Análise Léxica

Universidade Federal do Ceará - Campus de Quixadá

Lucas Ismaily
ismailybf@ufc.br

Semestre 2024.1

Compiladores
Baseado nos slides do Prof. Sandro Rigo (IC-Unicamp)

Seção 1

Introdução

Introdução

$$\text{Programa Fonte} \longrightarrow \boxed{\text{Compilador}} \longrightarrow \text{Programa Alvo}$$

Introdução

Programa Fonte $\longrightarrow$ $\boxed{\text{Compilador}}$ $\longrightarrow$ Programa Alvo

- O compilador traduz o programa de uma linguagem (fonte) para outra (de máquina).

Introdução

Programa Fonte $\longrightarrow$ $\boxed{\text{Compilador}}$ $\longrightarrow$ Programa Alvo

- O compilador traduz o programa de uma linguagem (fonte) para outra (de máquina).
- Esse processo demanda sua quebra em várias partes, o entendimento de sua estrutura e significado.

Introdução

Programa Fonte $\longrightarrow$ $\boxed{\text{Compilador}}$ $\longrightarrow$ Programa Alvo

- O compilador traduz o programa de uma linguagem (fonte) para outra (de máquina).
- Esse processo demanda sua quebra em várias partes, o entendimento de sua estrutura e significado.
- O responsável por esse tipo de análise é o front-end.

Front-end

- Análise Léxica:

Front-end

- Análise Léxica:
    - Quebra a entrada em palavras conhecidas como símbolos léxicos (tokens).

Front-end

- Análise Léxica:
  - Quebra a entrada em palavras conhecidas como símbolos léxicos (tokens).
- Análise Sintática:

Front-end

- Análise Léxica:
  - Quebra a entrada em palavras conhecidas como símbolos léxicos (tokens).
- Análise Sintática:
  - Analisa a estrutura de frases do programa.

Front-end

- Análise Léxica:
  - Quebra a entrada em palavras conhecidas como símbolos léxicos (tokens).
- Análise Sintática:
  - Analisa a estrutura de frases do programa.
- Análise Semântica:

Front-end

- Análise Léxica:
  - Quebra a entrada em palavras conhecidas como símbolos léxicos (tokens).
- Análise Sintática:
  - Analisa a estrutura de frases do programa.
- Análise Semântica:
  - Calcula o significado do programa.

Cadeias e Linguagens

- Alfabeto

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$
    - $\Sigma_2 = \{a, b, c, d, \ldots, z\}$

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$
    - $\Sigma_2 = \{a, b, c, d, \ldots, z\}$
- Cadeia

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$
    - $\Sigma_2 = \{a, b, c, d, \ldots, z\}$
- Cadeia
  - Sequência finita de símbolos de um alfabeto

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$
    - $\Sigma_2 = \{a, b, c, d, \ldots, z\}$
- Cadeia
  - Sequência finita de símbolos de um alfabeto
    - 010010 é cadeia sobre Σ_1 de tamanho 6.

Cadeias e Linguagens

- Alfabeto
  - Conjunto finito não vazio de símbolos.
  - Exemplos:
    - $\Sigma_1 = \{0, 1\}$
    - $\Sigma_2 = \{a, b, c, d, \ldots, z\}$
- Cadeia
  - Sequência finita de símbolos de um alfabeto
    - 010010 é cadeia sobre Σ_1 de tamanho 6.
  - O símbolo ε denota cadeia vazia de comprimento zero.

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)
- Dado um alfabeto Σ:

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)
- Dado um alfabeto Σ:
  - Σ^* é o conjunto de todas as cadeias finitas sobre Σ.

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)
- Dado um alfabeto Σ:
  - Σ^* é o conjunto de todas as cadeias finitas sobre Σ.
  - Σ^+ é o conjunto de todas as cadeias finitas sobre Σ menos a cadeia vazia.

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)
- Dado um alfabeto Σ:
  - Σ^* é o conjunto de todas as cadeias finitas sobre Σ.
  - Σ^+ é o conjunto de todas as cadeias finitas sobre Σ menos a cadeia vazia.
- Linguagem

Cadeias e Linguagens

- Dado uma cadeia w sobre Σ:
  - $w^0 = \varepsilon$
  - $w^k = w^{k-1}w$ para $k > 0$ (concatenação k vezes)
- Dado um alfabeto Σ:
  - Σ^* é o conjunto de todas as cadeias finitas sobre Σ.
  - Σ^+ é o conjunto de todas as cadeias finitas sobre Σ menos a cadeia vazia.
- Linguagem
  - Conjunto de todas cadeias de um alfabeto.

Cadeias e Linguagens

- Linguagem sobre Σ

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.
  - Subconjunto de Σ^*.

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.
  - Subconjunto de Σ^*.

- Dadas L_1 e L_2 linguagens sobre Σ:

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.
  - Subconjunto de Σ^*.

- Dadas L_1 e L_2 linguagens sobre Σ:
  - $L_1 \bigcup L_2 = \{x|x \in L_1$ ou $x \in L_2\}$

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.
  - Subconjunto de Σ^*.

- Dadas L_1 e L_2 linguagens sobre Σ:
  - $L_1 \bigcup L_2 = \{x | x \in L_1 \text{ ou } x \in L_2\}$
  - $L_1.L_2 = \{xy | x \in L_1 \text{ e } y \in L_2\}$

Cadeias e Linguagens

- Linguagem sobre Σ
  - Conjunto de cadeias em Σ.
  - Subconjunto de Σ^*.

- Dadas L_1 e L_2 linguagens sobre Σ:
  - $L_1 \bigcup L_2 = \{x|x \in L_1$ ou $x \in L_2\}$
  - $L_1.L_2 = \{xy|x \in L_1$ e $y \in L_2\}$
  - $L_1^* = \{x_1, x_2, \ldots, x_k|k \geqslant 0$ e cada $x_i \in L_1\}$

Cadeias e Linguagens

- Linguagem livre de contexto:

Cadeias e Linguagens

- Linguagem livre de contexto:
    - Linguagens descritas por uma gramática livre de contexto.

Cadeias e Linguagens

- Linguagem livre de contexto:
    - Linguagens descritas por uma gramática livre de contexto.
    - Úteis para especificar linguagens de programação.

Cadeias e Linguagens

- Linguagem livre de contexto:
  - Linguagens descritas por uma gramática livre de contexto.
  - Úteis para especificar linguagens de programação.
  - Vamos estudar com mais detalhes a seguir.

Cadeias e Linguagens

- Linguagem livre de contexto:
    - Linguagens descritas por uma gramática livre de contexto.
    - Úteis para especificar linguagens de programação.
    - Vamos estudar com mais detalhes a seguir.

- Linguagens regulares:

Cadeias e Linguagens

- Linguagem livre de contexto:
  - Linguagens descritas por uma gramática livre de contexto.
  - Úteis para especificar linguagens de programação.
  - Vamos estudar com mais detalhes a seguir.

- Linguagens regulares:
  - Caso particular de linguagens livre de contexto.

Cadeias e Linguagens

- Linguagem livre de contexto:
  - Linguagens descritas por uma gramática livre de contexto.
  - Úteis para especificar linguagens de programação.
  - Vamos estudar com mais detalhes a seguir.

- Linguagens regulares:
  - Caso particular de linguagens livre de contexto.
  - Linguagens que podem ser descritas através de expressões regulares ou reconhecidas por autômatos finitos.

Analisador Léxico

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

Analisador Léxico

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

- Descarta comentários e espaços em branco.

Símbolos Léxicos

Tipo	Exemplos
ID	foo n14 last var
NUM	15 13 2 1 2 5 121
REAL	1.2 .533 1e66.4 24.5e-10
IF	if
COMMA	,
NOTEQ	!=
EQ	==
LPAREN	(

Não Símbolos

Tipo	Exemplos
Comentário	/*try again*/
Diretivas de pré-processamento	#include <stdio.h>
Diretivas de pré-processamento	#define NUMS 5,6
macro	NUMS
Linhas em branco, tabs e novas linhas	

Exemplo

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

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

Exemplo

```
int paridade(int n) /* é par?*/
{    if (n%2==0))
      return 1;
      else return 0;
}
```

Retorno do analisador léxico:

???

Analisador Léxico

- Alguns símbolos têm um valor semântico associados a eles:

Analisador Léxico

- Alguns símbolos têm um valor semântico associados a eles:
  - IDs e NUMs.

Analisador Léxico

- Alguns símbolos têm um valor semântico associados a eles:
  - IDs e NUMs.
- Como são descritas as regras lexicográficas?

Analisador Léxico

- Alguns símbolos têm um valor semântico associados a eles:
  - IDs e NUMs.
- Como são descritas as regras lexicográficas?

Um identificador é uma sequência de letras e dígitos; o primeiro caractere deve ser uma letra. O underscore "_" conta como uma letra. Letras maiúsculas e minúsculas são diferentes. Se o fluxo de entrada resultar em um símbolo até um dado caractere, o próximo caractere é lido visando encontrar a maior string de caracteres possível que constitua um símbolo. Espaços, tabs, novas linhas e comentários são ignorados exceto quando eles servem de separadores de símbolos. Um espaço em branco é usado para separar identificadores adjacentes, palavras chaves e constantes.

Analisador Léxico

- Alguns símbolos têm um valor semântico associados a eles:
  - IDs e NUMs.
- Como são descritas as regras lexicográficas?

Um identificador é uma sequência de letras e dígitos; o primeiro caractere deve ser uma letra. O underscore "_" conta como uma letra. Letras maiúsculas e minúsculas são diferentes. Se o fluxo de entrada resultar em um símbolo até um dado caractere, o próximo caractere é lido visando encontrar a maior string de caracteres possível que constitua um símbolo. Espaços, tabs, novas linhas e comentários são ignorados exceto quando eles servem de separadores de símbolos. Um espaço em branco é usado para separar identificadores adjacentes, palavras chaves e constantes.

- Como os tokens são especificados?

Expressões Regulares

- Uma linguagem é um conjunto de strings.

Expressões Regulares

- Uma linguagem é um conjunto de strings.
- Uma string é uma sequência de símbolos.

Expressões Regulares

- Uma linguagem é um conjunto de strings.
- Uma string é uma sequência de símbolos.
- Estes símbolos estão definidos em um alfabeto finito

Expressões Regulares

- Uma linguagem é um conjunto de strings.
- Uma string é uma sequência de símbolos.
- Estes símbolos estão definidos em um alfabeto finito
  - Ex.: Linguagem C ou Pascal, linguagem dos primos, etc.

Expressões Regulares

- Uma linguagem é um conjunto de strings.
- Uma string é uma sequência de símbolos.
- Estes símbolos estão definidos em um alfabeto finito
  - Ex.: Linguagem C ou Pascal, linguagem dos primos, etc.
- Queremos poder dizer se uma string está ou não em uma linguagem.

Expressões Regulares

- **Símbolos:** Para cada símbolo a no alfabeto da linguagem, a expressão regular a representa a linguagem contendo somente a string a;

Expressões Regulares

- **Símbolos:** 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 .

Expressões Regulares

- **Símbolos:** 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 .
    - Ex.: A linguagem de $a|b$ contém as duas strings a e b.

Expressões Regulares

- **Concatenação:** Dadas duas expressões regulares M e N, o operador de concatenação ($\cdot$) gera uma nova expressão $M \cdot N$. Uma string está na linguagem de $M \cdot N$ 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

- **Concatenação:** Dadas duas expressões regulares M e N, o operador de concatenação ($\cdot$) gera uma nova expressão $M \cdot N$. Uma string está na linguagem de $M \cdot N$ 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.
    - Ex.: $(a|b) \ \cdot \ a$ contém as strings aa e ba.

Expressões Regulares

- **Concatenação:** Dadas duas expressões regulares M e N, o operador de concatenação ($\cdot$) gera uma nova expressão $M \cdot N$. Uma string está na linguagem de $M \cdot N$ 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.
  - Ex.: $(a|b) \;\cdot\; a$ contém as strings aa e ba.
- **Epsilon:** A expressão regular ε representa a linguagem cuja única string é a vazia.

Expressões Regulares

- **Concatenação:** Dadas duas expressões regulares M e N, o operador de concatenação ($\cdot$) gera uma nova expressão $M \cdot N$. Uma string está na linguagem de $M \cdot N$ 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.
  - Ex.: $(a|b) \cdot a$ contém as strings aa e ba.
- **Epsilon:** A expressão regular ε representa a linguagem cuja única string é a vazia.
  - Ex.: $(a \cdot b)|\varepsilon$ representa a linguagem {" ", "ab"}.

Expressões Regulares

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

Expressões Regulares

- **Repetição:** Dada uma expressão regular M, seu Kleene closure é M^*. Uma string está em M^* se ela é a concatenação de zero ou mais strings, todas em M.
  - Ex.: $((a|b) \cdot a)^*$ representa {" ", "aa", "ba", "$aaaa$", "$baaa$", "$aaba$", "$baba$","$aaaaa$", ...}.

Expressões Regulares

- $(0|1)^* \cdot 0$

Expressões Regulares

- $(0|1)^* \cdot 0$
  - Números binários múltiplos de 2.

Expressões Regulares

- $(0|1)^* \cdot \mathsf{0}$
  - Números binários múltiplos de 2.

- $b^*(abb^*)^*(a|\varepsilon)$

Expressões Regulares

- $(0|1)^* \cdot 0$
  - Números binários múltiplos de 2.

- $b^*(abb^*)^*(a|\varepsilon)$
  - Strings de a's e b's sem a's consecutivos.

Expressões Regulares

- $(0|1)^* \cdot \mathsf{0}$
  - Números binários múltiplos de 2.

- $b^*(abb^*)^*(a|\varepsilon)$
  - Strings de a's e b's sem a's consecutivos.

- $(a|b)^*aa(a|b)*$

Expressões Regulares

- $(0|1)^* \cdot 0$
  - Números binários múltiplos de 2.
- $b^*(abb^*)^*(a|\varepsilon)$
  - 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

- a - Um caractere ordinário por se só.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).
- M^+ - Repetição (uma ou mais vezes).

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).
- M^+ - Repetição (uma ou mais vezes).
- $M?$ - Opcional, zero ou uma ocorrência de M.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).
- M^+ - Repetição (uma ou mais vezes).
- $M?$ - Opcional, zero ou uma ocorrência de M.
- $[a-zA-Z]$ - Conjunto de caracteres alterados.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).
- M^+ - Repetição (uma ou mais vezes).
- $M?$ - Opcional, zero ou uma ocorrência de M.
- $[a-zA-Z]$ - Conjunto de caracteres alterados.
- . - Representa qualquer caractere, exceto nova linha.

Notação

- a - Um caractere ordinário por se só.
- ε - A string vazia.
- - Outra forma de escrever a string vazia.
- $M|N$ - Alternação.
- $M \cdot N$ - Concatenação.
- MN - Outra forma de concatenação.
- M^* - Repetição (zero ou mais vezes).
- M^+ - Repetição (uma ou mais vezes).
- $M?$ - Opcional, zero ou uma ocorrência de M.
- $[a - zA - Z]$ - Conjunto de caracteres alterados.
- . - Representa qualquer caractere, exceto nova linha.
- “a.+*” - Uma string de caracteres.

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF
    - if

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF
    - if
  - ID

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF
    - if
  - ID
    - $[a-z][a-z0-9]^*$

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF
    - if
  - ID
    - $[a - z][a - z0 - 9]^*$
  - NUM

Expressões Regulares

- Como seriam as expressões regulares para os seguintes tokens?
  - IF
    - if
  - ID
    - $[a - z][a - z0 - 9]^*$
  - NUM
    - $[0 - 9]+$

Expressões Regulares

- Quais símbolos representam as seguintes expressões regulares?
    - $[0-9]+$"."$[0-9]^*)|([0-9]^*$"."$[0-9]^+)$
        - ???
    - ("- -"$[a-z]^*$"\n")|(" "|"\n" | "\t")$^+$
        - ???
    - .
        - ???

Expressões Regulares

- Quais símbolos representam as seguintes expressões regulares?
  - $[0-9]+$“ . ”$[0-9]*)|([0-9]*$“ . ”$[0-9]^{+})$
    - ??? REAL
  - (“- -”$[a-z]^{*}$“ \n”)|(“ ”|“\n” | “\t”)$^{+}$
    - ??? nenhum token, somente comentário, brancos, nova linha e tab
  - .
    - ??? ERROR

Analisador Léxico

- Ambiguidade:

Analisador Léxico

- Ambiguidade:
  - if8 é um ID ou dois tokens IF e NUM(8)?

Analisador Léxico

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

Analisador Léxico

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

Analisador Léxico

- Ambiguidade:
  - 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 símbolo sempre é a substring mais longa possível de ser casada.

Analisador Léxico

- Ambiguidade:
  - 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 símbolo 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.

Analisador Léxico

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

Analisador Léxico

- A especificação deve ser completa, sempre reconhecendo uma substring da entrada
  - Mas quando estiver errada? Use uma regra com o "."

Analisador 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?

Analisador 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?)

Seção 2

Autômatos Finitos

Autômatos Finitos

- Expressões regulares são convenientes para especificar os símbolos.

Autômatos Finitos

- Expressões regulares são convenientes para especificar os símbolos.
- Precisamos de um formalismo que possa ser convertido em um programa de computador.

Autômatos Finitos

- Expressões regulares são convenientes para especificar os símbolos.
- 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:

Autômatos Finitos

- Um autômato finito possui:
  - Um conjunto finito de estados.

Autômatos Finitos

- Um autômato finito possui:
    - Um conjunto finito de estados.
    - Arestas levando de um estado a outro, anotada com um símbolo.

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.

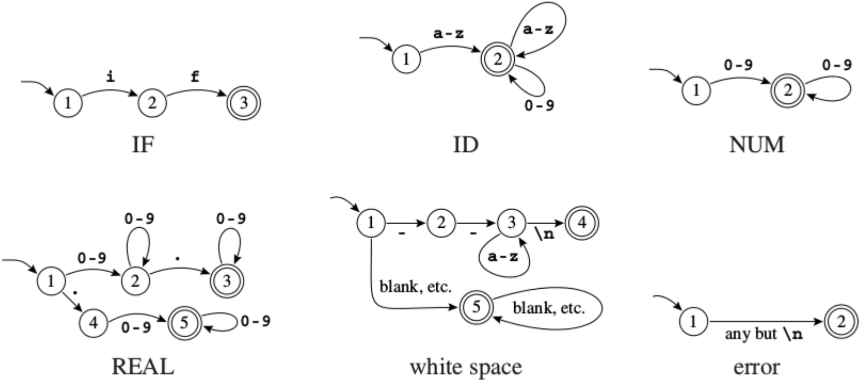
Autômatos Finitos

- 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.

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



Autômato Finito Determinístico

- DFAs não podem apresentar duas arestas que deixam o mesmo estado, anotadas com o mesmo símbolo.

Autômato Finito Determinístico

- 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.

Autômato Finito Determinístico

- 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

Autômato Finito Determinístico

- 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.

Autômato Finito Determinístico

- 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.

Autômato Finito Determinístico

- Consigo combinar os autômatos definidos para cada símbolo de maneira a ter um único autômato que possa ser usado como analisador léxico?

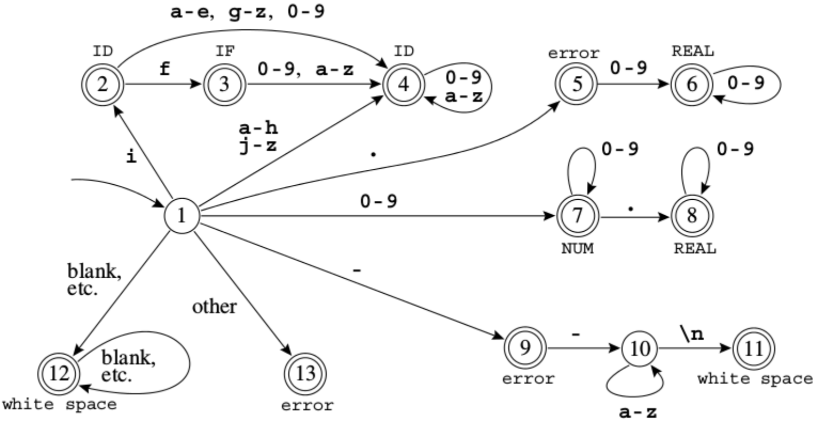
Autômato Finito Determinístico

- Consigo combinar os autômatos definidos para cada símbolo de maneira a ter um único autômato que possa ser usado como analisador léxico?
    - Sim.

Autômato Finito Determinístico

- Consigo combinar os autômatos definidos para cada símbolo de maneira a ter um único autômato que possa ser usado como analisador léxico?
  - Sim.
  - Veremos um exemplo ad-hoc e mais adiante mecanismos formais para esta tarefa.

Autômatos Finitos



Autômato Combinado

- Estados finais nomeados com o respectivo símbolo.

Autômato Combinado

- Estados finais nomeados com o respectivo símbolo.
- Alguns estados apresentam características de mais de um autômato anterior.

Autômato Combinado

- Estados finais nomeados com o respectivo símbolo.
- Alguns estados apresentam características de mais de um autômato anterior.
  - Ex.: 2.

Autômato Combinado

- Estados finais nomeados com o respectivo símbolo.
- Alguns estados apresentam características de mais de um autômato anterior.
    - Ex.: 2.
- Como ocorre a quebra de ambiguidade entre ID e IF?

Autômato Combinado

```
int edges[][] = { /* ... 0 1 2 ... - ... e f g h i j ... */ —ENTRADA—
/* state 0 */    {0,0, ... 0,0,0 ... 0 ... 0,0,0,0,0,0 ... }, -N ARESTAS-
/* state 1 */    {0,0, ... 7,7,7 ... 9 ... 4,4,4,4,2,4 ... },
/* state 2 */    {0,0, ... 4,4,4 ... 0 ... 4,3,4,4,4,4 ... },
/* state 3 */    {0,0, ... 4,4,4 ... 0 ... 4,4,4,4,4,4 ... },
/* state 4 */    {0,0, ... 4,4,4 ... 0 ... 4,4,4,4,4,4 ... },
/* state 5 */    {0,0, ... 6,6,6 ... 0 ... 0,0,0,0,0,0 ... },
/* state 6 */    {0,0, ... 6,6,6 ... 0 ... 0,0,0,0,0,0 ... },
/* state 7 */    {0,0, ... 7,7,7 ... 0 ... 0,0,0,0,0,0 ... },
/* state 8 */    {0,0, ... 8,8,8 ... 0 ... 0,0,0,0,0,0 ... },
etc}
```

Maior Substring

- A tabela anterior é usada para aceitar ou recusar uma string.

Maior Substring

- A tabela anterior é usada para aceitar ou recusar uma string.
- Porém, precisamos garantir que a maior string seja reconhecida.

Maior Substring

- A tabela anterior é usada para aceitar ou recusar uma string.
- Porém, precisamos garantir que a maior string seja reconhecida.
- Necessitamos de duas informações.

Maior Substring

- A tabela anterior é usada para aceitar ou recusar uma string.
- Porém, precisamos garantir que a maior string seja reconhecida.
- Necessitamos de duas informações.
  - Último estado final.

Maior Substring

- A tabela anterior é usada para aceitar ou recusar uma string.
- Porém, precisamos garantir que a maior string seja reconhecida.
- Necessitamos de duas informações.
    - Último estado final.
    - Posição da entrada no último estado final.

Maior Substring

Last Final	Current State	Current Input	Accept Action
0	1	`if --not-a-com`	
2	2	`if --not-a-com`	
3	3	`if --not-a-com`	
3	0	`if --not-a-com`	*return* IF
0	1	`if --not-a-com`	
12	12	`if --not-a-com`	
12	0	`if --not-a-com`	*found white space; resume*
0	1	`if --not-a-com`	
9	9	`if --not-a-com`	
9	10	`if --not-a-com`	
9	10	`if --not-a-com`	
9	10	`if --not-a-com`	
9	10	`if --not-a-com`	
9	0	`if --not-a-com`	*error, illegal token '-'; resume*
0	1	`if --not-a-com`	
9	9	`if --not-a-com`	
9	0	`if --not-a-com`	*error, illegal token '-'; resume*

Autômato Finito Não-Determinístico

- Pode ter mais de uma aresta saindo do mesmo estado com o mesmo símbolo.

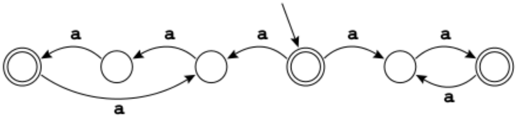
Autômato Finito Não-Determinístico

- Pode ter mais de uma aresta saindo do mesmo estado com o mesmo símbolo.
- Pode ter arestas anotadas com o símbolo ε.

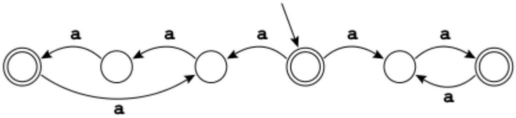
Autômato Finito Não-Determinístico

- Pode ter mais de uma aresta saindo do mesmo estado com o mesmo símbolo.
- Pode ter arestas anotadas com o símbolo ε.
    - Essa aresta pode ser percorrida sem consumir nenhum caractere de entrada.

Autômato Finito Não-Determinístico

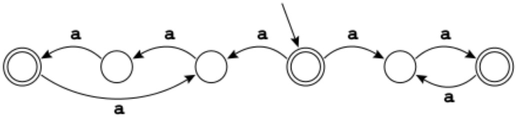


Autômato Finito Não-Determinístico



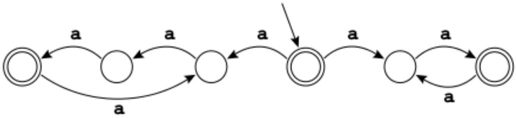
- Que linguagem este autômato reconhece?

Autômato Finito Não-Determinístico



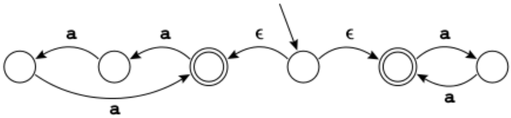
- Que linguagem este autômato reconhece?
  - **Obs.:** Ele é obrigado a aceitar a string se existir alguma escolha de caminho que leva à aceitação.

Autômato Finito Não-Determinístico

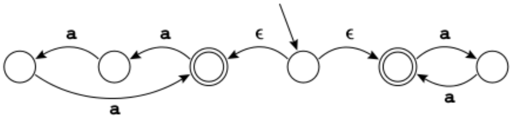


- Que linguagem este autômato reconhece?
  - **Obs.:** Ele é obrigado a aceitar a string se existir alguma escolha de caminho que leva à aceitação.
  - vazio ou um número de 'as' que são múltiplos de 3 ou de 2.

Autômato Finito Não-Determinístico



Autômato Finito Não-Determinístico



- Que linguagem este autômato reconhece?

Autômato Finito Não-Determinístico

- Não são apropriados para transformar em programas de computador.

Autômato Finito Não-Determinístico

- Não são apropriados para transformar em programas de computador.
  - “Adivinhar” qual caminho deve ser seguido não é uma tarefa facilmente executada pelo hardware dos computadores.

Autômato Finito Não-Determinístico

- Não são apropriados para transformar em programas de computador.
  - "Adivinhar" qual caminho deve ser seguido não é uma tarefa facilmente executada pelo hardware dos computadores.

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

Autômato Finito Não-Determinístico

- Exemplos:

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).

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).

Convertendo ER's para NFA's

- Podemos definir essa conversão de maneira indutiva pois:

Convertendo ER's para NFA's

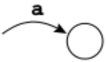
- Podemos definir essa conversão de maneira indutiva pois:
    - Uma ER é primitiva (único símbolo ou vazio) ou é uma combinação de outras ERs.

Convertendo ER's para NFA's

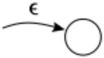
- Podemos definir essa conversão de maneira indutiva pois:
    - Uma ER é primitiva (único símbolo ou vazio) ou é uma combinação de outras ERs.
    - O mesmo vale para NFAs.

Convertendo ER's para NFA's

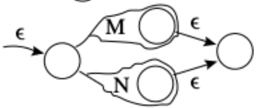
a



ϵ

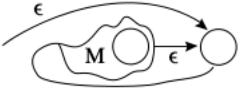


$M \mid N$



$M \cdot N$

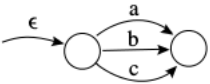
M^*



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

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

[abc]



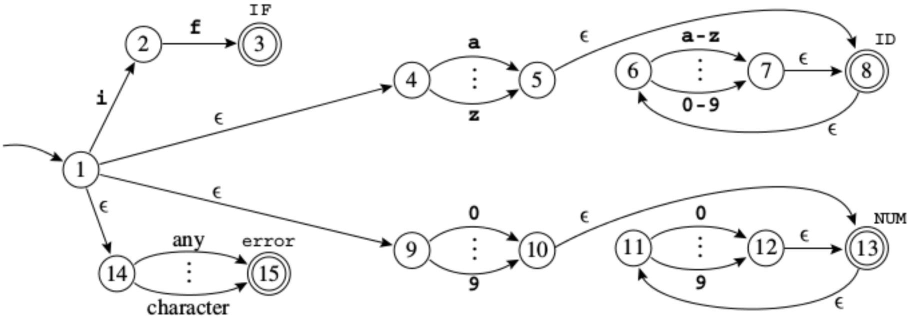
"abc" constructed as $\mathbf{a} \cdot \mathbf{b} \cdot \mathbf{c}$

Exemplo

- ERs para IF, ID, NUM e ERROR

Exemplo

- ERs para IF, ID, NUM e ERROR



Algoritmo de Thompson - ER para AFN

```
Função converter_er_para_afn(pilha):
    símbolo = pilha desempilhar
    Se o símbolo for uma concatenação ('.'):
        AFN1 = desempilhar pilha
        AFN2 = desempilhar pilha
        Criar um AFN com uma transição do final de AFN1 para o início de AFN2
        O do início do AFN é o início de AFN1 e o final é o final de AFN2
        Empilhar AFN
    Se o símbolo for uma união ('|'):
        AFN1 = desempilhar pilha
        AFN2 = desempilhar pilha
        Criar um AFN com um novo estado inicial q1 e um novo estado final q2
        Adicionar uma transição de q1 para os estados iniciais de AFN1 e AFN2 com epsilon
        Adicionar uma transição dos finais de AFN1 e AFN2 para q2 com epsilon
        Empilhar AFN
    Se o símbolo for um fechamento de Kleene ('*'):
        AFN1 = desempilhar pilha
        Criar AFN com um novo estado inicial q1 e um novo estado final q2
        Adicionar uma transição de q1 para q2 com epsilon
        Adicionar uma transição de q1 para o início do AFN1 com epsilon
        Adicionar uma transição do final de AFN1 para q2 com epsilon
        Adicionar uma transição de q2 para o início do AFN1 com epsilon
        Empilhar AFN
    Retornar o AFN gerado e a pilha
```

Algoritmo de Thompson - ER para AFN

```
Função preparar_lista(expressao_regular):
    lista_ER = []
    Para cada símbolo na expressao_regular:
        Se o símbolo for um operando (caractere alfabético ou epsilon):
            Criar um AFN com um estado inicial q1 e um estado final q2
            Adicionar uma transição do estado q1 para o estado q2, com o símbolo lido ou epsilon
            Adicionar AFN na lista_ER
        Se não:
            Adicionar símbolo na lista_ER
    Retornar lista_ER


Função converter_ER_to_AFN (expressao_regular):
    pilha = []
    AFN = []
    lista_ER = preparar_lista (expressao_regular)
    Para cada símbolo na lista_ER:
        Se símbolo não for ')':
            Empilhar símbolo
        Se símbolo for ')':
            AFN, pilha = Executar converter_er_para_afn(pilha) até o primeiro '('
    Retornar AFN gerado
```

Algoritmo de Thompson - ER para AFN

- O algoritmo de Thompson exige que as expressões regulares estejam bem formuladas, com o uso de parênteses corretos.
- Ex. $(a|b|c)$ deve ser escrito como $((a|b)|c)$.
- Bem como $(a|b)^*$ deve ser escrito como $((a|b)^*)$.

NFA vs. DFA

- DFAs são facilmente simuláveis por programas de computador.

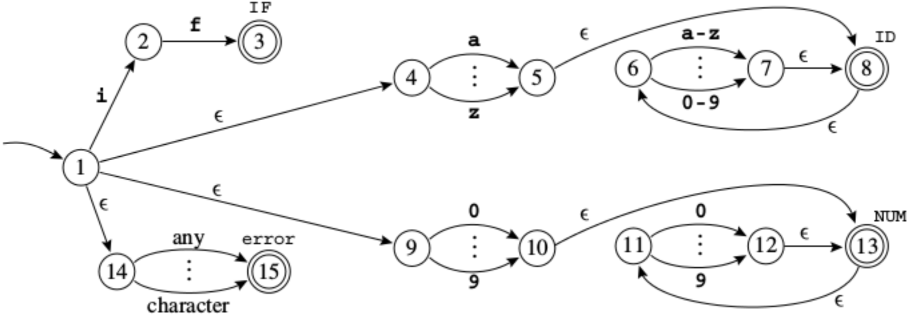
NFA vs. DFA

- DFAs são facilmente simuláveis por programas de computador.
- NFAs são mais complexos, pois o programa teria que “adivinhar” o melhor caminho em alguns momentos.

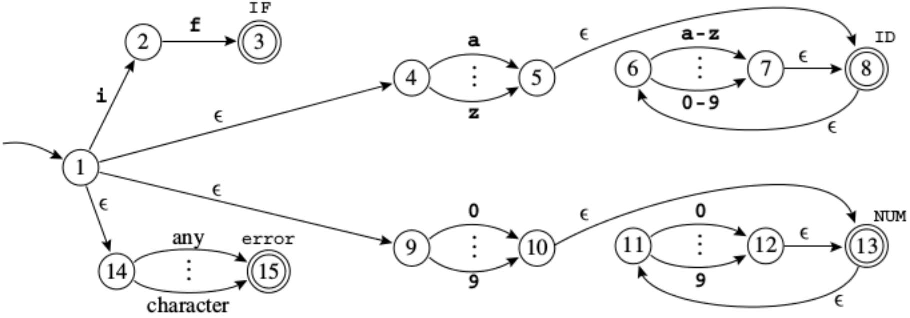
NFA vs. DFA

- DFAs são facilmente simuláveis por programas de computador.
- NFAs são mais complexos, pois o programa teria que “adivinhar” o melhor caminho em alguns momentos.
- Outra alternativa seria tentar todas as possibilidades.

Simulando NFA para "in"

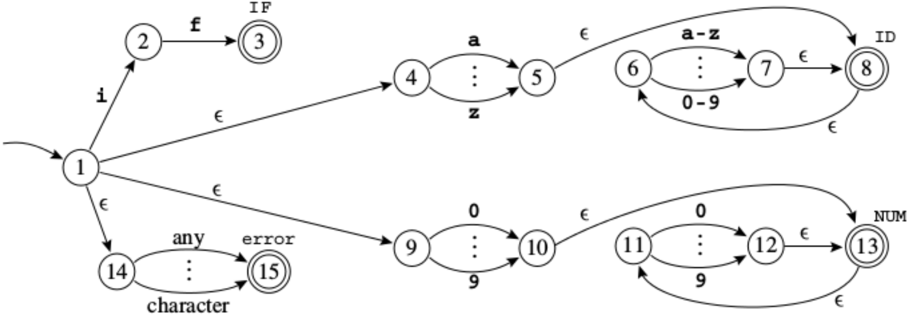


Simulando NFA para “in”



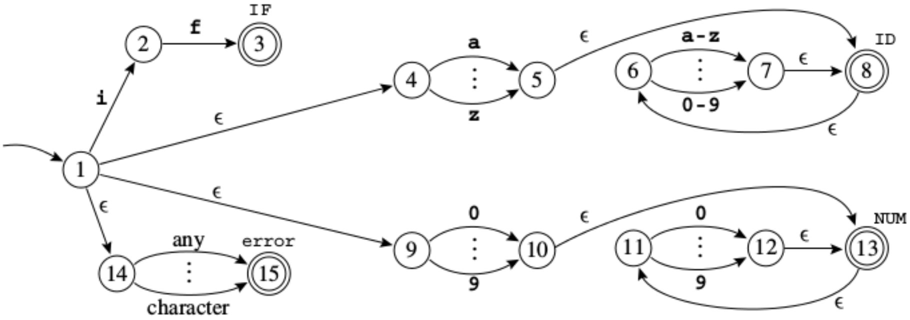
- Início (1) $\Longrightarrow$ NFA pode estar em 1,4,9,14.

Simulando NFA para “in”



- Início (1) $\Longrightarrow$ NFA pode estar em 1,4,9,14.
- Consome i $\Longrightarrow$ NFA pode estar em 2,5,6,8,15.

Simulando NFA para "in"



- Início (1) $\Longrightarrow$ NFA pode estar em 1,4,9,14.
- Consome i $\Longrightarrow$ NFA pode estar em 2,5,6,8,15.
- Consome n $\Longrightarrow$ NFA pode estar em 6,7,8.

ε-Closure

- Edge(s, c): todos os estados alcançáveis a partir de s, consumindo c.

ε-Closure

- Edge(s, c): todos os estados alcançáveis a partir de s, consumindo c.
- Closure(S): todos os estados alcançáveis a partir do conjunto S, sem consumir caractere de entrada.

ε-Closure

- Edge(s, c): todos os estados alcançáveis a partir de s, consumindo c.
- Closure(S): todos os estados alcançáveis a partir do conjunto S, sem consumir caractere de entrada.
- Closure(S) é o menor conjunto T, tal que :

ε-Closure

- Edge(s, c): todos os estados alcançáveis a partir de s, consumindo c.
- Closure(S): todos os estados alcançáveis a partir do conjunto S, sem consumir caractere de entrada.
- Closure(S) é o menor conjunto T, tal que :

$$T = S \bigcup (\bigcup_{S \in T} \textbf{edge}(s, \varepsilon)) \tag{1}$$

Algoritmo

Computado por iteração:

$$
\begin{array}{l}
T \longleftarrow S \\
\texttt{\textbf{repita}}\ \ T' \longleftarrow T \\
\qquad\quad T = T' \bigcup (\bigcup_{S \in T'} \texttt{\textbf{edge}}(s, \varepsilon)) \\
\texttt{\textbf{até}}\ \ T = T'
\end{array}
$$

Algoritmo

Da maneira que fizemos a simulação, vamos definir:

Algoritmo

Da maneira que fizemos a simulação, vamos definir:

$$\mathbf{DFAedge}\,(d, c) \;=\; \mathbf{closure}\left(\bigcup_{s \in d} \mathbf{edge}(s, c)\right)$$

Algoritmo

Da maneira que fizemos a simulação, vamos definir:

$$\texttt{\textbf{DFAedge}}(d, c) \;=\; \texttt{\textbf{closure}}\left(\bigcup_{s \in d} \texttt{\textbf{edge}}(s, c)\right)$$

como o conjunto de estados do NFA que podemos atingir a partir do conjunto d, consumindo c.

Algoritmo

Da maneira que fizemos a simulação, vamos definir:

$$\texttt{\textbf{DFAedge}}(d, c) \;=\; \texttt{\textbf{closure}}(\textstyle\bigcup_{s \in d} \texttt{\textbf{edge}}(s, c))$$

como o conjunto de estados do NFA que podemos atingir a partir do conjunto d, consumindo c.
Estado inicial s_1 e string $c_1, \ldots, c_k$.

Algoritmo

Da maneira que fizemos a simulação, vamos definir:

$$\textbf{\texttt{DFAedge}}\,(d, c) \;=\; \textbf{\texttt{closure}}\left(\bigcup_{s \in d} \textbf{\texttt{edge}}(s, c)\right)$$

como o conjunto de estados do NFA que podemos atingir a partir do conjunto d, consumindo c.

Estado inicial s_1 e string $c_1, \ldots, c_k$.

$$
\begin{array}{l}
d \longleftarrow \textbf{\texttt{closure}}(\{s_1\}) \\
\texttt{para } i \longleftarrow 1 \texttt{ até } k \\
\qquad d \longleftarrow \textbf{\texttt{DFAedge}}(d, c_i)
\end{array}
$$

Convertendo NFA em DFA

- Manipular esses conjuntos de estados é muito caro durante a simulação.

Convertendo NFA em DFA

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

Convertendo NFA em DFA

- Manipular esses conjuntos de estados é muito caro durante a simulação.
- Solução:
  - Calcular todos eles antecipadamente.

Convertendo NFA em DFA

- Manipular esses conjuntos de estados é muito caro durante a simulação.
- Solução:
  - Calcular todos eles antecipadamente.
- Isso converte um NFA em um DFA!!!

Convertendo NFA em DFA

- Manipular esses conjuntos de estados é muito caro durante a simulação.
- Solução:
  - Calcular todos eles antecipadamente.
- Isso converte um NFA em um DFA!!!
  - Cada conjunto de estados no NFA se torna um estado no DFA.

Algoritmo

states[0] $\leftarrow$ {};   states[1] $\leftarrow$ **closure**($\{s_1\}$)
$p \leftarrow 1$; $j \leftarrow 0$
while $j \leq p$
  foreach $c \in \Sigma$
    $e \leftarrow$ **DFAedge**(states[j], c)
    if $e =$ states[i] for some $i \leq p$
      then trans[j, c] $\leftarrow i$
      else $p \leftarrow p + 1$
        states[p] $\leftarrow e$
        trans[j, c] $\leftarrow p$
  $j \leftarrow j + 1$

Convertendo NFA em DFA

- O estado d é final se qualquer um dos estados de `states[`d`]` for final.

Convertendo NFA em DFA

- O estado d é final se qualquer um dos estados de `states[`d`]` for final.
- Pode haver vários estados finais em `states[`d`]`.

Convertendo NFA em DFA

- O estado d é final se qualquer um dos estados de `states`[d] for final.
- Pode haver vários estados finais em `states`[d].
  - d será anotado com o token que ocorrer primeiro na especificação léxica (ERs) $\Longrightarrow$ Regra de prioridade

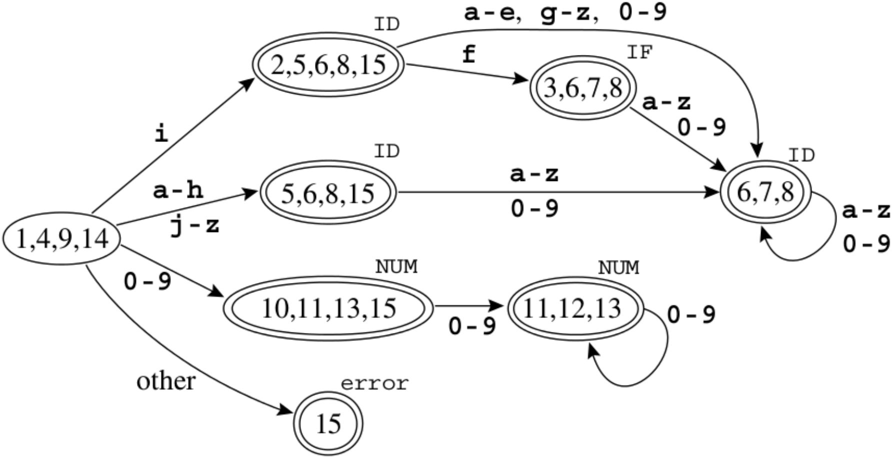
Convertendo NFA em DFA

- O estado d é final se qualquer um dos estados de `states[`d`]` for final.
- Pode haver vários estados finais em `states[`d`]`.
  - d será anotado com o token que ocorrer primeiro na especificação léxica (ERs) $\Longrightarrow$ Regra de prioridade
- Ao final

Convertendo NFA em DFA

- O estado d é final se qualquer um dos estados de `states[`d`]` for final.
- Pode haver vários estados finais em `states[`d`]`.
  - d será anotado com o token que ocorrer primeiro na especificação léxica (ERs) $\Longrightarrow$ Regra de prioridade
- Ao final
  - Descartar `states[]` e usar `trans[]` para análise léxica.

Convertendo NFA em DFA



Convertendo NFA em DFA

- Esse é o menor autômato possível para essa linguagem?

Convertendo NFA em DFA

- Esse é o menor autômato possível para essa linguagem?
  - Não!

Convertendo NFA em DFA

- Esse é o menor autômato possível para essa linguagem?
  - Não!
  - Existem estados que são equivalentes!

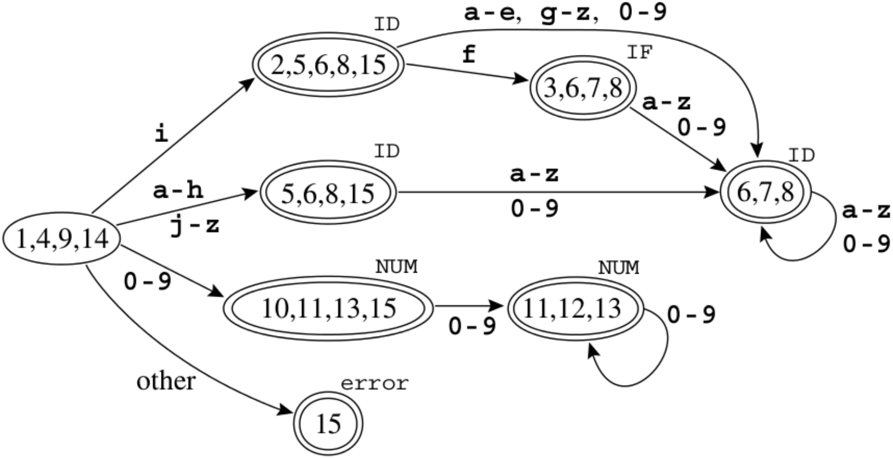
Convertendo NFA em DFA

- Esse é o menor autômato possível para essa linguagem?
  - Não!
  - Existem estados que são equivalentes!

- s_1 e s_2 são equivalentes quando o autômato aceita σ começando em s_1 se e somente se ele também aceita σ começando em s_2.

Convertendo NFA em DFA

Quais estados são equivalentes no autômato?



Convertendo NFA em DFA

- Como encontrar estados equivalentes?

Convertendo NFA em DFA

- Como encontrar estados equivalentes?
  - $\texttt{trans}[s_1, c] = \texttt{trans}[s_2, c]$ para para todo c.

Convertendo NFA em DFA

- Como encontrar estados equivalentes?
  - $\texttt{trans}[s_1, c] = \texttt{trans}[s_2, c]$ para para todo c.
  - Não é suficiente!!!

Convertendo NFA em DFA

- Como encontrar estados equivalentes?
  - $\texttt{trans}[s_1, c] = \texttt{trans}[s_2, c]$ para para todo c.
  - Não é suficiente!!!

- s_1 e s_2 são equivalentes quando o autômato aceita σ começando em s_1 se e somente se ele também aceita σ começando em s_2.

Análise Léxica

Universidade Federal do Ceará - Campus de Quixadá

`Lucas Ismaily`
`ismailybf@ufc.br`

Semestre 2024.1

Compiladores
Baseado nos slides do Prof. Sandro Rigo (IC-Unicamp)