Conceitos Preliminares

Alfabetos, Palavras e Linguagens

Alfabeto (Σ)

• É um conjunto finito e não vazio de símbolos. Assim são alfabetos os conjuntos:

{0, 1}

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

{a, b, ab, abc}

{♣, ♦, ♥, ♠}

Cada elemento no alfabeto pode ser chamado de uma letra, a qual tem um significado diferente do usual. O terceiro alfabeto apresentado possui 4 letras.

Alfabetos, Palavras e Linguagens

Palavra

• Uma palavra ou cadeia sobre um alfabeto Σ é uma tupla ordenada de letras de Σ. Assim:

$\langle 0, 1, 0, 1, 1, 0 \rangle$ é uma palavra sobre {0, 1}

$\langle 2, 1, 0, 8 \rangle$ é uma palavra sobre {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

$\langle c, o, m, p, i, l, a, d, o, r \rangle$ é uma palavra sobre {a, b, …, z}

$\langle a, ab, b, abc \rangle$ é uma palavra sobre {a, b, ab, abc}

Alfabetos, Palavras e Linguagens

Em geral, pode-se representar uma palavra apenas aglutinando, na ordem correta, as letras que a compõem. Assim:

$\langle 0, 1, 0, 1, 1, 0 \rangle$ como *010110*

$\langle 2, 1, 0, 8 \rangle$ como *2108*

$\langle c, o, m, p, i, l, a, d, o, r \rangle$ como *compilador*

E a palavra $\langle a, ab, b, abc \rangle$?

Alfabetos, Palavras e Linguagens

Porém, ⟨a, ab, b, abc⟩ não pode ser representada simplesmente por aabbabc, pois essa representação poderia indicar outras palavras além daquela que se deseja representar.

Neste caso, pode-se utilizar espaços entre as letras para indicar como "separar" as letras da palavra.

A palavra ⟨a, ab, b, abc⟩ seria representada como *a ab b abc*

A palavra ⟨a, a, b, b, abc⟩ seria representada como *a a b b abc*.

Deve-se ressaltar que os espaços NÃO fazem parte das palavras.

Alfabetos, Palavras e Linguagens

Tamanho de uma Palavra - $|x|$

Define-se o tamanho de uma palavra x, denotado por $|x|$ como o número de letras de x. Assim:

|010110| = 6

|2108| = 4

|compilador| = 10

|a ab b abc| = 4

Alfabetos, Palavras e Linguagens

Cadeia Vazia - λ ε

Sobre qualquer alfabeto Σ, define-se uma única palavra de tamanho 0 (zero) que é denotada por λ ou ε.

Definem-se, também, os conjuntos:

Σ^k = { palavras x sobre Σ tal que, $|x| = k$ }

$$\sum^* = \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots$$ (Fecho de Kleene)

$$\sum^+ = \sum^* - \{\lambda\}$$

Alfabetos, Palavras e Linguagens

Fecho de Kleene

$\{ ab, c \}^* = ?$

$\{ a, b, c\}^+ = ?$

Alfabetos, Palavras e Linguagens

Fecho de Kleene

$\{ ab, c \}^* = \{ \varepsilon, ab, c, abab, abc, cab, cc, ababab, ababc, abcab, \ldots \}$

$\{ a, b, c\}^+ = \{ a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, abc, \ldots \}$

Alfabetos, Palavras e Linguagens

Linguagem

• Dado um alfabeto Σ, uma linguagem L sobre um alfabeto Σ é um subconjunto qualquer de Σ^*. Assim:

{ 0, 1, 00, 01, 10, 11} é uma linguagem sobre **{ 0, 1}** que contém 6 palavras

{ $x \in$ **{0,1, 2, 3, 4, 5, 6, 7, 8, 9}*** tal que x representa um número decimal ímpar**}** é uma linguagem sobre **{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}** com um número infinito de palavras

o conjunto de todos os programas **C++** válidos (sintaticamente corretos) é uma linguagem sobre um alfabeto Σ composto de:

- palavras reservadas como for, while, if, etc
- símbolos especiais como {, }, *, +, etc
- nomes de variáveis, métodos, classes, etc.

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

- **União** ($\cup$): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 \cup L_2$ como $\{ x \in \Sigma^* \text{ tal que } x \in L_1 \vee x \in L_2 \}$

- **Concatenação** (**.**): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 . L_2$ como $\{ x.y \in \Sigma^* \text{ tal que } x \in L_1 \wedge y \in L_2 \}$

- **Fecho de Kleene (*)**: dada uma linguagem L sobre o alfabeto Σ, define-se L^* como sendo $\lambda \cup L \cup (L.L) \cup (L.L.L) \cup (L.L.L.L) \cup \ldots$

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

- **União** ($\cup$): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 \cup L_2$ como $\{ x \in \Sigma^* \text{ tal que } x \in L_1 \vee x \in L_2 \}$

{a, b} $\cup$ {c, d} = ?

{100, 010, 110} $\cup$ {00, 01, 11} = ?

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

- **União** ($\cup$): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 \cup L_2$ como $\{ x \in \Sigma^* \text{ tal que } x \in L_1 \vee x \in L_2 \}$

{a, b} $\cup$ {c, d} = {a, b, c, d}

{100, 010, 110} $\cup$ {00, 01, 11} = {100, 010, 110, 00, 01, 11}

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

• **Concatenação** (**.**): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 \,.\, L_2$ como $\{ x.y \in \Sigma^* \text{ tal que } x \in L_1 \wedge y \in L_2 \}$

{101, 110} **.** {00, 11} = ?

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

- **Concatenação** (**.**): dadas as linguagens L_1 e L_2, sobre o alfabeto Σ, define-se $L_1 \,.\, L_2$ como $\{ x.y \in \Sigma^* \text{ tal que } x \in L_1 \wedge y \in L_2 \}$

{101, 110} **.** {00, 11} = {10100, 10111, 11000, 11011}

Alfabetos, Palavras e Linguagens

Operações sobre Linguagens

- **Fecho de Kleene (*)**: dada uma linguagem L sobre o alfabeto Σ, define-se L^* como sendo $\lambda \cup L \cup (L.L) \cup (L.L.L) \cup (L.L.L.L) \cup \ldots$

$\{abc\}^* = ?$

Alfabetos, Palavras e Linguagens

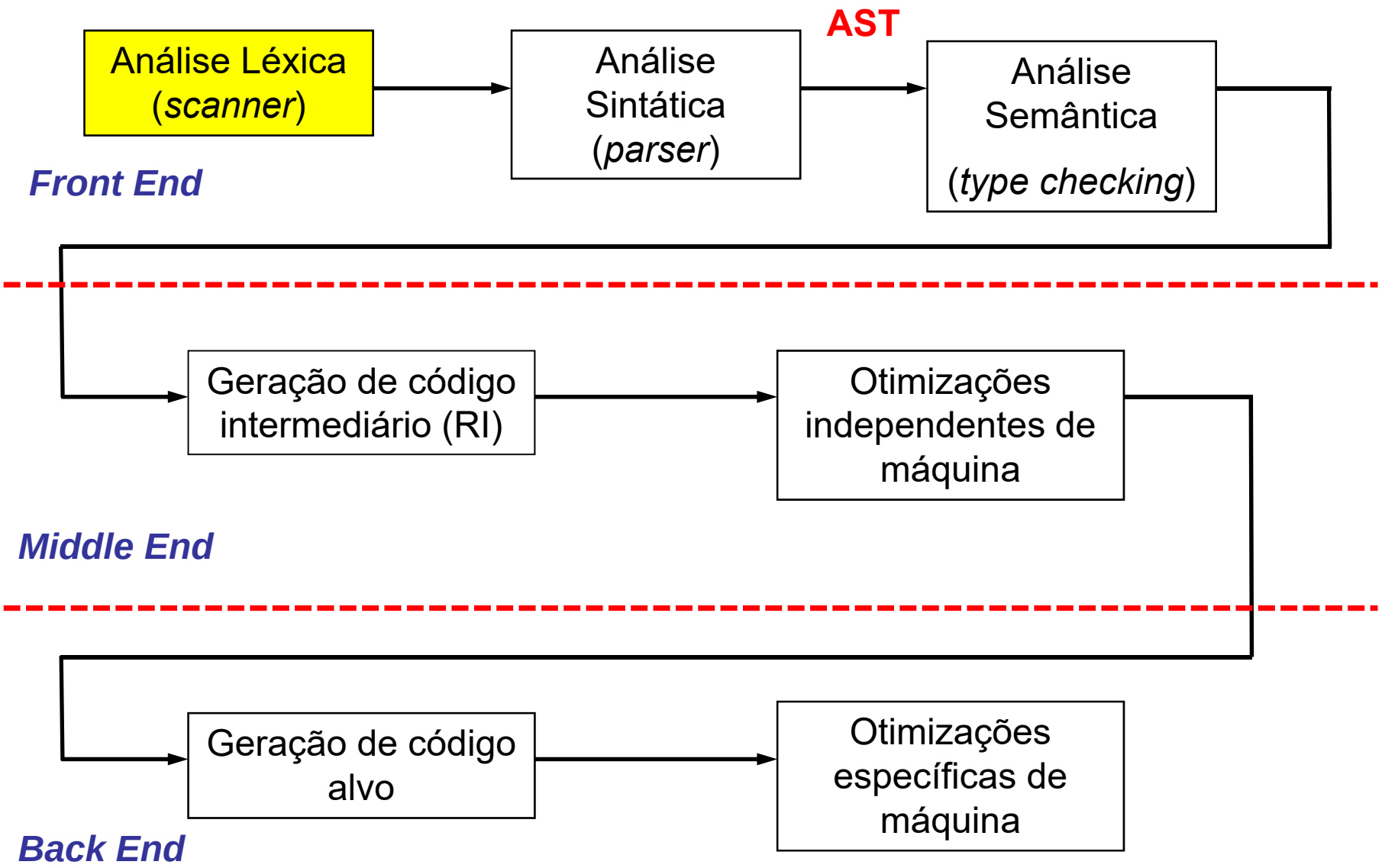
Operações sobre Linguagens

- **Fecho de Kleene (*)**: dada uma linguagem L sobre o alfabeto Σ, define-se L^* como sendo $\lambda \cup L \cup (L.L) \cup (L.L.L) \cup (L.L.L.L) \cup \ldots$

$\{abc\}^* = \{ \lambda, abc, abcabc, abcabcabc, abcabcabcabc, \ldots\}$

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

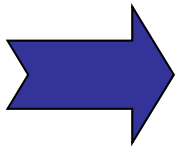
- Um linguagem L é um conjunto de *strings* (palavras) ($L \subseteq \Sigma^*$)
- Uma *string* (palavra) é uma sequência de símbolos (cadeia)
- Estes símbolos estão definidos em um alfabeto finito (Σ)
- Ex: Linguagem C ou Pascal, linguagem dos primos, etc
- Deseja-se poder dizer se uma *string* (palavra)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

Uma expressão regular (ER) utiliza linguagens regulares "primitivas" e combina-as por meio de alguns operadores.

Essa expressão formada define, então, uma outra linguagem

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

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

- **a** An ordinary character stands for itself.
- ε The empty string.
- Another way to write the empty string.
- ***M* | *N*** Alternation, choosing from *M* or *N*.
- ***M* · *N*** Concatenation, an *M* followed by an *N*.
- ***MN*** Another way to write concatenation.
- ***M**** Repetition (zero or more times).
- M^+ Repetition, one or more times.
- ***M*?** Optional, zero or one occurrence of *M*.
- **[a − zA − Z]** Character set alternation.
- **.** A period stands for any single character except newline.
- "a.+*" Quotation, a string in quotes stands for itself literally.

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

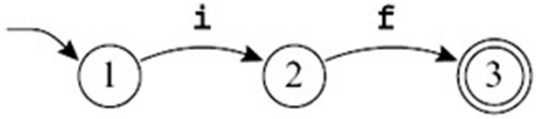
- Este formalismo são os autômatos finitos

Autômatos Finitos

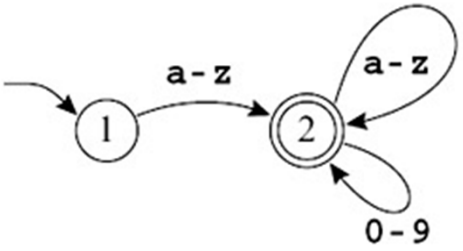
Um autômato finito possui:

- 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

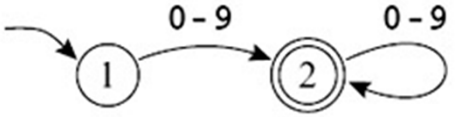
Autômatos Finitos



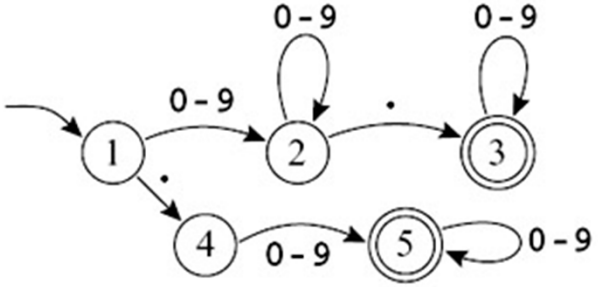
IF



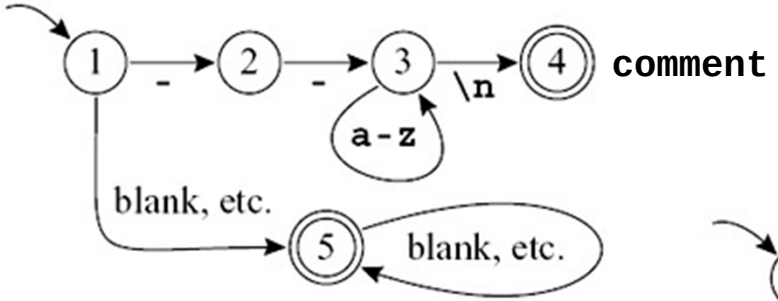
ID



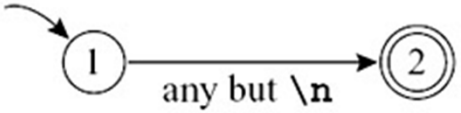
NUM



REAL



white space



error

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

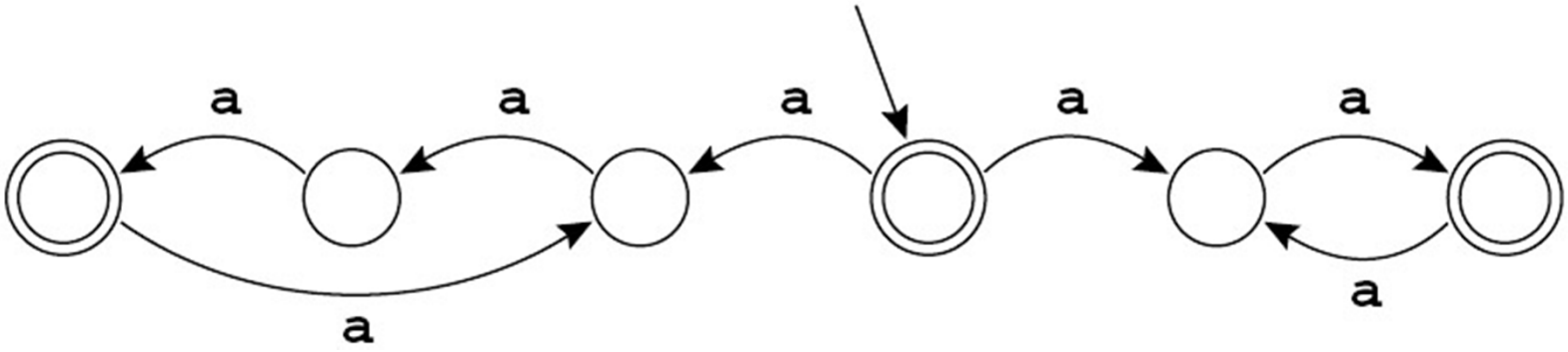
Nondeterministic Finite Automata (NFA)

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

- Podem existir arestas anotadas com o símbolo Є
  - 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

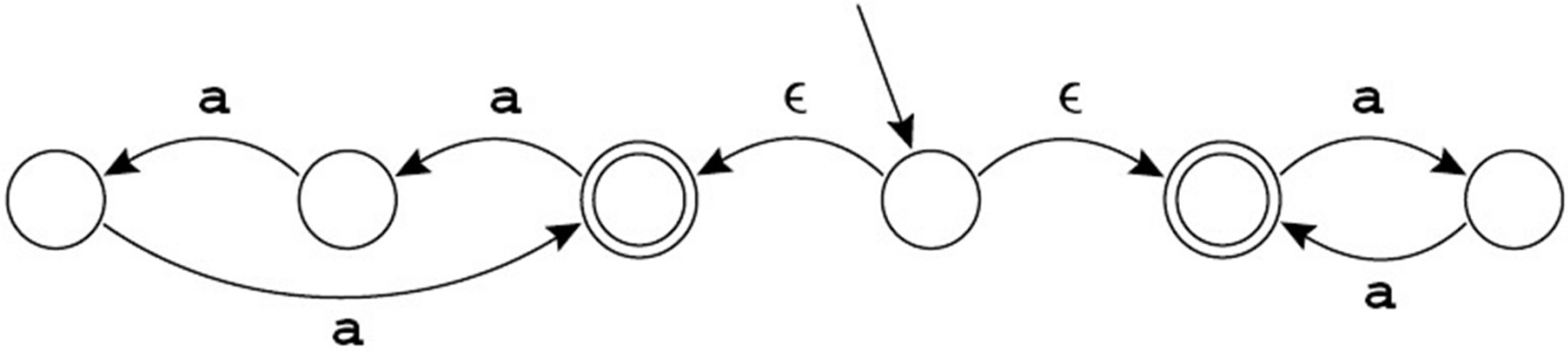
Nondeterministic Finite Automata (NFA)



Que linguagem este autômato reconhece?

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

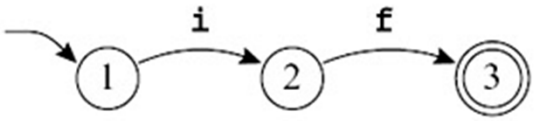
Nondeterministic Finite Automata (NFA)



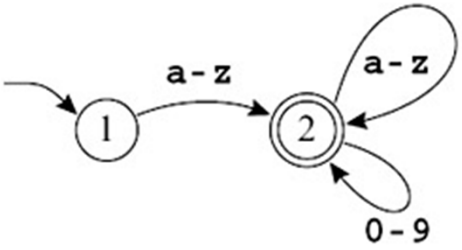
Que linguagem este autômato reconhece?

Autômatos Finitos

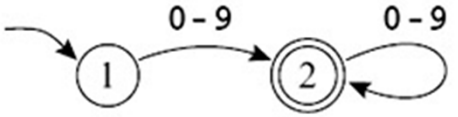
- É possível combinar os autômatos definidos para cada *token* de maneira a ter um único autômato que possa ser usado como analisador léxico?



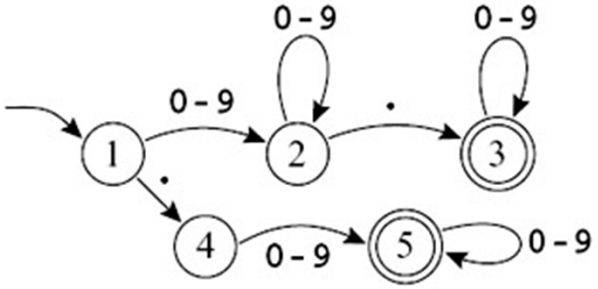
IF



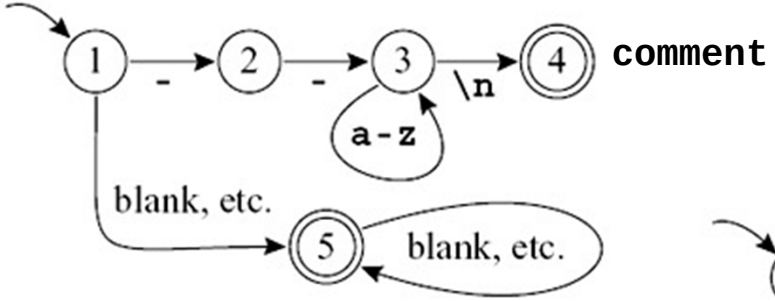
ID



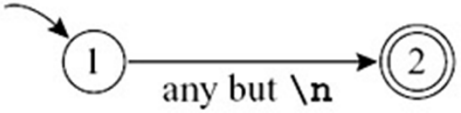
NUM



REAL

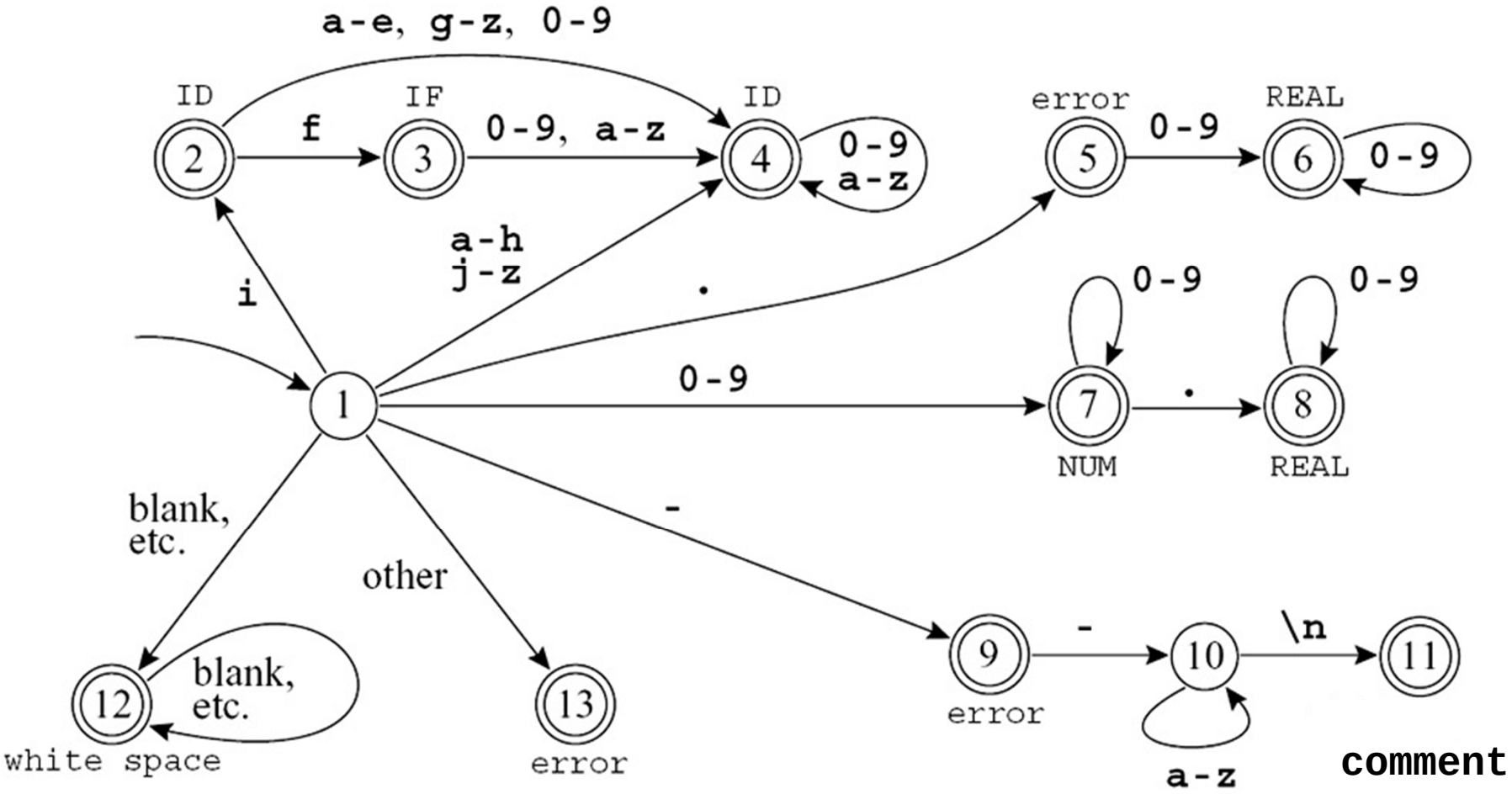


white space

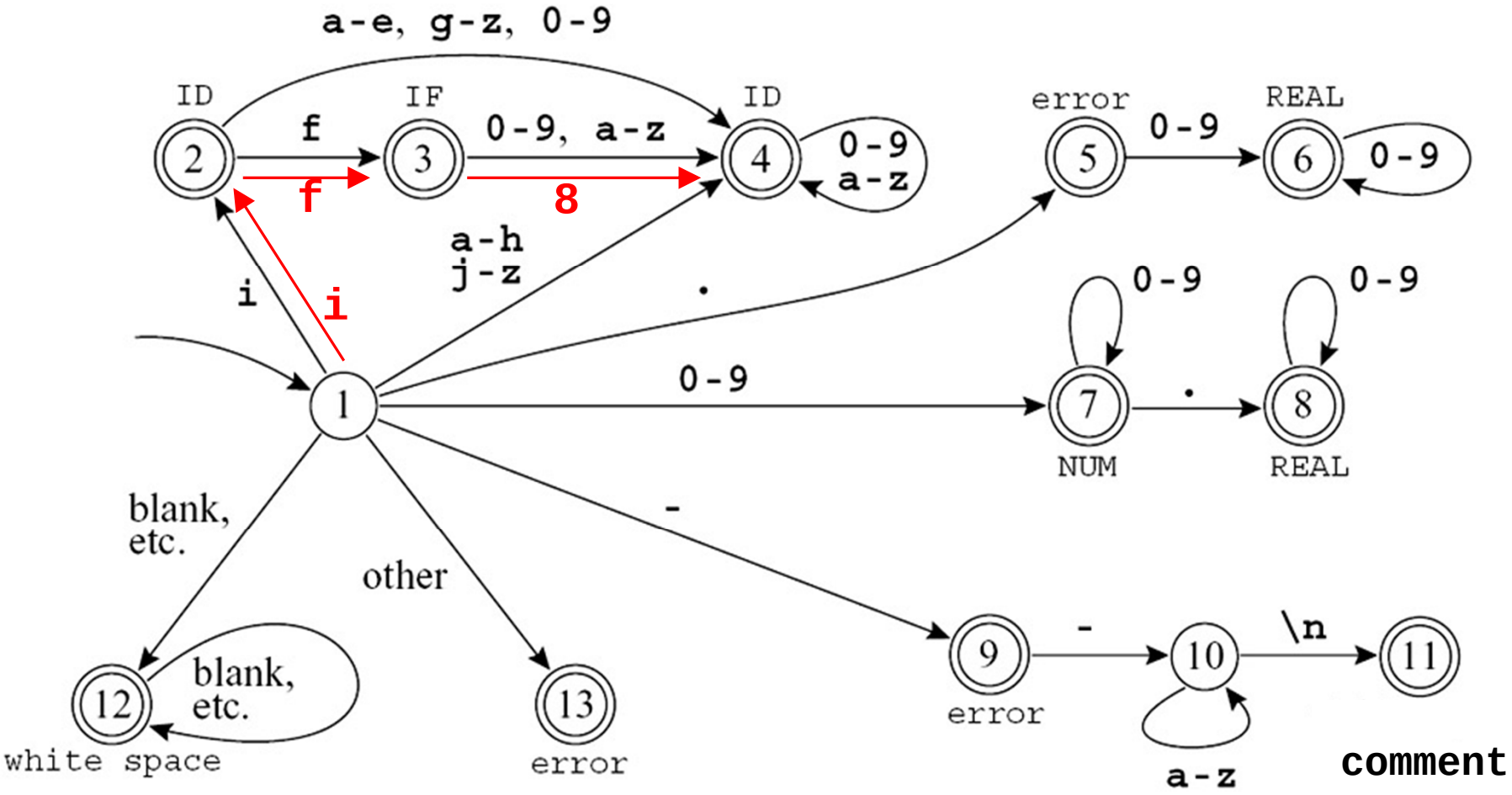


error

Autômato Combinado



Autômato Combinado: if8



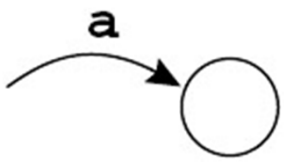
Autômato Combinado

- 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:

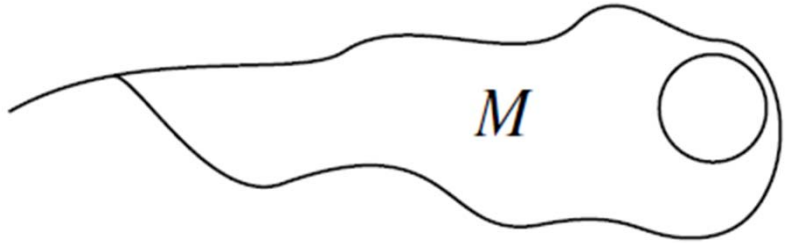
a



ab

Convertendo ER's para NFA's

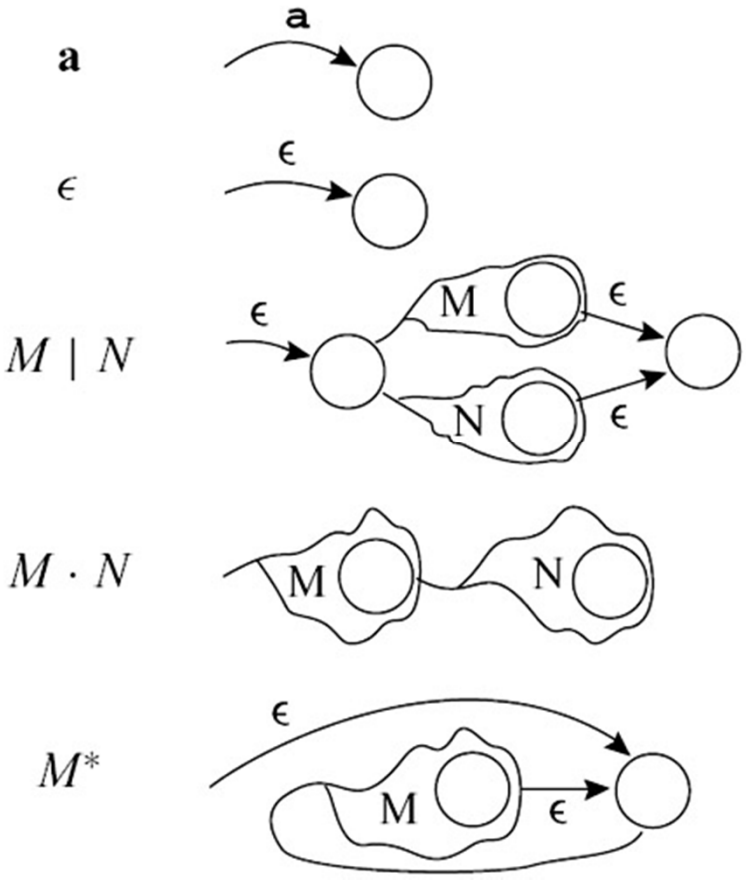
De maneira geral, toda ER terá um NFA com uma cauda (aresta de entrada) e uma cabeça (estado final).



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.

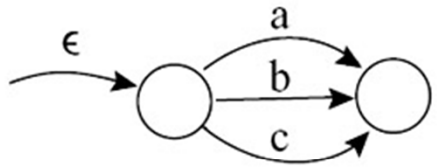
Convertendo ER's para NFA's



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

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

[abc]



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

Exemplo

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