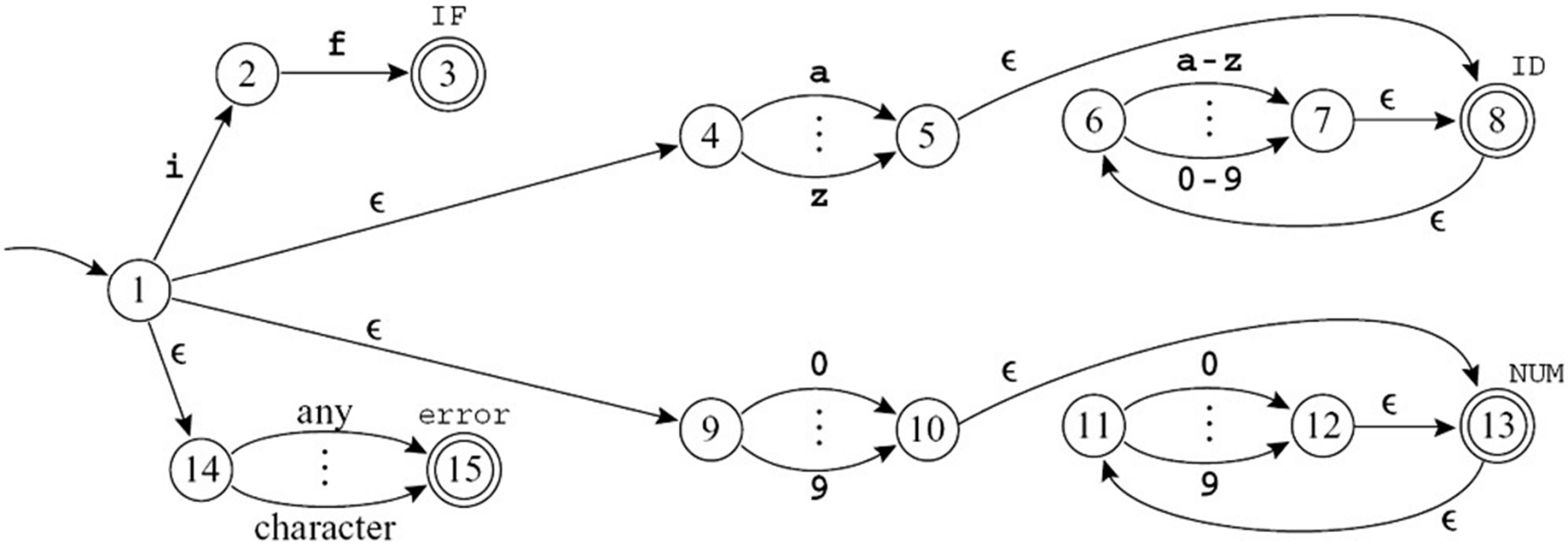
Simulando NFA para “in”

Início (1) -> NFA pode estar em {1,4,9,14}

Consome `i` -> NFA pode estar em {2,5,6,8,15}

Consome n -> NFA pode estar em {6,7,8}



NFA vs DFA

DFAs são facilmente simuláveis por programas de computador

NFAs são mais complexos, pois o programa teria que “adivinhar” o melhor caminho em alguns momentos

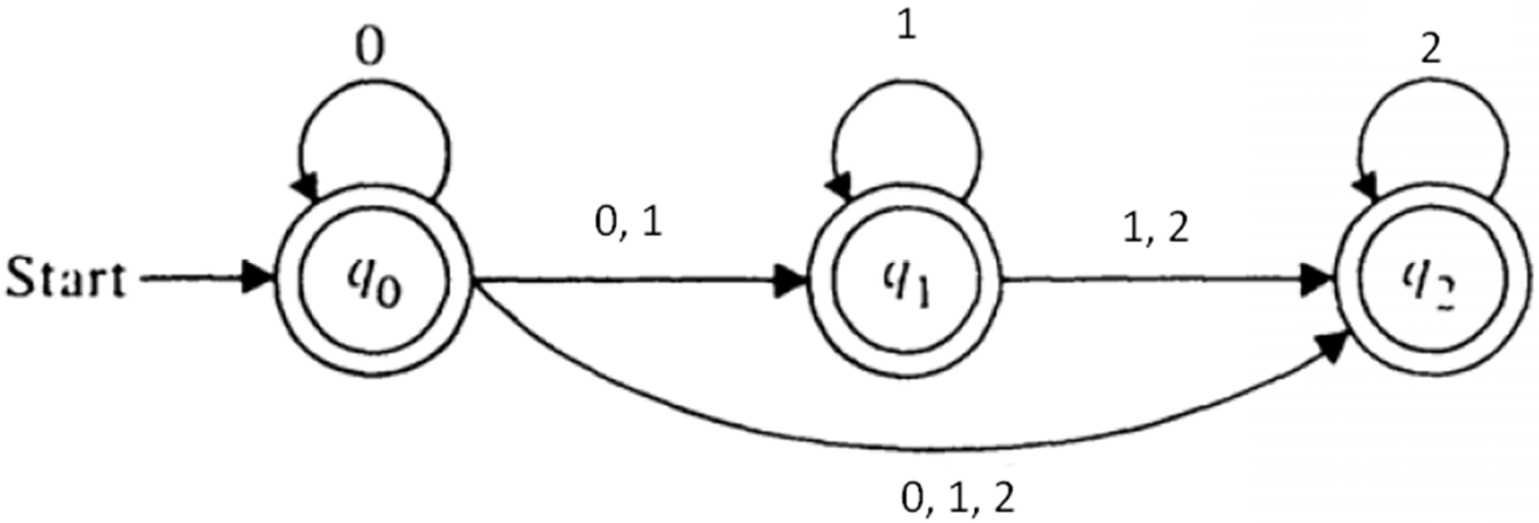
Outra alternativa seria tentar todas as possibilidades

Convertendo NFA em DFA

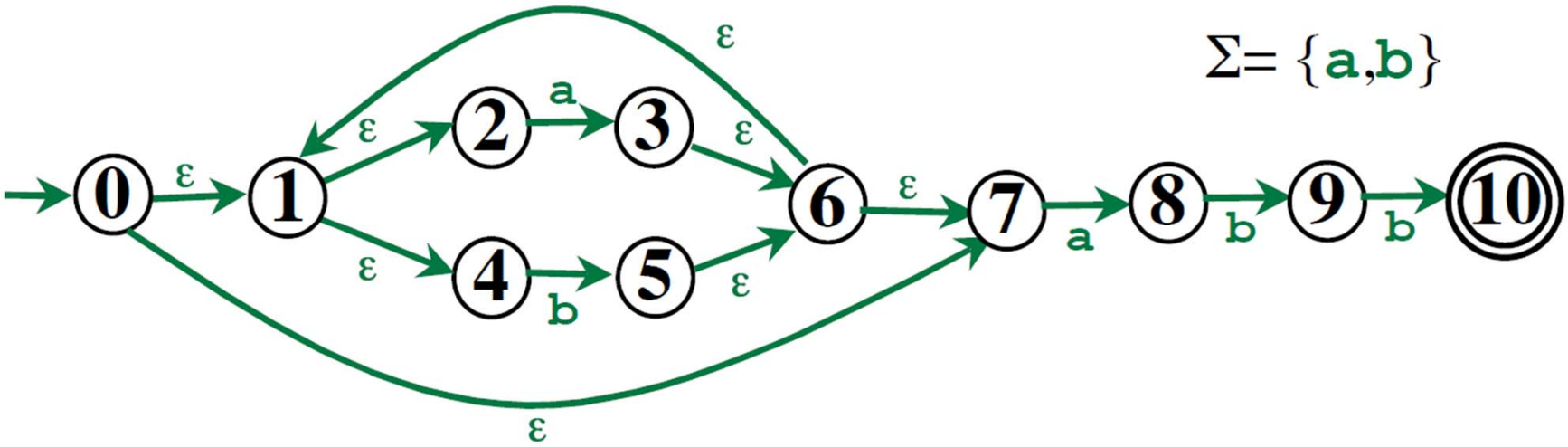
Manipular esses conjuntos de estados é muito caro durante a simulação
Solução:

- Calcular todos eles antecipadamente
- Isto converte o NFA em um DFA !!!
- Cada conjunto de estados no NFA se torna um estado no DFA

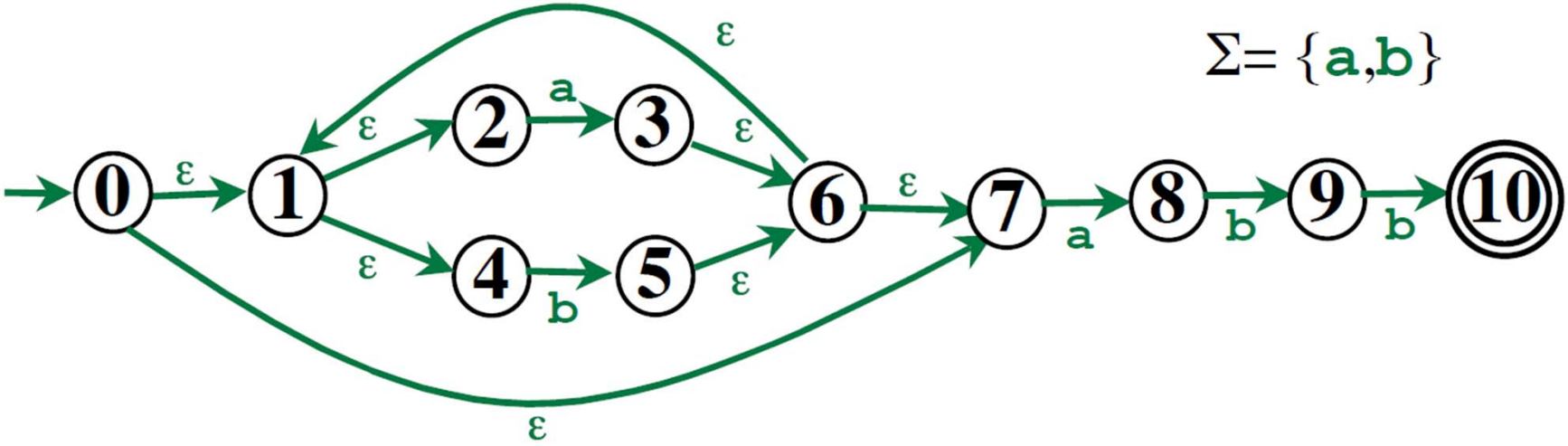
Convertendo NFA em DFA

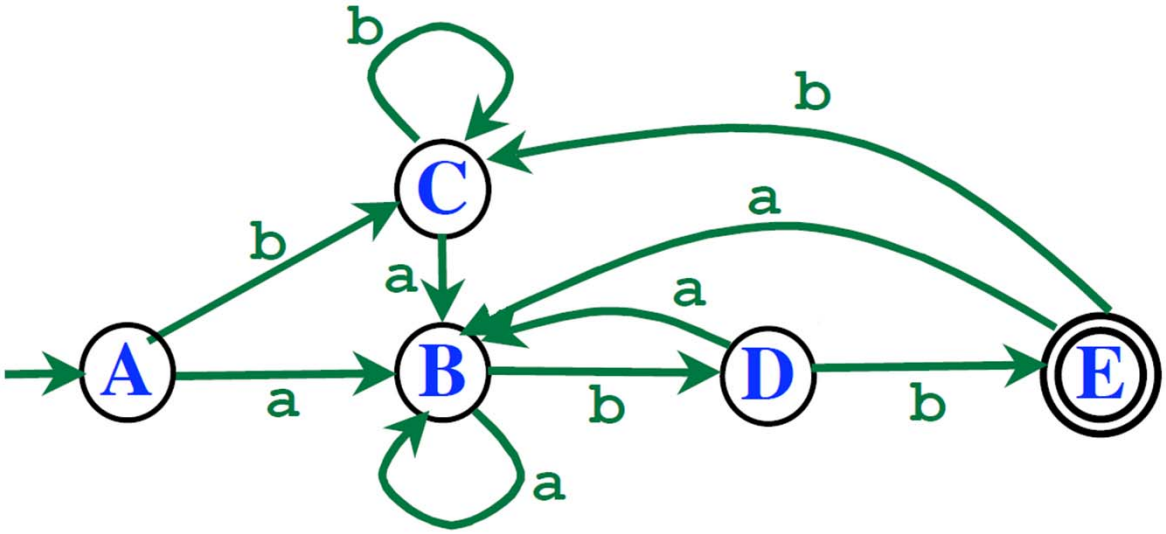


Convertendo NFA-Є em DFA



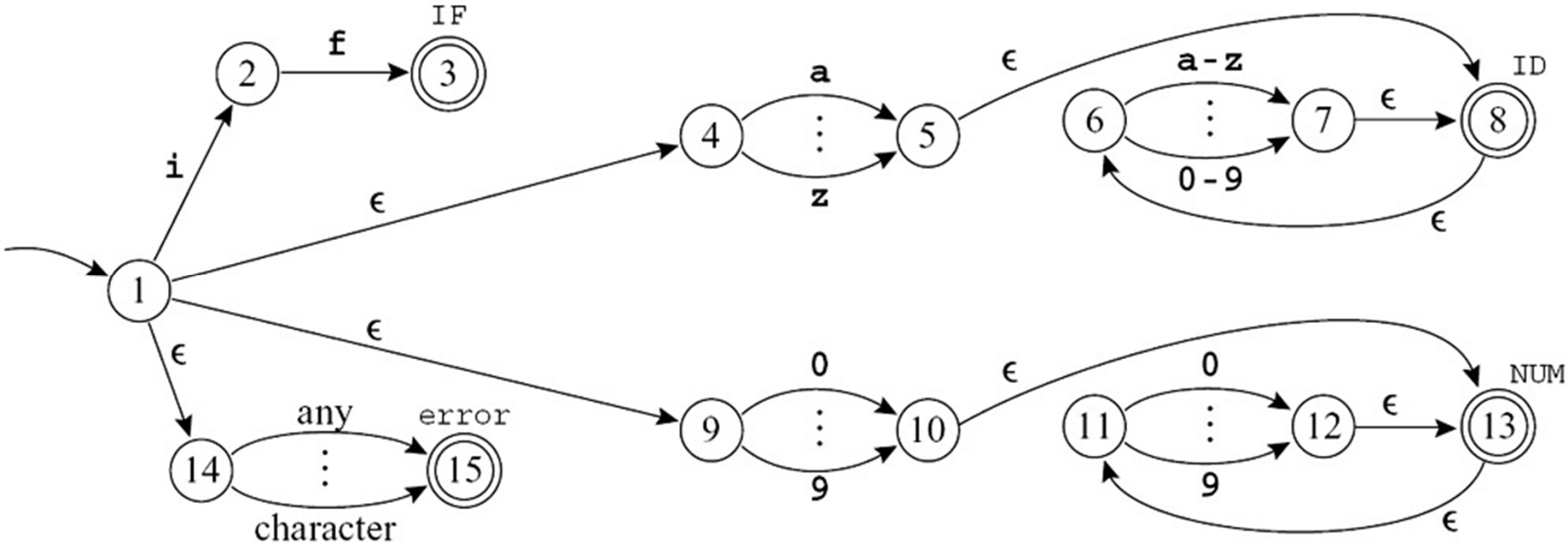
Convertendo NFA-Є em DFA



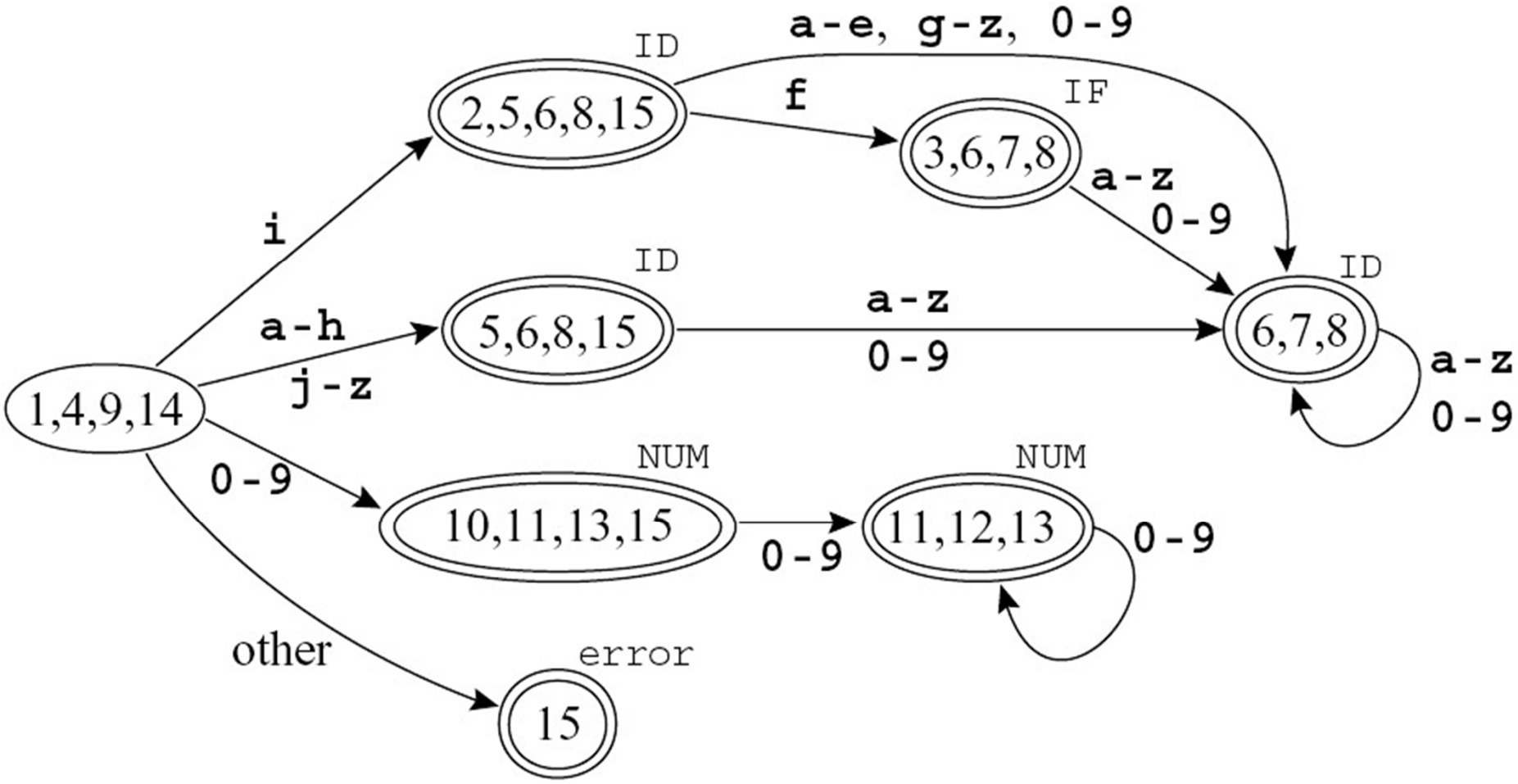


Convertendo NFA-Є em DFA → ANTES

ERs para **`IF`**, **`ID`**, **`NUM`** e **`error`**



Convertendo NFA-Є em DFA → DEPOIS



Estados Equivalentes

Estados Equivalentes

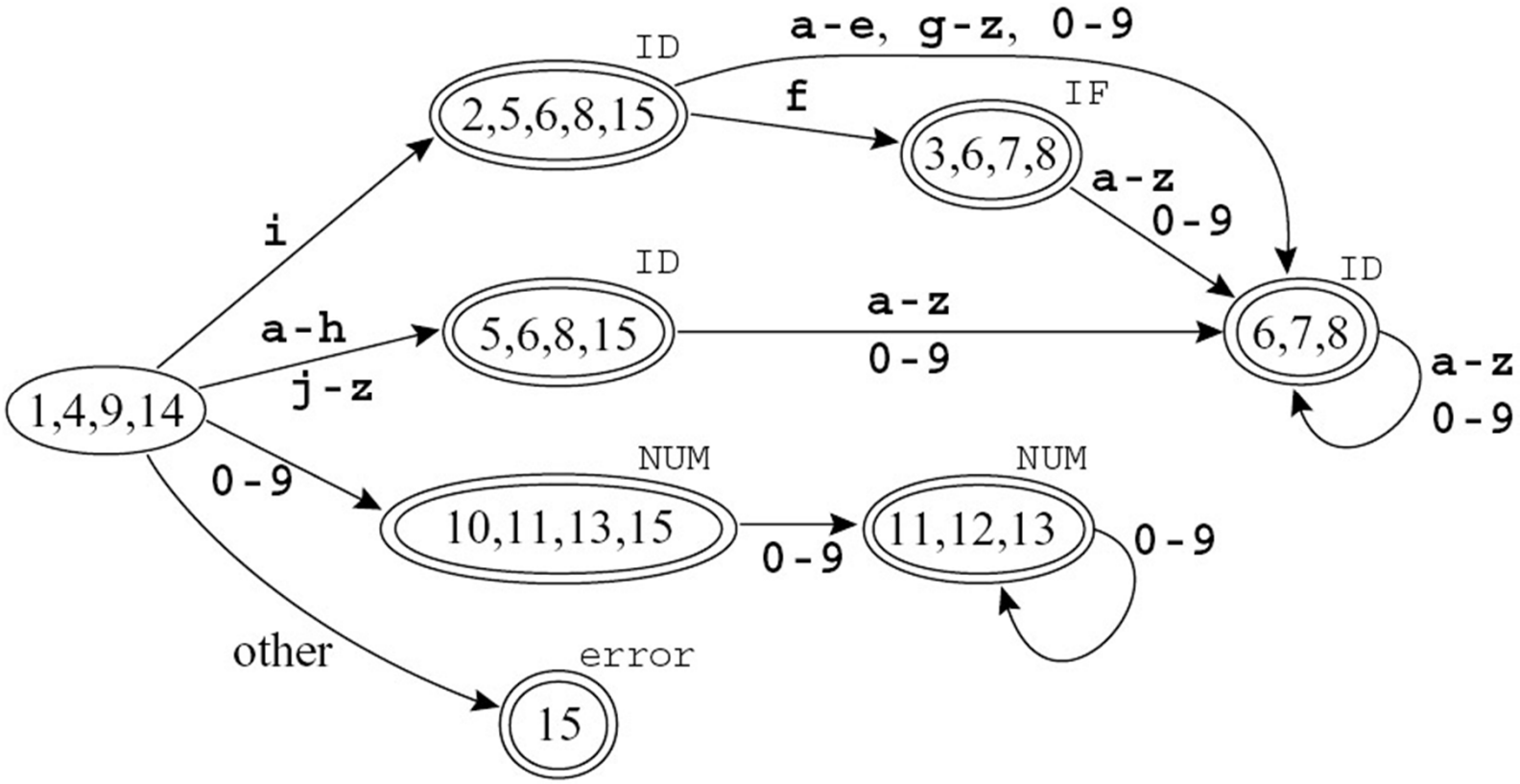
- Dois estados s_1 e s_2 são equivalentes quando o autômato aceita uma cadeia w começando em s_1 ⇔ ele também aceita a mesma cadeia w começando em s_2

- Para cada símbolo do alfabeto, tem-se que:

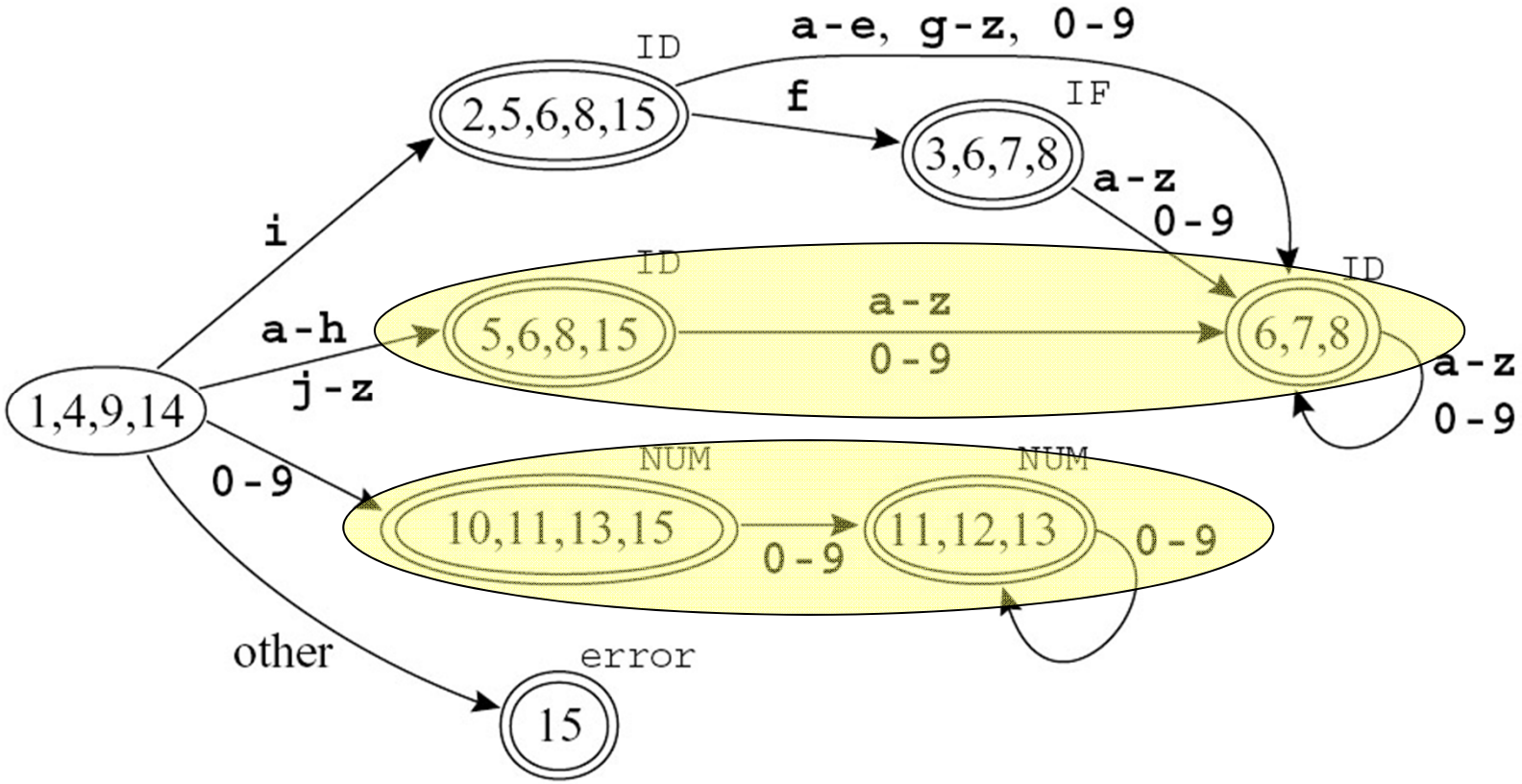
$$\text{trans}[\, s_1, c \,] = \text{trans}[\, s_2, c \,] \text{ para } \forall\, c$$

Estados Equivalentes

- Quais estados são equivalentes no autômato?

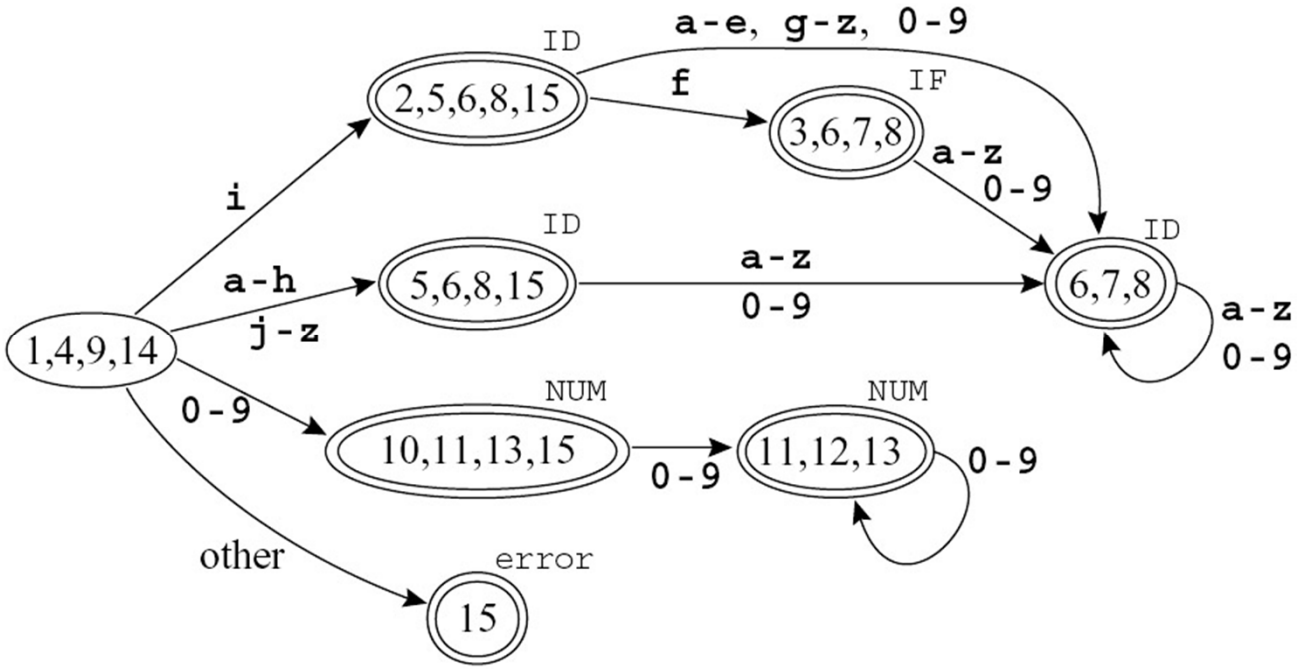


Estados Equivalentes



Estados Equivalentes

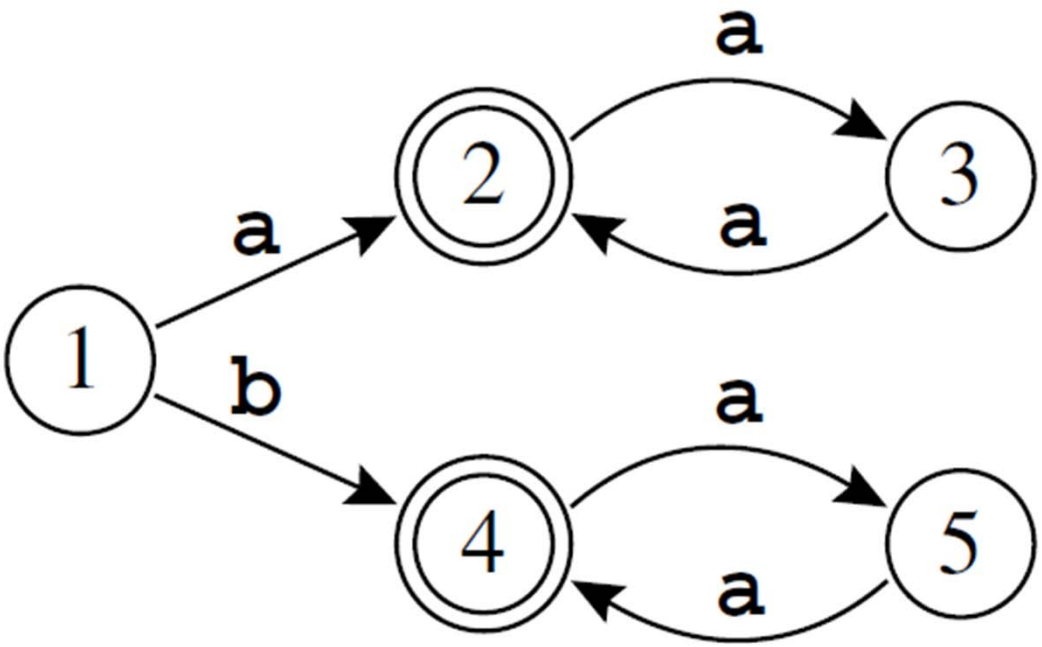
- Como encontrar estados equivalentes?
  - trans[s_1, c] = trans[s_2, c] para $\forall$ c
  - Não é suficiente!!!



Estados Equivalentes

- Contra exemplo:

Os estados 2 e 4 são equivalentes, mas trans[2, **a**] ≠ trans[4, **a**]



Estados Equivalentes

Estados Equivalentes

- Dois estados s_1 e s_2 são equivalentes quando o autômato aceita uma cadeia w começando em s_1 ⇔ ele também aceita a mesma cadeia w começando em s_2

$\delta(s_1, w) \rightarrow$ cadeia aceita

e

$\delta(s_2, w) \rightarrow$ cadeia aceita

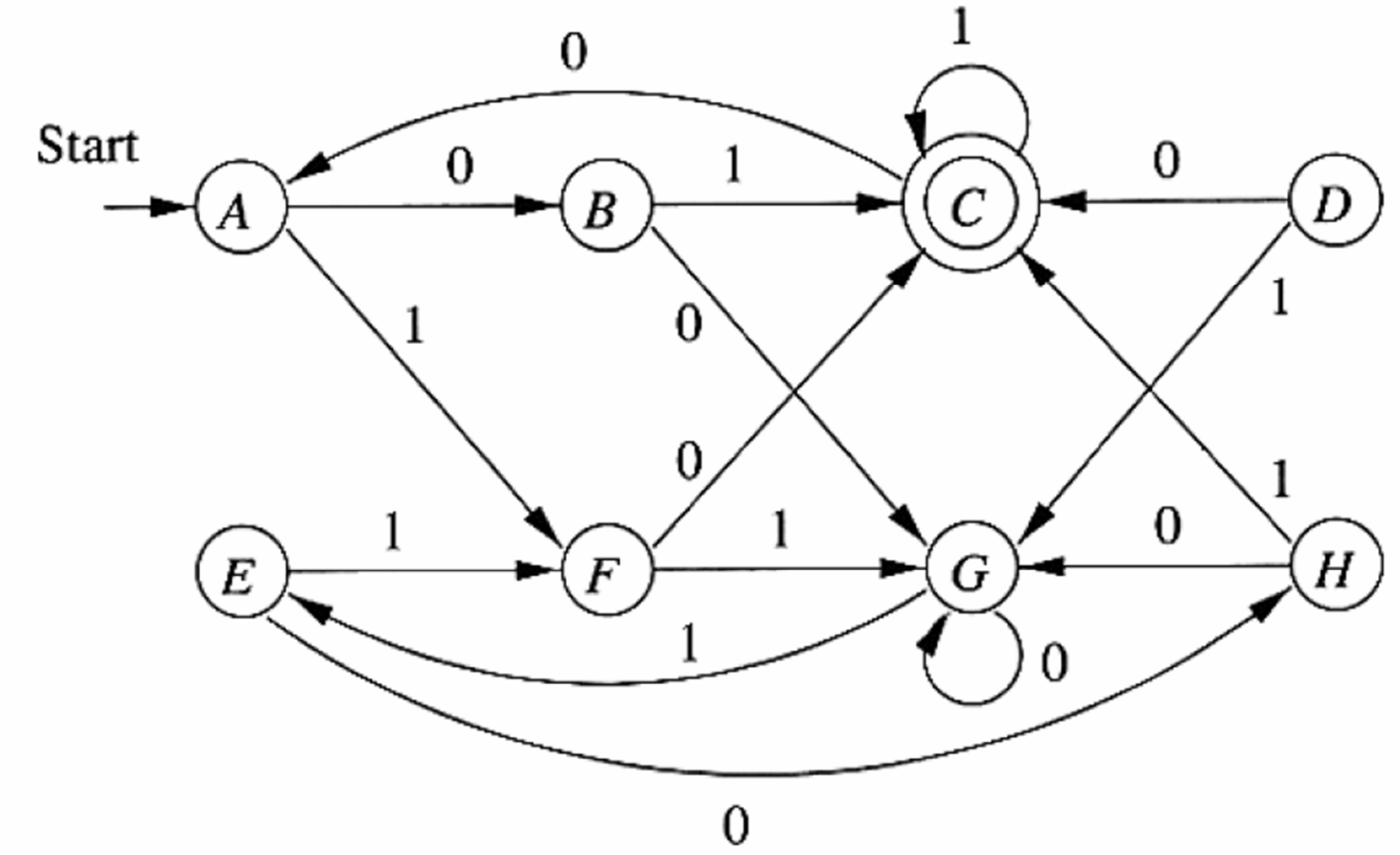
OU

$\delta(s_1, w) \rightarrow$ cadeia NÃO aceita

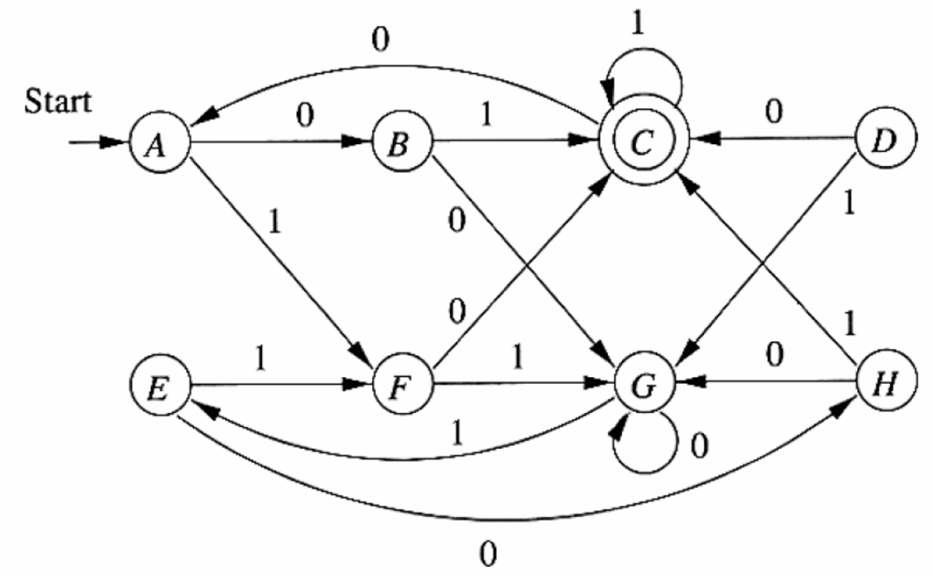
e

$\delta(s_2, w) \rightarrow$ cadeia NÃO aceita

Minimização de DFA

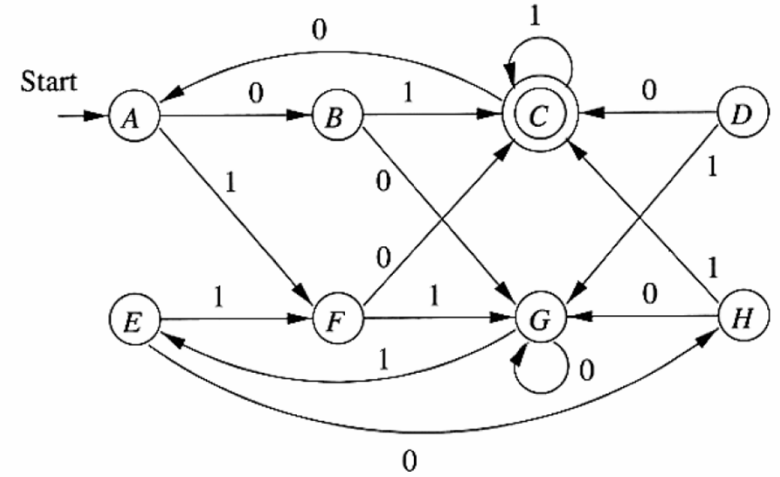


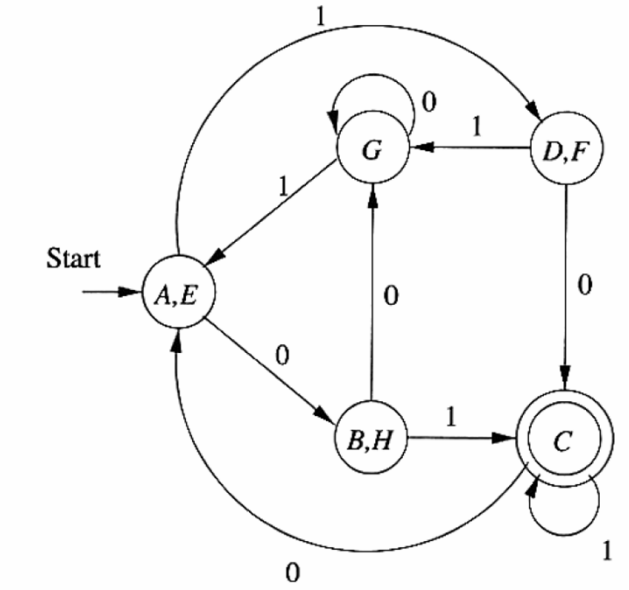
Minimização de DFA



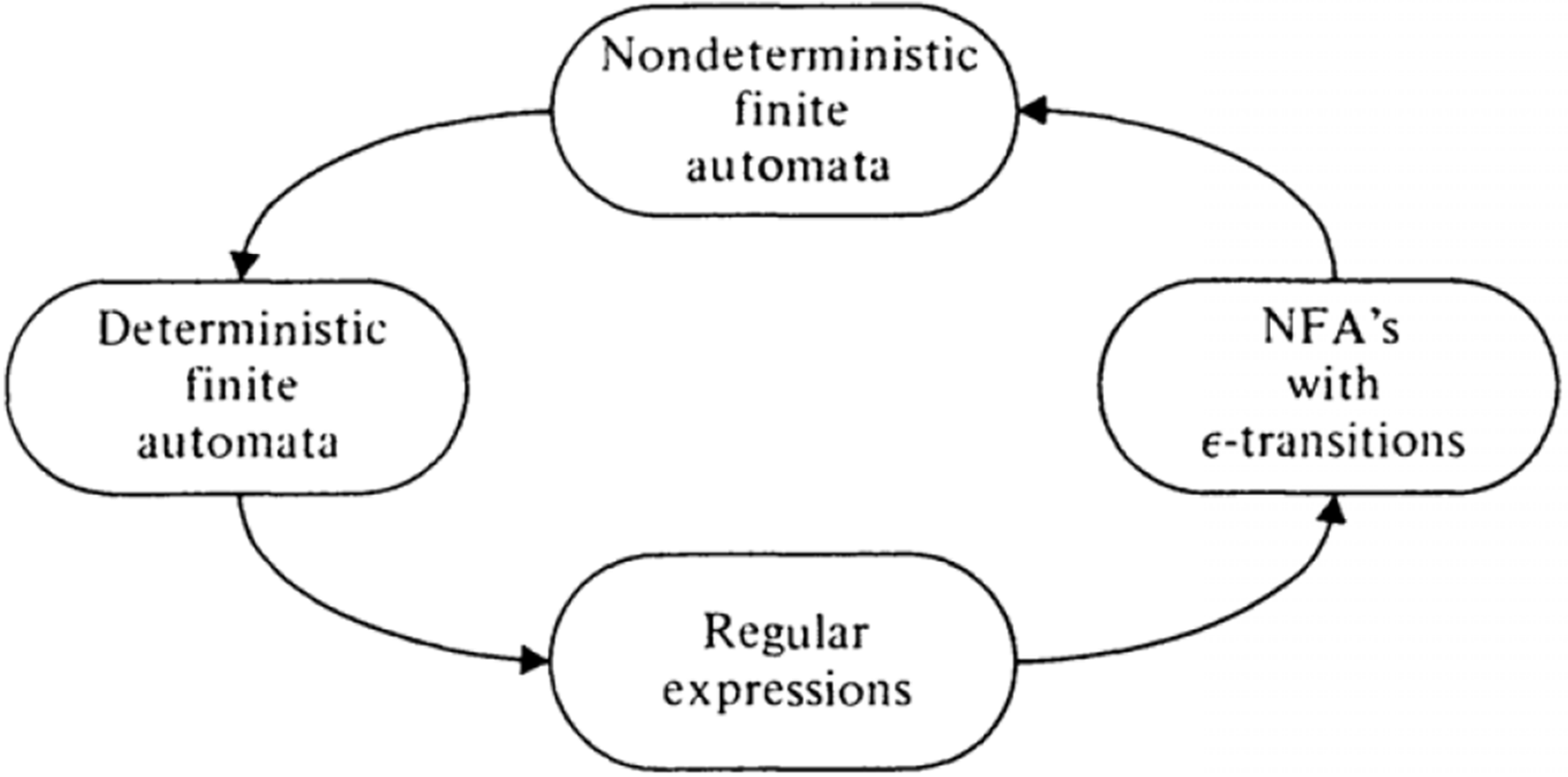
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Minimização de DFA

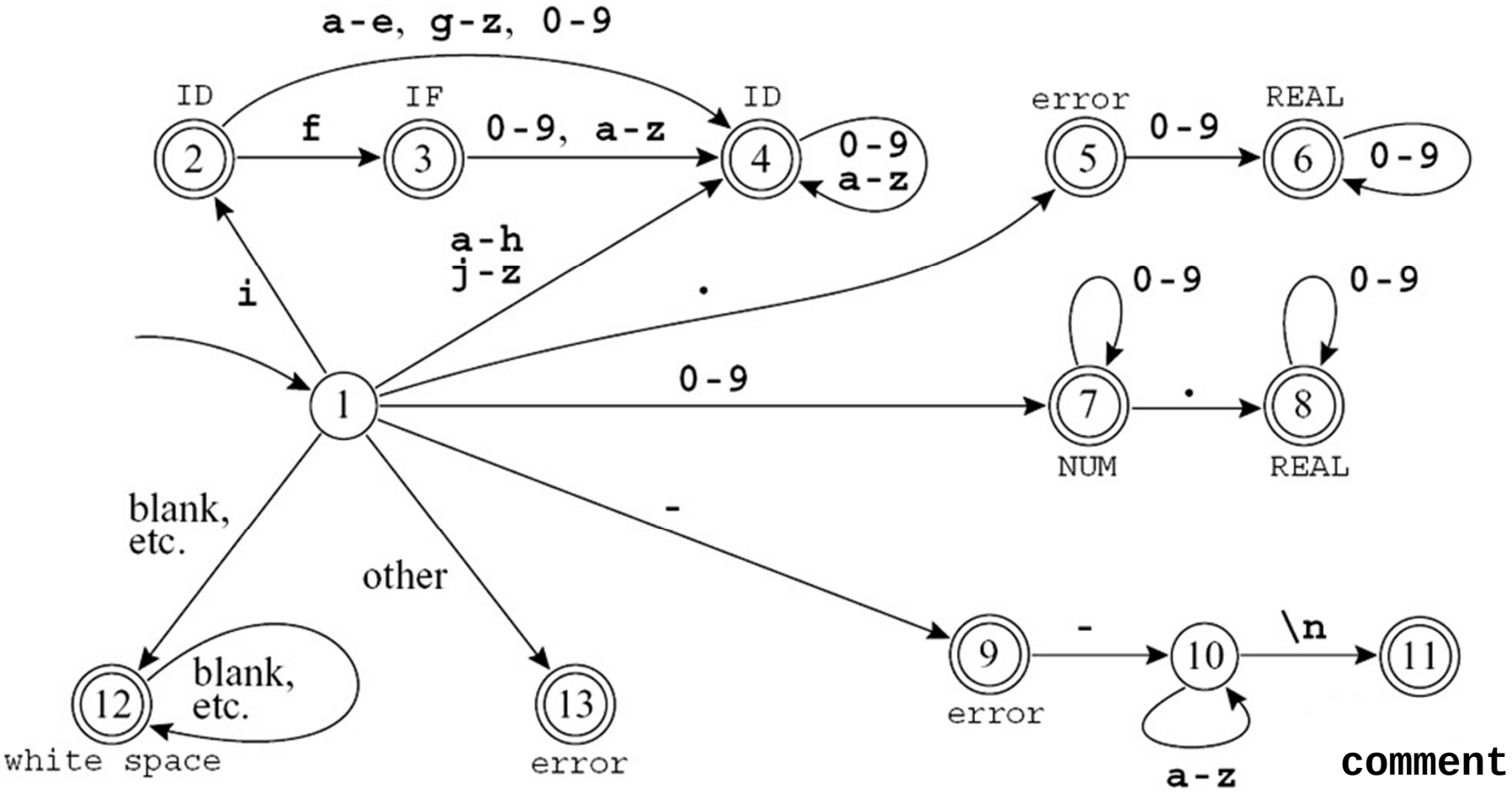




Equivalência entre Autômatos Finitos e ER



Autômato Combinado



Autômato Combinado

```
int edges[ ][ ] = { /* ...012...-...e f g h i j... */
/* state 0 */ {0,0,...0,0,0...0...0,0,0,0,0,0...},
/* state 1 */ {0,0,...7,7,7...9...4,4,4,4,2,4...},
/* state 2 */ {0,0,...4,4,4...0...4,3,4,4,4,4...},
/* state 3 */ {0,0,...4,4,4...0...4,4,4,4,4,4...},
/* state 4 */ {0,0,...4,4,4...0...4,4,4,4,4,4...},
/* state 5 */ {0,0,...6,6,6...0...0,0,0,0,0,0...},
/* state 6 */ {0,0,...6,6,6...0...0,0,0,0,0,0...},
/* state 7 */ {0,0,...7,7,7...0...0,0,0,0,0,0...},
/* state 8 */ {0,0,...8,8,8...0...0,0,0,0,0,0...},
   et cetera }
```

entrada

Ausência de aresta

Reconhecimento da Maior SubString

A tabela anterior é usada para aceitar ou recusar uma string

Porém, precisa-se garantir que a maior string seja reconhecida

Duas informações são necessárias:

- Último estado final
- Posição da entrada no último estado final

Reconhecimento da Maior SubString

Last Final	Current State	Current Input	Accept Action
0	1	⌶if --not-a-com	
2	2	|i⌶f --not-a-com	
3	3	|if⌶ --not-a-com	
3	0	|if⊤⊥--not-a-com	*return* IF
0	1	if⌶ --not-a-com	
12	12	if|⌶--not-a-com	
12	0	if|⊤-⊥-not-a-com	*found white space; resume*
0	1	if ⌶--not-a-com	
9	9	if |-⌶-not-a-com	
9	10	if |-⊤-⌶not-a-com	
9	10	if |-⊤-n⊥ot-a-com	
9	10	if |-⊤-no⊥t-a-com	
9	10	if |-⊤-not⊥-a-com	
9	0	if |-⊤-not-⊥a-com	*error, illegal token '-'; resume*
0	1	if -⌶-not-a-com	
9	9	if -|-⌶not-a-com	
9	0	if -|-⊤n⊥ot-a-com	*error, illegal token '-'; resume*