

Tema 1

Compiladores, Linguagens e Gramáticas

Introdução

Compiladores, 2º semestre 2024-2025

Miguel Oliveira e Silva
DETI, Universidade de Aveiro

Enquadramento

Linguagens de programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do código

Linguagens: Definição como Conjunto

Conceito básicos e terminologia

Operações sobre palavras

Operações sobre linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente limitados

Autómatos de pilha

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

**Estrutura de um
Compilador**

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

**Implementação de um
Compilador**

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

**Linguagens: Definição
como Conjunto**

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

**Introdução às
gramáticas**

Hierarquia de Chomsky
Autómatos

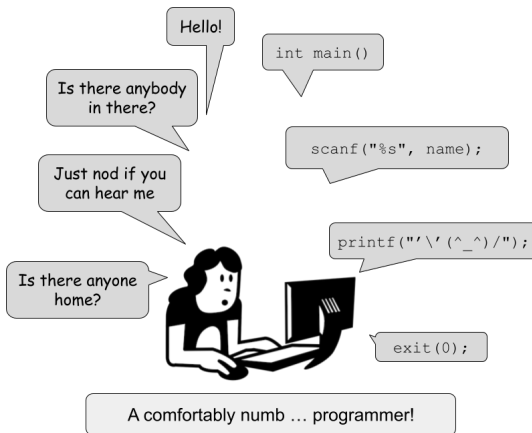
Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Enquadramento

Enquadramento

- Nesta disciplina vamos falar sobre **linguagens** – o que são e como as podemos definir – e sobre **compiladores** – ferramentas que as reconhecem e que permitem realizar acções como consequência desse processo.



Enquadramento

Linguagens de programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do código

Linguagens: Definição
como Conjunto

Conceito básicos e terminologia

Operações sobre palavras


Operações sobre linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente limitados
Autómatos de pilha

Enquadramento (2)

- Se tivesse que definir **linguagem** como é que o faria?
 - Permite **expressar, transmitir e receber ideias**.
 - **Comunicação** entre pessoas ou seres vivos em geral.
 - Inclui a comunicação com e entre máquinas.
 - Requer várias **entidade comunicantes**, um código e regras para que a comunicação seja inteligível.
 - Requer o uso de (pelo menos) um meio ou **canal de comunicação** (som, texto, ...).
- Necessário: codificação e um conjunto de regras comuns, e interlocutores que as conheçam.
- Vejamos, como exemplo, algumas linguagens naturais.
- Palavras diferentes, em linguagens diferentes, podem ter o mesmo significado:
 - “adeus”, “goodbye”, “au revoir”,

- Por outro lado, também existem palavras iguais com significados diferentes (dependendo do contexto):
 - *morro, rio, caminho, ...*

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados
Autómatos de pilha

Enquadramento (3)

- Diferentes linguagens podem utilizar símbolos (letras ou caracteres) diferentes, ou partilhar muitos deles.
- Compreensão de uma palavra é feita letra a letra, mas isso não acontece com um texto.
- Assim, podemos ver uma linguagem natural como o português como sendo composta por mais do que uma linguagem:
 - Uma que explicita as regras para construir palavras a partir do alfabeto das letras:

$a + d + e + u + s \rightarrow \text{adeus}$

- E outra que contém as regras gramaticais para construir frases a partir das palavras resultantes da linguagem anterior:

$\text{adeus} + e + \text{até} + \text{amanhã} \rightarrow \text{adeus e até amanhã}$

Neste caso o alfabeto deixa de ser o conjunto de letras e passa a ser o conjunto de palavras existentes.

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Enquadramento (4)

- Inerente às linguagens, é a necessidade de decidir se uma sequência de símbolos do alfabeto é válida.
 - **correcto:**
 $a + d + e + u + s \rightarrow \text{adeus}$
 $\text{adeus} + e + \text{até} + \text{amanhã} \rightarrow \text{adeus e até amanhã}$
 - **incorrecto:**
 $e + d + u + a + s \rightarrow \text{edues}$
 $\text{até} + \text{adeus} + \text{amanhã} + e \rightarrow \text{até adeus amanhã e}$
- Só sequências válidas é que permitem uma comunicação correcta.
- Por outro lado, essa comunicação tem muitas vezes um efeito.
- Seja esse efeito uma resposta à mensagem inicial, ou o despoletar de uma qualquer acção.

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Linguagens de programação

- As linguagens de comunicação com computadores – designadas por linguagens de programação – partilham todas estas características.
- Diferem, no facto de não poderem ter nenhuma **ambiguidade**, e de as acções despoletadas serem muitas vezes a mudança do estado do sistema computacional, podendo este estar ligado a entidades externas como sejam outros computadores, pessoas, sistemas robóticos, máquinas de lavar, etc..
- Vamos ver que podemos definir linguagens de programação por estruturas formais matematicamente bem comportadas.
- Para além disso, veremos também que essas definições nos ajudam a implementar acções interessantes.

O desenvolvimento das linguagens de programação está umbilicalmente ligado com as tecnologias de compilação!

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Compiladores: Introdução

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

**Estrutura de um
Compilador**

Análise Lexical
Análise Sintáctica
Análise Semântica
Síntese

**Implementação de um
Compilador**

Análise léxica
Análise sintáctica
Análise semântica
Síntese: interpretação do
código

**Linguagens: Definição
como Conjunto**

Conceito básicos e
terminologia
Operações sobre palavras
Operações sobre
linguagens

**Introdução às
gramáticas**

Hierarquia de Chomsky
Autómatos
Máquina de Turing
Autómatos linearmente
limitados
Autómatos de pilha

Compiladores

Compreensão, interpretação e/ou tradução automática de linguagens.

Compiladores (Processadores de Linguagens)

- Os **compiladores** são programas que permitem:
 - decidir sobre a correcção de sequências de símbolos do respectivo alfabeto;
 - despoletar acções resultantes dessas decisões.
- Frequentemente, os compiladores “limitam-se” a fazer a tradução entre linguagens.



- É o caso dos compiladores das linguagens de programação de alto nível (Java, C++, Eiffel, etc.), que traduzem o código fonte dessas linguagens em código de linguagens mais próximas do *hardware* do sistema computacional (e.g. *assembly* ou *Java bytecode*).
- Nestes casos, na inexistência de erros, é gerado um programa composto por código executável directa ou indirectamente pelo sistema computacional:



Exemplo: Java *bytecode*

```
public class Hello
{
    public static void main(String [] args)
    {
        System.out.println ("Hello!");
    }
}
```

```
javac Hello.java
```

```
javap -c Hello.class
```

Compiled from "Hello.java"

```
public class Hello {
    public Hello();
        Code:
        0: aload_0
        1: invokespecial #1 // Method java/lang/Object."<init>":()V
        4: return

    public static void main(java.lang.String []);
        Code:
        0: getstatic     #2 // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #3 // String Hello!
        5: invokevirtual #4 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        8: return
}
```

Exemplo 2: Calculadora

- Código fonte:

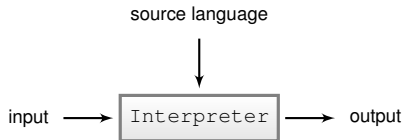
```
1+2*3:4
```

- Uma possível compilação para Java:

```
public class CodeGen {  
    public static void main( String [] args ) {  
        double v2 = 1;  
        double v5 = 2;  
        double v6 = 3;  
        double v4 = v5 * v6;  
        double v7 = 4;  
        double v3 = v4 / v7;  
        double v1 = v2 + v3;  
        System.out.println( v1 );  
    }  
}
```

Compiladores: Processadores de Linguagens (2)

- Uma variante possível consiste num **interpretador**:



- Neste caso a execução é feita durante a interpretação do código fonte, instrução a instrução.
- `Python` e `bash` são exemplos de linguagens interpretadas.
- Existem também aproximações híbridas em que existe compilação de código para uma linguagem intermédia, que depois é interpretada na execução.
- A linguagem `Java` utiliza uma estratégia deste género em que o código fonte é compilado para *Java bytecode*, que depois é interpretado pela máquina virtual `Java`.
- Em geral os compiladores processam código fonte em formato de *texto*, havendo uma grande variedade no formato do código gerado (texto, binário, interpretado, ...).

Exemplo: Calculadora

- Código fonte:



1+2*3:4

- Uma possível interpretação:

2.5

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

**Estrutura de um
Compilador**

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

**Implementação de um
Compilador**

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

**Linguagens: Definição
como Conjunto**

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

**Introdução às
gramáticas**

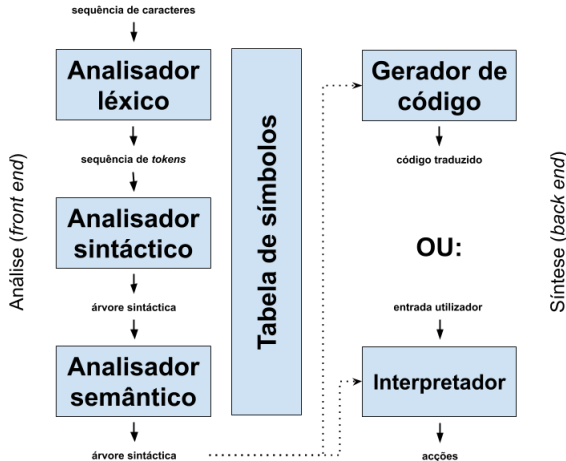
Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Estrutura de um Compilador

Estrutura de um Compilador



Compiladores,
Linguagens e
Gramáticas

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical
Análise Sintáctica
Análise Semântica
Síntese

Implementação de um
Compilador

Análise léxica
Análise sintáctica
Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia
Operações sobre palavras
Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados
Autómatos de pilha

Estrutura de um Compilador (2)

- Uma característica interessante da compilação de linguagens de alto nível, é o facto de, tal como no caso das linguagens naturais, essa compilação envolver mais do que uma linguagem:
 - **análise léxica**: composição de letras e outros caracteres em palavras (*tokens*);
 - **análise sintáctica**: composição de *tokens* numa estrutura sintáctica adequada.
 - **análise semântica**: verificação – tanto quanto possível ou desejável – se a estrutura sintáctica tem significado.
- As acções consistem na geração do programa na linguagem destino e podem envolver também diferentes fases de geração de código e optimização.

- Conversão da sequência de caracteres de entrada numa sequência de elementos lexicais.
- Esta estratégia simplifica fortemente a gramática da análise sintáctica, e permite uma implementação muito eficiente do analisador léxico (mais tarde veremos em detalhe como e porquê).
- Cada elemento lexical pode ser definido por um tuplo¹ com uma identificação do elemento e o seu valor (o valor pode ser omitido quando não se aplica):

```
<token_name , attribute_value >
```

- Exemplo 1:

```
pos = pos + vel * 5;
```

pode ser convertido pelo analisador léxico (*scanner*) em:

```
<id , pos> <=> <id , pos> <+> <id , vel> <*> <int , 5> <;>
```

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

¹ Sequência ordenada de valores, não necessariamente do mesmo tipo.

Análise Lexical (2)

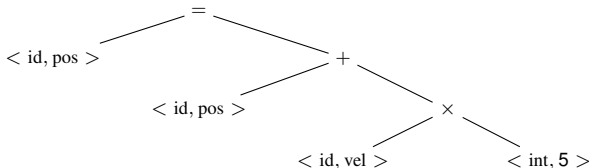
- Em geral os espaços em branco, e as mudanças de linha e os comentários não são relevantes nas linguagens de programação, pelo que podem ser eliminados pelo analisador lexical.
- Exemplo 2: esboço de linguagem de processamento geométrico:

```
distance ( 0 , 0 ) ( 4 , 3 )
```

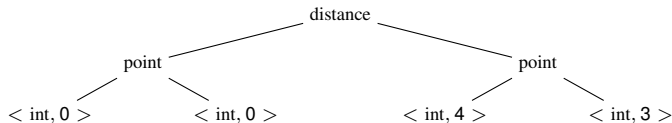
pode ser convertido pelo analisador léxico (*scanner*) em:

```
<distance> <( > <num,0> <,> <num,0> <)>  
<( > <num,4> <,> <num,3> <)>
```

- Após a análise lexical segue-se a *análise sintáctica* (*parsing*), onde se verifica a conformidade da sequência de elementos lexicais com a estrutura sintáctica da linguagem.
- Nas linguagens que se pretende sintacticamente processar, podemos sempre fazer uma aproximação à sua estrutura formal através duma representação tipo *árvore*.
- Para esse fim é necessário uma *gramática* que especifique a estrutura desejada (voltaremos a este problema mais à frente).
- No exemplo 1 (`pos = pos + vel * 5;`):



- No exemplo 2 (`distance (0 , 0) (4 , 3)`):



- Chama-se a atenção para duas características das árvores sintáticas:
 - não incluem alguns elementos lexicais (que apenas são relevantes para a estrutura formal);
 - definem sem ambiguidade a ordem das operações (havemos de voltar a este problema).

- A parte final do *front end* do compilador é a *análise semântica*.
- Nesta fase são verificadas, tanto quando possível, restrições que não é possível (ou tão só desejável) que sejam feitas nas duas fases anteriores.
- Por exemplo: verificar se um identificador foi declarado, verificar a conformidade no sistema de tipos da linguagem, etc.
- Note-se que apenas restrições com verificação estática (i.e. em tempo de compilação), podem ser objecto de análise semântica pelo compilador.
- Se no exemplo 2 existisse a instrução de um círculo do qual fizesse parte a definição do seu raio, não seria em geral possível, durante a análise semântica, garantir um valor não negativo para esse raio (essa semântica apenas poderia ser verificada dinamicamente, i.e., em tempo de execução).

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados
Autómatos de pilha

Análise Semântica (2)

- Utiliza a árvore sintáctica da análise sintáctica assim como uma estrutura de dados designada por tabela de símbolos (assente em arrays associativos).
- Esta última fase de análise deve garantir o sucesso das fases subsequentes (geração e eventual optimização de código, ou interpretação).

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

- Havendo garantia de que o código da linguagem fonte é válido, então podemos passar aos efeitos pretendidos com esse código.
- Os efeitos podem ser:
 - 1 simplesmente a indicação de validade do código fonte²;
 - 2 a tradução do código fonte numa linguagem destino;
 - 3 ou a interpretação e execução imediata.
- Em todos os casos, pode haver interesse na identificação e localização precisa de eventuais erros.
- Como a maioria do código fonte assenta em texto, é usual indicar não só a instrução mas também a linha onde cada erro ocorre.

²Útil, por exemplo, para *syntax highlighting*.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Geração de código: exemplo

- No processo de compilação, pode haver o interesse em gerar uma representação intermédia do código que facilite a geração final de código.
- Uma forma possível para essa representação intermédia é o chamado *código de triplo endereço*.
- Para o exemplo 1 (`pos = pos + vel * 5;`) poderíamos ter:

```
t1 = inttofloat(5)
t2 = id(vel) * t1
t3 = id(pos) + t2
id(pos) = t3
```

- Este código poderia depois ser otimizado na fase seguinte da compilação:

```
t1 = id(vel) * 5.0
id(pos) = id(pos) + t1
```

- E por fim, poder-se-ia gerar *assembly* (pseudo-código):

```
LOAD R2, id(vel) // load value from memory to register R2
MULT R2, R2, #5.0 // mult. 5 with R2 and store result in R2
LOAD R1, id(pos) // load value from memory to register R1
ADD R1, R1, R2 // add R1 with R2 and store result in R1
STORE id(pos), R1 // store value to memory from register R1
```

Linguagens: Definição como Conjunto

**Compiladores,
Linguagens e
Gramáticas**

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Linguagens: Definição como Conjunto

- As linguagens servem para **comunicar** recorrendo a mensagens.
- Uma mensagem pode ser vista como uma sequência de **símbolos**.
- No entanto, uma linguagem não aceita todo o tipo de símbolos e de sequências.
- Uma linguagem é caracterizada por um conjunto de símbolos e uma forma de descrever sequências válidas desses símbolos (i.e. o conjunto de sequências válidas).
- Se as linguagens naturais admitem alguma subjectividade e ambiguidade, as linguagens de programação requerem total objectividade.

Problema

Como definir linguagens de forma sintética e objectiva?

- Definir por **extensão** é uma possibilidade.
- No entanto, para linguagens minimamente interessantes não só teríamos uma descrição gigantesca como também, provavelmente, incompleta.
- As linguagens de programação tendem a aceitar variantes infinitas de entradas.
- Alternativamente podemos descrevê-la por **compreensão**.
- Uma possibilidade é utilizar os formalismos ligados à definição de **conjuntos**.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Conceito básicos

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

**Estrutura de um
Compilador**

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

**Implementação de um
Compilador**

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

**Linguagens: Definição
como Conjunto**

**Conceito básicos e
terminologia**

Operações sobre palavras

Operações sobre
linguagens

**Introdução às
gramáticas**

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Conceito básicos e terminologia

- Um conjunto pode ser definido por **extensão** (ou enumeração) ou por **compreensão**.
- Um exemplo de um conjunto definido por extensão é o conjunto dos algarismos binários $\{0, 1\}$.
- Na definição por compreensão utiliza-se a seguinte notação:

$$\{x \mid p(x)\}$$

ou

$$\{x : p(x)\}$$

- x é a variável que representa um qualquer elemento do conjunto, e $p(x)$ um predicado sobre essa variável.
- Assim, este conjunto é definido contendo todos os valores de x em que o predicado $p(x)$ é verdadeiro.
- Por exemplo:
$$\{n \mid n \in \mathbb{N} \wedge n \leq 9\} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Conceito básicos e terminologia (2)

- Um **símbolo** (ou **letra**) é a unidade atômica (indivisível) das linguagens.
- Em linguagens assentes em texto, um símbolo será um carácter.
- Um **alfabeto** é um conjunto finito não vazio de símbolos.
- Por exemplo:
 - $A = \{0, 1\}$ é o alfabeto dos algarismos binários.
 - $A = \{0, 1, \dots, 9\}$ é o alfabeto dos algarismos decimais.
- Uma **palavra** (*string* ou cadeia) é uma sequência de símbolos sobre um dado alfabeto A .

$$U = a_1 a_2 \cdots a_n, \quad \text{com} \quad a_i \in A \wedge n \geq 0$$

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Conceito básicos e terminologia (3)

- Por exemplo:
 - $A = \{0, 1\}$ é o alfabeto dos algarismos binários.
01101, 11, 0
 - $A = \{0, 1, \dots, 9\}$ é o alfabeto dos algarismos decimais.
2016, 234523, 999999999999999, 0
 - $A = \{0, 1, \dots, 0, a, b, \dots, z, @, \dots\}$
mos@ua.pt, Bom dia!

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Conceito básicos e terminologia (4)

- A **palavra vazia** é uma sequência de zero símbolos e denota-se por ε (épsilon).
- Note que ε não pertence ao alfabeto.
- Uma **sub-palavra** de uma palavra u é uma sequência contígua de 0 ou mais símbolos de u .
- Um **prefixo** de uma palavra u é uma sequência contígua de 0 ou mais símbolos iniciais de u .
- Um **sufixo** de uma palavra u é uma sequência contígua de 0 ou mais símbolos terminais de u .
- Por exemplo:
 - as é uma sub-palavra de casa, mas não prefixo nem sufixo
 - 001 é prefixo e sub-palavra de 00100111 mas não é sufixo
 - ε é prefixo, sufixo e sub-palavra de qualquer palavra u
 - qualquer palavra u é prefixo, sufixo e sub-palavra de si própria

Conceito básicos e terminologia (5)

- O **fecho** (ou conjunto de cadeias) do alfabeto A denominado por A^* , representa o conjunto de todas as palavras definíveis sobre o alfabeto A , incluindo a palavra vazia.
- Por exemplo:
 - $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
 - $\{\clubsuit, \diamond, \heartsuit, \spadesuit\}^* = \{\epsilon, \clubsuit, \diamond, \heartsuit, \spadesuit, \clubsuit\diamond, \dots\}$
- Dado um alfabeto A , uma **linguagem** L sobre A é um conjunto finito ou infinito de palavras consideradas válidas definidas com símbolos de A .
Isto é: $L \subseteq A^*$

Conceito básicos e terminologia (6)

- Exemplo de linguagens sobre o alfabeto $A = \{0, 1\}$
 - $L_1 = \{u \mid u \in A^* \wedge |u| \leq 2\} = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$
 - $L_2 = \{u \mid u \in A^* \wedge \forall_i u_i = 0\} = \{\varepsilon, 0, 00, 000, 0000, \dots\}$
 - $L_3 = \{u \mid u \in A^* \wedge u.\text{count}(1) \bmod 2 = 0\} = \{000, 11, 000110101, \dots\}$
 - $L_4 = \{\} = \emptyset$ (conjunto vazio)
 - $L_5 = \{\varepsilon\}$
 - $L_6 = A$
 - $L_7 = A^*$
- Note que $\{\}$, $\{\varepsilon\}$, A e A^* são linguagens sobre o alfabeto A qualquer que seja A
- Uma vez que as linguagens são conjuntos, todas as operações matemáticas sobre conjuntos são aplicáveis: reunião, intercepção, complemento, diferença, etc.



Operações sobre palavras

**Compiladores,
Linguagens e
Gramáticas**

Enquadramento

Linguagens de
programação

**Compiladores:
Introdução**

**Estrutura de um
Compilador**

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

**Implementação de um
Compilador**

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

**Linguagens: Definição
como Conjunto**

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

**Introdução às
gramáticas**

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Operações sobre palavras

- O **comprimento** de uma palavra u denota-se por $|u|$ e representa o seu número de símbolos.
- O comprimento da palavra vazia é zero

$$|\varepsilon| = 0$$

- É habitual interpretar-se a palavra u como uma função de acesso aos seus símbolos (tipo *array*):

$$u : \{1, 2, \dots, n\} \rightarrow A, \quad \text{com } n = |u|$$

em que u_i representa o *i*ésimo símbolo de u

- O **reverso** de uma palavra u é a palavra, denota-se por u^R , e é obtida invertendo a ordem dos símbolos de u

$$u = \{u_1, u_2, \dots, u_n\} \implies u^R = \{u_n, \dots, u_2, u_1\}$$

Operações sobre palavras (2)

- A **concatenação** (ou **produto**) das palavras u e v denota-se por $u.v$, ou simplesmente uv , e representa a justaposição de u e v , i.e., a palavra constituída pelos símbolos de u seguidos pelos símbolos de v .
- Propriedades da concatenação:
 - $|u.v| = |u| + |v|$
 - $u.(v.w) = (u.v).w = u.v.w$ (associatividade)
 - $u.\varepsilon = \varepsilon.u = u$ (elemento neutro)
 - $u \neq \varepsilon \wedge v \neq \varepsilon \wedge u \neq v \implies u.v \neq v.u$ (não comutativo)
- A **potência** de ordem n , com $n \geq 0$, de uma palavra u denota-se por u^n e representa a concatenação de n réplicas de u , ou seja, $\underbrace{uu \cdots u}_{n \times}$.
- $u^0 = \varepsilon$

Operações sobre linguagens

**Compiladores,
Linguagens e
Gramáticas**

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

**Operações sobre
linguagens**

Introdução às
gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Operações sobre linguagens: reunião

- A **reunião** de duas linguagens L_1 e L_2 denota-se por $L_1 \cup L_2$ e é dada por:

$$L_1 \cup L_2 = \{u \mid u \in L_1 \vee u \in L_2\}$$

- Por exemplo, se definirmos as linguagens L_1 e L_2 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{a w \mid w \in A^*\}$$

$$L_2 = \{u \mid u \text{ termina com } a\} = \{w a \mid w \in A^*\}$$

- qual será o resultado da reunião destas linguagens?

$$L = L_1 \cup L_2 = ?$$

- Resposta:

$$L = \{w_1 a w_2 \mid w_1, w_2 \in A^* \wedge (w_1 = \varepsilon \vee w_2 = \varepsilon)\}$$

Operações sobre linguagens: intercepção

- A **intercepção** de duas linguagens L_1 e L_2 denota-se por $L_1 \cap L_2$ e é dada por:

$$L_1 \cap L_2 = \{u \mid u \in L_1 \wedge u \in L_2\}$$

- Por exemplo, se definirmos as linguagens L_1 e L_2 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{a w \mid w \in A^*\}$$

$$L_2 = \{u \mid u \text{ termina com } a\} = \{w a \mid w \in A^*\}$$

- qual será o resultado da intercepção destas linguagens?

$$L = L_1 \cap L_2 = ?$$

- Resposta:

$$L = \{a w a \mid w \in A^*\} \cup \{a\}$$

Operações sobre linguagens: diferença

- A **diferença** de duas linguagens L_1 e L_2 denota-se por $L_1 - L_2$ e é dada por:

$$L_1 - L_2 = \{u \mid u \in L_1 \wedge u \notin L_2\}$$

- Por exemplo, se definirmos as linguagens L_1 e L_2 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$$

$$L_2 = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\}$$

- qual será o resultado da diferença destas linguagens?

$$L = L_1 - L_2 = ?$$

- Resposta:

$$L = \{awx \mid w \in A^* \wedge x \in A \wedge x \neq a\}$$

- ou:

$$L = \{awb \mid w \in A^*\}$$

Operações sobre linguagens: complementação

- A **complementação** da linguagem L denota-se por \bar{L} e é dada por:

$$\bar{L} = A^* - L = \{u \mid u \notin L\}$$

- Por exemplo, se definirmos a linguagem L_1 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{a w \mid w \in A^*\}$$

- qual será o resultado da complementação desta linguagem?

$$L = \bar{L_1} = ?$$

- Resposta:

$$L = \{x w \mid w \in A^* \wedge x \in A \wedge x \neq a\} \cup \{\varepsilon\}$$

- ou:

$$L = \{b w \mid w \in A^*\} \cup \{\varepsilon\}$$

Operações sobre linguagens: concatenação

- A **concatenação** de duas linguagens L_1 e L_2 denota-se por $L_1.L_2$ e é dada por:

$$L_1.L_2 = \{uv \mid u \in L_1 \wedge v \in L_2\}$$

- Por exemplo, se definirmos as linguagens L_1 e L_2 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{aw \mid w \in A^*\}$$

$$L_2 = \{u \mid u \text{ termina com } a\} = \{wa \mid w \in A^*\}$$

- qual será o resultado da concatenação destas linguagens?

$$L = L_1.L_2 = ?$$

- Resposta:

$$L = \{aw a \mid w \in A^*\}$$

Operações sobre linguagens: potenciação

- A **potência** de ordem n da linguagem L denota-se por L^n e é definida indutivamente por:

$$\begin{aligned}L^0 &= \{\varepsilon\} \\ L^{n+1} &= L^n \cdot L\end{aligned}$$

- Por exemplo, se definirmos a linguagem L_1 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{a w \mid w \in A^*\}$$

- qual será o resultado da potência de ordem 2 desta linguagem?

$$L = L_1^2 = ?$$

- Resposta:

$$L = \{a w_1 a w_2 \mid w_1, w_2 \in A^*\}$$

Operações sobre linguagens: fecho de Kleene

- O **fecho de Kleene** da linguagem L denota-se por L^* e é dado por:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i=0}^{\infty} L^i$$

- Por exemplo, se definirmos a linguagem L_1 sobre o alfabeto $A = \{a, b\}$:

$$L_1 = \{u \mid u \text{ começa por } a\} = \{a w \mid w \in A^*\}$$

- qual será o fecho de Kleene desta linguagem?

$$L = L_1^* = ?$$

- Resposta:

$$L = L_1 \cup \{\varepsilon\}$$

- Note que para $n > 1$ $L_1^n \subset L_1$

Operações sobre linguagens: notas adicionais

- Note que nas operações binárias sobre conjuntos não é requerido que as duas linguagens estejam definidos sobre o mesmo alfabeto.
- Assim se tivermos duas linguagens L_1 e L_2 definidas respectivamente sobre os alfabetos A_1 e A_2 , então o alfabeto resultante da aplicação duma qualquer operação binária sobre as linguagens é: $A_1 \cup A_2$

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical
Análise Sintáctica
Análise Semântica
Síntese

Implementação de um Compilador

Análise léxica
Análise sintáctica
Análise semântica
Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia
Operações sobre palavras

Operações sobre linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos
Máquina de Turing
Autómatos linearmente
limitados
Autómatos de pilha

Introdução às gramáticas

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

- A utilização de conjuntos para definir linguagens não é frequentemente a forma mais adequada e versátil para as descrever.
- Muitas vezes é preferível identificar estruturas intermédias, que abstraem partes ou subconjuntos importantes, da linguagem.
- Tal como em programação, muitas vezes descrições recursivas são bem mais simples, sem perda da objectividade e do rigor necessários.
- É nesse caminho que encontramos as **gramáticas**.
- As **gramáticas** descrevem linguagens por compreensão recorrendo a representações **formais** e (muitas vezes) **recursivas**.
- Vendo as linguagens como sequências de símbolos (ou palavras), as gramáticas definem formalmente as sequências **válidas**.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Gramáticas (2)

- Por exemplo, em português a frase “O cão ladra” pode ser gramaticalmente descrita por:

frase	→	sujeito predicado
sujeito	→	artigo substantivo
predicado	→	verbo
artigo	→	O Um
substantivo	→	cão lobo
verbo	→	ladra uiva

- Esta gramática descreve 8 possíveis frases e contém mais informação do que a frase original.
- Contém 6 **símbolos terminais** e 6 **símbolos não terminais**.
- Um símbolo não terminal é definido por uma **produção** descrevendo possíveis representações desse símbolo, em função de símbolos terminais e/ou não terminais.

- Formalmente, uma gramática é um quádruplo $G = (T, N, S, P)$, onde:
 - T é um conjunto finito não vazio designado por alfabeto terminal, onde cada elemento é designado por símbolo **terminal**;
 - N é um conjunto finito não vazio, disjunto de T ($N \cap T = \emptyset$), cujos elementos são designados por símbolos **não terminais**;
 - $S \in N$ é um símbolo não terminal específico designado por **símbolo inicial**;
 - P é um conjunto finito de **regras** (ou produções) da forma $\alpha \rightarrow \beta$ onde $\alpha \in (T \cup N)^* N (T \cup N)^*$ e $\beta \in (T \cup N)^*$, isto é, α é uma cadeia de símbolos terminais e não terminais contendo, pelo menos, um símbolo não terminal; e β é uma cadeia de símbolos, eventualmente vazia, terminais e não terminais.

Gramáticas: exemplos

- Formalmente, a gramática anterior será:

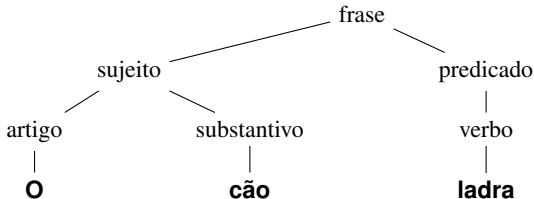
$$G = (\{\mathbf{O, Um, cão, lobo, ladra, uiva}\}, \\ \{\text{frase, sujeito, predicado, artigo, substantivo, verbo}\}, \\ \text{frase}, P)$$

- P é constituído pelas regras já apresentadas:

frase \rightarrow sujeito predicado
sujeito \rightarrow artigo substantivo
predicado \rightarrow verbo
artigo \rightarrow **O | Um**
substantivo \rightarrow **cão | lobo**
verbo \rightarrow **ladra | uiva**

Gramáticas: exemplos (2)

- Podemos descrever a frase “O cão ladra” com a seguinte árvore (denominada sintáctica).



Gramáticas: exemplos (3)

- Considere a seguinte gramática
 $G = (\{0, 1\}, \{S, A\}, S, P)$, onde P é constituído pelas regras:

$$S \rightarrow 0S$$

$$S \rightarrow 0A$$

$$A \rightarrow 0A1$$

$$A \rightarrow \varepsilon$$

- Qual será a linguagem definida por esta gramática?

$$L = \{0^n 1^m : n \in \mathbb{N} \wedge m \in \mathbb{N}_0 \wedge n > m\}$$

Gramáticas: exemplos (4)

- Sendo $A = \{a, b\}$, defina uma gramática para a seguinte linguagem:

$$L_1 = \{a w \mid w \in A^*\}$$

- A gramática $G = (\{a, b\}, \{S, X\}, S, P)$, onde P é constituído pelas regras:

$$S \rightarrow aX$$

$$X \rightarrow aX$$

$$X \rightarrow bX$$

$$X \rightarrow \varepsilon$$

ou:

$$S \rightarrow aX$$

$$X \rightarrow aX \mid bX \mid \varepsilon$$

Gramáticas: exemplos (5)

- Sendo $A = \{0, 1\}$, defina uma gramática para a seguinte linguagem:

$$L_3 = \{u \mid u \in A^* \wedge u.\text{count}(1) \bmod 2 = 0\}$$

- A gramática $G = (\{0, 1\}, \{S, A\}, S, P)$, onde P é constituído pelas regras:

$$S \rightarrow S1S1S \mid A$$

$$A \rightarrow 0A \mid \varepsilon$$

Hierarquia de Chomsky

**Compiladores,
Linguagens e
Gramáticas**

Enquadramento

Linguagens de
programação

Compiladores:
Introdução

Estrutura de um
Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um
Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição
como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às
gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Autómatos finitos

- Restrições sobre α e β permitem definir uma taxonomia das linguagens – hierarquia de Chomsky:
 - 1 Se não houver nenhuma restrição, G é designada por gramática do **tipo-0**.
 - 2 G será do **tipo-1**, ou gramática **dependente do contexto**, se cada regra $\alpha \rightarrow \beta$ de P obedece a $|\alpha| \leq |\beta|$ (com a exceção de também poder existir a produção vazia: $S \rightarrow \varepsilon$).
 - 3 G será do **tipo-2**, ou gramática **independente, ou livre, do contexto**, se cada regra $\alpha \rightarrow \beta$ de P obedece a $|\alpha| = 1$, isto é: α é constituído por um só não terminal.
 - 4 G será do **tipo-3**, ou gramática **regular**, se cada regra tiver uma das formas: $A \rightarrow cB$, $A \rightarrow c$ ou $A \rightarrow \varepsilon$, onde A e B são símbolos não terminais (A pode ser igual a B) e c um símbolo terminal. Isto é, em todas as produções, o β só pode ter no máximo um símbolo não terminal sempre à direita (ou, alternativamente, sempre à esquerda).

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical
Análise Sintáctica
Análise Semântica
Síntese

Implementação de um Compilador

Análise léxica
Análise sintáctica
Análise semântica
Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia
Operações sobre palavras
Operações sobre
linguagens

Introdução às gramáticas

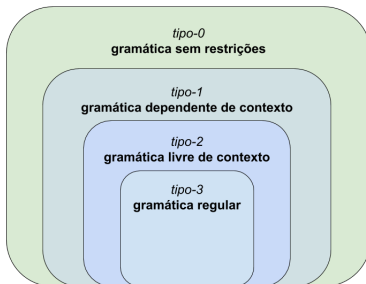
Hierarquia de Chomsky

Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha

Hierarquia de Chomsky (2)



- Para cada um desses tipos podem ser definidos diferentes tipos de máquinas (algoritmos, autómatos) que as podem reconhecer.
- Quanto mais simples for a gramática, mas simples e eficiente é a máquina que reconhece essas linguagens.

Enquadramento

Linguagens de programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do código

Linguagens: Definição como Conjunto

Conceito básicos e terminologia

Operações sobre palavras

Operações sobre linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente limitados

Autómatos de pilha

Hierarquia de Chomsky (3)

- Cada classe de linguagens do **tipo- i** contém a classe de linguagens **tipo- $(i+1)$** ($i = 0, 1, 2$)
- Esta hierarquia não traduz apenas as características formais das linguagens, mas também expressam os requisitos de computação necessários:
 - 1 As **máquinas de Turing** processam gramáticas sem restrições (tipo-0);
 - 2 Os **autômatos linearmente limitados** processam gramáticas dependentes do contexto (tipo-1);
 - 3 Os **autômatos de pilha** processam gramáticas independentes do contexto (tipo-2);
 - 4 Os **autômatos finitos** processam gramáticas regulares (tipo-3).

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autômatos

Máquina de Turing

Autômatos linearmente
limitados

Autômatos de pilha

Autômatos finitos

Autómatos

Compiladores, Linguagens e Gramáticas

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky

Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Autómatos finitos

Máquina de Turing

- (Alan Turing, 1936)
- Modelo abstracto de computação.
- Permite (em teoria) implementar qualquer programa computável.
- Assenta numa máquina de estados finita, numa "cabeça" de leitura/escrita de símbolos e numa fita infinita (onde se escreve ou lê esses símbolos).
- A "cabeça" de leitura/escrita pode movimentar-se uma posição para esquerda ou direita.
- Modelo muito importante na teoria da computação.
- Pouco relevante na implementação prática de processadores de linguagens.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

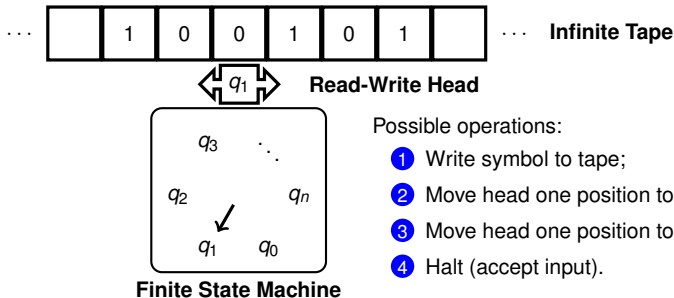
Hierarquia de Chomsky
Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Máquina de Turing (2)



Possible operations:

- 1 Write symbol to tape;
- 2 Move head one position to the right;
- 3 Move head one position to the left;
- 4 Halt (accept input).

- A máquina de estados finita (FSM) tem acesso ao símbolo actual e decide a próxima acção a ser realizada.
- A acção consiste na transição de estado e qual a operação sobre a fita.
- Se não for possível nenhuma acção, a entrada é rejeitada.

Máquina de Turing: exemplo

- Dado o alfabeto $A = \{0, 1\}$, e considerando que um número inteiro não negativo n é representado pela sequência de $n + 1$ símbolos 1, vamos implementar uma MT que some os próximos (i.e à direita da posição actual) dois números inteiros existentes na fita (separados apenas por um 0).
- O algoritmo pode ser simplesmente trocar o símbolo 0 entre os dois números por 1, e trocar os dois últimos símbolos 1 por 0.
- Por exemplo: $3 + 2$ a que corresponde o seguinte estado na fita (símbolo a negrito é a posição da "cabeça"):
... **0**111101110... (o resultado pretendido será:
... **0**111111000...).
- Considerando que os estados são designados por $E_i, i \geq 1$ (sendo E_1 o estado inicial); e as operações:
 - d mover uma posição para a direita;
 - e mover uma posição para a esquerda;
 - 0 escrever o símbolo 0 na fita;
 - 1 escrever o símbolo 1 na fita;
 - h aceitar e terminar autómato.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Máquina de Turing: exemplo (2)

- Uma solução possível é dada pela seguinte diagrama de transição de estados:

Estado	Entrada	
	0	1
E_1	E_1/d	E_2/d
E_2	$E_3/1$	E_2/d
E_3	E_4/e	E_3/d
E_4	—	$E_5/0$
E_5	E_5/e	$E_6/0$
E_6	E_7/e	—
E_7	E_1/h	E_7/e

- $E_1 \dots 0111101110 \dots \rightarrow E_1 \dots 0111101110 \dots \xrightarrow{*} E_2 \dots 0111101110 \dots \rightarrow$
 $E_3 \dots 0111111110 \dots \rightarrow E_3 \dots 0111111110 \dots \xrightarrow{*} E_3 \dots 0111111110 \dots \rightarrow$
 $E_4 \dots 0111111110 \dots \rightarrow E_5 \dots 0111111100 \dots \rightarrow E_5 \dots 0111111100 \dots \rightarrow$
 $E_6 \dots 0111111000 \dots \rightarrow E_7 \dots 0111111000 \dots \xrightarrow{*} E_7 \dots 0111111000 \dots$

Autômatos linearmente limitados

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

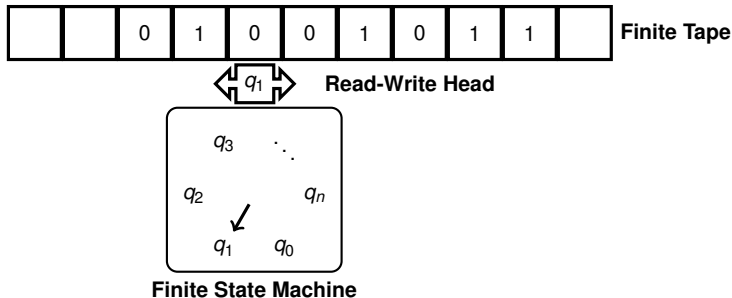
Hierarquia de Chomsky

Autômatos

Máquina de Turing

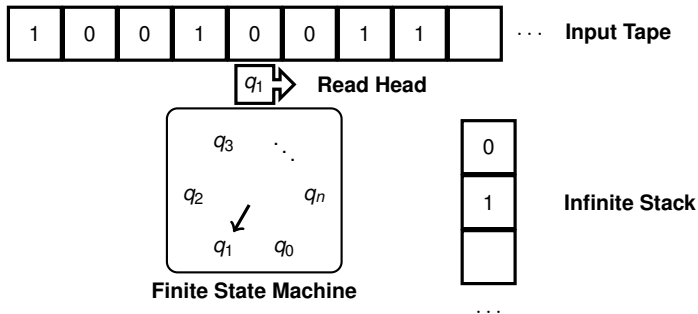
Autômatos linearmente
limitados

Autômatos finitos



- Diferem das MT pela finitude da fita.

Autómatos de pilha



- "Cabeça" apenas de leitura e suporte de uma pilha sem limites.
- Movimento da "cabeça" apenas numa direcção.
- Autómatos adequados para análise sintáctica.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky

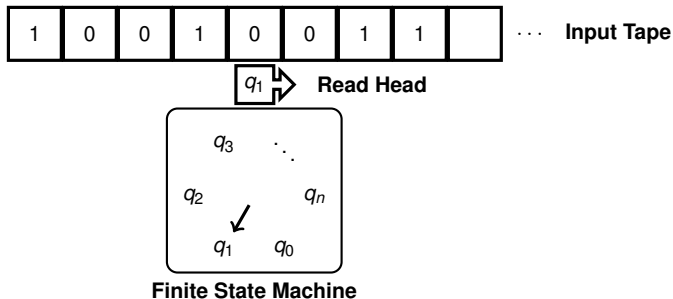
Autómatos

Máquina de Turing

Autómatos linearmente
limitados

Autómatos de pilha

Autómatos finitos



- Sem escrita de apoio à máquina de estados.
- Autómatos adequados para análise léxica.

Enquadramento

Linguagens de
programação

Compiladores: Introdução

Estrutura de um Compilador

Análise Lexical

Análise Sintáctica

Análise Semântica

Síntese

Implementação de um Compilador

Análise léxica

Análise sintáctica

Análise semântica

Síntese: interpretação do
código

Linguagens: Definição como Conjunto

Conceito básicos e
terminologia

Operações sobre palavras

Operações sobre
linguagens

Introdução às gramáticas

Hierarquia de Chomsky
Autómatos

Máquina de Turing
Autómatos linearmente
limitados

Autómatos de pilha