Prof. Dr. Daniel Leal Souza

Autômatos e Compiladores (EC8MA)

Compiladores (CC6MA)

daniel.leal.souza@gmail.com
daniel.souza@prof.cesupa.br

2025

Introdução

Compiladores e Linguagens Formais:

- Normalmente pensamos em linguagens de programação
  - Programa fonte = algoritmo
  - Programa objeto = executável
- Mas compiladores podem ser utilizados em outros contextos
  - SQL
    - Programas SQL não são algoritmos
    - Mas ainda assim existe um compilador (ou interpretador)
  - HTML
    - O compilador (interpretador) no navegador lê o programa (página) HTML e o traduz em ações
      - Que desenham uma página Latex

Introdução

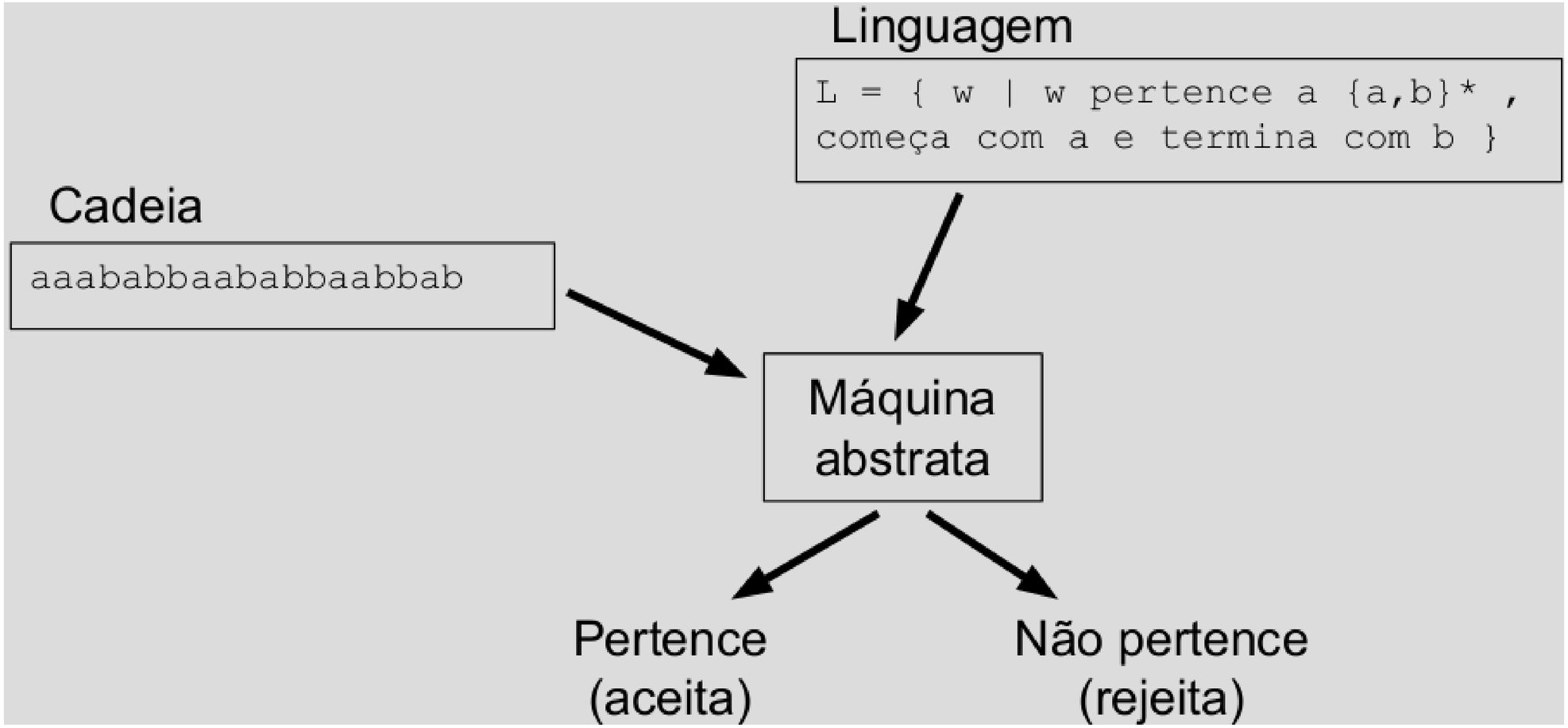
Compiladores e Linguagens Formais:

- Conceito de linguagem
  - É o mesmo visto na disciplina de Linguagens Formais e Autômatos (LFA)
  - Linguagens são descrições formais de problemas
- Mas aqui, o objetivo é fazer com que o computador entenda a semântica da linguagem
  - Ou seja: não basta decidir se uma cadeia faz parte ou não da linguagem
  - É necessário “entender” o que significa a cadeia
  - E traduzi-la para as ações desejadas!

Introdução

LFA x Compiladores:

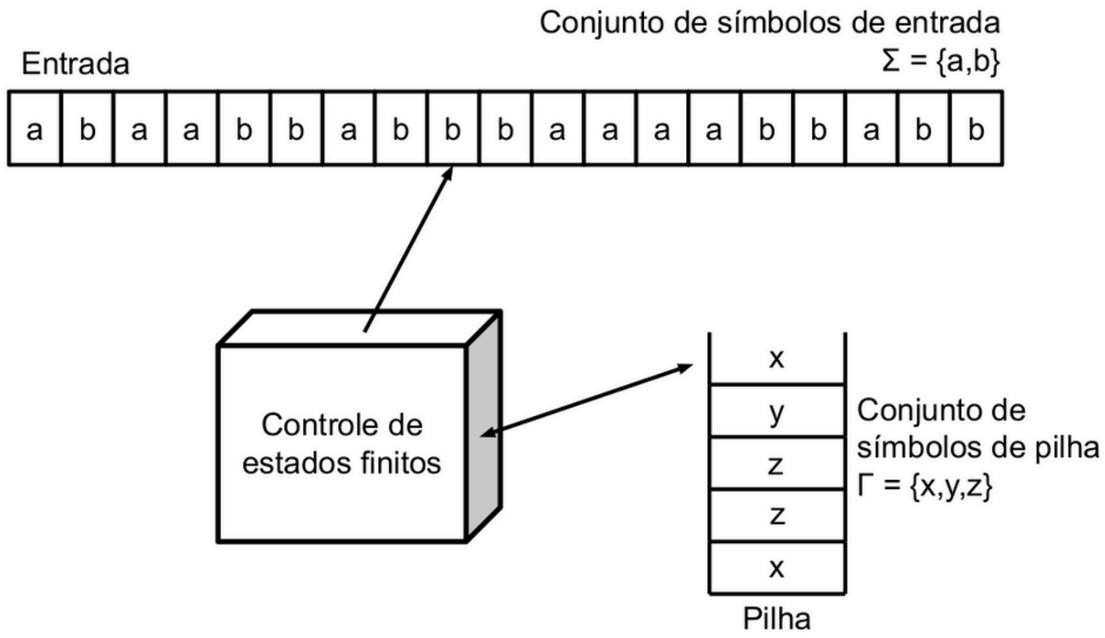
- Em Linguagens Formais e Autômatos (LFA):



Introdução

LFA x Compiladores:

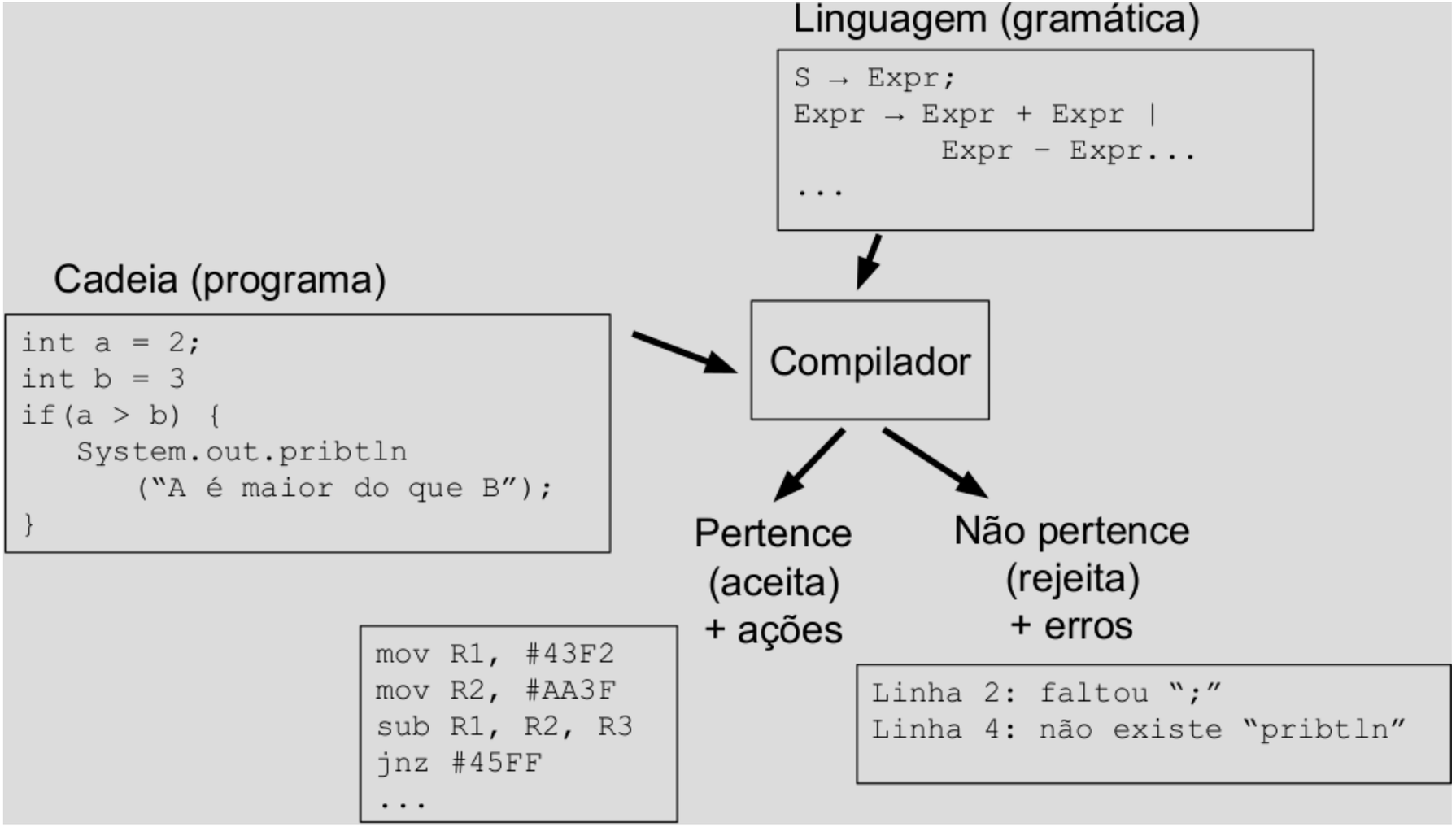
- Trabalharemos com ***linguagens livres de contexto (LLC)***
  - Linguagens livres de contexto são bons modelos de programas que tipicamente queremos escrever
- Portanto, precisamos de um **PDA (Autômato de Pilha)**
  - Modelo abstrato simples (Autômato Finito com uma pilha de memória)
    - "Fácil" de implementar;
    - Falaremos sobre PDAs mais adiante;



Introdução

LFA x Compiladores:

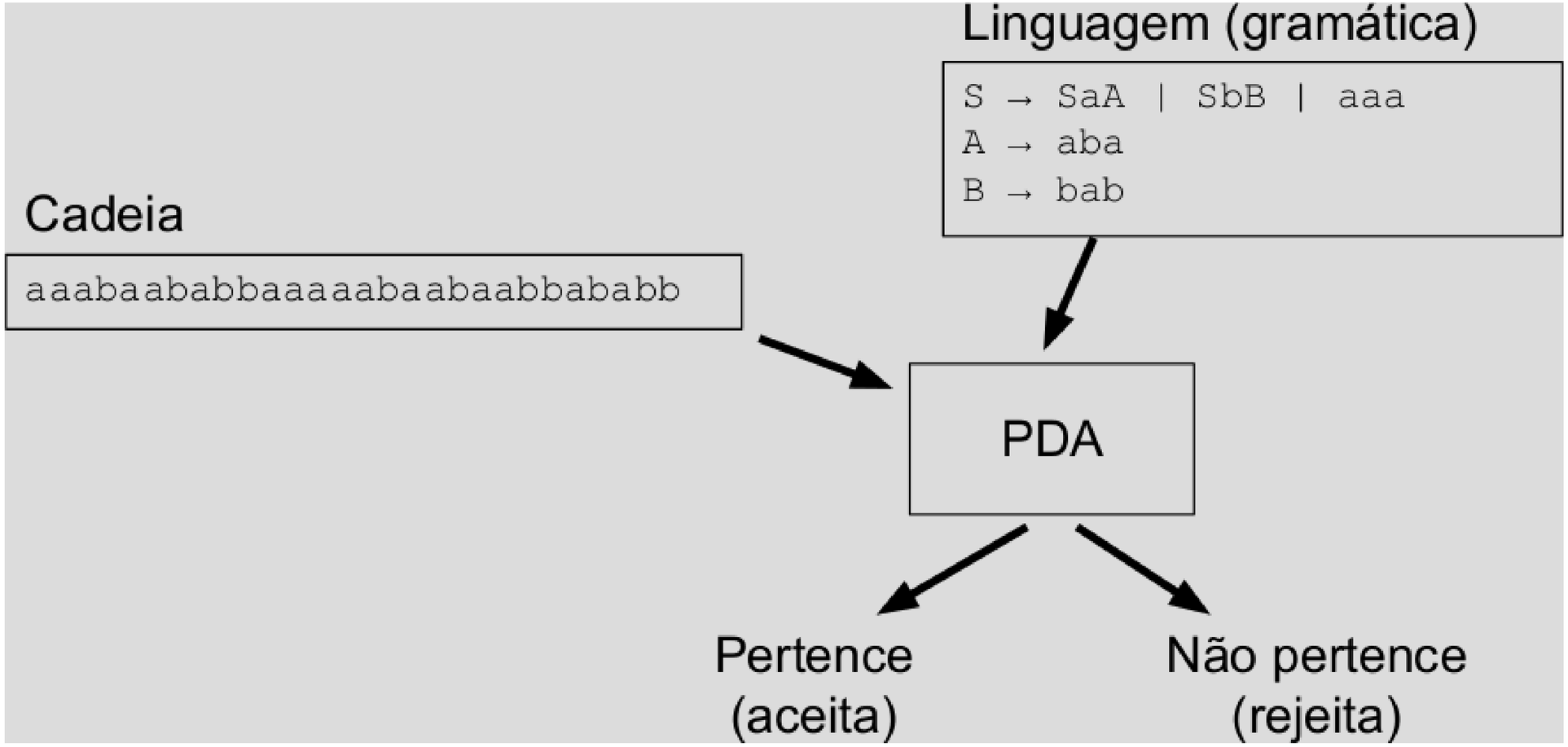
- Em Compiladores:



Introdução

LFA x Compiladores:

- Em LFA:



Introdução

LFA x Compiladores:

- Portanto, um compilador é essencialmente um Autômato de Pilha
- Ele usa uma pilha e estados para reconhecer as cadeias
  - Análise sintática!
- E traduz (ou executa) para "ações semânticas" (contexto)
  - Definidas sobre as regras da linguagem
  - Ex:

    - ***Expr → Expr + Expr {ação de soma} \| Expr – Expr {ação de subtração}***

- Mas tem (sempre tem) um problema...

Introdução

Linguagens não livres de contexto:

- **Considere a seguinte cadeia, em uma linguagem de programação típica. Em algumas LP, variáveis precisam ser declaradas antes de serem utilizadas:**

```
String numero = 0;
if (nmero > 0){
  System.out.println(“Nunca vai entrar aqui”);
}
```

Irá acusar erro aqui, pois a variável "***nmero"*** não foi declarada

Introdução

Linguagens não livres de contexto:

- Outros exemplos: declaração de pacotes, macros, chamada de funções, etc.
- Ou seja, gramáticas livres de contexto não conseguem impor todas as restrições de uma linguagem de programação típica.
- Portanto, fica a pergunta: podemos usar um PDA?
  - Refraseando: precisamos de um autômato mais poderoso?
    - Uma máquina de Turing com fita limitada?
    - Um PDA com duas pilhas?
  - Mas o PDA simples é tão ... simples!!
    - Eu queria MUITO poder usar um PDA simples

Introdução

Hierarquia de Chomsky:

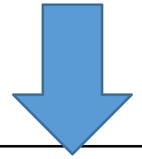
Hierarquia	Gramáticas	Linguagens	Autômato mínimo
Tipo 0	Recursivamente enumeráveis ou Irrestritas	Recursivamente enumeráveis	Máquinas de Turing
Tipo 1	Sensíveis ao contexto	Sensíveis ao contexto	MT com fita limitada
Tipo 2	Livres de Contexto	Livres de Contexto	Autômatos com pilha (PDA)
Tipo 3	Regulares (Expressão regular)	Regulares	Autômatos finitos

Introdução

Resolvendo a limitação da LLC...

```
int a = 2;
int b = 3;
if(a > b){
  System.out.println("A é maior do que B");
}
```

```
NOME1 = a
NOME2 = b
CONSTANTE1 = 2
CONSTANTE2 = 3
CLASSE1 = System
MEMBRO1 = out
METODO1 = println
CONSTANTE_STR1 =
  "A é maior do que B"
```



```
TIPO NOME = CONSTANTE;
TIPO NOME = CONSTANTE;
if(NOME > NOME){
CLASSE.MEMBRO.METODO(CONSTANTE_STR);
}
```

Introdução

Resolvendo a limitação da LLC...

```
int a = 2;
int b = 3;
if(a > b){
  System.out.println("A é maior do que B");
}
```

Não é livre de contexto (SEMÂNTICA)

```
TIPO NOME = CONSTANTE;
TIPO NOME = CONSTANTE;
if(NOME > NOME){
CLASSE.MEMBRO.METODO(CONSTANTE_STR);
}
```

É livre de contexto (SINTÁTICA)

```
NOME1 = a
NOME2 = b
CONSTANTE1 = 2
CONSTANTE2 = 3
CLASSE1 = System
MEMBRO1 = out
METODO1 = println
CONSTANTE_STR1 =
  "A é maior do que B"
```

Introdução

Compiladores e LLC:

- Sintaxe (forma e ordem) vs semântica (significado e contexto)
  - Compilador precisa lidar com ambos
  - Mas até onde vai a sintaxe?
  - Onde começa a semântica?

- int a = “Alo mundo”;
  - Aqui tem um erro sintático ou semântico?
    - R: Erro semântico.
- No mundo das Linguagens de Programação:
  - Variável inteira PEDE valor inteiro!
  - Uma variável deve ter sido declarada antes de ser usada!

Introdução

Compiladores e Linguagens Livres de Contexto:

- Em compiladores:
  - Tudo que está na Gramática Livre de Contexto (LLC) é **sintático**
  - O resto é considerado **semântico**
- Motivo: o uso de Autômatos de Pilha simples
  - Ou seja, adotamos o ponto de vista das linguagens livres de contexto, por praticidade
- Faz sentido, pois em LFA, temos:
  - Árvore de análise sintática
  - Somente com elementos da gramática

DÚVIDAS?

Referências

Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compiladores - princípios, técnicas e ferramentas. Pearson, 2007.

José Neto, João. Apresentação à compilação. 2ª edição. Rio de Janeiro: Elsevier, 2016.

Notas de aula do professor Daniel Lucrédio - UFSCar.