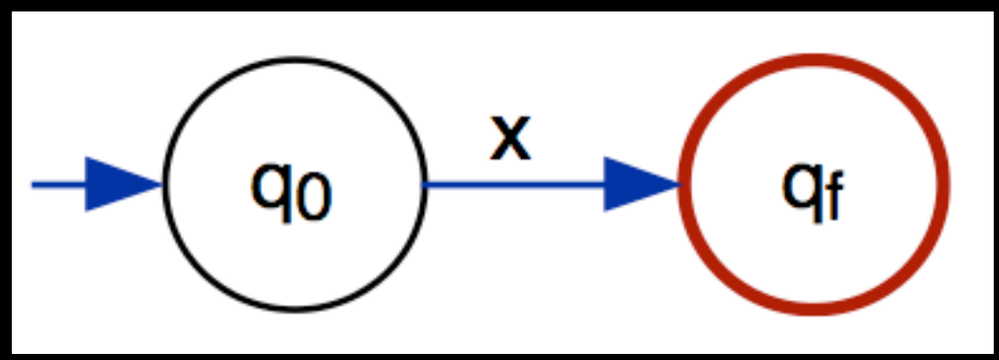
ERs como autômatos

r=ε,



$M_1=(\varnothing,\{q_f\},\delta_1,q_f,\{q_f\})$

r = x,



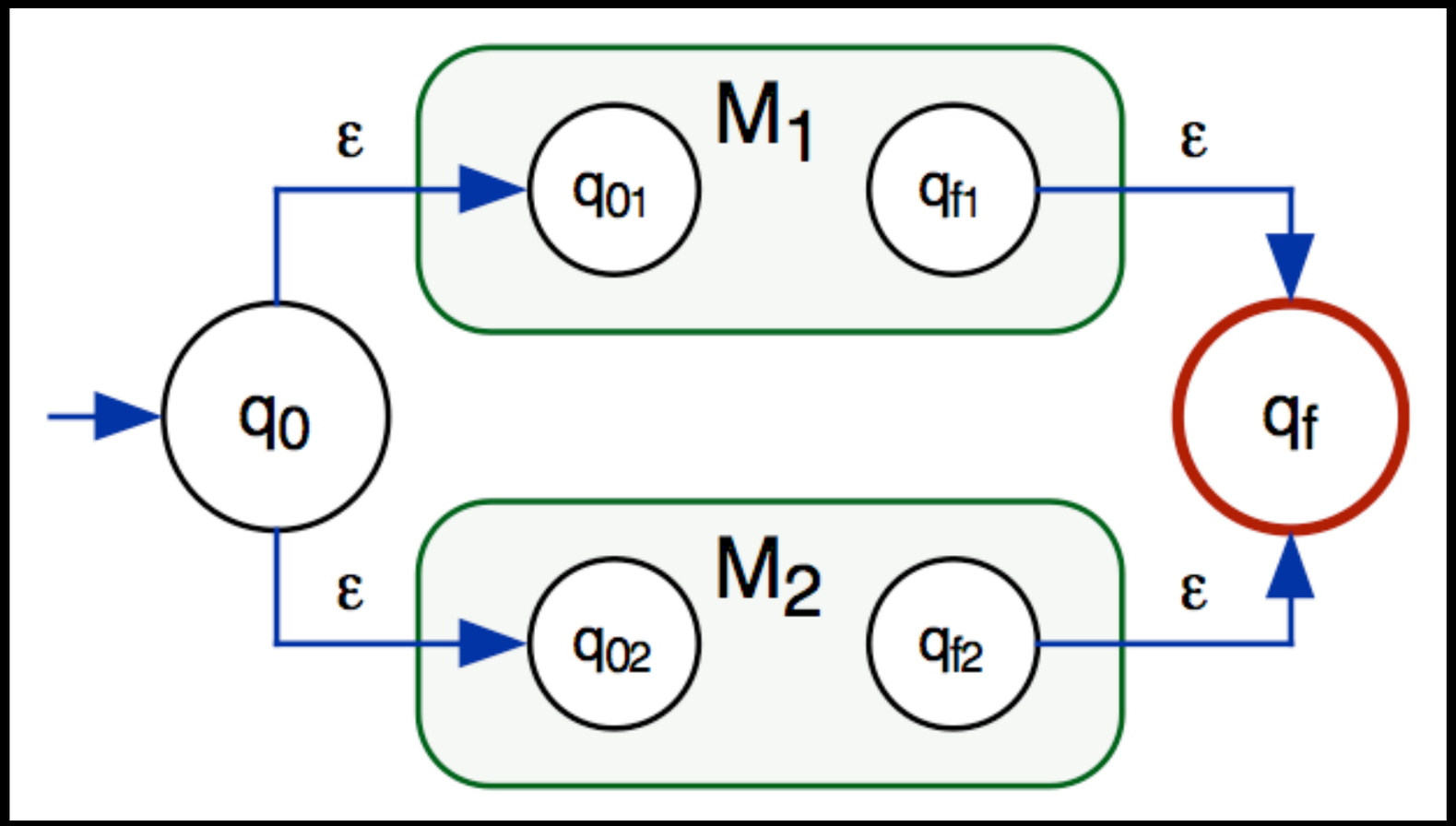
$M_2=(\{x\},\{q_0,q_f\},\delta_2,q_0,\{q_f\})$

Passo de indução

- Se r é uma ER com n+1 operadores, pode ser representada por:
  - $r=r_1+r_2$
  - $r=r_1r_2$
  - $r = r_1^*$
- É possível representar cada uma destas com autômatos?

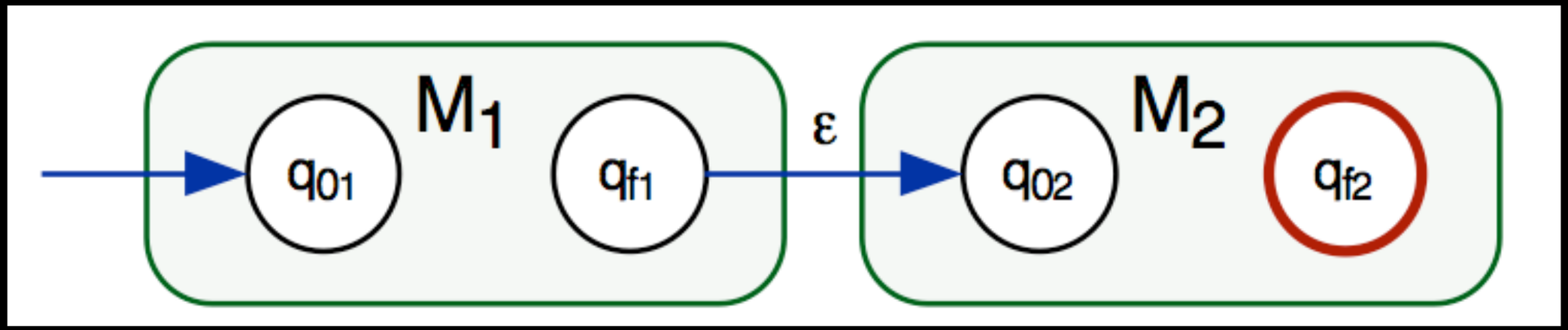
$$r = r_1 + r_2$$

$$M_3 = (\Sigma_1 \cup \Sigma_2,\ Q_1 \cup Q_2 \cup \{ q_0, q_f \},\ \delta,\ q_0,\ \{ q_f \})$$



$$r = r_1 r_2$$

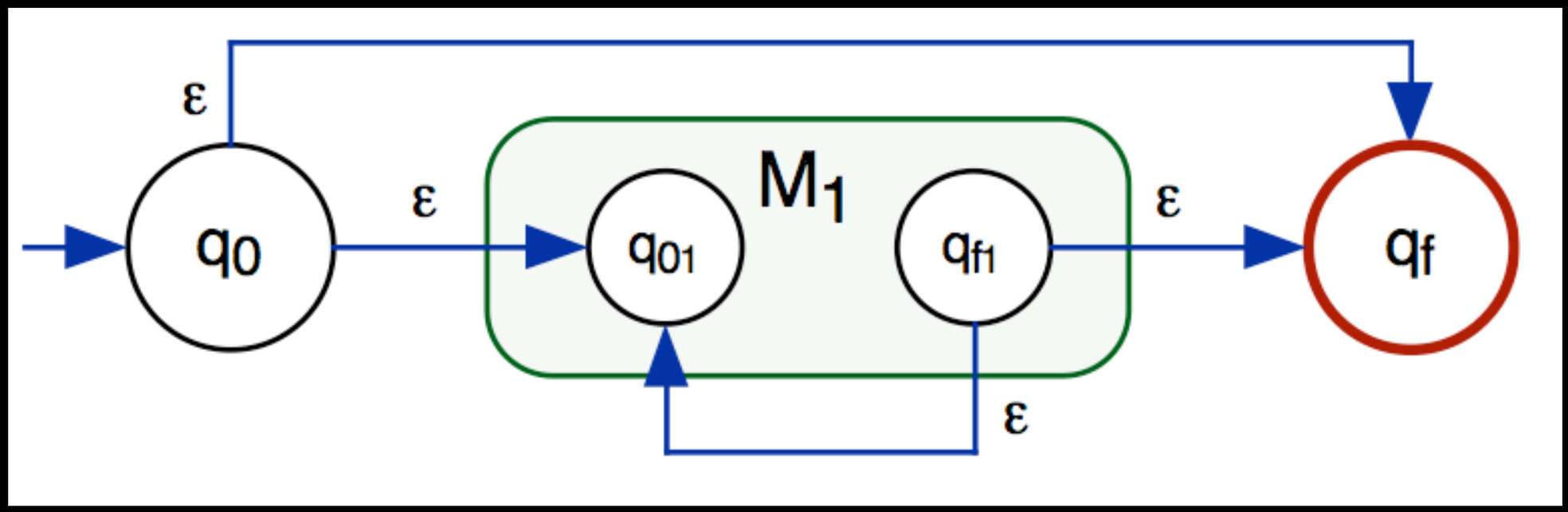
$$M_4 \equiv (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{ q_0, q_f \}, \delta, q_{01}, \{ q_{f2} \})$$



$$r = r_1^*$$

(suponha $q_0 \notin Q1$, $q_f \notin Q1$)

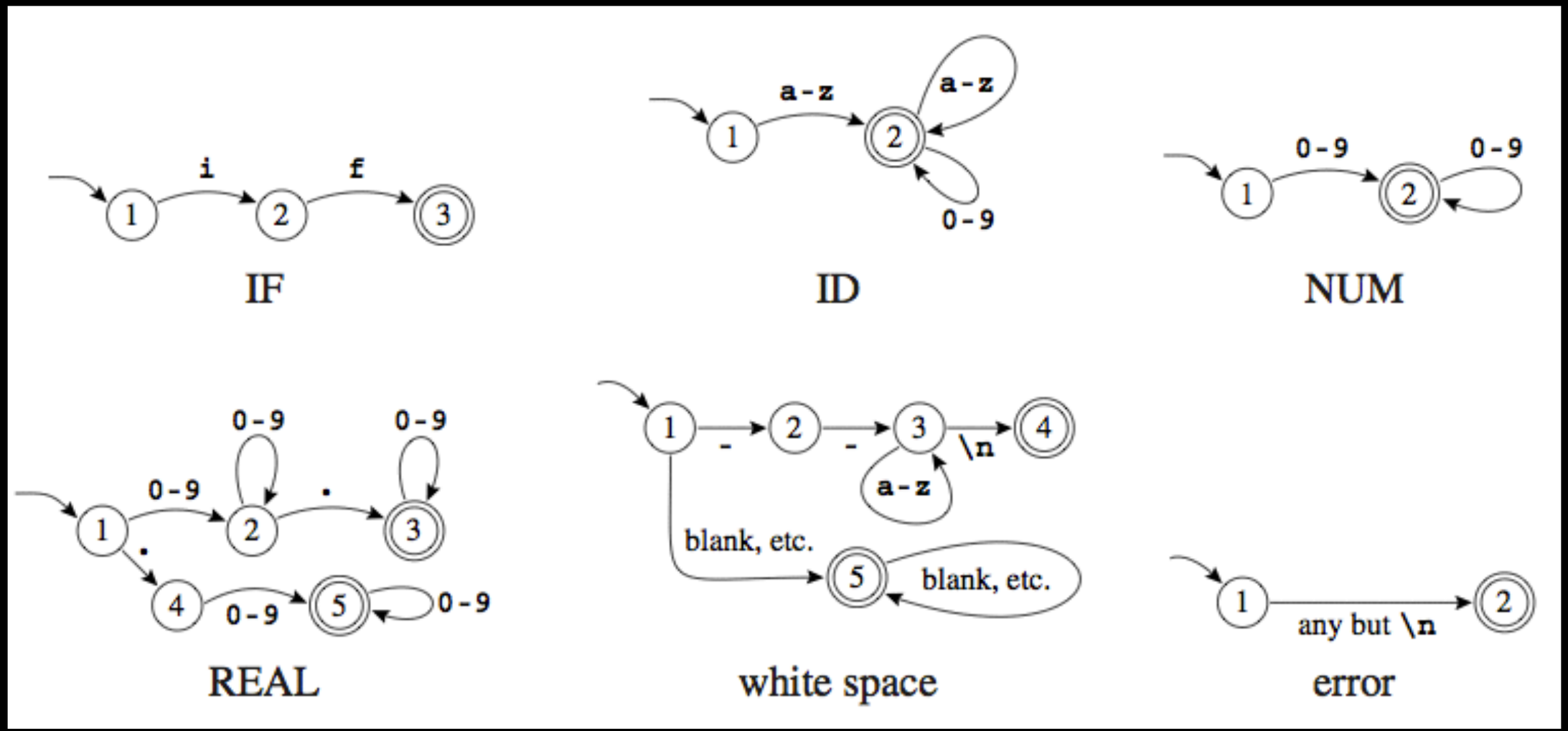
$$M_5 = (\Sigma_1, Q_1 \cup \{ q_0, q_f \}, \delta, q_0, \{ q_f \})$$



Exemplo

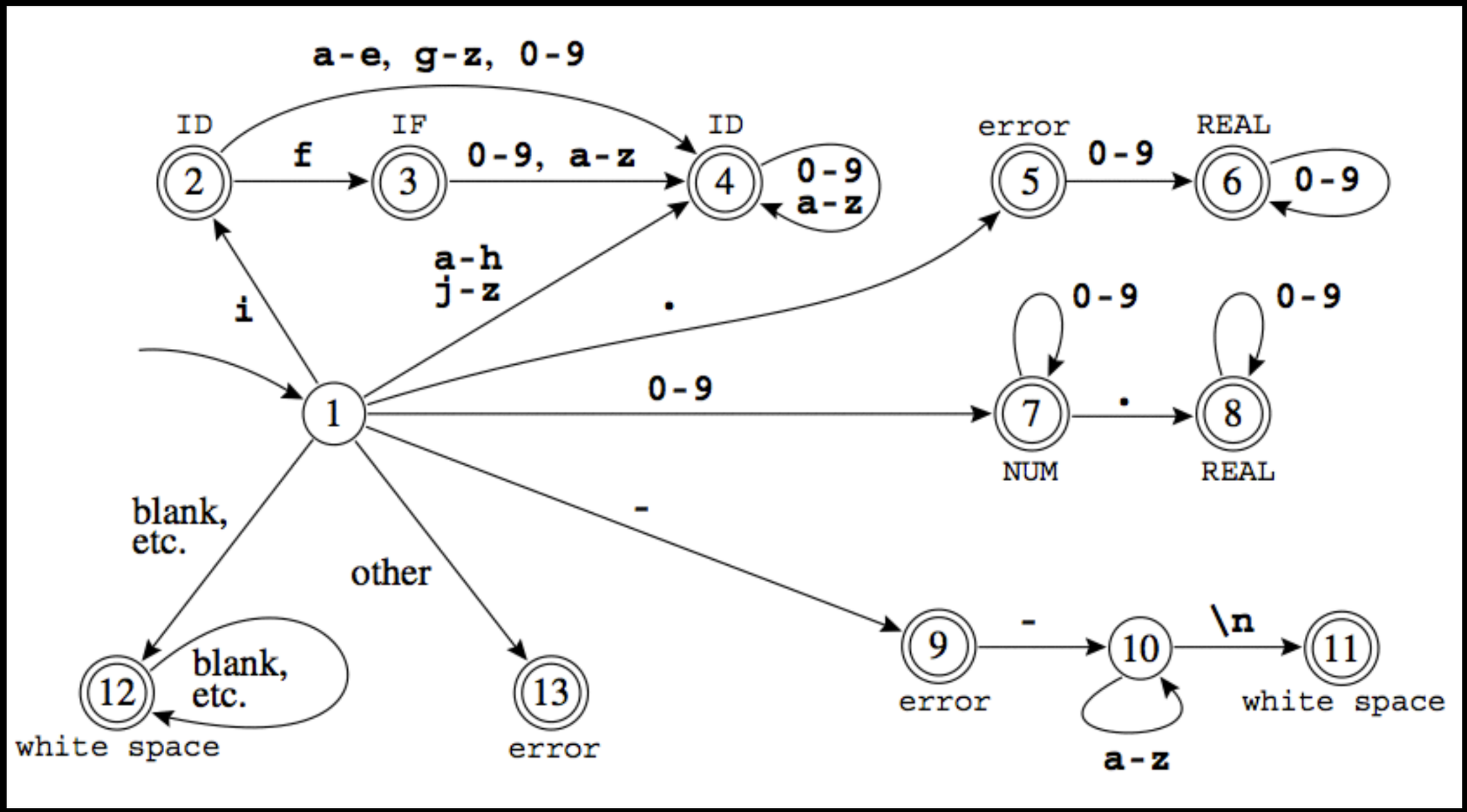
if → if
id → [a–z][a–z0–9]*
num → [0–9]+
real → ([0–9]+"."[0–9]*)|([0–9]*"."[0–9]+)
ws → ("––"[a–z]*"\n")|(" "|"\n"|"\t")+
error → **.**

Autômatos Exemplo



Adaptado de Andrew Appel
Modern Compiler Implementation in Java

Combinando os Autômatos



Adaptado de Andrew Appel
Modern Compiler Implementation in Java

Próxima aula:

implementando analisadores léxicos