Conversão de um NFA para um DFA

- É possível converter um autómato não determinístico num autómato determinístico.
- A ideia é construir um DFA que simula todos os possíveis estados em que o NFA pode estar após ter consumido um determinado input.
- Este método tem um nome: Subset Construction

Podem ver descrição detalhada no livro do dragão, incluindo pseudocódigo.

- Aqui vou ilustrar apenas usando o autómato da aula passada como exemplo.

NFA → DFA (Subset Construction)

NFA

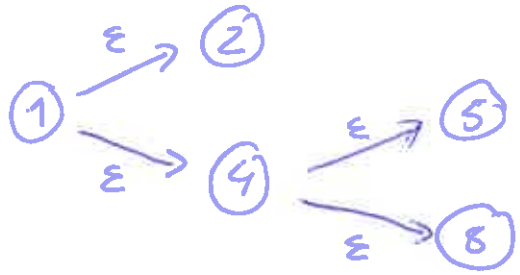
		d	•	ε
→	1			{2,4}
	2	{3}		
	3			{2,4}
	4			{5,8}
	5		{6}	
	6	{7}		
	7			{11}
	8	{9}		
	9		{10}	
	10			{11}
	11			{12,14}
	12	{13}		
	13			{12,14}
*	14			

DFA

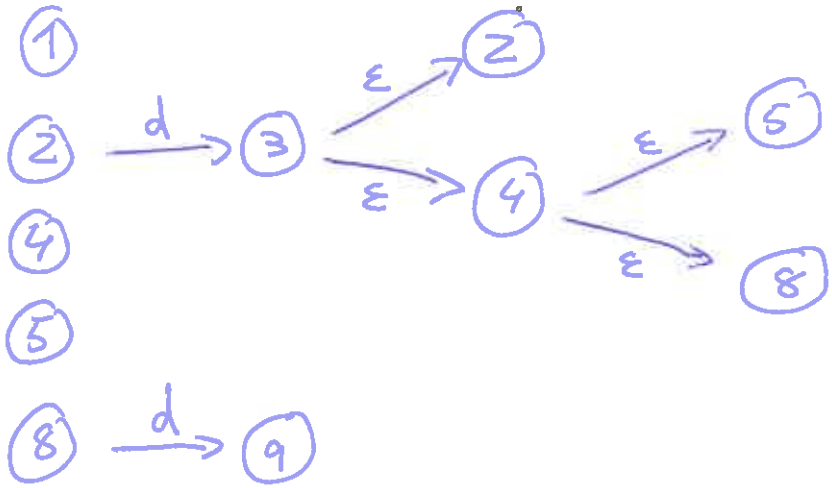
	d	•
→ {1,2,4,5,8}	{2,3,4,5,8,9}	{6}
{2,3,4,5,8,9}	{2,3,4,5,8,9}	{6,10,11,12,14}
{6}	{7,11,12,14}	
* {6,10,11,12,14}	{7,11,12,13,14}	
* {7,11,12,14}	{12,13,14}	
* {7,11,12,13,14}	{12,13,14}	
* {12,13,14}	{12,13,14}	

entradas vazias correspondem a ∅

- Inicialmente o NFA pode estar no estado 1 mas também pode estar nos estados 2, 4, 5, 8 através de transições ε.



- Se estivermos num destes estados do NFA, e o input for um d, para onde podemos ir?

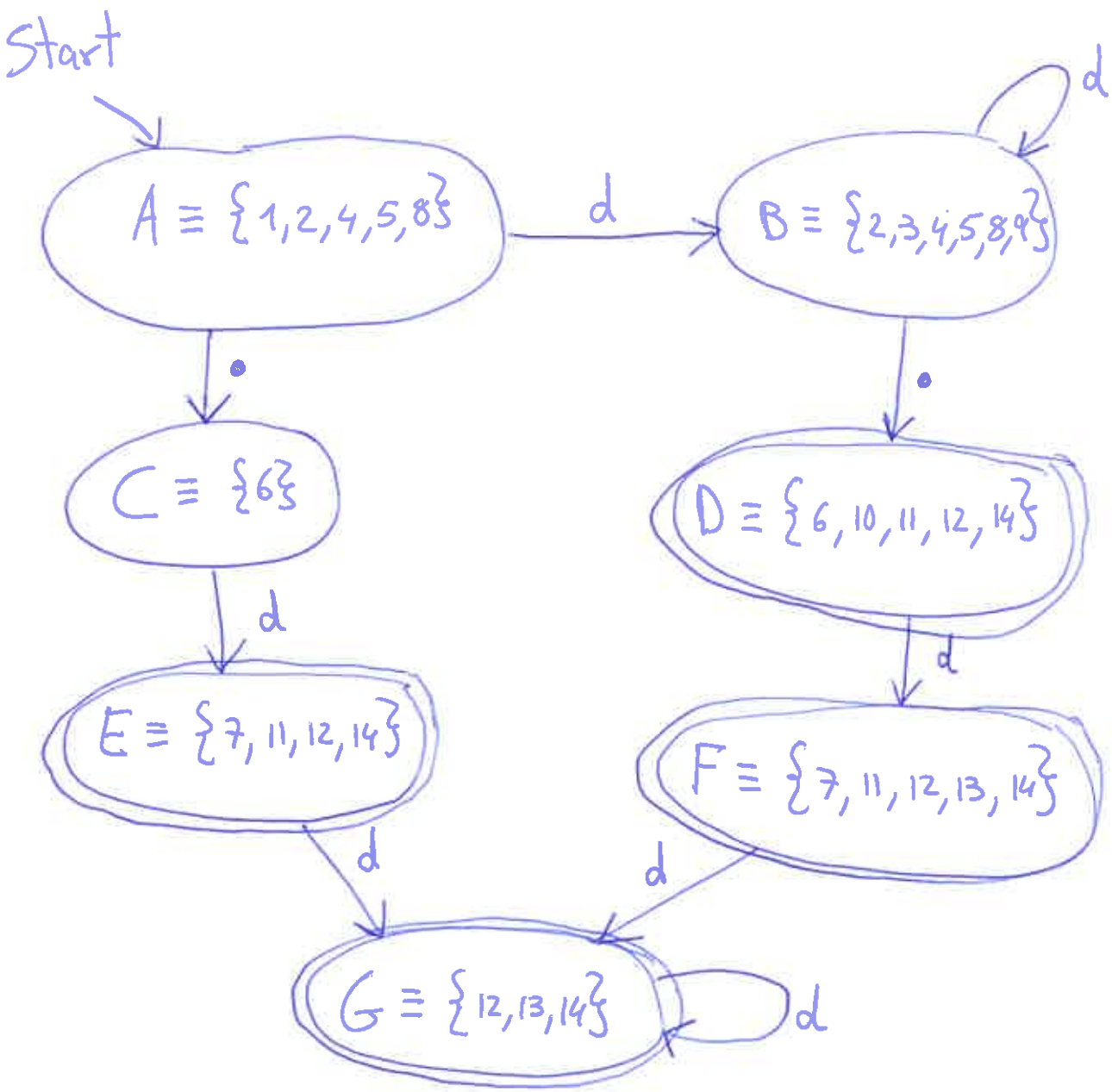


Resposta: {2, 3, 4, 5, 8, 9}

- E assim sucessivamente...

- Os estados do DFA correspondem a conjuntos de estados em que o NFA poderia estar.

- Todos os estados do DFA que contenham o estado final do NFA passam a ser estados finais. (no exemplo, todos aqueles que têm o estado 14.)



Em resumo

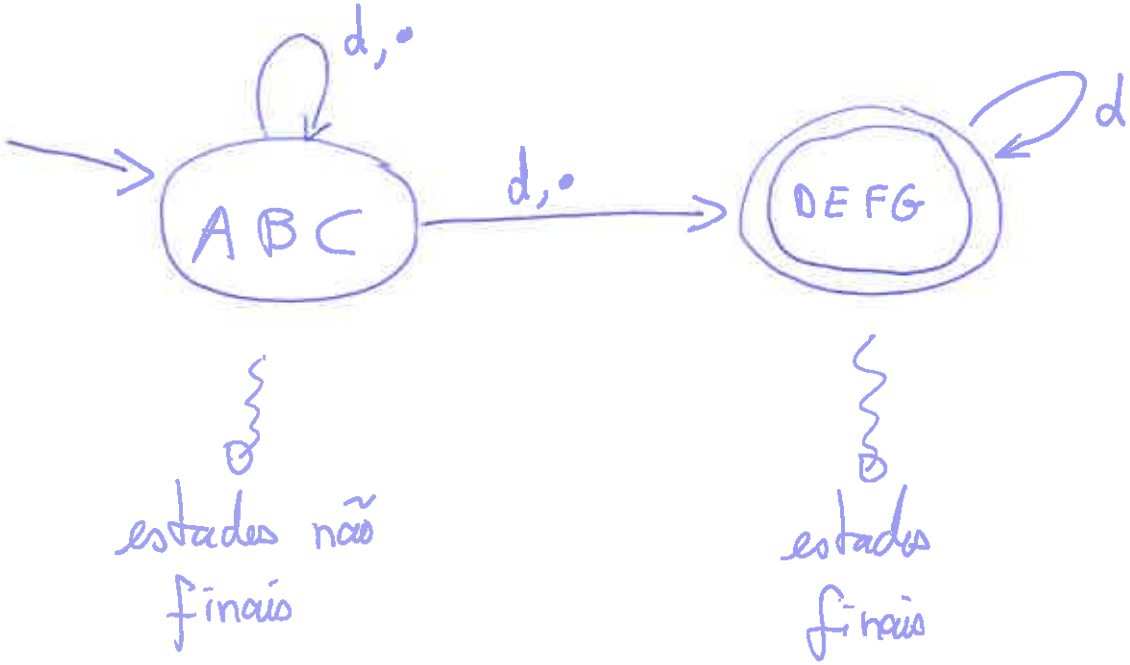
Exp. Regular ⟶ NFA ⟶ DFA

construção de Thompson

subset construction

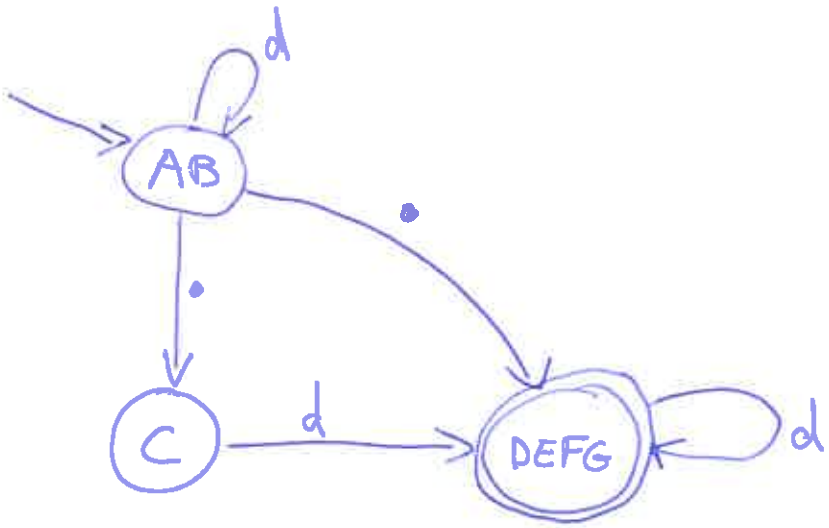
Minimização de estados num DFA

- O DFA obtido por subset construction pode ser simplificado de modo a ter o número mínimo de estados.



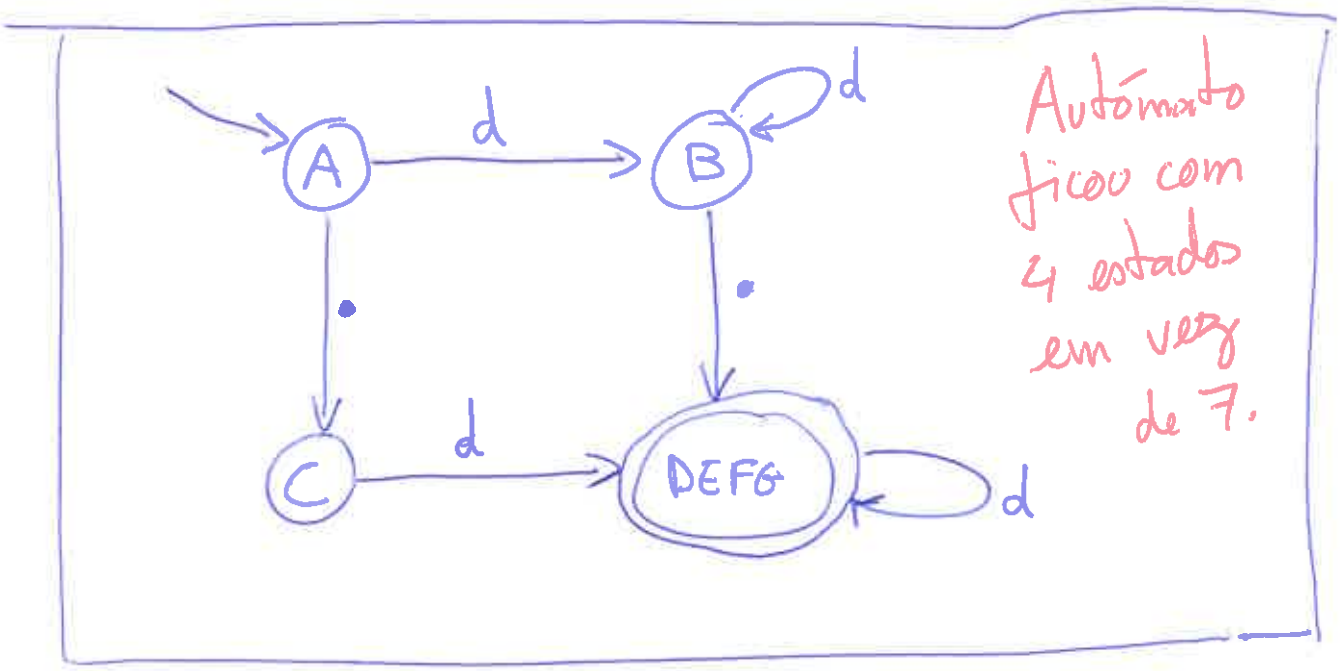
- Ficou não determinístico porque (ABC) tem transições em 'd' e '.' para mais do que um estado.

  → Divide-se (ABC) em 2 estados.



- (AB) continua a dar problemas com '.'

  → Divide (AB) em 2 estados



DFA → código

- Uma vez tendo um DFA é fácil escrever código que simula o DFA.

- Pode ser feito de maneira adhoc como fizemos há algumas aulas.

- Ou de maneira sistemática (automática)

  → com um switch e um case para cada estado.

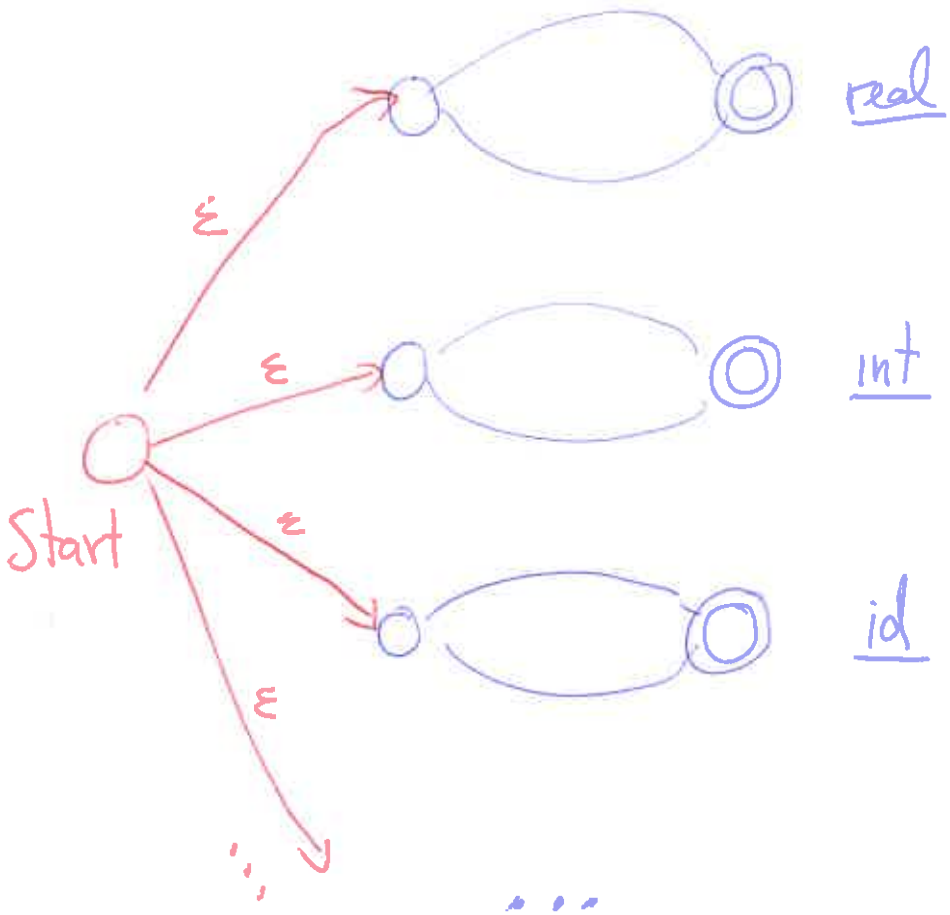
  → ou recorrendo a uma tabela de transição de estados.

```
s = s0   // estado inicial
c = nextsymbol()
while c ≠ EOF do {
|    s = move(s, c)
|    c = nextsymbol()
}
if s ∈ F return "ACCEPT"
else return "REJECT"
```

Construção automática de um analisador léxico

Input: expressões regulares, uma para cada tipo de token.

Output: analisador léxico que reconhece os tokens



- Cria-se um estado inicial com transições ε, para o estado inicial do autómato que reconhece cada tipo de token.
- Faz-se NFA → DFA → código.

Lex / Flex em UNIX/Linux