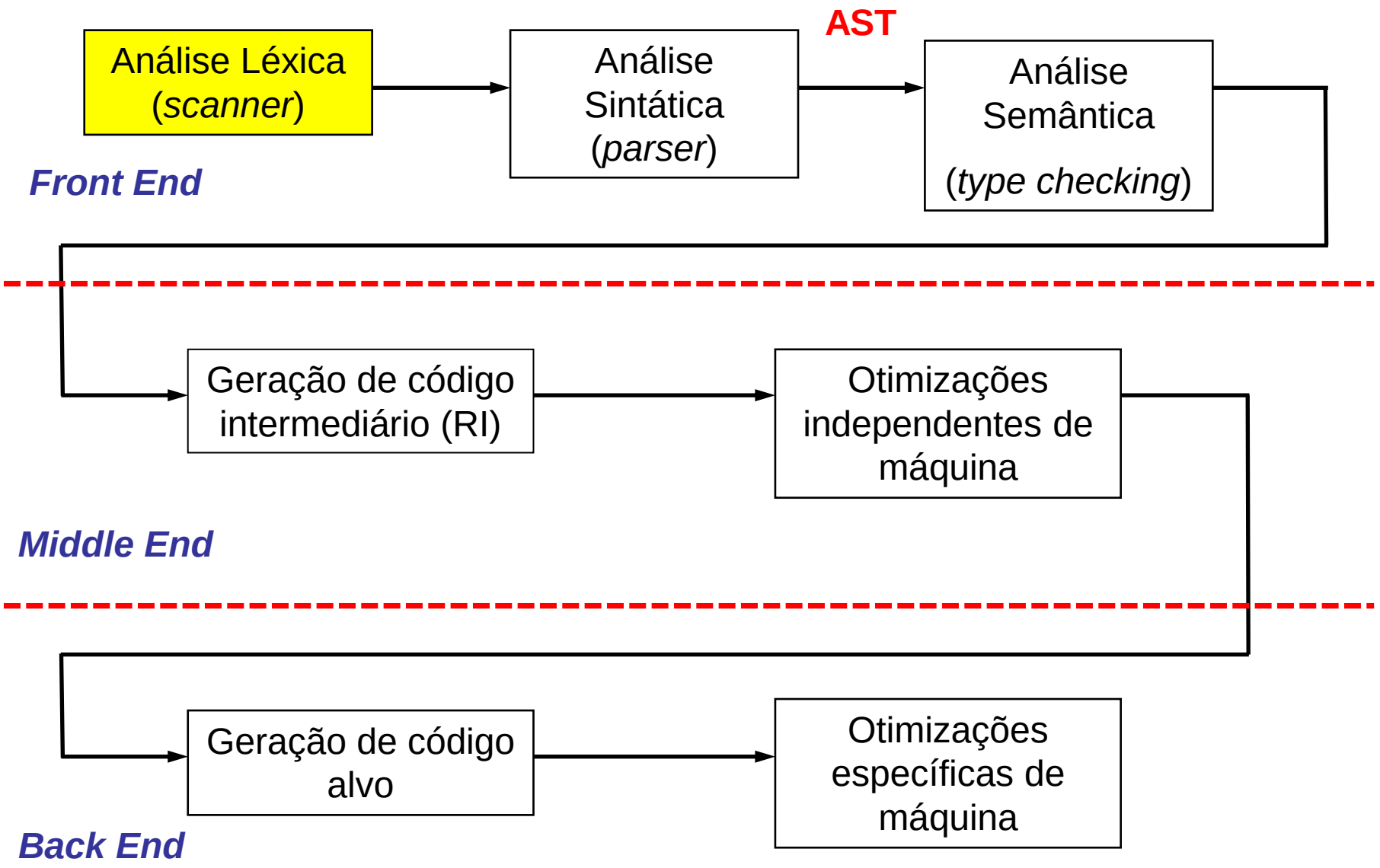
Análise Léxica

Fluxo do Compilador



Linguagem de Programação

Uma linguagem de programação, de uma forma genérica, pode ser definida como uma linguagem $L \subseteq \Sigma^*$, onde o alfabeto Σ é formado pelos símbolos da tabela ASCII e as palavras $x \in L$, são formadas de acordo com regras específicas de cada linguagem de programação.

Analisador Léxico

- Recebe uma seqüência de caracteres e produz uma seqüência de palavras chaves, pontuação e nomes

- Descarta comentários e espaços em branco

- Cada elemento produzido pelo analisador léxico recebe o nome de **TOKEN**

- Cada **TOKEN** é uma cadeia que pertence a linguagem que se deseja compilar, onde a construção de cada TOKEN obedece a regras que são aplicadas ao alfabeto Σ

- Cada instância de um **TOKEN** recebe o nome de lexema

Tokens

Tipos de Tokens	Exemplos (lexemas)
ID	`foo n14 last`
NUM	`73 0 00 515 082`
REAL	`66.1 .5 10. 1e67 5.5e-10`
IF	`if`
COMMA	`,`
NOTEQ	`!=`
LPAREN	`(`

Não-Tokens

comment	`/* try again */`
preprocessor directive	`#include<stdio.h>`
preprocessor directive	`#define NUMS 5, 6`
macro	`NUMS`
blanks, tabs, and newlines	

Obs.: O pré-processador passa pelo programa antes do léxico

Exemplo

```
float match0(char *s) /* find a zero */
  {if (!strncmp(s, "0.0", 3))
    return 0.;
 }
```

Retorno do analisador léxico:

FLOAT	ID(match0)	LPAREN	CHAR	STAR	ID(s)
RPAREN	LBRACE	IF	LPAREN	BANG	ID(strncmp)
LPAREN	ID(s)	COMMA	STRING(0.0)	COMMA	NUM(3)
RPAREN	RPAREN	RETURN	REAL(0.0)	SEMI	RBRACE
EOF					

Analisador Léxico

- **Alguns tokens têm um valor semântico associados a eles:**
  - IDs e NUMs (identificadores e números)

- **Como são descritas as regras léxicográficas?**
  Descrição de identificadores em C++ na língua portuguesa:

  - Um IDENTIFICADOR é uma seqüência de letras e dígitos; o primeiro caractere deve ser uma letra. O símbolo *underscore* _ conta como uma letra. Maiúsculas e minúsculas são diferentes.
  - Se uma dada cadeia foi analisada para formar ***tokens*** até um dado caractere, o próximo ***token*** é tal que deve incluir a maior cadeia de caracteres que podem possivelmente formar um ***token***.
  - Espaços, tabulações, fim-de-linha e comentários são ignorados, pois os mesmos servem para separar ***tokens***.
  - Alguns espaços em branco são necessários para separar identificadores, palavras reservadas e constantes que estejam próximos.

- **Como os *tokens* são especificados?**

Analisador Léxico

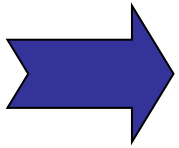
- Um linguagem L é um conjunto de *strings* (palavras) ($L \subseteq \Sigma^*$)
- Uma *string* (palavra) é uma sequência de símbolos (cadeia)
- Estes símbolos estão definidos em um alfabeto finito (Σ)
- Ex: Linguagem C ou Pascal, linguagem dos primos, etc
- Deseja-se poder dizer se uma *string* (palavra) está ou não em uma linguagem, isto é:

dada uma cadeia x sobre Σ;
dada uma linguagem L sobre Σ^*;
será que $x \in L$?

Analisador Léxico

- Um linguagem L é um conjunto de *strings* (palavras) ($L \subseteq \Sigma^*$)
- Uma *string* (palavra) é uma sequência de símbolos (cadeia)
- Estes símbolos estão definidos em um alfabeto finito (Σ)
- Ex: Linguagem C ou Pascal, linguagem dos primos, etc
- Deseja-se poder dizer se uma *string* (palavra) está ou não em uma linguagem, isto é:

dada uma cadeia x sobre Σ;
dada uma linguagem L sobre Σ^*;
será que $x \in L$?



Expresões Regulares

Expressões Regulares

Expressão Regular é um método formal de se especificar um padrão de texto.

É formada por uma composição de símbolos, caracteres com funções especiais, que, agrupados entre si e com caracteres literais, formam uma sequência; uma expressão.

Essa expressão formada, define então, uma outra linguagem.

A expressão é interpretada como uma regra que indicará sucesso se uma entrada de dados qualquer casar com essa regra, ou seja, obedecer exatamente a todas as suas condições.

Pode-se utilizar um autômato finito para dizer se uma determinada palavra pertence ou não a linguagem gerada por uma expressão regular.

Expressões Regulares

São uma maneira de procurar um texto, com base em variações do mesmo.

São uma maneira de um programador especificar padrões complexos que podem ser procurados e casados em uma cadeia de caracteres.

São uma forma de especificar os elementos formadores de uma linguagem de programação.

Expressões Regulares

Símbolo: Para cada símbolo ***a*** no alfabeto da linguagem, a expressão regular ***a*** representa a linguagem contendo somente a string ***a***.

Alternação: Dadas duas expressões regulares ***M*** e ***N***, o operador de alternação (**|**) gera uma nova expressão ***M*** **|** ***N***. Uma string está na linguagem de ***M*** **|** ***N*** se ela está na linguagem de ***M*** ou na linguagem de ***N***.

Concatenação: Dadas duas expressões regulares ***M*** e ***N***, o operador de concatenação (·) gera uma nova expressão ***M*** · ***N***. Uma string está na linguagem de ***M*** · ***N*** se ela é a concatenação de quaisquer duas strings α e β, tal que α está na linguagem de ***M*** e β está na linguagem de ***N***.

Expressões Regulares

Símbolo: Para cada símbolo ***a*** no alfabeto da linguagem, a expressão regular ***a*** representa a linguagem contendo somente a string ***a***.

Dado um alfabeto $\Sigma = \{a_1, a_2, \ldots, a_n\}$, os símbolos $a_1, a_2, \ldots, a_n$ são Expressões Regulares que correspondem às linguagens $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$, respectivamente.

Expressões Regulares

Alternação: Dadas duas expressões regulares ***M*** e ***N***, o operador de alternação (**|**) gera uma nova expressão ***M*** **|** ***N***. Uma string está na linguagem de ***M*** **|** ***N*** se ela está na linguagem de ***M*** ou na linguagem de ***N***.

Se e_1 e e_2 são ER que correspondem às linguagens L_1 e L_2, então e_1 **|** e_2 é igual a $e_1 \cup e_2$ que é uma expressão regular que corresponde a $L_1 \cup L_2$.

(**a** **|** **b**) = ?

Expressões Regulares

Alternação: Dadas duas expressões regulares ***M*** e ***N***, o operador de alternação (**|**) gera uma nova expressão ***M*** **|** ***N***. Uma string está na linguagem de ***M*** **|** ***N*** se ela está na linguagem de ***M*** ou na linguagem de ***N***.

Se e_1 e e_2 são ER que correspondem às linguagens L_1 e L_2, então e_1 **|** e_2 é igual a $e_1 \cup e_2$ que é uma expressão regular que corresponde a $L_1 \cup L_2$.

(**a** **|** **b**) = { ***a*** , ***b*** }

Expressões Regulares

Concatenação: Dadas duas expressões regulares ***M*** e ***N***, o operador de concatenação (·) gera uma nova expressão ***M*** · ***N***. Uma string está na linguagem de ***M*** · ***N*** se ela é a concatenação de quaisquer duas strings α e β, tal que α está na linguagem de ***M*** e β está na linguagem de ***N***.

Se e_1 e e_2 são ER que correspondem às linguagens L_1 e L_2, então $e_1 \cdot e_2$

é uma expressão regular que corresponde a $L_1 \cdot L_2$.

(**a** | **b**) · **a** = ?

Expressões Regulares

Concatenação: Dadas duas expressões regulares ***M*** e ***N***, o operador de concatenação (·) gera uma nova expressão ***M*** · ***N***. Uma string está na linguagem de ***M*** · ***N*** se ela é a concatenação de quaisquer duas strings α e β, tal que α está na linguagem de ***M*** e β está na linguagem de ***N***.

Se e_1 e e_2 são ER que correspondem às linguagens L_1 e L_2, então $e_1 . e_2$

é uma expressão regular que corresponde a $L_1 . L_2$.

(a | b) · a = { ***aa*** , ***ba***}

Expressões Regulares

- **Epsilon:** A expressão regular **ε** representa a linguagem cuja única string é a vazia. Ex.:

(a · b) | ε = ?

- **Repetição:** Dada uma expressão regular ***M***, seu fecho de Kleene é ***M****. Uma string está em ***M**** se ela é a concatenação de zero ou mais strings, todas em ***M***. Ex.:

((a | b) · a)* = ?

Expressões Regulares

- **Epsilon:** A expressão regular **ε** representa a linguagem cuja única string é a vazia. Ex.:

(a · b) | ε = {"", "ab"}

- **Repetição:** Dada uma expressão regular ***M***, seu fecho de Kleene é ***M****. Uma string está em ***M**** se ela é a concatenação de zero ou mais strings, todas em ***M***. Ex.:

((a | b) · a)* = **{"", "aa", "ba", "aaaa", "baaa", "aaba", "baba", "aaaaaa", …}**

Expressões Regulares

Informalmente, o que gera cada uma das expressões regulares a seguir?

(0 | 1)* · 0 = ?

b*(abb*)*(a|ε) = ?

(a|b)*aa(a|b)* = ?

Expressões Regulares

Informalmente, o que gera cada uma das expressões regulares a seguir?

(0 | 1)* · 0 =

- Números binários múltiplos de 2.

b*(**abb***)*(**a**|ε) **=**

- Strings de a's e b's sem a's consecutivos.

(a|b)*aa(a|b)* **=**

- Strings de a's e b's com a's consecutivos.

Notação para Expressões Regulares

Metacaracteres: **. ? * + | [ ] ( )**

Metacaractere	Nome	Função
.	Ponto	Um caractere qualquer
[...]	Lista	Lista de caracteres permitidos
?	Opcional	Zero ou um
*	Asterisco	Zero, um ou mais
+	Mais	Um ou mais
|	Ou	Ou um ou outro
(...)	Grupo	Delimita um grupo

Notação para Expressões Regulares

a Um símbolo denota a si próprio.

ε A palavra vazia.

M* | *N Alternação: escolher *M* ou *N*.

M* · *N Concatenação, a expressão *M* seguida pela expressão *N*.

MN Outra forma de expressar a concatenação.

M* Repetição: zero, uma ou mais vezes.

***M*⁺** Repetição: uma ou mais vezes.

***M*?** Opcional: zero ou apenas uma ocorrência da expressão *M*.

[a−zA−Z] Lista de caracteres.

. Um único símbolo qualquer.

"a.+*" Uma cadeia entre aspas, denota a si própria.

Notação para Expressões Regulares

Ponto:

Considere o alfabeto {ã, a, b, …, z, A, B, …, Z}

Expressão	Casa com
`n.o`	não, nao, ...
`.eclado`	teclado, Teclado, ...
`e.tendido`	estendido, extendido, eztendido, ...

Notação para Expressões Regulares

Lista de Caracteres:

Expressão	Casa com
n[ãa]o	não, nao
[Tt]eclado	Teclado, teclado
e[sx]tendido	estendido, extendido
12[:.]45	12:45, 12.45, 12 45
<[BIP]>	<B>, <I>, <P>

Notação para Expressões Regulares

Intervalos em Listas:

[0-9] é igual a [0123456789]

[a−zA−Z**]** é igual a

[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]

Exemplos

Quais palavras geram as seguintes expressões regulares?

`if` =

`[a-z][a-z0-9]*` =

`[0-9]+` =

`([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)` =

Exemplos

Quais palavras geram as seguintes expressões regulares?

`if` = `if`

`[a-z][a-z0-9]*` = `chuchu`, `abobrinha`, `var`, `func`, ...

`[0-9]+` = `0`, `00`, `1`, `666`, `42`, `65535`, ...

`([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)` =

`0.`, `0.0`, `1.`, `.5`, `4.2`, `65.535`, ...

Exemplos

Como seriam as expressões regulares para os seguintes tokens?

- IF = **?**

- ID = **?**

- NUM = **?**

Exemplos

Como seriam as expressões regulares para os seguintes tokens?

- IF = `if`
- ID = `[a-z][a-z0-9]*`
- NUM = `[0-9]+`

Exemplos

Quais tokens representam as seguintes expressões regulares?

- `([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)`

- `("//"[a-z]*"\n")|(" "|"\n"|"\t")+`

- `.`

Exemplos

Quais tokens representam as seguintes expressões regulares?

- `([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)`
  - Números Reais

- `("//"[a-z]*"\n")|(" "|"\n"|"\t")+`
  - nenhum token: somente comentário, brancos, nova linha e tabulação

- `.`
  - qualquer coisa, menos nova linha (`\n`)

Anlisador Léxico

Ambiguidades

- `if8` é um ID ou dois tokens: IF e NUM(8) ?
- `if 89` começa com um ID ou uma palavra-reservada?

Duas regras:

- Maior casamento: o próximo *token* sempre é a ***substring*** mais longa possível de ser casada.

- Prioridade: Para uma dada *substring* mais longa, a primeira regra a ser casada produzirá o *token*

Anlisador Léxico

A especificação deve ser completa, sempre reconhecendo uma *substring* da entrada

- Mas quando estiver errada? Use uma regra com o “**.**”
- Em que lugar da sua especificação deve estar esta regra?
  - Esta regra deve ser a última! (Por que?)

Autômatos Finitos

- Expressões Regulares são convenientes para especificar os *tokens*

- Precisamos de um formalismo que possa ser convertido em um programa de computador

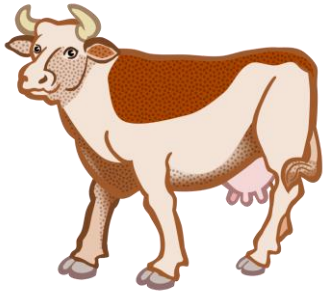
- Este formalismo são os autômatos finitos

Autômatos Finitos

Um autômato finito possui:

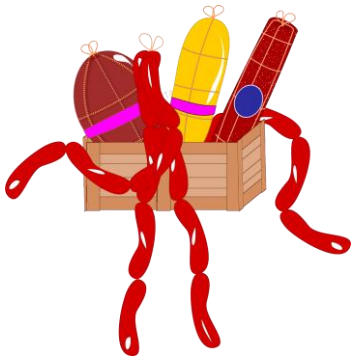
- Um conjunto finito de estados
- Arestas levando de um estado a outro, anotada com um símbolo
- Um estado inicial
- Um ou mais estados finais
- Normalmente os estados são numerados ou nomeados para facilitar a manipulação e discussão

Autômatos Finitos

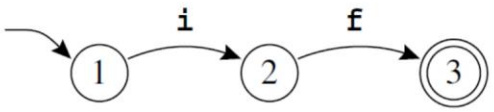


A, B, C

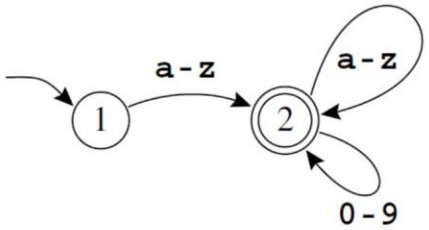
1, 2, 3



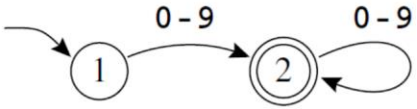
Autômatos Finitos



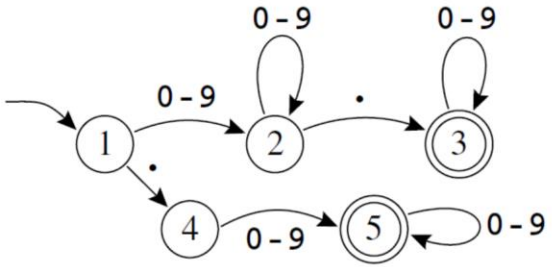
IF



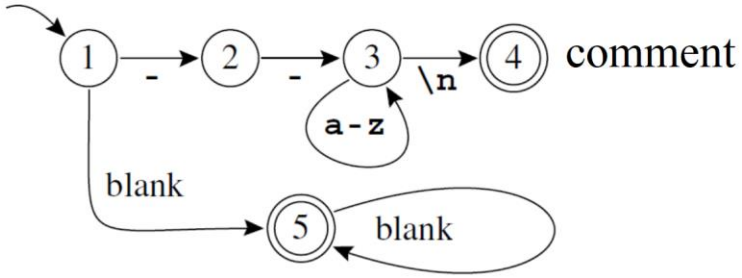
ID



NUM



REAL

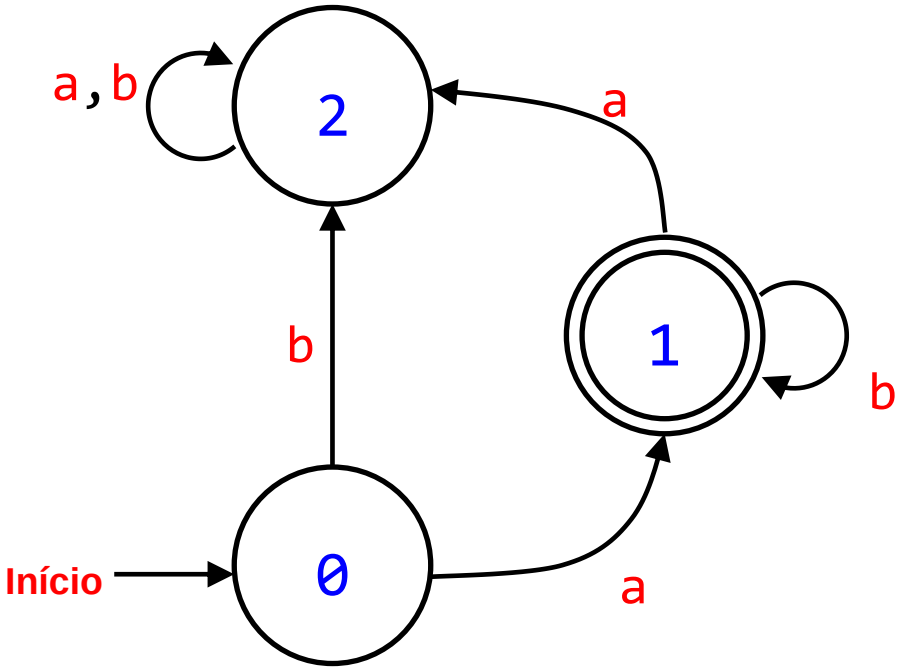


white space

Deterministic Finite Automata (DFA)

- DFAs não podem apresentar duas arestas que deixam o mesmo estado, anotadas com o mesmo símbolo
- Saindo do estado inicial, o autômato segue exatamente uma aresta para cada caractere da entrada
- O DFA aceita a *string* se, após percorrer todos os caracteres, ele estiver em um estado final
- Se em algum momento não houver uma aresta a ser percorrida para um determinado caractere ou ele terminar em um estado não-final, a *string* é rejeitada
- A linguagem reconhecida pelo autômato é o conjunto de todas as *strings* que ele aceita

Deterministic Finite Automata (DFA)



Expressão Regular = ab^*

abbbbb ✓

a ✓

baa ✗

Nondeterministic Finite Automata (NFA)

Pode existir mais de uma aresta saindo de um determinado estado com o mesmo símbolo

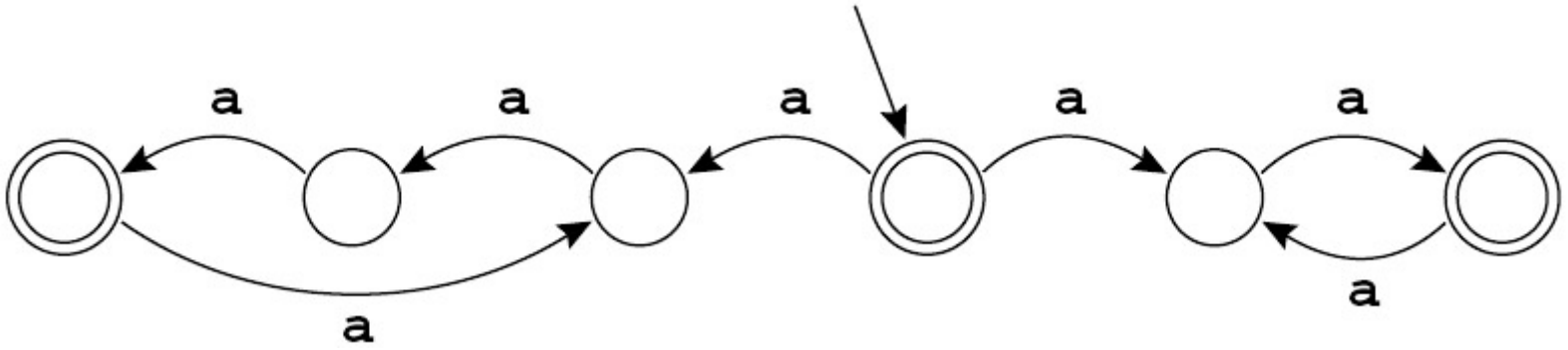
Podem existir arestas anotadas com o símbolo **ε** (*cadeia vazia*)

- Essa aresta pode ser percorrida sem consumir nenhum caractere da entrada!

Não são apropriados para serem transformados em programas de computador

- “Adivinhar” qual caminho deve ser seguido não é uma tarefa facilmente executada pelo hardware dos computadores

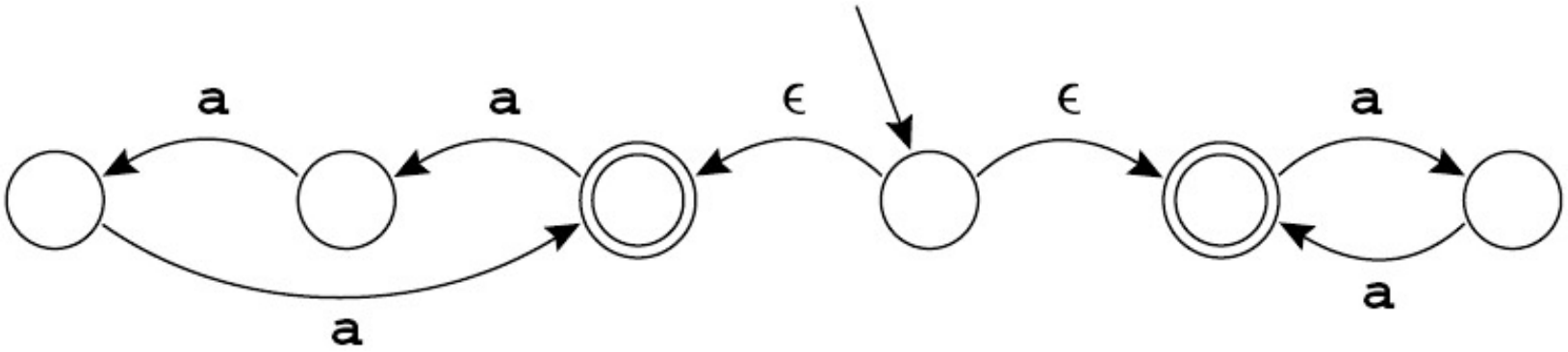
Nondeterministic Finite Automata (NFA)



Que linguagem este autômato reconhece?

Obs: Ele é obrigado a aceitar a *string* se existe alguma escolha de caminho que leva à aceitação

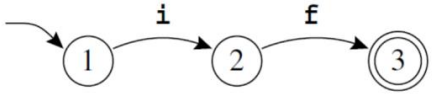
Nondeterministic Finite Automata (NFA)



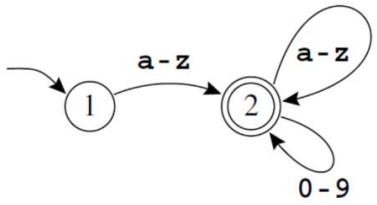
Que linguagem este autômato reconhece?

Autômatos Finitos

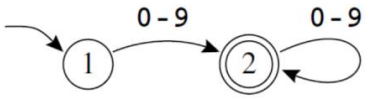
- É possível combinar os autômatos definidos para cada *token* de maneira a ter um único autômato que possa ser usado como analisador léxico?



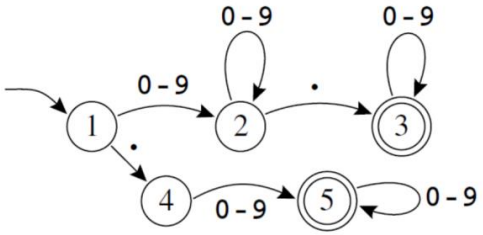
IF



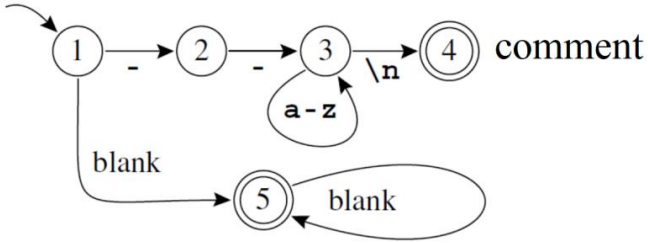
ID



NUM

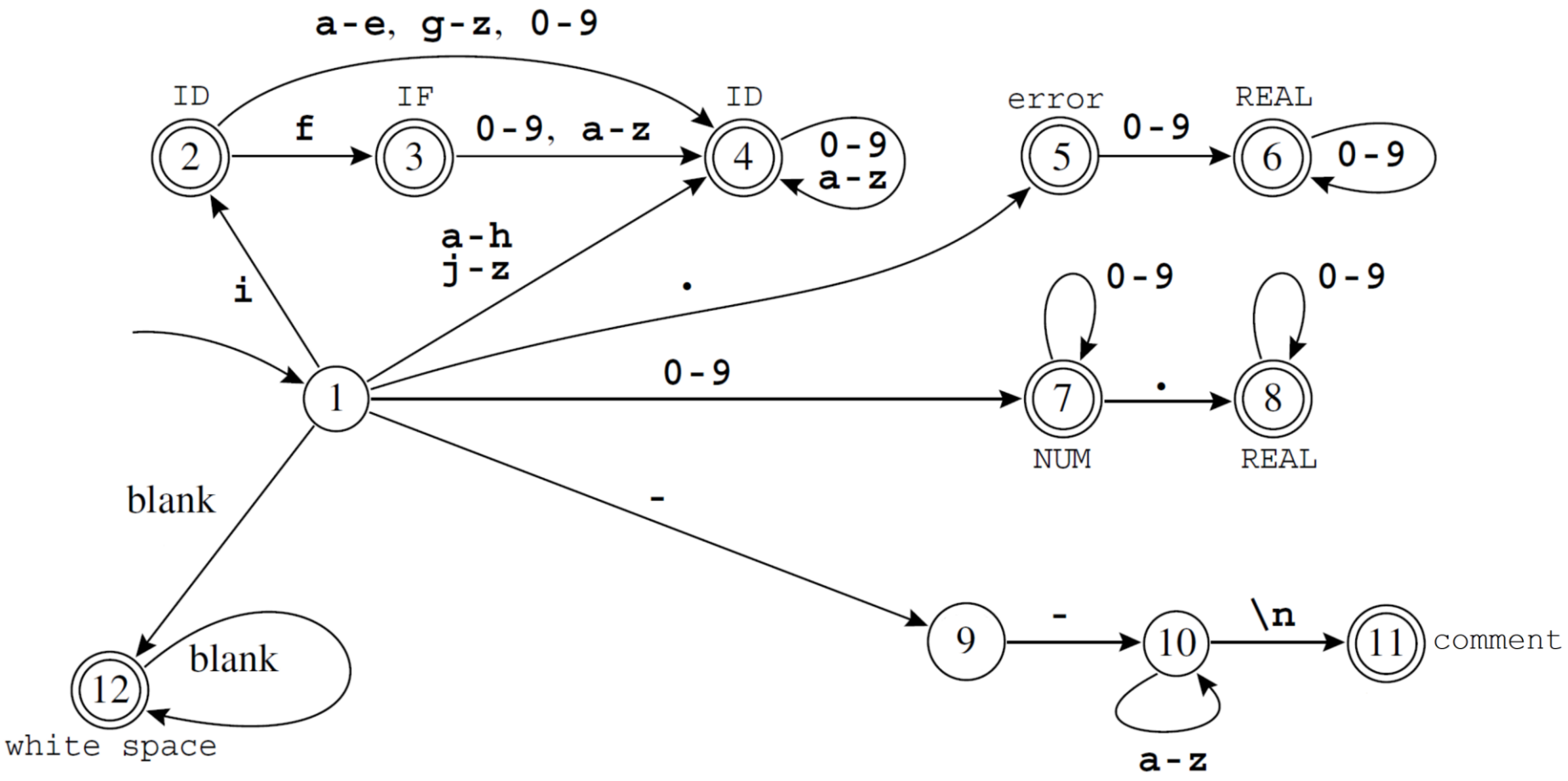


REAL

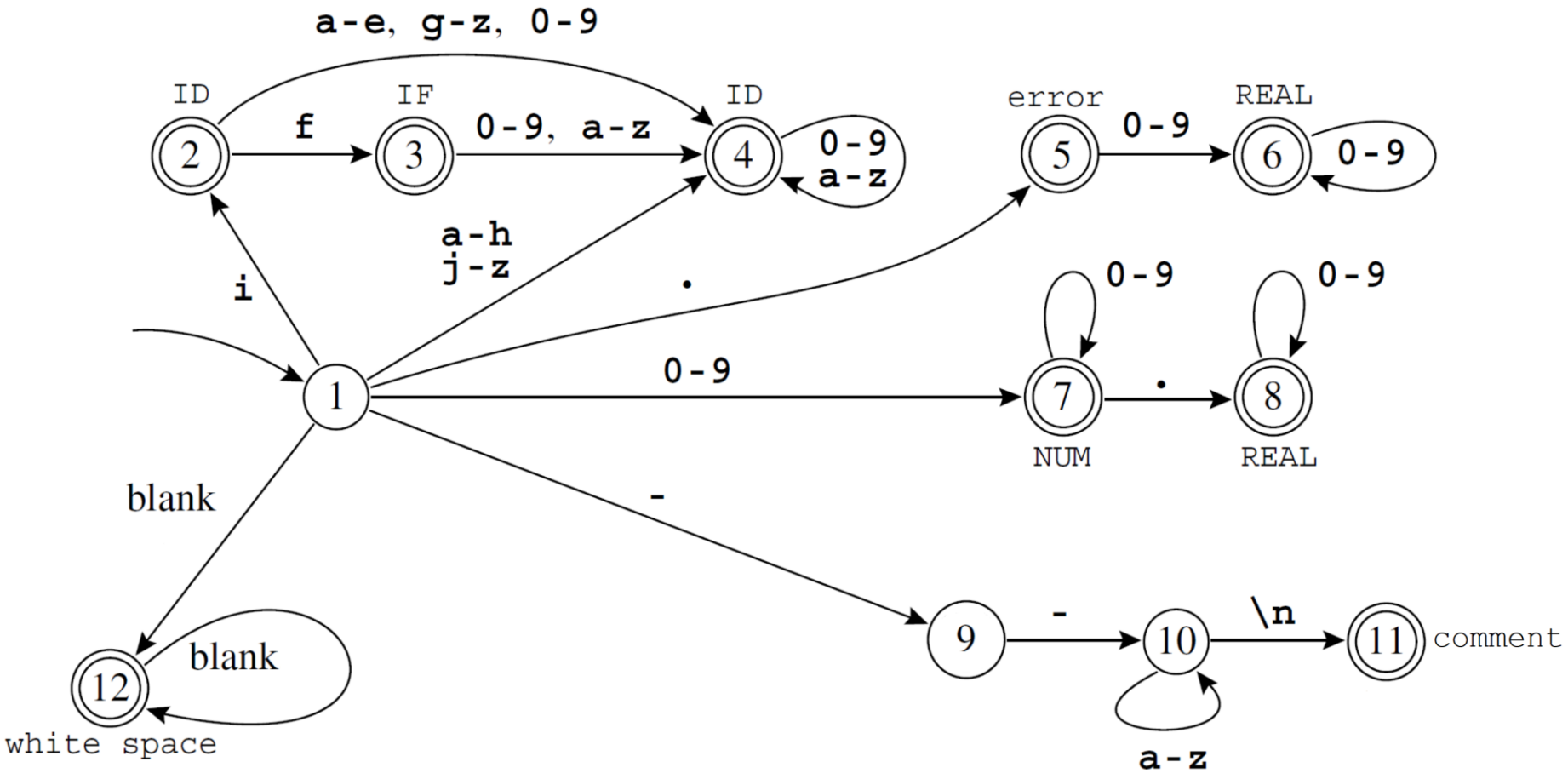


white space

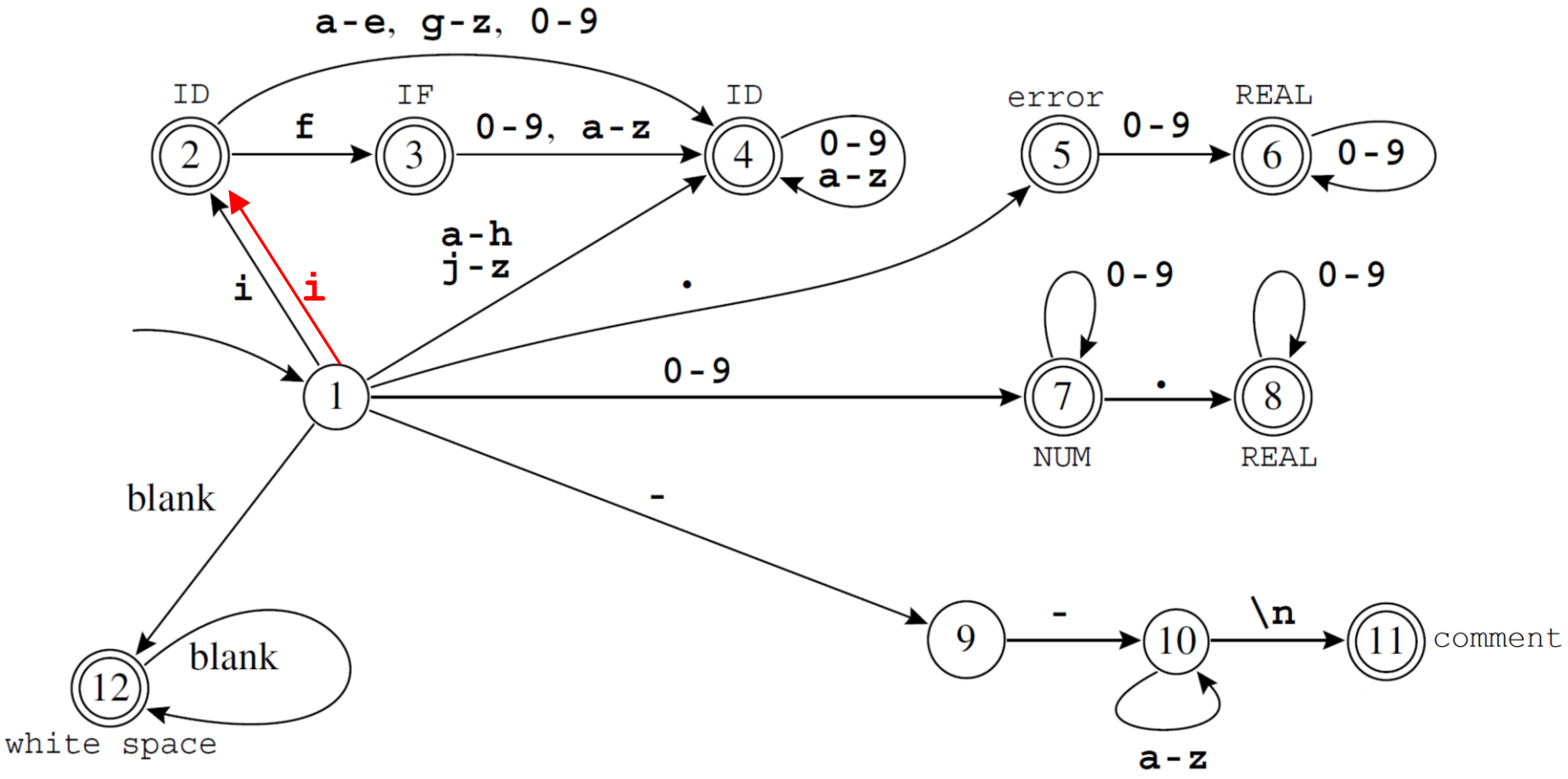
Autômato Combinado



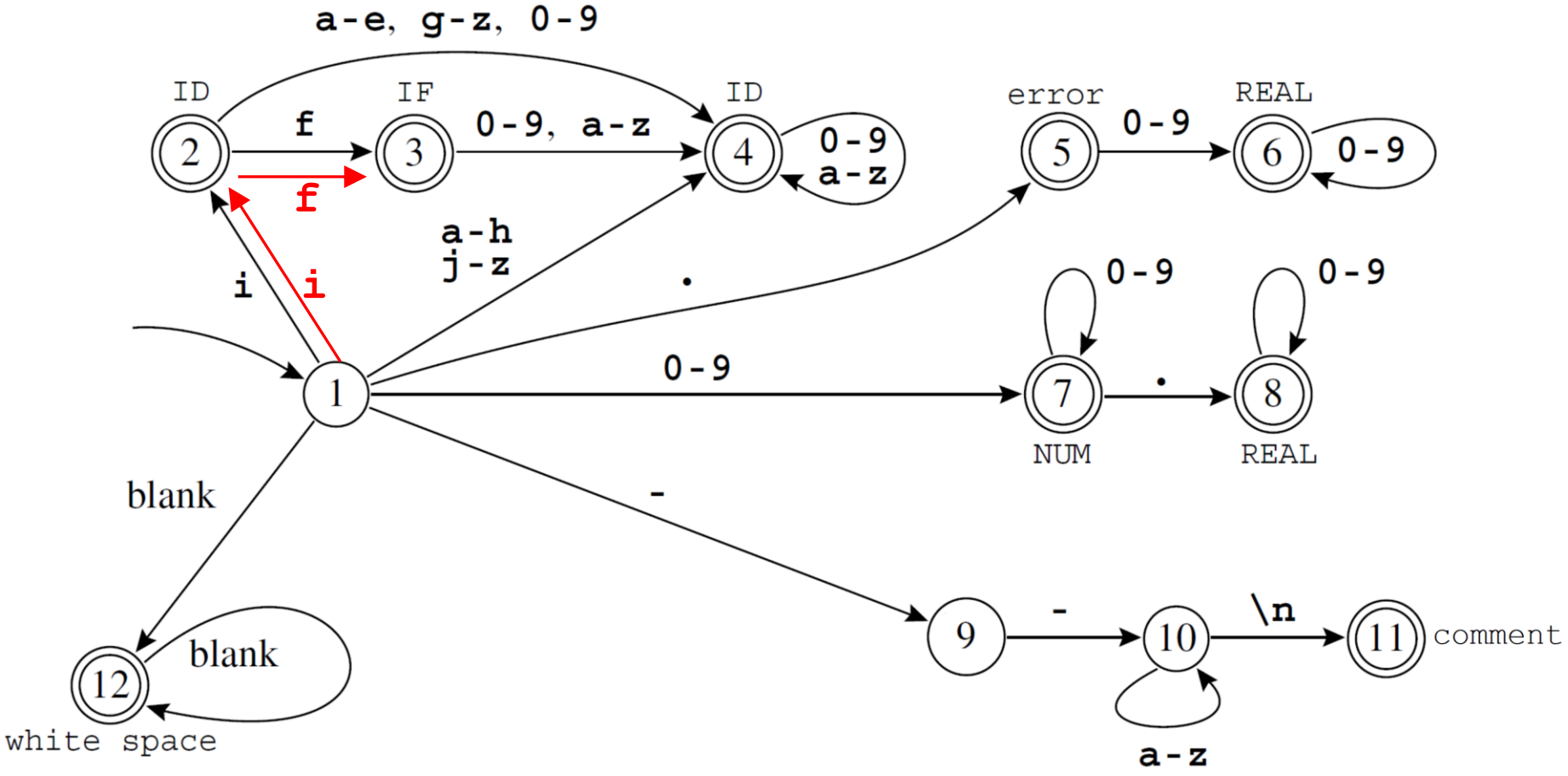
Autômato Combinado: if8



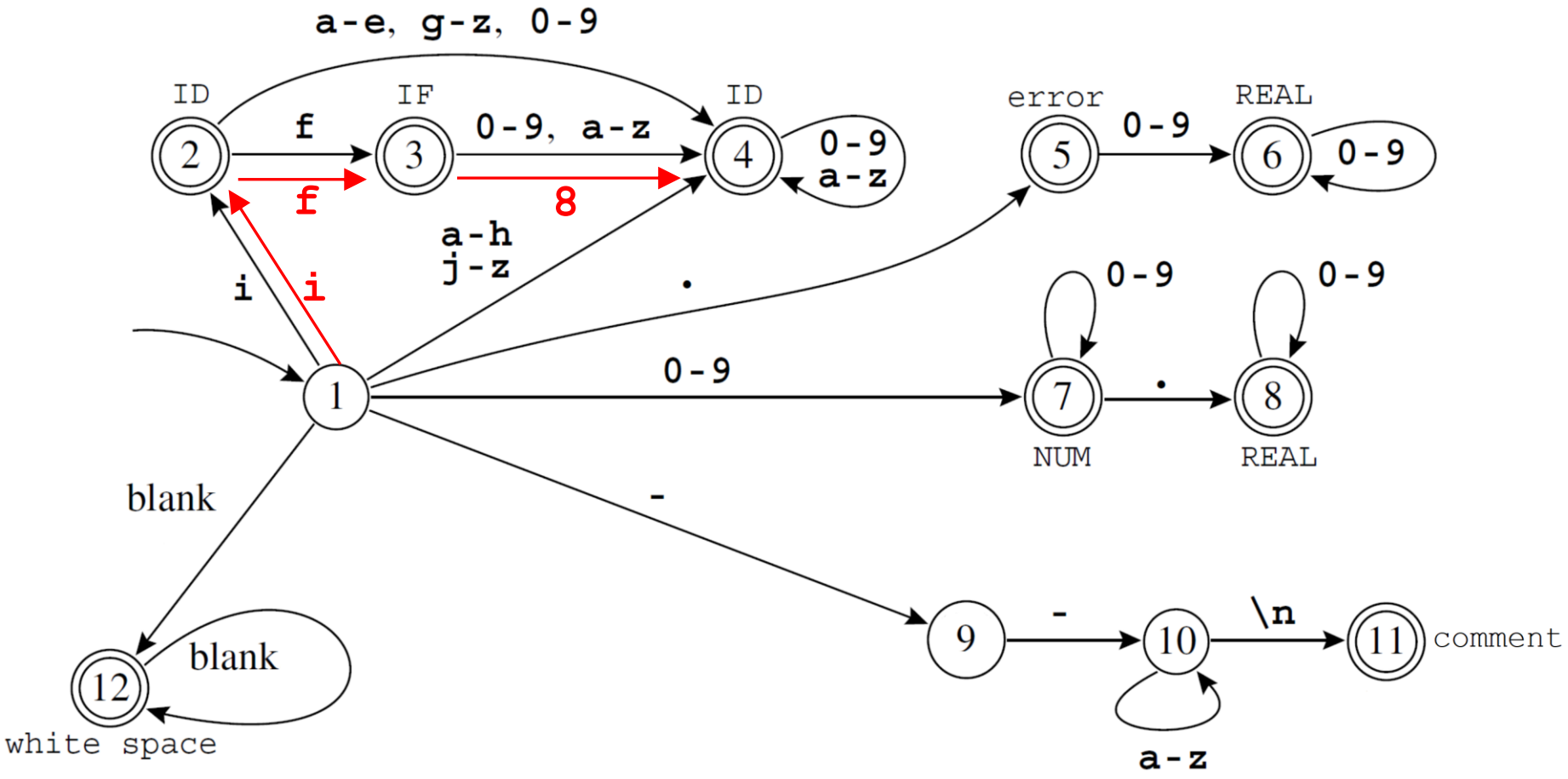
Autômato Combinado: if8



Autômato Combinado: if8



Autômato Combinado: if8



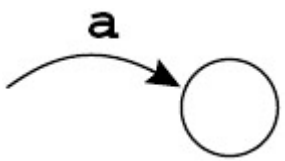
Autômato Combinado

- Estados finais nomeados com o respectivo *token*
- Alguns estados apresentam características de mais de um autômato anterior. Ex.: 3
- Como ocorre a quebra de ambiguidade entre ID e IF?

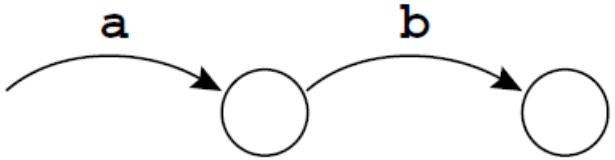
Convertendo ER's para NFA's

- NFAs se tornam úteis porque é fácil converter expressões regulares (ER) para NFA
- Exemplos:

a

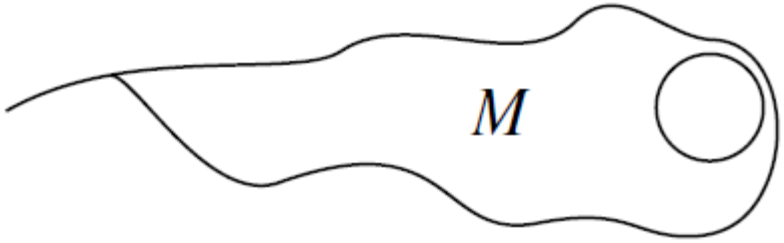


ab



Convertendo ER's para NFA's

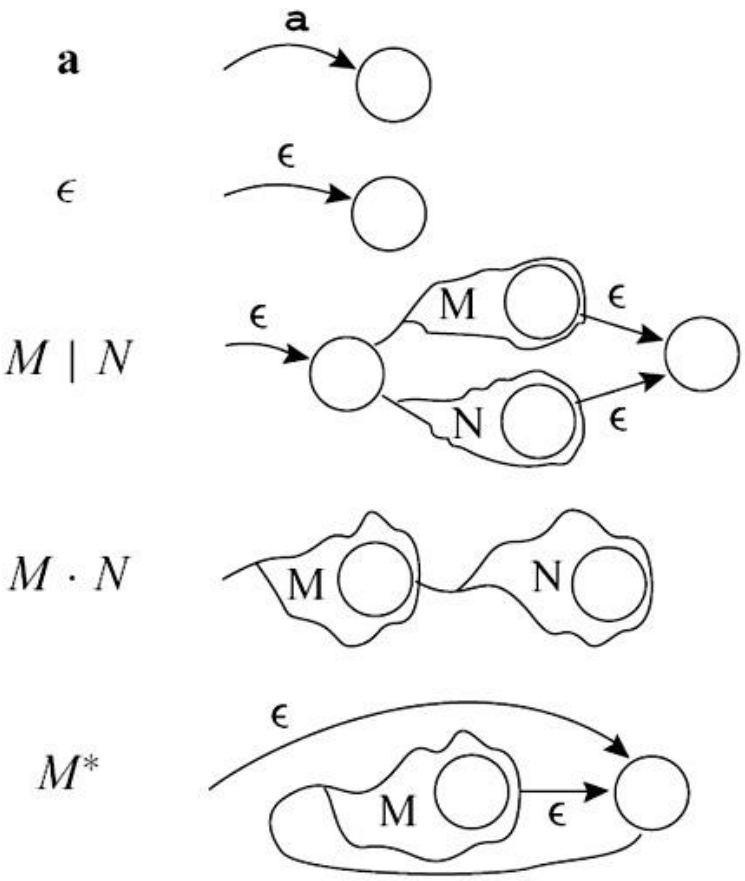
De maneira geral, toda ER terá um NFA com uma cauda (aresta de entrada) e uma cabeça (estado final).



Pode-se definir essa conversão de maneira indutiva pois uma ER é primitiva (único símbolo ou vazio) ou é uma combinação de outras ERs.

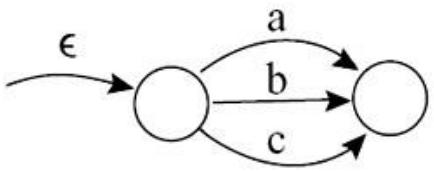
O mesmo vale para NFAs.

Convertendo ER's para NFA's



M^+ constructed as $M \cdot M^*$

$M?$ constructed as $M \mid \epsilon$



"**abc**" constructed as **a** · **b** · **c**

Conversão de Expressões Regulares para AFNDs

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

$a|b$

ab

a^*

Conversão de Expressões Regulares para AFNDs

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

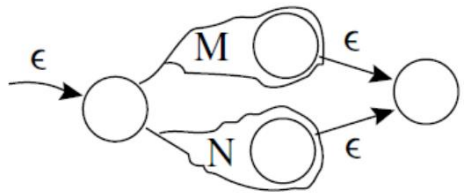
$a|b$

Conversão de Expressões Regulares para AFNDs

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

a|b

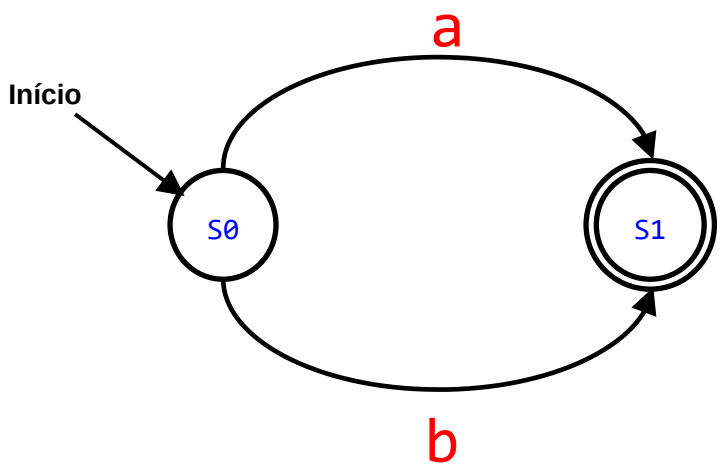
$M \mid N$



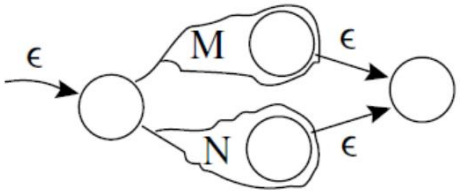
Conversão de Expressões Regulares para AFNDs

Σ = {a, b}

a|b



$M \mid N$



Conversão de Expressões Regulares para AFNDs

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

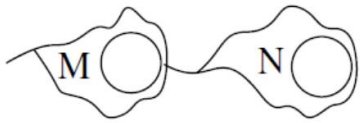
ab

Conversão de Expressões Regulares para AFNDs

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

ab

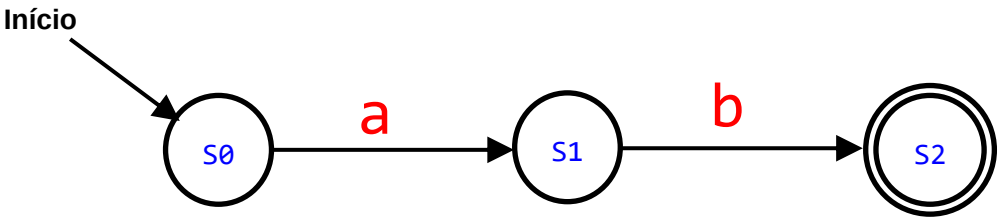
$M \cdot N$



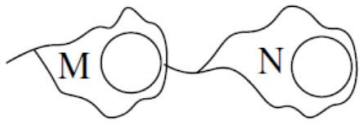
Conversão de Expressões Regulares para AFNDs

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

ab



$M \cdot N$



Conversão de Expressões Regulares para AFNDs

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

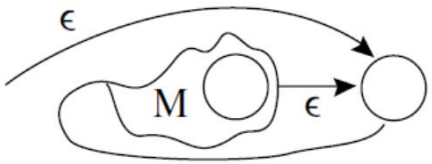
a^*

Conversão de Expressões Regulares para AFNDs

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

a^*

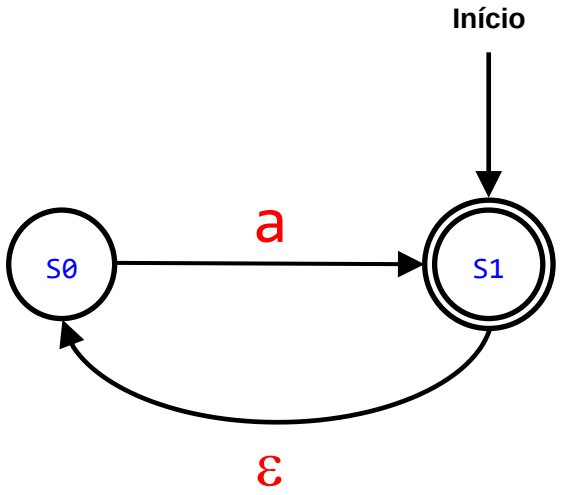
M^*



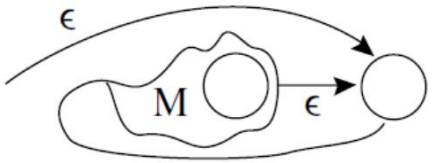
Conversão de Expressões Regulares para AFNDs

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

a^*



M^*



Conversão de Expressões Regulares para AFNDs

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

$a|bc$

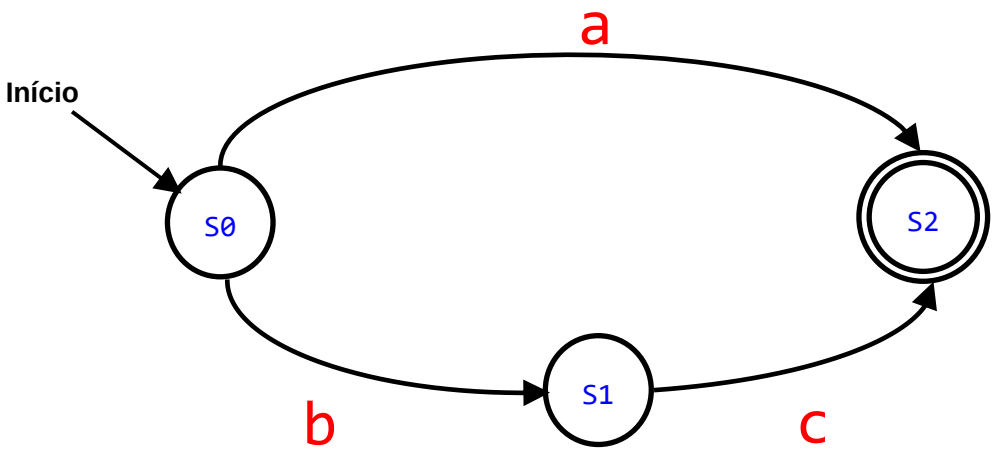
$(a|b)^*$

a^*bc

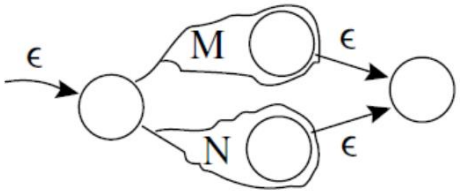
Conversão de Expressões Regulares para AFNDs

Σ = {a, b, c}

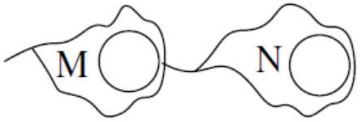
a|bc



$M \mid N$



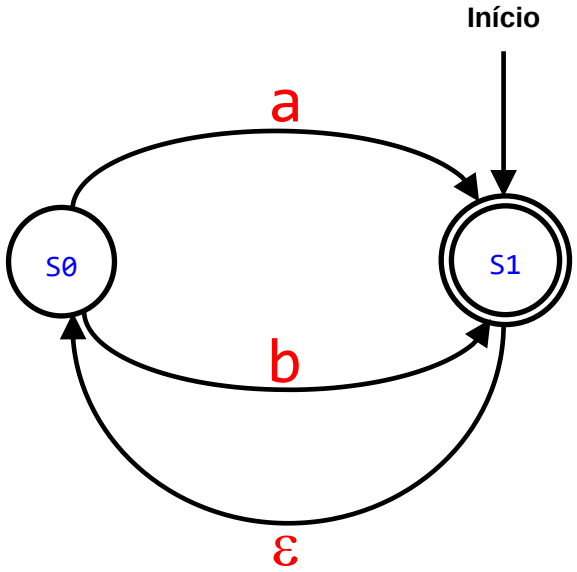
$M \cdot N$



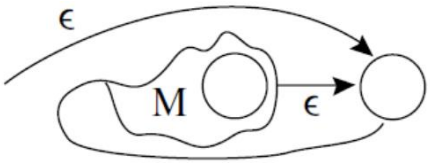
Conversão de Expressões Regulares para AFNDs

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

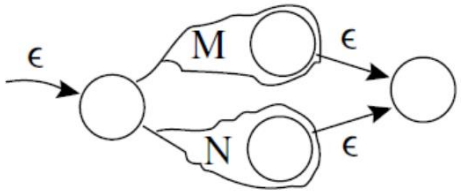
$(a|b)^*$



M^*



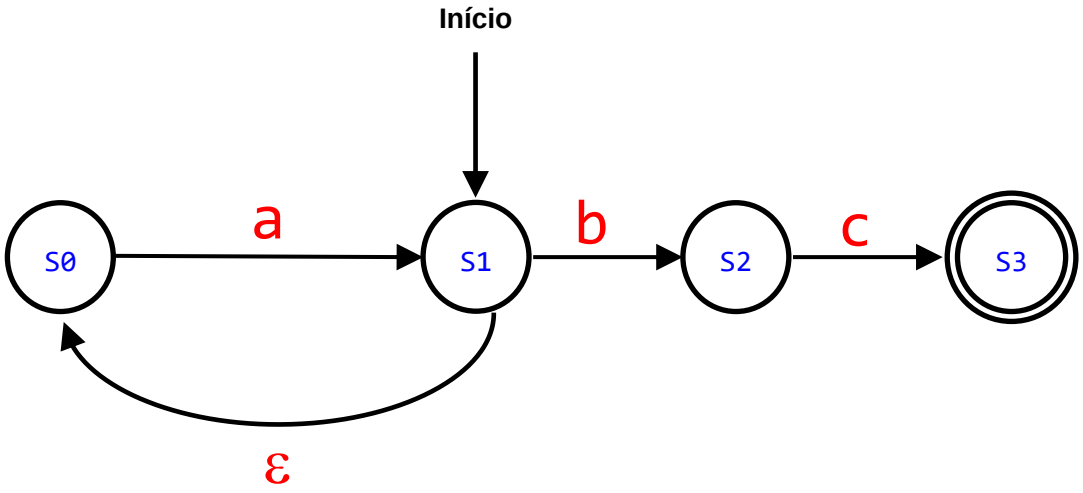
$M \mid N$

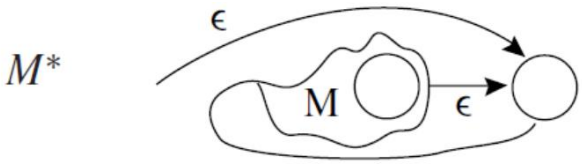


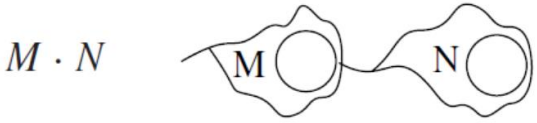
Conversão de Expressões Regulares para AFNDs

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

a^*bc







Lista de Exercícios

Lista 2

- Exercícios Teóricos