Teoria da Computação

Aula Teórica 12:
Compilação de expressões regulares para AFDs

António Ravara

Departamento de Informática

15 de Abril de 2019

Linguagens regulares versus AFDs

Teorema de Kleene

Linguagens regulares = Linguagens aceites pelos AFDs.

Questões relevantes:

- Como obter o AFD que aceita a linguagem denotada por dada expressão regular?
- Como obter a expressão regular que denota a linguagem de dado AFD?

São problemas *algorítmicos*!

Vamos ver a tradução de expressões regulares em AFDs.

Linguagens regulares versus AFDs

Teorema de Kleene

Linguagens regulares = Linguagens aceites pelos AFDs.

Questões relevantes:

- Como obter o AFD que aceita a linguagem denotada por dada expressão regular?
- Como obter a expressão regular que denota a linguagem de dado AFD?

São problemas *algorítmicos*!
Vamos ver a tradução de expressões regulares em AFDs.

Linguagens regulares versus AFDs

Teorema de Kleene

Linguagens regulares = Linguagens aceites pelos AFDs.

Questões relevantes:

- Como obter o AFD que aceita a linguagem denotada por dada expressão regular?
- Como obter a expressão regular que denota a linguagem de dado AFD?

São problemas *algorítmicos*!
Vamos ver a tradução de expressões regulares em AFDs.

Linguagens regulares versus AFDs

Teorema de Kleene

Linguagens regulares = Linguagens aceites pelos AFDs.

Questões relevantes:

- Como obter o AFD que aceita a linguagem denotada por dada expressão regular?
- Como obter a expressão regular que denota a linguagem de dado AFD?

São problemas *algorítmicos*!

Vamos ver a tradução de expressões regulares em AFDs.

Linguagens regulares versus AFDs

Teorema de Kleene

Linguagens regulares = Linguagens aceites pelos AFDs.

Questões relevantes:

- Como obter o AFD que aceita a linguagem denotada por dada expressão regular?
- Como obter a expressão regular que denota a linguagem de dado AFD?

São problemas *algorítmicos*!
Vamos ver a tradução de expressões regulares em AFDs.

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:

  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFD}_\Sigma$$

Linguagens regulares em AFDs

Como traduzir?

- Uma expressão regular arbitrária pode ser muito complexa...
- A abordagem tipicamente usada (nos compiladores, por exemplo) é *composicional*:
  - define-se *por casos* uma função (recursiva) que traduz cada tipo de operação.
  - os casos da definição (indutiva) da função são os da definição (indutiva) da linguagem a traduzir.

A função *Compile*

Dado um alfabeto Σ, seja $\mathcal{AFD}_\Sigma$ o conjunto de todos os AFDs sobre Σ. O objectivo é definir

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFD}_\Sigma$$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

A função *Compile*

- Caso $\emptyset$: $\mathcal{L}(\emptyset) = \emptyset$. Para aceitar a linguagem vazia basta ter um só estado, inicial, nenhum estado final, e também não são precisas acções.
  $\mathit{Compile}(\emptyset) = \langle \{1\}, \emptyset, 1, \emptyset, \emptyset \rangle$
- Caso ε: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$. Para aceitar (só) a palavra vazia (*i.e.*, a linguagem singular só com a palavra ϵ) basta ter um estado, inicial e final, e não são precisas acções.
  $\mathit{Compile}(\varepsilon) = \langle \{1\}, \emptyset, 1, \emptyset, \{1\} \rangle$
- Caso $a \in \Sigma$: $\mathcal{L}(a) = \{a\}$. Para aceitar (só) a palavra de tamanho 1 (um símbolo de Σ) basta ter dois estados, um inicial e outro final, e transitar do inicial para o final pelo símbolo.
  $\mathit{Compile}(a) = \langle \{1, 2\}, \{a\}, 1, \{(1, a) \mapsto 2\}, \{2\} \rangle$

Compile(EF)

Para aceitar uma palavra que resulta de concatenar uma palavra da linguagem de E e outra da linguagem de F, é necessário "ligar" o AFD que implementa E com o que implementa F.

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como passar dos estados finais de *Compile*(E) ao inicial de *Compile*(F) *sem consumir acções da palavra concatenada*?

Compile(EF)

Para aceitar uma palavra que resulta de concatenar uma palavra da linguagem de E e outra da linguagem de F, é necessário "ligar" o AFD que implementa E com o que implementa F.

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como passar dos estados finais de *Compile*(E) ao inicial de *Compile*(F) *sem consumir acções da palavra concatenada*?

Compile(*EF*)

Para aceitar uma palavra que resulta de concatenar uma palavra da linguagem de E e outra da linguagem de F, é necessário "ligar" o AFD que implementa E com o que implementa F.

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como passar dos estados finais de *Compile*(E) ao inicial de *Compile*(F) *sem consumir acções da palavra concatenada*?

Compile(*EF*)

Para aceitar uma palavra que resulta de concatenar uma palavra da linguagem de E e outra da linguagem de F, é necessário "ligar" o AFD que implementa E com o que implementa F.

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como passar dos estados finais de *Compile*(E) ao inicial de *Compile*(F) *sem consumir acções da palavra concatenada?*

Compile(*EF*)

Para aceitar uma palavra que resulta de concatenar uma palavra da linguagem de E e outra da linguagem de F, é necessário "ligar" o AFD que implementa E com o que implementa F.

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como passar dos estados finais de *Compile*(E) ao inicial de *Compile*(F) *sem consumir acções da palavra concatenada*?

Compile(*EF*)

Dificuldade

Como o conjunto dos estados finais pode ter mais que um elemento,

- replica-se em vários o estado inicial de *Compile*(*F*) para os fundir com os finais de *Compile*(*E*)?!
  mas um autómato só tem um estado inicial...
- unificam-se num só os estados finais de *Compile*(*E*) para o fundir com o inicial de de *Compile*(*F*)?!
  mas isso pode alterar a linguagem aceite...

Compile(*EF*)

Dificuldade

Como o conjunto dos estados finais pode ter mais que um elemento,

- replica-se em vários o estado inicial de *Compile*(*F*) para os fundir com os finais de *Compile*(*E*)?!
  mas um autómato só tem um estado inicial...
- unificam-se num só os estados finais de *Compile*(*E*) para o fundir com o inicial de de *Compile*(*F*)?!
  mas isso pode alterar a linguagem aceite...

Compile(*EF*)

Dificuldade

Como o conjunto dos estados finais pode ter mais que um elemento,

- replica-se em vários o estado inicial de *Compile*(*F*) para os fundir com os finais de *Compile*(*E*)?!
  mas um autómato só tem um estado inicial...
- unificam-se num só os estados finais de *Compile*(*E*) para o fundir com o inicial de de *Compile*(*F*)?!
  mas isso pode alterar a linguagem aceite...

$Compile(EF)$

Dificuldade

Como o conjunto dos estados finais pode ter mais que um elemento,

- replica-se em vários o estado inicial de $Compile(F)$ para os fundir com os finais de $Compile(E)$?!
  mas um autómato só tem um estado inicial...
- unificam-se num só os estados finais de $Compile(E)$ para o fundir com o inicial de de $Compile(F)$?!
  mas isso pode alterar a linguagem aceite...

Compile(*EF*)

Dificuldade

Como o conjunto dos estados finais pode ter mais que um elemento,

- replica-se em vários o estado inicial de *Compile*(*F*) para os fundir com os finais de *Compile*(*E*)?!
  mas um autómato só tem um estado inicial...
- unificam-se num só os estados finais de *Compile*(*E*) para o fundir com o inicial de de *Compile*(*F*)?!
  mas isso pode alterar a linguagem aceite...

Compile(*EF*)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$\mathit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$
sendo
$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\}$ e $S_E \cap S_F = \emptyset$

Compile(*EF*)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$\mathit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$
sendo
$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\}$ e $S_E \cap S_F = \emptyset$

Compile(EF)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\textit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\textit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$\textit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$
sendo
$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\}$ e $S_E \cap S_F = \emptyset$

Compile(EF)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$\mathit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$
sendo
$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\}$ e $S_E \cap S_F = \emptyset$

$Compile(EF)$

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de $Compile(E)$ para o inicial de $Compile(F)$ *sem consumir acções* de $w_1 w_2$ por uma *transição-*ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$$Compile(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$$

sendo

$$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\} \text{ e } S_E \cap S_F = \emptyset$$

$Compile(EF)$

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de $Compile(E)$ para o inicial de $Compile(F)$ *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$$Compile(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$$

sendo

$$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\} \text{ e } S_E \cap S_F = \emptyset$$

Compile(*EF*)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$$\mathit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$$

sendo

$$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\} \text{ e } S_E \cap S_F = \emptyset$$

Compile(*EF*)

Relembre-se que

$$\mathcal{L}(EF) = \mathcal{L}(E) \cdot \mathcal{L}(F)$$

Assumindo então

$$\mathit{Compile}(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$\mathit{Compile}(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

como $w_1 w_2 = w_1 \epsilon w_2$, pode-se passar dos estados finais de *Compile*(E) para o inicial de *Compile*(F) *sem consumir acções* de $w_1 w_2$ por uma *transição*-ϵ.

O que é um autómato com transições-ϵ?

Basta incluir ϵ em Σ; a definição é então a de AFD.

$\mathit{Compile}(EF) = \langle S_E \cup S_F, \{\epsilon\} \cup \Sigma_E \cup \Sigma_F, s_E, \delta_\epsilon \cup \delta_E \cup \delta_F, F_F \rangle$
sendo
$\delta_\epsilon = \{(f_E, \epsilon) \mapsto s_F \mid f_E \in F_E\}$ e $S_E \cap S_F = \emptyset$

$Compile(E + F)$

Relembre-se que

$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

é preciso no início decidir se a palavra $w \in \mathcal{L}(E + F)$ vai ser executada:

- por $Compile(E)$ (se $w \in \mathcal{L}(E)$); ou
- por $Compile(F)$ (se $w \in \mathcal{L}(F)$).

Quem será o estado inicial de $Compile(E + F)$?

$Compile(E + F)$

Relembre-se que

$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

é preciso no início decidir se a palavra $w \in \mathcal{L}(E + F)$ vai ser executada:

- por $Compile(E)$ (se $w \in \mathcal{L}(E)$); ou
- por $Compile(F)$ (se $w \in \mathcal{L}(F)$).

Quem será o estado inicial de $Compile(E + F)$?

$Compile(E + F)$

Relembre-se que

$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

é preciso no início decidir se a palavra $w \in \mathcal{L}(E + F)$ vai ser executada:

- por $Compile(E)$ (se $w \in \mathcal{L}(E)$); ou
- por $Compile(F)$ (se $w \in \mathcal{L}(F)$).

Quem será o estado inicial de $Compile(E + F)$?

$Compile(E + F)$

Relembre-se que

$$\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \delta_F, F_F \rangle$$

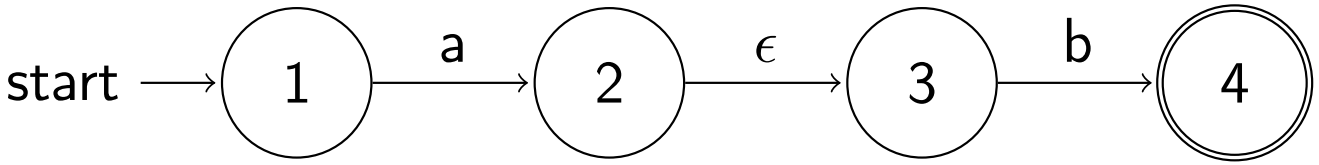
é preciso no início decidir se a palavra $w \in \mathcal{L}(E + F)$ vai ser executada:

- por $Compile(E)$ (se $w \in \mathcal{L}(E)$); ou
- por $Compile(F)$ (se $w \in \mathcal{L}(F)$).

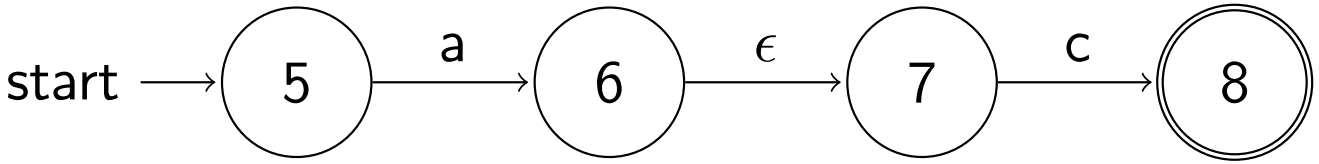
Quem será o estado inicial de $Compile(E + F)$?

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



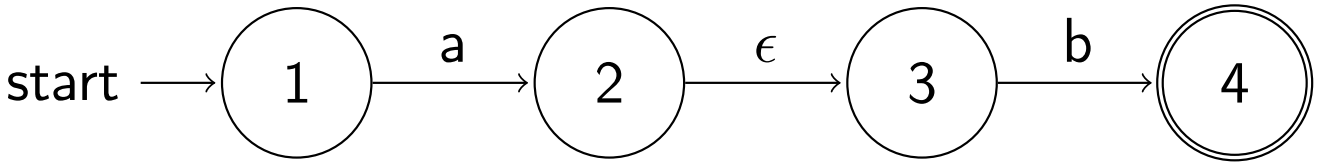
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

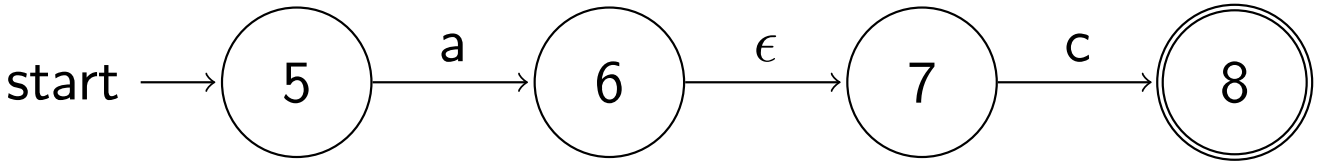
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



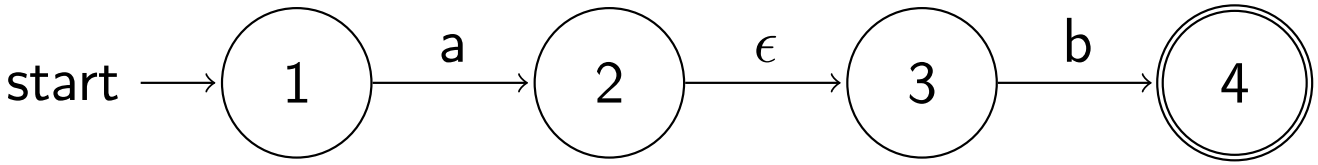
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

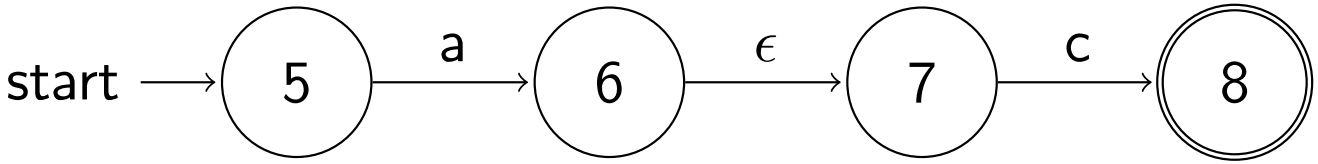
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



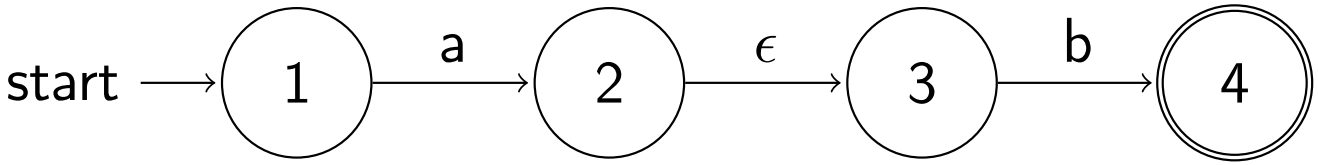
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

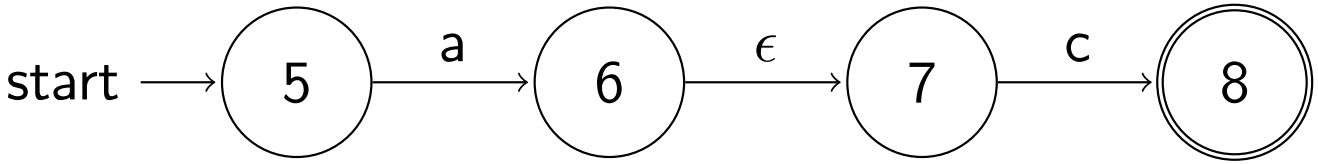
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



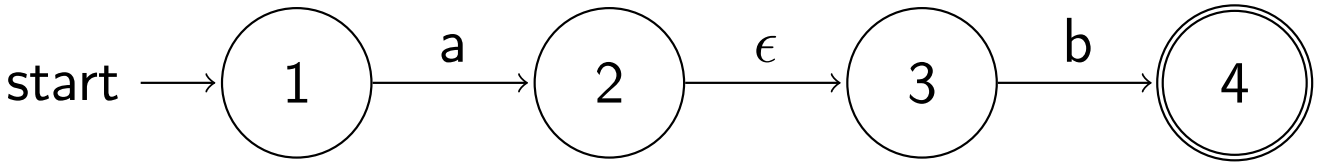
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

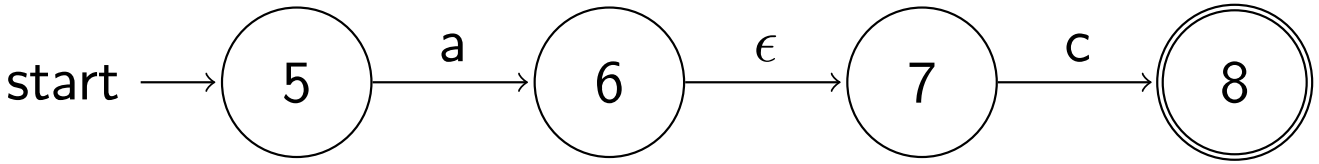
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



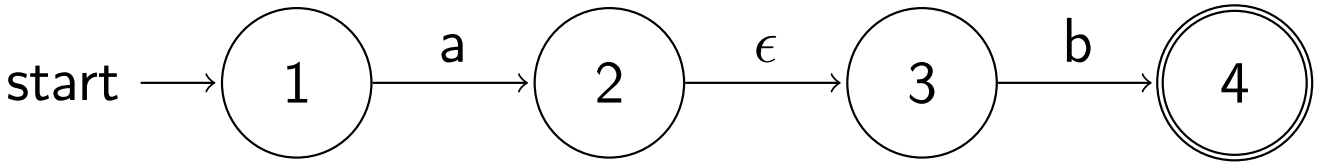
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

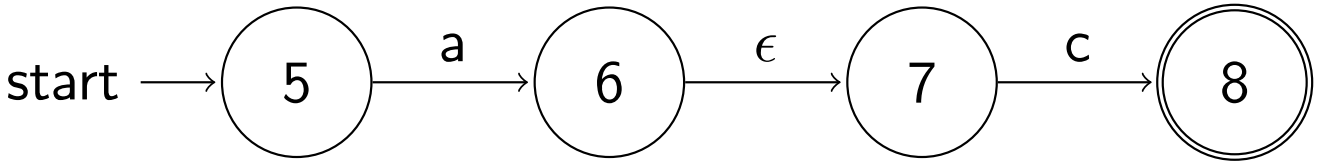
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



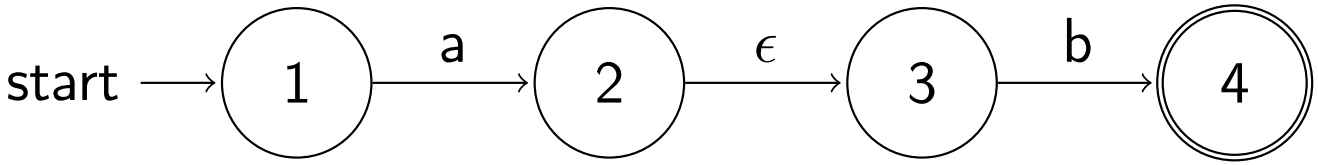
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

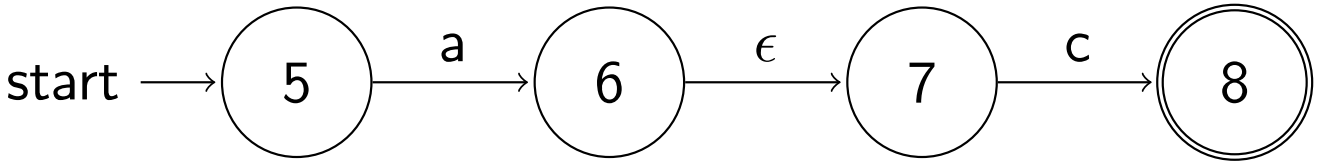
A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

Compile(ab):



Compile(ac):



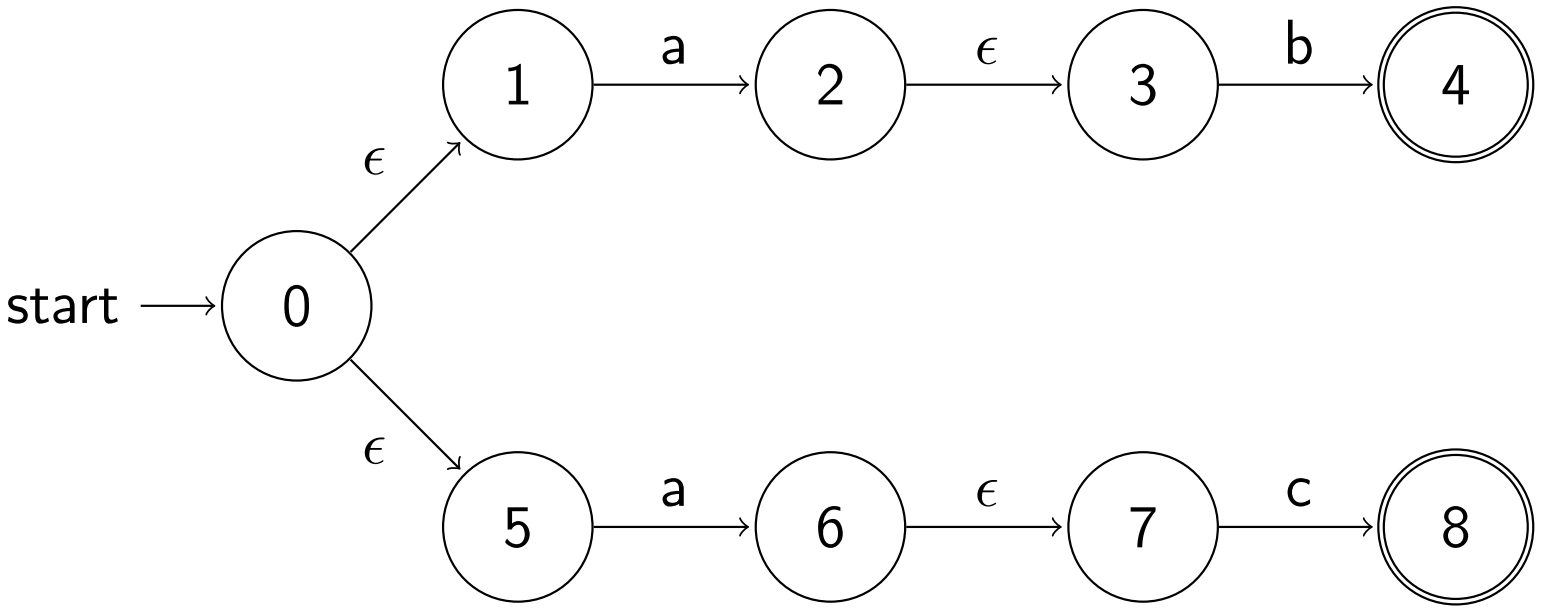
Dada uma palavra $w \in \mathcal{L}((ab + ac)$, o autómato que junta os dois tem que decidir *não-deterministicamente*:

- se $w \in \mathcal{L}(ab)$ e então vai executar *Compile*(ab); ou
- se $w \in \mathcal{L}(ac)$ e então vai executar *Compile*(ac).

A "decisão" pode ser uma transição-ϵ a partir dum novo estado inicial para um dos "antigos" iniciais.

Um exemplo: *Compile*$(ab + ac)$

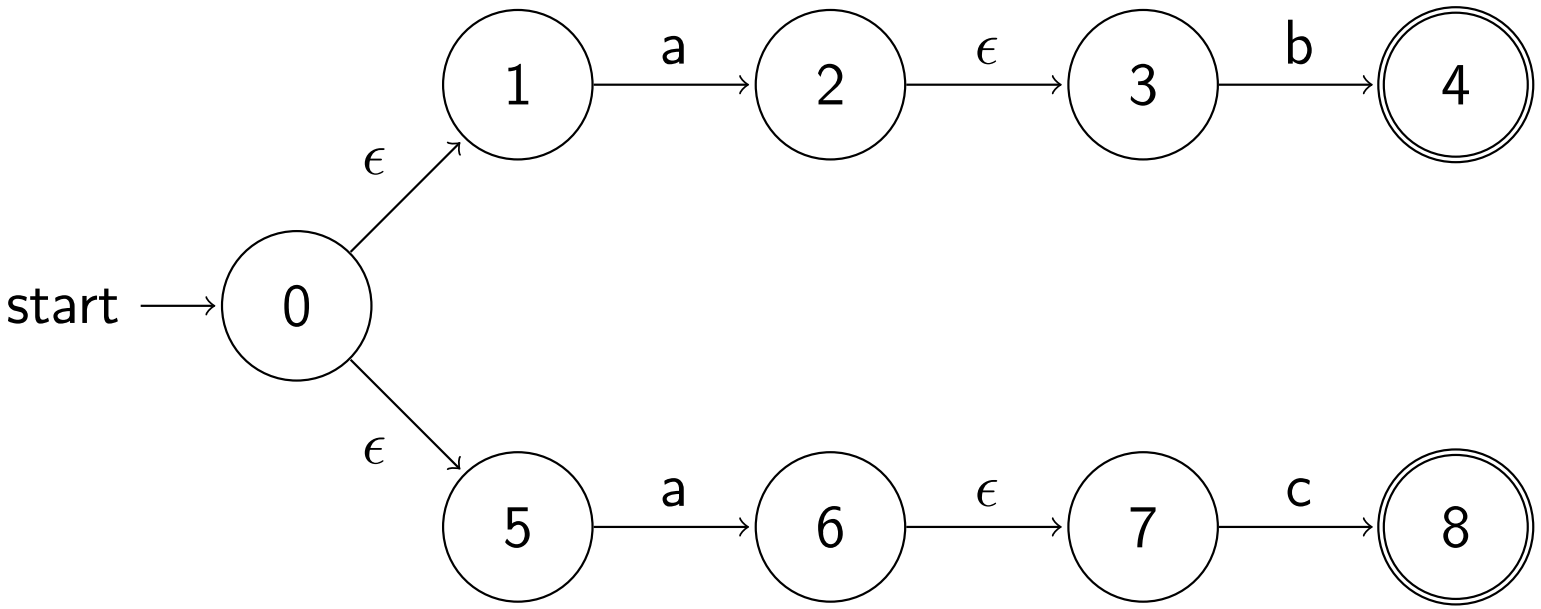
Compile$(ab + ac)$:



- O problema é que este autómato NÃO é determinista...
- É um autómato finito não determinista (AFN).
- Transições não deterministas exigem uma RELAÇÃO de transição, em vez de uma função!

Um exemplo: *Compile*$(ab + ac)$

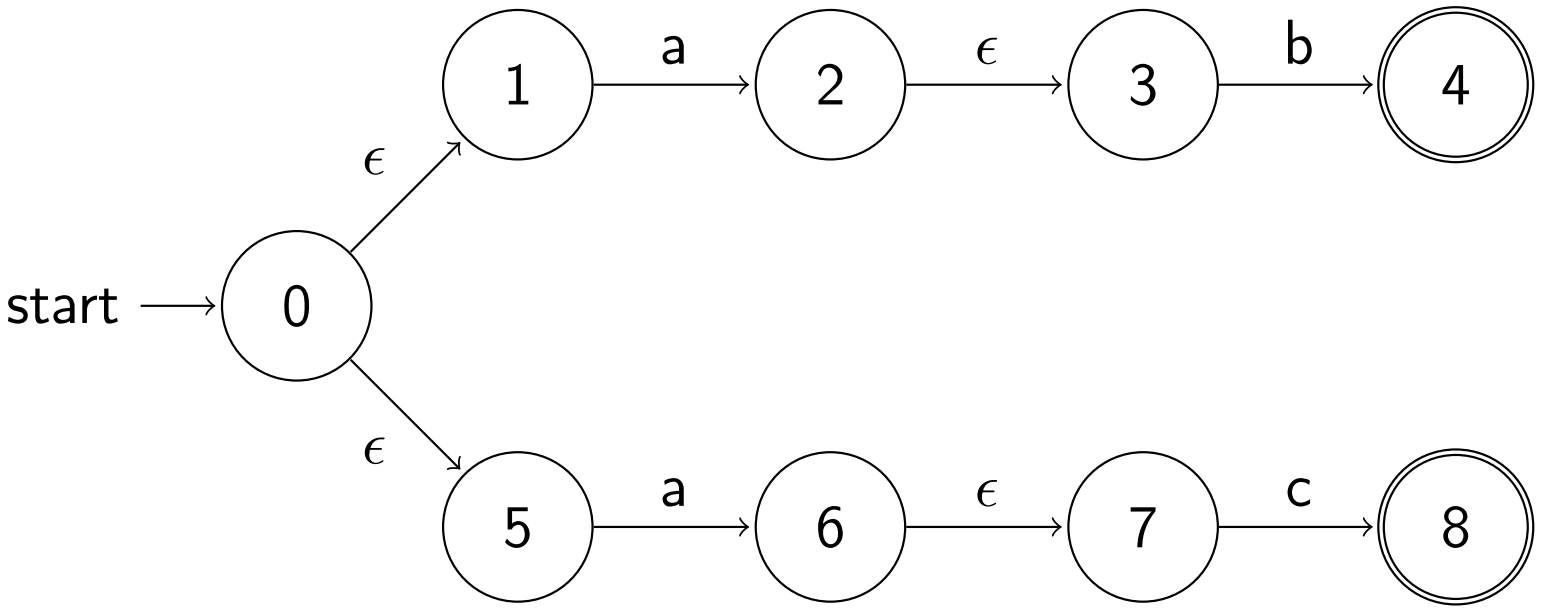
Compile$(ab + ac)$:



- O problema é que este autómato NÃO é determinista...
- É um autómato finito não determinista (AFN).
- Transições não deterministas exigem uma RELAÇÃO de transição, em vez de uma função!

Um exemplo: *Compile*$(ab + ac)$

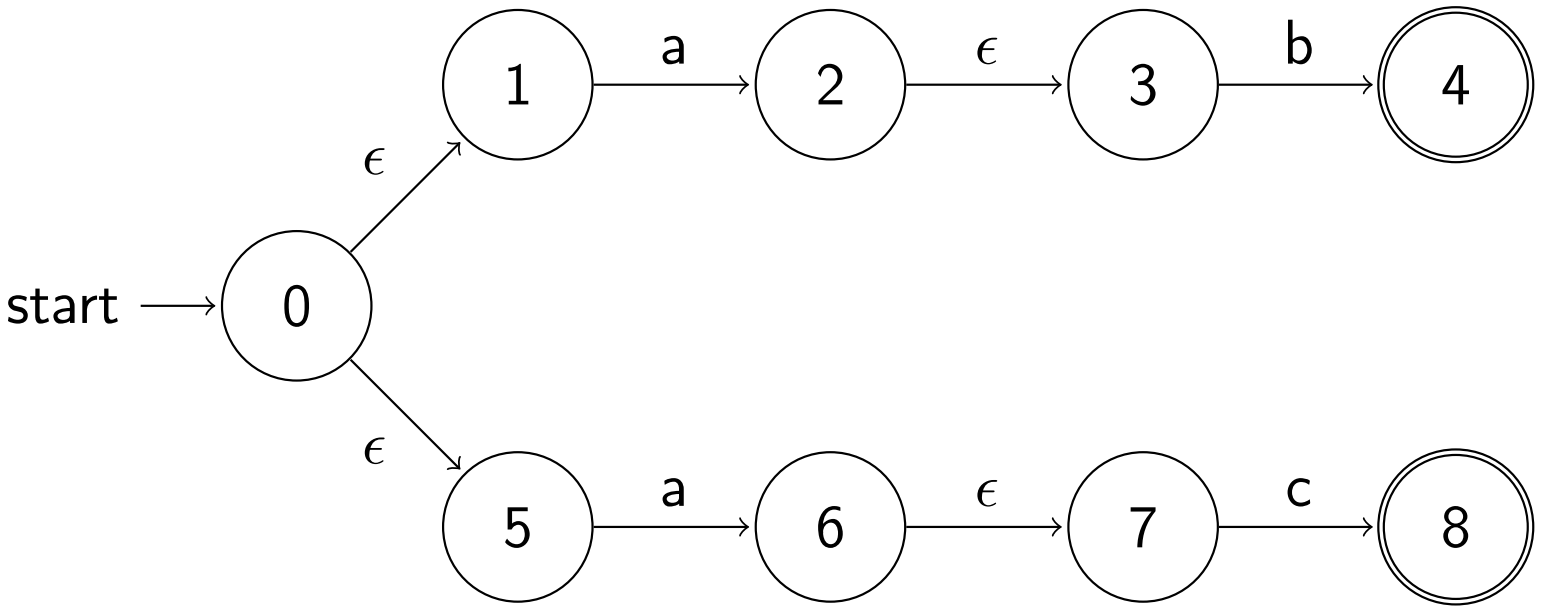
Compile$(ab + ac)$:



- O problema é que este autómato NÃO é determinista...
- É um autómato finito não determinista (AFN).
- Transições não deterministas exigem uma RELAÇÃO de transição, em vez de uma função!

Um exemplo: *Compile*$(ab + ac)$

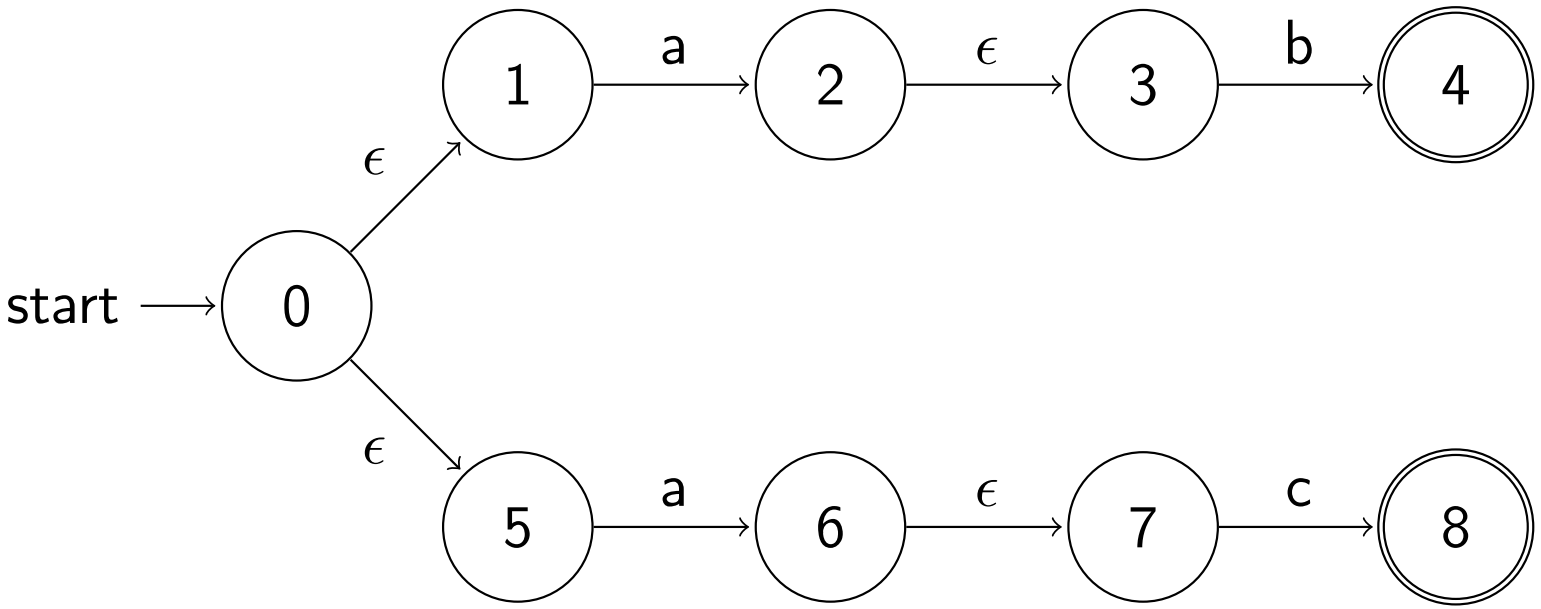
Compile$(ab + ac)$:



- O problema é que este autómato NÃO é determinista...
- É um autómato finito não determinista (AFN).
- Transições não deterministas exigem uma RELAÇÃO de transição, em vez de uma função!

Um exemplo: *Compile*$(ab + ac)$

Compile$(ab + ac)$:



- O problema é que este autómato NÃO é determinista...
- É um autómato finito não determinista (AFN).
- Transições não deterministas exigem uma RELAÇÃO de transição, em vez de uma função!

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

Definição Formal

Autómatos Finitos Não Deterministas

Um AFN é um quíntuplo $A = \langle S, \Sigma, s, \Delta, F \rangle$, sendo:

- S um conjunto finito, dos *estados* de A.
- Σ um conjunto finito, das *acções* de A.
- $s \in S$ é o *estado inicial* de A.
- $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ é a *relação* de *transição* de A.
- $F \subseteq S$ é o conjunto dos *estados finais* (ou de aceitação) de A.

- Apesar de não ser um modelo "realista" por não ser realizável, é muito conveniente, não só por razões "teóricas".
- A especificação de sistemas complexos torna-se mais simples.

AFDs versus AFNs

Autómato que reconhece $(ab + aba)^*$

AFD:

AFN:

AFDs versus AFNs

Autómato que reconhece $(ab + aba)^*$

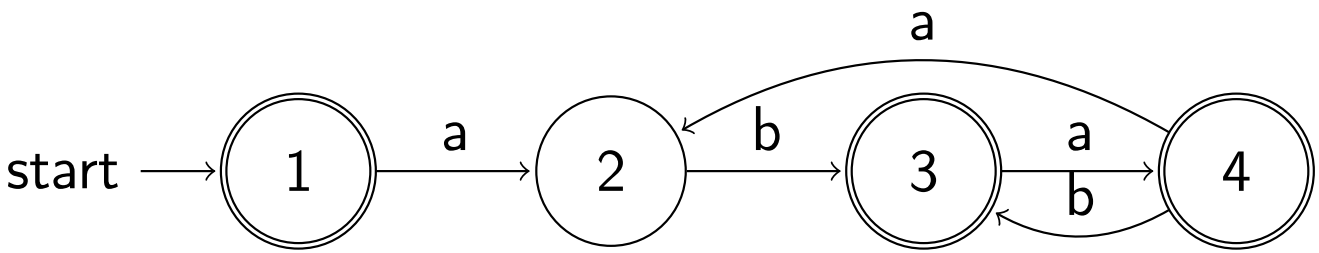
AFD:

AFN:

AFDs versus AFNs

Autómato que reconhece $(ab + aba)^*$

AFD:

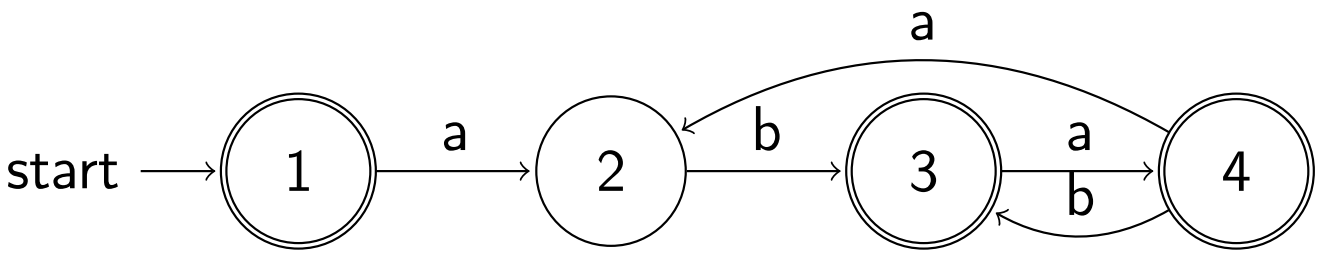


AFN:

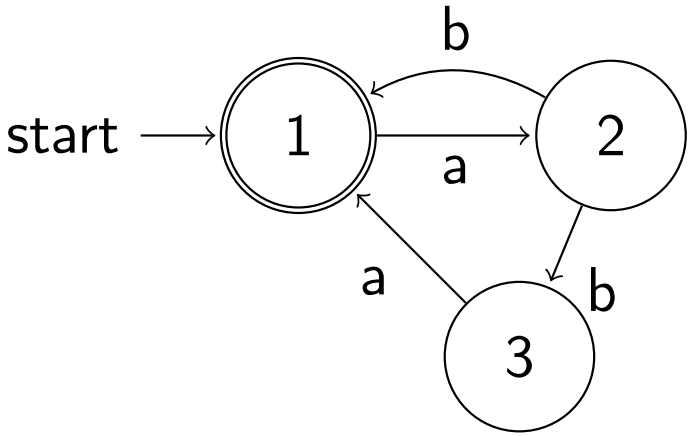
AFDs versus AFNs

Autómato que reconhece $(ab + aba)^*$

AFD:

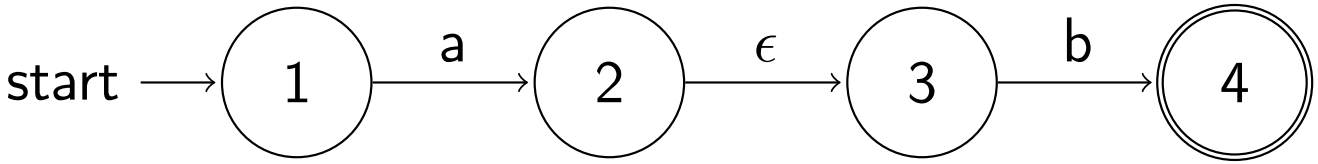


AFN:



Relação de transição extendida: intuição

Considere-se agora o AFN



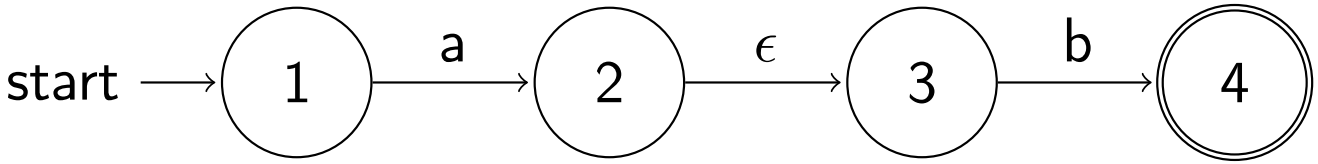
Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\langle 1, |a\epsilon b\rangle \xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: intuição

Considere-se agora o AFN



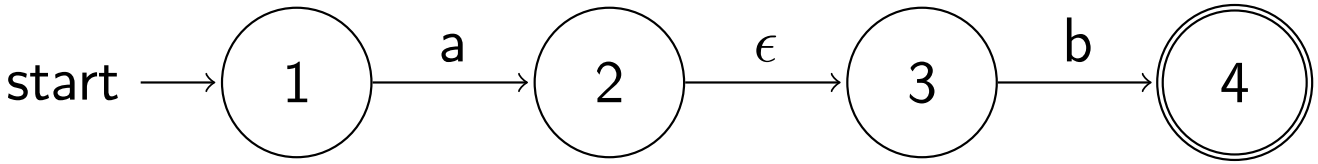
Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\langle 1, |a\epsilon b\rangle \xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: intuição

Considere-se agora o AFN



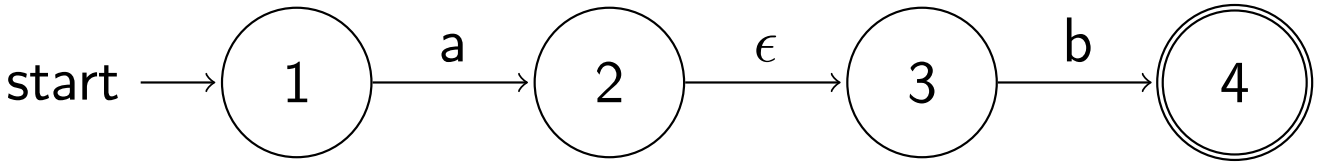
Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\langle 1, |a\epsilon b\rangle \xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: intuição

Considere-se agora o AFN



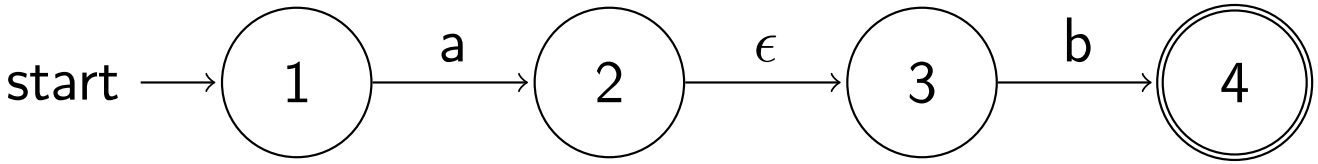
Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\begin{aligned} \langle 1, |a\epsilon b\rangle &\xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \\ &\xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \\ &\xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle \end{aligned}$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: intuição

Considere-se agora o AFN



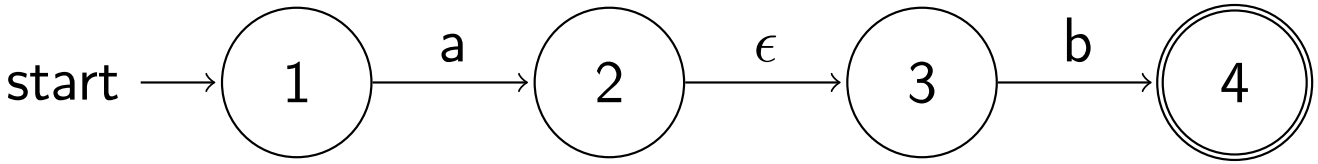
Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\begin{aligned} \langle 1, |a\epsilon b\rangle &\xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \\ &\xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \\ &\xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle \end{aligned}$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: intuição

Considere-se agora o AFN



Como mostrar que aceita a palavra ab? Note que $ab = a\epsilon b$. Então

$$\begin{aligned} \langle 1, |a\epsilon b\rangle &\xrightarrow{(1,a,2)\in\Delta} \langle 2, a|\epsilon b\rangle \\ &\xrightarrow{(2,\epsilon,3)\in\Delta} \langle 3, a\epsilon|b\rangle \\ &\xrightarrow{(3,b,4)\in\Delta} \langle 4, a\epsilon b|\rangle \end{aligned}$$

Define-se a execução de palavras por um AFN (*i.e.*, as computações do autómato não determinista), de forma semelhante ao que se fez para um AFD.

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Relação de transição extendida: definição e exemplo

Dado um AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$, a relação $\Delta^* \subseteq S \times \Sigma^* \to S$ é definida indutivamente:

$$s \in S \Rightarrow (s, \epsilon, s) \in \Delta^*$$
$$(s, a, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, au, s'') \in \Delta^*$$
$$(s, \epsilon, s') \in \Delta \wedge (s', u, s'') \in \Delta^* \Rightarrow (s, u, s'') \in \Delta^*$$

O AFN da página anterior (chamemos-lhe AB) executa ab, pois $(1, ab, 4) \in \Delta^*_{AB}$:

$$(3, b, 4) \in \Delta_{AB} \wedge (4, \epsilon, 4) \in \Delta^*_{AB} \quad \text{logo } (3, b, 4) \in \Delta^*_{AB}$$
$$(2, \epsilon, 3) \in \Delta_{AB} \wedge (3, b, 4) \in \Delta^*_{AB} \quad \text{logo } (2, b, 4) \in \Delta^*_{AB}$$
$$(1, a, 2) \in \Delta_{AB} \wedge (2, b, 4) \in \Delta^*_{AB} \quad \text{logo } (1, ab, 4) \in \Delta^*_{AB}$$

Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?

Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?

Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?

Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?

Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?

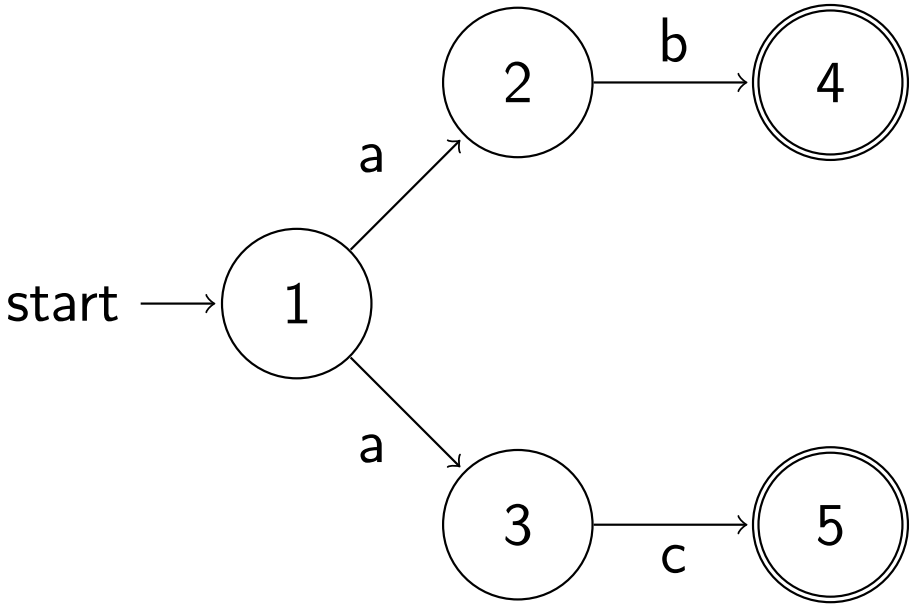
Linguagem de um AFN

Considere-se o AFN $A = \langle S, \Sigma, s, \Delta, F \rangle$. A linguagem aceite é:

$$\mathcal{L}(A) = \{w \in Words(\Sigma) \mid \exists f . f \in F_A \wedge (s_A, w, f) \in \Delta^*\}$$

Como Δ é uma relação (não determinista), ao executar dada palavra o AFN pode ter que *adivinhar* o caminho a tomar!

Exemplo: como executar *ac*?



A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$Compile \subseteq RegExp(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

A função *Compile*

- Dado um alfabeto Σ, seja $\mathcal{AFN}_\Sigma$ o conjunto de todos os AFNs sobre Σ.
- Quer-se definir a função

$$\mathit{Compile} \subseteq \mathit{RegExp}(\Sigma) \to \mathcal{AFN}_\Sigma$$

- Como um AFD (com transições-ϵ) é um (caso particular de) um AFN, as traduções definidas para os casos anteriores são ainda válidas.
- Falta definir a função para dois casos: $(E + F)$ e E^*.

$Compile(E + F)$

Relembre-se que $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$

Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \Delta_F, F_F \rangle$$

Tem-se

$$Compile(E+F) = \langle \{i\} \cup S_E \cup S_F, \Sigma_E \cup \Sigma_F, i, \Delta_\epsilon \cup \Delta_E \cup \Delta_F, F_E \cup F_F \rangle$$

considerando:

- $(i \notin S_E \cup S_F) \wedge (S_E \cap S_F = \emptyset)$
- $\Delta_\epsilon = \{(i, \epsilon, s_E), (i, \epsilon, s_F)\}$

$Compile(E + F)$

Relembre-se que $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \Delta_F, F_F \rangle$$

Tem-se

$$Compile(E+F) = \langle \{i\} \cup S_E \cup S_F, \Sigma_E \cup \Sigma_F, i, \Delta_\epsilon \cup \Delta_E \cup \Delta_F, F_E \cup F_F \rangle$$

considerando:

- $(i \notin S_E \cup S_F) \wedge (S_E \cap S_F = \emptyset)$
- $\Delta_\epsilon = \{(i, \epsilon, s_E), (i, \epsilon, s_F)\}$

$Compile(E + F)$

Relembre-se que $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \Delta_F, F_F \rangle$$

Tem-se

$$Compile(E+F) = \langle \{i\} \cup S_E \cup S_F, \Sigma_E \cup \Sigma_F, i, \Delta_\epsilon \cup \Delta_E \cup \Delta_F, F_E \cup F_F \rangle$$

considerando:

- $(i \notin S_E \cup S_F) \wedge (S_E \cap S_F = \emptyset)$
- $\Delta_\epsilon = \{(i, \epsilon, s_E), (i, \epsilon, s_F)\}$

$Compile(E + F)$

Relembre-se que $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \Delta_F, F_F \rangle$$

Tem-se

$$Compile(E+F) = \langle \{i\} \cup S_E \cup S_F, \Sigma_E \cup \Sigma_F, i, \Delta_\epsilon \cup \Delta_E \cup \Delta_F, F_E \cup F_F \rangle$$

considerando:

- $(i \notin S_E \cup S_F) \wedge (S_E \cap S_F = \emptyset)$
- $\Delta_\epsilon = \{(i, \epsilon, s_E), (i, \epsilon, s_F)\}$

$Compile(E + F)$

Relembre-se que $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
Assumindo então

$$Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$$
$$Compile(F) = \langle S_F, \Sigma_F, s_F, \Delta_F, F_F \rangle$$

Tem-se

$$Compile(E+F) = \langle \{i\} \cup S_E \cup S_F, \Sigma_E \cup \Sigma_F, i, \Delta_\epsilon \cup \Delta_E \cup \Delta_F, F_E \cup F_F \rangle$$

considerando:

- $(i \notin S_E \cup S_F) \wedge (S_E \cap S_F = \emptyset)$
- $\Delta_\epsilon = \{(i, \epsilon, s_E), (i, \epsilon, s_F)\}$

$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN $Compile(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter $Compile(EE)$, liga-se os estados finais de $Compile(E)$ ao inicial de $Compile(E)$, com uma transição-ϵ;
  - para obter $Compile(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN $Compile(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter $Compile(EE)$, liga-se os estados finais de $Compile(E)$ ao inicial de $Compile(E)$, com uma transição-ϵ;
  - para obter $Compile(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Compile(E^*)

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

- Como para qualquer L se tem $\epsilon \in L$, então o estado inicial do autómato que se vai construir tem também que ser final.
- Pode-se pensar em E^* como $\epsilon + E + EE + EEE + \cdots$
- Então, para construir o AFN *Compile*$(E)^*$, tem que se usar as ideias da *concatenação* e da *soma*:
  - para obter *Compile*(EE), liga-se os estados finais de *Compile*(E) ao inicial de *Compile*(E), com uma transição-ϵ;
  - para obter *Compile*$(E^n + E^{n+1})$, cria-se um novo estado inicial que se liga aos estados iniciais de cada um, com uma transição-ϵ

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

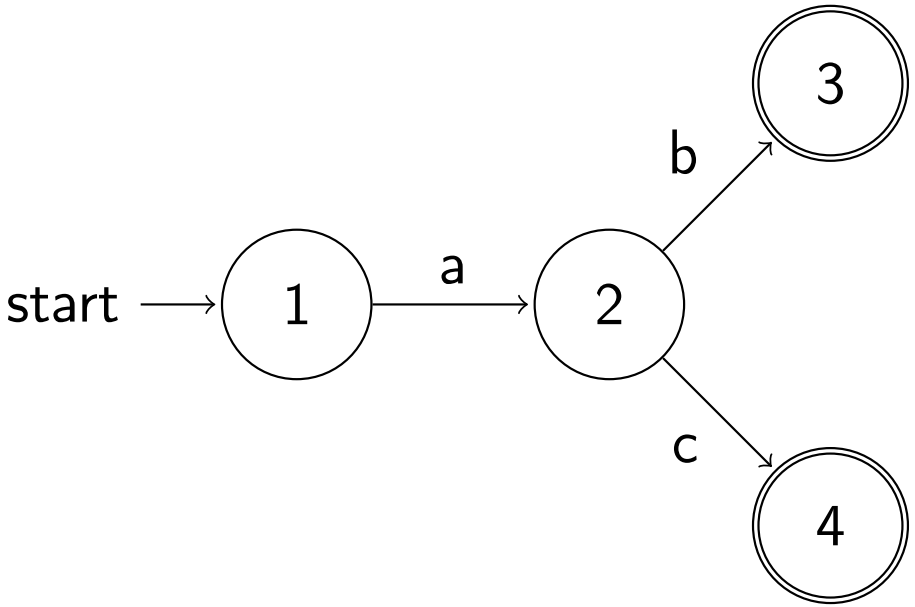
Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

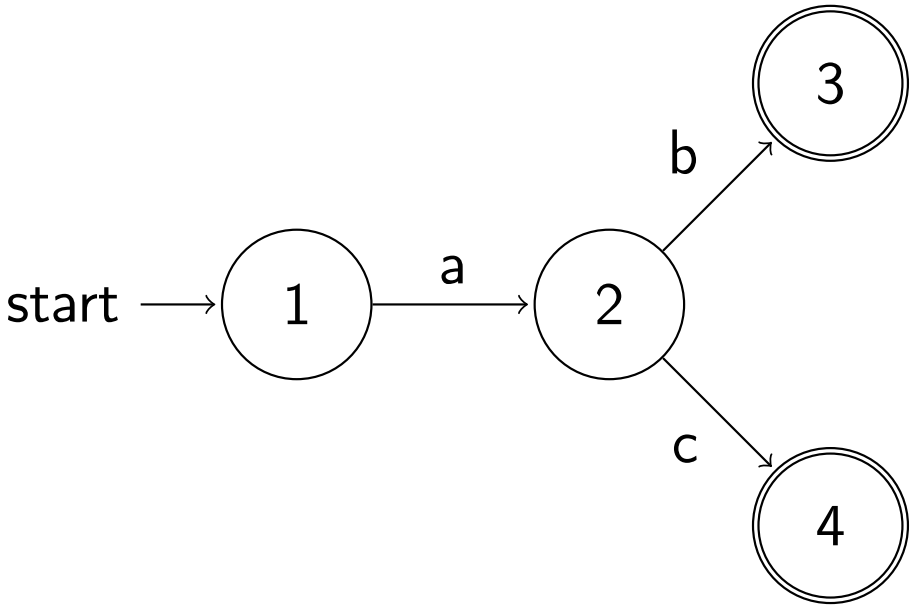


Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

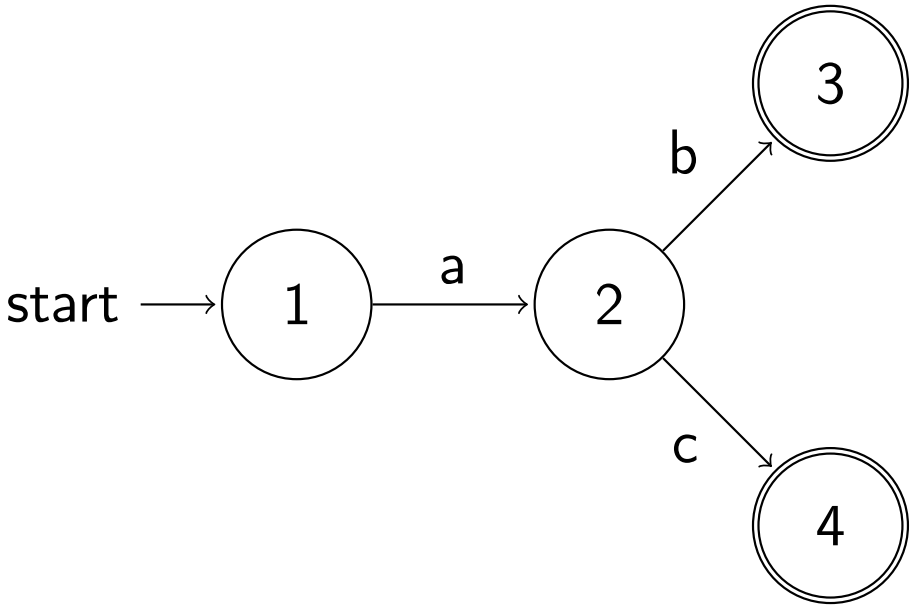


Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

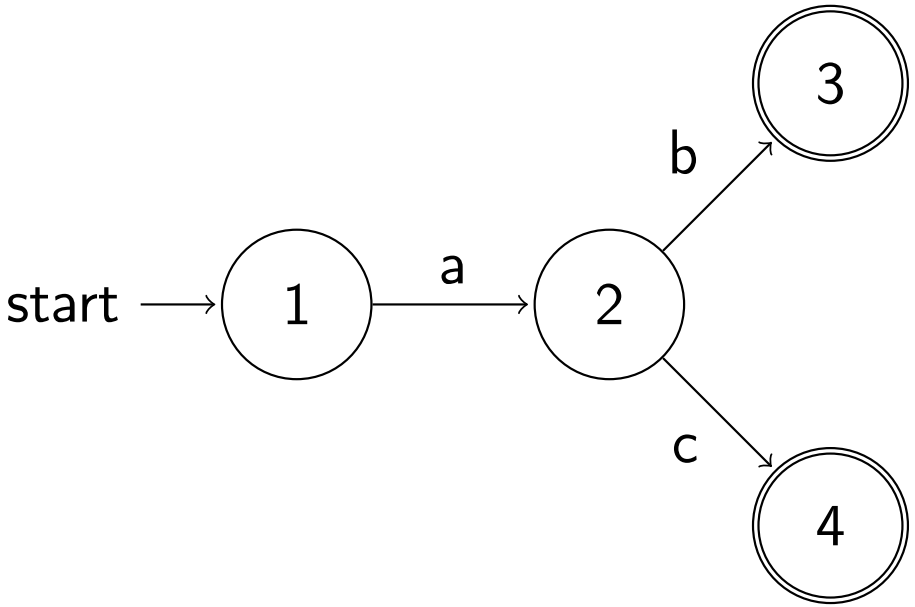


Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

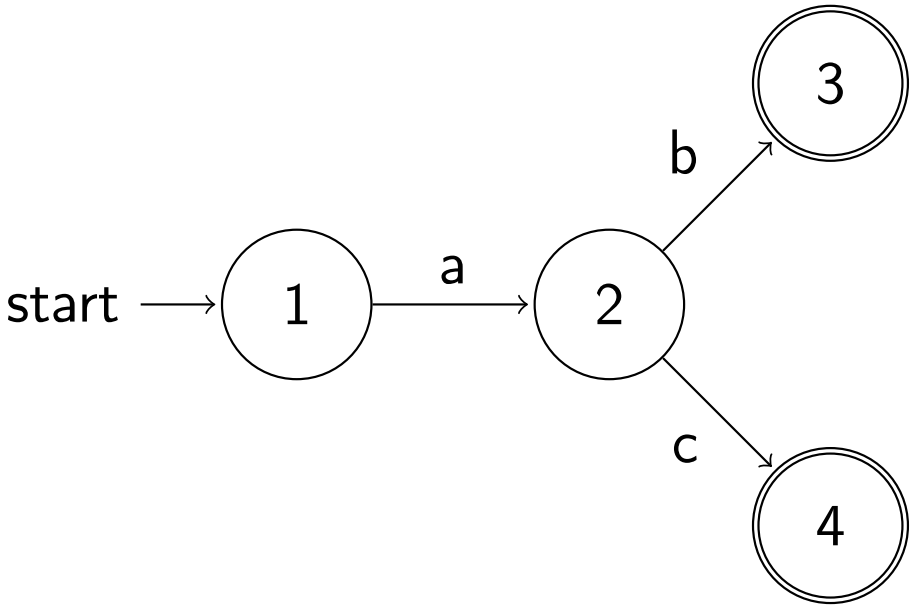


Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):

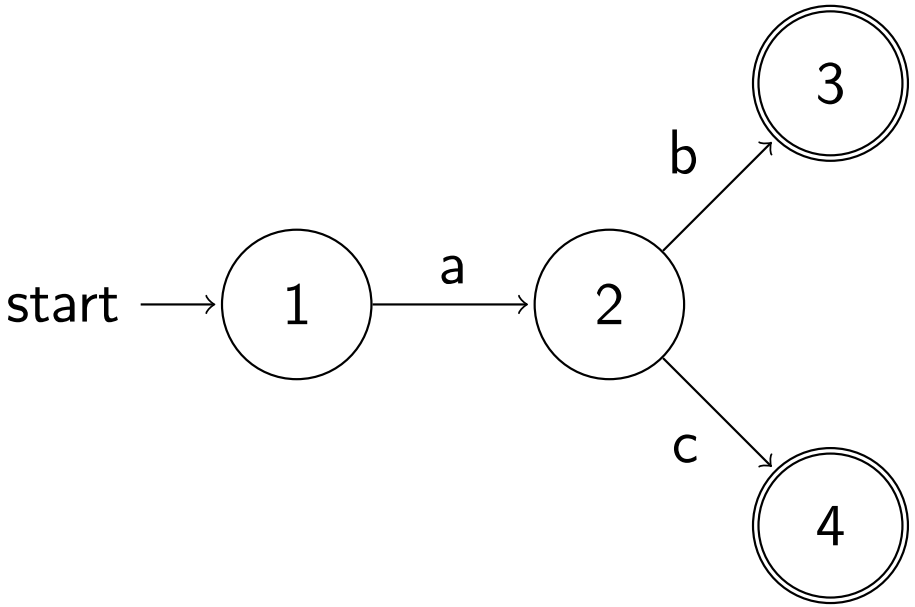


Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

Um exemplo: considere-se $(a(b+c))^*$

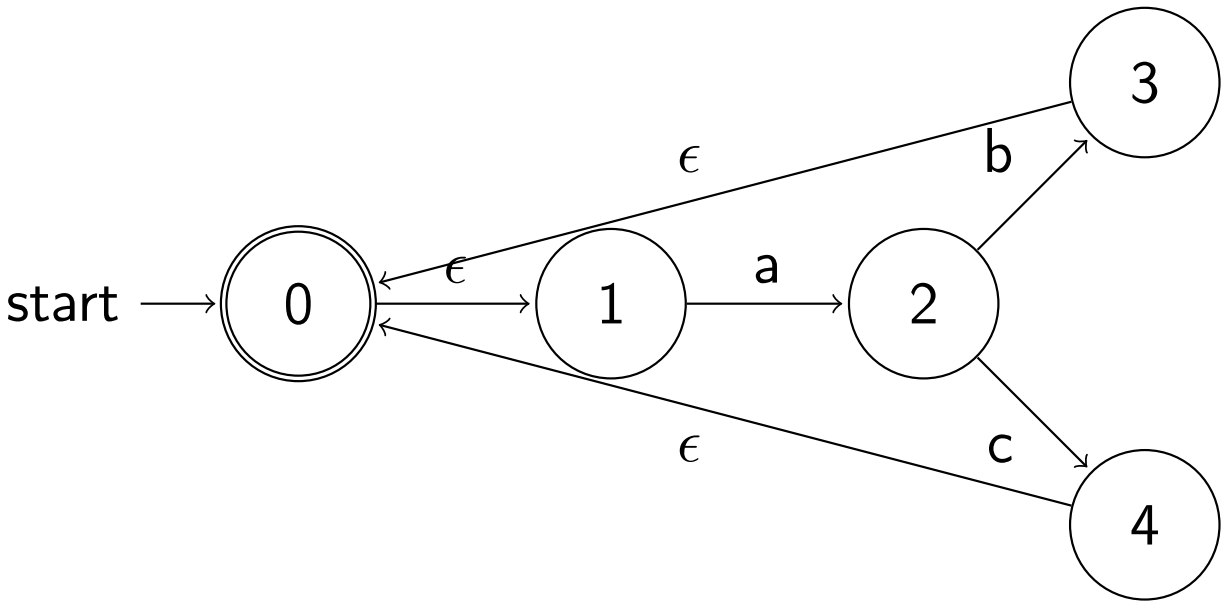
Assume-se o AFD seguinte que aceita $\mathcal{L}(a(b+c))$ (não é a tradução):



Como modificá-lo para aceitar $\mathcal{L}(a(b+c)^*)$?

- acrescenta-se um novo estado inicial, que se liga ao estado 1 com uma transição-ϵ (para tratar a soma);
- ligam-se cada antigo estado final ao novo inicial (também final) com uma transição-ϵ (para tratar a concatenação).

AFN que aceita $(a(b+c))^*$



$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$

Assumindo então $Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$ Tem-se

$$Compile(E^*) = \langle \{i\} \cup S_E, \Sigma_E, i, \Delta_\epsilon \cup \Delta_E, \{i\} \rangle$$

considerando:

- $i \notin S_E$
- $\Delta_\epsilon = \{(i, \epsilon, s_E)\} \cup \{(f, \epsilon, i) \mid f \in S_F\}$

$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$
Assumindo então $Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$ Tem-se

$$Compile(E^*) = \langle \{i\} \cup S_E, \Sigma_E, i, \Delta_\epsilon \cup \Delta_E, \{i\} \rangle$$

considerando:

- $i \notin S_E$
- $\Delta_\epsilon = \{(i, \epsilon, s_E)\} \cup \{(f, \epsilon, i) \mid f \in S_F\}$

$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$
Assumindo então $Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$ Tem-se

$$Compile(E^*) = \langle \{i\} \cup S_E, \Sigma_E, i, \Delta_\epsilon \cup \Delta_E, \{i\} \rangle$$

considerando:

- $i \notin S_E$
- $\Delta_\epsilon = \{(i, \epsilon, s_E)\} \cup \{(f, \epsilon, i) \mid f \in S_F\}$

$Compile(E^*)$

Relembre-se que $\mathcal{L}(E^*) = \mathcal{L}(E)^*$
Assumindo então $Compile(E) = \langle S_E, \Sigma_E, s_E, \Delta_E, F_E \rangle$ Tem-se

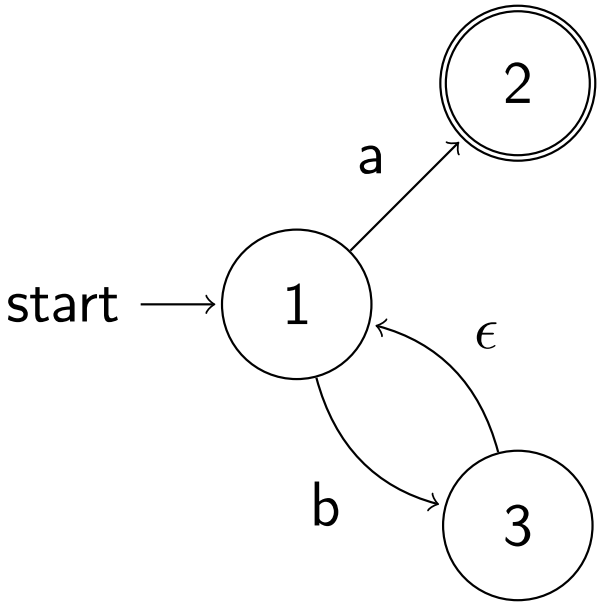
$$Compile(E^*) = \langle \{i\} \cup S_E, \Sigma_E, i, \Delta_\epsilon \cup \Delta_E, \{i\} \rangle$$

considerando:

- $i \notin S_E$
- $\Delta_\epsilon = \{(i, \epsilon, s_E)\} \cup \{(f, \epsilon, i) \mid f \in S_F\}$

É mesmo necessário acrescentar um novo estado inicial?

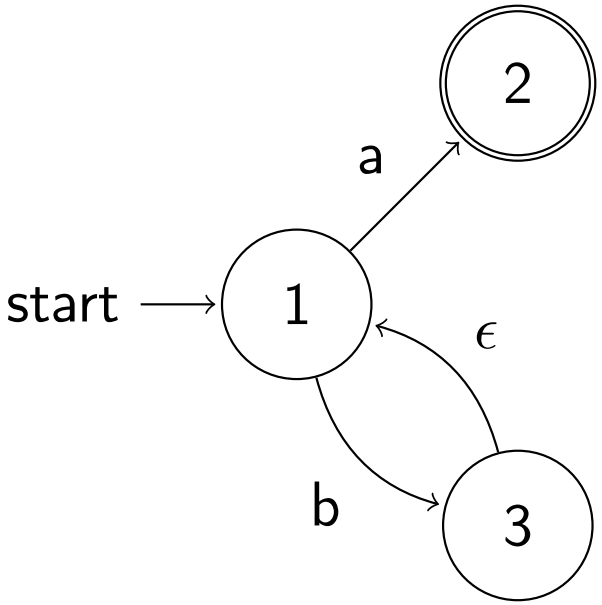
Considere-se o seguinte AFD com transições-ϵ que reconhece a linguagem da expressão regular $a + b^+a$.



Para obter um AFD que reconheça $(a + b^+a)^*$ não basta ligarmos o antigo final ao inicial por ϵ e tornar o inicial em final?
Não! Passaria a aceitar b^*.

É mesmo necessário acrescentar um novo estado inicial?

Considere-se o seguinte AFD com transições-ϵ que reconhece a linguagem da expressão regular $a + b^+a$.

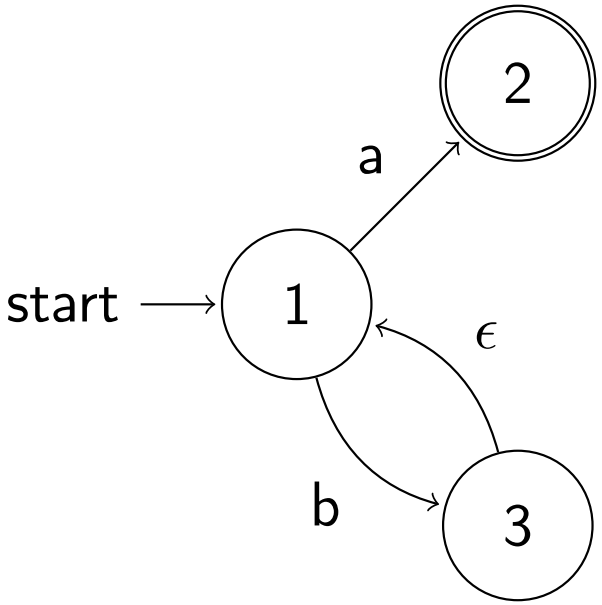


Para obter um AFD que reconheça $(a + b^+a)^*$ não basta ligarmos o antigo final ao inicial por ϵ e tornar o inicial em final?

Não! Passaria a aceitar b^*.

É mesmo necessário acrescentar um novo estado inicial?

Considere-se o seguinte AFD com transições-ϵ que reconhece a linguagem da expressão regular $a + b^+a$.



Para obter um AFD que reconheça $(a + b^+a)^*$ não basta ligarmos o antigo final ao inicial por ϵ e tornar o inicial em final?
Não! Passaria a aceitar b^*.

Exercício

- Traduzir $(a + b)(ad + ae)$ num AFN
- Definir directamente um AFN que aceita $(a + b)(ad + ae)$

Exercício

- Traduzir $(a + b)(ad + ae)$ num AFN
- Definir directamente um AFN que aceita $(a + b)(ad + ae)$