Teoria da Computação

Introdução e Conceitos Básicos

Aula 01

Prof. Felipe A. Louza

UFU

Roteiro

Roteiro

Introdução

A Ciência da Computação corresponde à todo o conhecimento relacionado à computação (não apenas computadores).

- O seu início pode ser rastreado pelo desenvolvimento de algoritmos como os de *Euclides e outros*.

Figura: Euclides ($\approx$ 300 a.C.)

Máximo divisor comum: `pt.wikipedia.org/wiki/Algoritmo_de_Euclides`

Introdução

O interesse *"moderno"* na Ciência da Computação é baseado em dois importantes eventos:

1. A invenção do *computador digital* capaz de realizer bilhões de operações por segundos;
2. A formalização do conceito de *procedimento efetivo* (o que pode e o que não pode ser computado);

Introdução

O interesse *“moderno”* na Ciência da Computação é baseado em dois importantes eventos:

1. A invenção do *computador digital* capaz de realizer bilhões de operações por segundos;
2. A formalização do conceito de *procedimento efetivo* (o que pode e o que não pode ser computado);

Introdução

Podemos dividir a Ciência da Computação em duas principais áreas:

1. Modelos e fundamentos implícitos sobre computação;
2. Técnicas de engenharia para o desenvolvimento de sistemas de computação (hardware e software)

Nesse curso, veremos uma introdução à primeira área.

Introdução

Podemos dividir a Ciência da Computação em duas principais áreas:

1. Modelos e fundamentos implícitos sobre computação;
2. Técnicas de engenharia para o desenvolvimento de sistemas de computação (hardware e software)

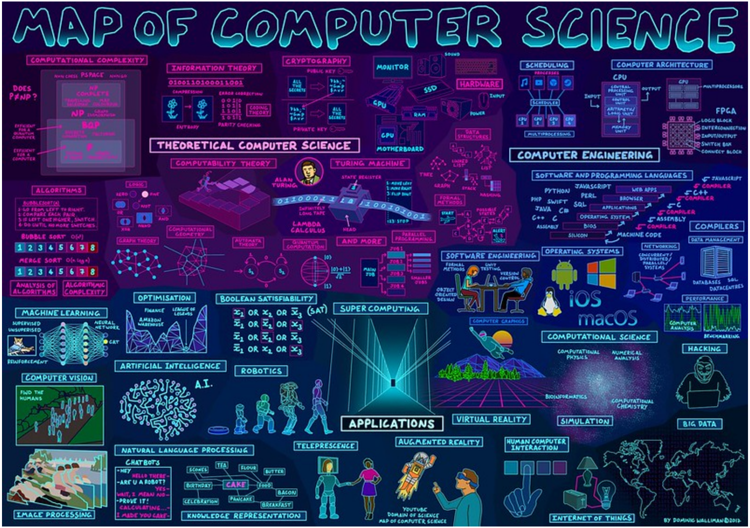
Nesse curso, veremos uma introdução à primeira área.

Introdução

Podemos dividir a Ciência da Computação em duas principais áreas:

1. Modelos e fundamentos implícitos sobre computação;
2. Técnicas de engenharia para o desenvolvimento de sistemas de computação (hardware e software)

Nesse curso, veremos uma introdução à primeira área.



https://youtu.be/SzJ46YA_RaA

Roteiro

Um pouco de história

Antes dos computadores existirem, na década de 1930:

- *Alan Turing* estudou uma máquina abstrata que possui **todas as capacidades** dos computadores da atualidade.
  1. O objetivo de Turing era descrever precisamente os limites entre o que uma *"máquina de computação"* pode ou não fazer.
  2. Seus resultados aplicam-se não somente à sua *máquina*, mas também às máquinas reais de hoje em dia.

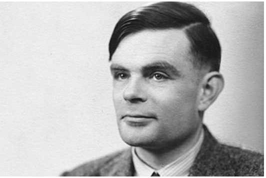


Figura: Alan Turing

Um pouco de história

Antes dos computadores existirem, na década de 1930:

- *Alan Turing* estudou uma máquina abstrata que possui **todas as capacidades** dos computadores da atualidade.
  1. O objetivo de Turing era descrever precisamente os limites entre o que uma *"máquina de computação"* pode ou não fazer.
  2. Seus resultados aplicam-se não somente à sua *máquina*, mas também às máquinas reais de hoje em dia.

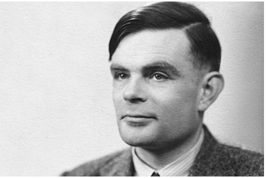


Figura: Alan Turing

Um pouco de história

Antes dos computadores existirem, na década de 1930:

- *Alan Turing* estudou uma máquina abstrata que possui **todas as capacidades** dos computadores da atualidade.
  1. O objetivo de Turing era descrever precisamente os limites entre o que uma *"máquina de computação"* pode ou não fazer.
  2. Seus resultados aplicam-se não somente à sua *máquina*, mas também às máquinas reais de hoje em dia.

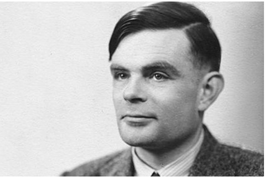


Figura: Alan Turing

Um pouco de história

Nas décadas de 1940 e 1950:

- Tipos mais simples de máquinas, que chamamos hoje de *"autômatos finitos"* (AF) foram estudados.
- AF foram propostos inicialmente para modelar redes neurais[1] e tornaram-se úteis para uma variadade de problemas.

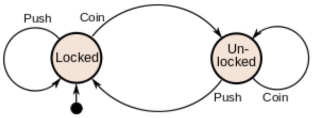


Figura: AF de uma catraca

[1] https://link.springer.com/article/10.1007%2FBF02478259

Um pouco de história

Nas décadas de 1940 e 1950:

- Tipos mais simples de máquinas, que chamamos hoje de *"autômatos finitos"* (AF) foram estudados.
- AF foram propostos inicialmente para modelar redes neurais[1] e tornaram-se úteis para uma variadade de problemas.

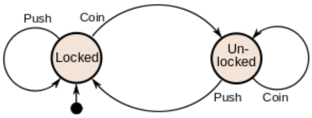


Figura: AF de uma catraca

[1] https://link.springer.com/article/10.1007%2FBF02478259

Um pouco de história

No final da década de 1950, o linguista *Noam Chomsky* iniciou o estudo formal das *"gramáticas"*.

- Embora não envolva estritamente *"máquinas"*, gramáticas estão fortemente relacionadas ao formalismo abstrato dos autômatos.
- Esses resultados servem como base para importantes componentes de software, como partes de um *Compilador*.

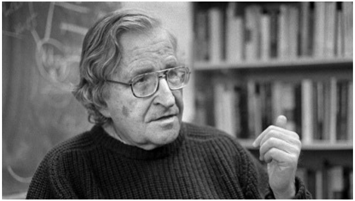


Figura: Noam Chomsky

Um pouco de história

No final da década de 1950, o linguista *Noam Chomsky* iniciou o estudo formal das *"gramáticas"*.

- Embora não envolva estritamente *"máquinas"*, gramáticas estão fortemente relacionadas ao formalismo abstrato dos autômatos.
- Esses resultados servem como base para importantes componentes de software, como partes de um *Compilador*.

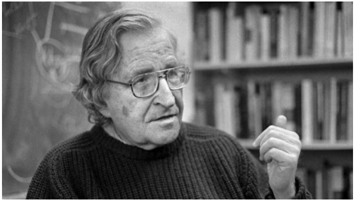


Figura: Noam Chomsky

Um pouco de história

Em 1969:

1. *Stephen Cook* estendeu o estudo de Turing sobre o que pode e o que não pode ser computável.
2. Cook também separou os problemas que são computáveis mas não podem ser resolvidos na prática:
    - Esse problemas são chamados de "intratáveis".

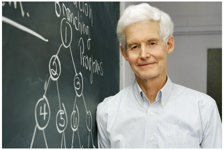


Figura: Stephen Cook

Um pouco de história

Em 1969:

1. *Stephen Cook* estendeu o estudo de Turing sobre o que pode e o que não pode ser computável.
2. Cook também separou os problemas que são computáveis mas não podem ser resolvidos na prática:
   - Esse problemas são chamados de *"intratáveis"*.

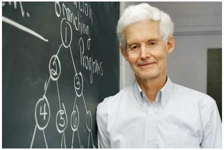


Figura: Stephen Cook

Um pouco de história

Em 1969:

1. *Stephen Cook* estendeu o estudo de Turing sobre o que pode e o que não pode ser computável.
2. Cook também separou os problemas que são computáveis mas não podem ser resolvidos na prática:
   - Esse problemas são chamados de *"intratáveis"*.

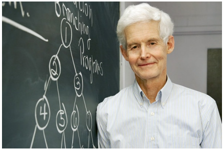
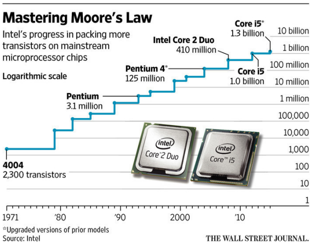


Figura: Stephen Cook

Um pouco de história

Observação importante:

1. Mesmo com o crescimente exponencial da capacidade de computação (tempo e espaço), conhecido como *"Lei de Moore"*, é improvável que isso impacte em nossa capacidade de resolver problemas intratáveis.
2. O que é *big data* hoje, amanhã não é mais.

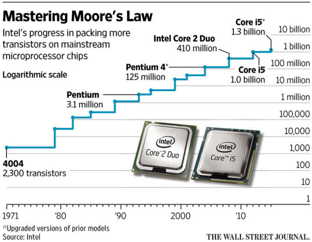


https://www.weforum.org/agenda/2016/01/why-the-internet-is-outgrowing-moore-s-law

Um pouco de história

Observação importante:

1. Mesmo com o crescimente exponencial da capacidade de computação (tempo e espaço), conhecido como *"Lei de Moore"*, é improvável que isso impacte em nossa capacidade de resolver problemas intratáveis.
2. O que é *big data* hoje, amanhã não é mais.



---

https://www.weforum.org/agenda/2016/01/why-the-internet-is-outgrowing-moore-s-law

Um pouco de história

Importância dos Modelos e Fundamentos:

- Todos esses desenvolvimentos dão suporte diretamente no que Cientistas/Engenheiros da Computação fazem hoje.

Um pouco de história

A *Teoria da Computação* tem sido desenvolvida em diferentes áreas:

1. Biólogos pesquisando *redes de neurônios*.
2. Engenheiros desenvolvendo *teoria de circuitos elétricos*.
3. Matemáticos estudando *fundamentos da lógica*.
4. Linguistas investigando *gramáticas* e *linguagens naturais*.
5. Cientistas da Computação pesquisando *mais teoria!!*

Um pouco de história

A *Teoria da Computação* tem sido desenvolvida em diferentes áreas:

1. Biólogos pesquisando *redes de neurônios*.
2. Engenheiros desenvolvendo *teoria de circuitos elétricos*.
3. Matemáticos estudando *fundamentos da lógica*.
4. Linguistas investigando *gramáticas* e *linguagens naturais*.
5. Cientistas da Computação pesquisando *mais teoria!!*

Um pouco de história

A *Teoria da Computação* tem sido desenvolvida em diferentes áreas:

1. Biólogos pesquisando *redes de neurônios*.
2. Engenheiros desenvolvendo *teoria de circuitos elétricos*.
3. Matemáticos estudando *fundamentos da lógica*.
4. Linguistas investigando *gramáticas* e *linguagens naturais*.
5. Cientistas da Computação pesquisando *mais teoria!!*

Um pouco de história

A *Teoria da Computação* tem sido desenvolvida em diferentes áreas:

1. Biólogos pesquisando *redes de neurônios*.
2. Engenheiros desenvolvendo *teoria de circuitos elétricos*.
3. Matemáticos estudando *fundamentos da lógica*.
4. Linguistas investigando *gramáticas* e *linguagens naturais*.
5. Cientistas da Computação pesquisando *mais teoria!!*

Um pouco de história

A *Teoria da Computação* tem sido desenvolvida em diferentes áreas:

1. Biólogos pesquisando *redes de neurônios*.
2. Engenheiros desenvolvendo *teoria de circuitos elétricos*.
3. Matemáticos estudando *fundamentos da lógica*.
4. Linguistas investigando *gramáticas* e *linguagens naturais*.
5. Cientistas da Computação pesquisando *mais teoria!!*

Roteiro

1. Introdução
   - O que é a Computação?
   - Um pouco de história
   - O que vamos aprender neste curso
2. Conceitos Básicos
   - Alfabetos, Palavras e Linguagens
3. Sistemas de Estados Finitos
4. Referências

O que vamos aprender neste curso

A Teoria da Computação é composta de três partes centrais:

1. Linguagens Formais e dos Autômatos,
2. Computabilidade e
3. Complexidade.

Essas áreas são interligadas pela questão:

Quais são as capacidades e as limitações fundamentais dos computadores?

O que vamos aprender neste curso

A Teoria da Computação é composta de três partes centrais:

1. Linguagens Formais e dos Autômatos,
2. Computabilidade e
3. Complexidade.

Essas áreas são interligadas pela questão:

> *Quais são as capacidades e as limitações fundamentais dos computadores?*

A primeira parte (Teoria das Linguagens Formais e dos Autômatos):

- Trata das definições e propriedades de modelos matemáticos de computação.
- Excelente para se começar a estudar a teoria da computação (para depois definir o que é ou não computável).

Esses modelos desempenham um papel em diversas áreas aplicadas:

Teoria das Linguagens Formais e dos Autômatos

A primeira parte (Teoria das Linguagens Formais e dos Autômatos):

- Trata das definições e propriedades de modelos matemáticos de computação.
- Excelente para se começar a estudar a teoria da computação (para depois definir o que é ou não computável).

Esses modelos desempenham um papel em diversas áreas aplicadas:

Teoria das Linguagens Formais e dos Autômatos

A primeira parte (Teoria das Linguagens Formais e dos Autômatos):

- Trata das definições e propriedades de modelos matemáticos de computação.
- Excelente para se começar a estudar a teoria da computação (para depois definir o que é ou não computável).

Esses modelos desempenham um papel em diversas áreas aplicadas:

1. Um modelo, chamado de *Autômato Finito*, é usando em processamento de textos, projeto de hardware, . . .
2. Outro modelo, denominado *Gramática Livre-de-Contexto*, é utilizado em compiladores, processamento de linguagem natural, . . .

Teoria das Linguagens Formais e dos Autômatos

A primeira parte (Teoria das Linguagens Formais e dos Autômatos):

- Trata das definições e propriedades de modelos matemáticos de computação.
- Excelente para se começar a estudar a teoria da computação (para depois definir o que é ou não computável).

Esses modelos desempenham um papel em diversas áreas aplicadas:

1. Um modelo, chamado de *Autômato Finito*, é usando em processamento de textos, projeto de hardware, ...
2. Outro modelo, denominado *Gramática Livre-de-Contexto*, é utilizado em compiladores, processamento de linguagem natural, ...

A primeira parte (Teoria das Linguagens Formais e dos Autômatos):

- Trata das definições e propriedades de modelos matemáticos de computação.
- Excelente para se começar a estudar a teoria da computação (para depois definir o que é ou não computável).

Esses modelos desempenham um papel em diversas áreas aplicadas:

1. Um modelo, chamado de *Autômato Finito*, é usando em processamento de textos, projeto de hardware, . . .
2. Outro modelo, denominado *Gramática Livre-de-Contexto*, é utilizado em compiladores, processamento de linguagem natural, . . .

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

- Parece uma questão natural para ser resolvida por um computador.
- Mas **nenhum algoritmo** de computador (ou MT) pode realizar essa tarefa.

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

- Parece uma questão natural para ser resolvida por um computador.
- Mas **nenhum algoritmo** de computador (ou MT) pode realizar essa tarefa.

Teoria da Computabilidade

A segunda parte (Teoria da Computabilidade):

- Apresenta a Máquina de Turing (MT).
- Estabelece a conexão entre a *noção informal de algoritmo* (solúvel efetivamente) e a definição precisa por uma MT.
- Veremos que se um problema não pode ser resolvido por uma MT, então não existe nenhuma solução computável para ele.

Um exemplo é o problema de verificar se um *enunciado matemático* é verdadeiro ou falso.

- Parece uma questão natural para ser resolvida por um computador.
- Mas **nenhum algoritmo** de computador (ou MT) pode realizar essa tarefa.

Teoria da Complexidade

A terceira parte (Teoria da Complexidade):

- Trata da classificação de problemas de acordo com a dificuldade computacional[2].
- Veremos que nem todos os problemas computáveis, podem ser resolvidos na prática: os recursos computacionais requeridos (tempo ou espaço) podem ser proibitivos.

Um exemplo é o problema de *fatoração de números grandes*:

[2] O escalonamento de salas de aula (pode levar 1 século).

Teoria da Complexidade

A terceira parte (Teoria da Complexidade):

- Trata da classificação de problemas de acordo com a dificuldade computacional[2].
- Veremos que nem todos os problemas computáveis, podem ser resolvidos na prática: os recursos computacionais requeridos (tempo ou espaço) podem ser proibitivos.

Um exemplo é o problema de *fatoração de números grandes*:

[2] O escalonamento de salas de aula (pode levar 1 século).

Teoria da Complexidade

A terceira parte (Teoria da Complexidade):

- Trata da classificação de problemas de acordo com a dificuldade computacional[2].
- Veremos que nem todos os problemas computáveis, podem ser resolvidos na prática: os recursos computacionais requeridos (tempo ou espaço) podem ser proibitivos.

Um exemplo é o problema de *fatoração de números grandes*:

- Atualmente nem mesmo um supercomputador podem encontrar todos os fatores de um número grande ($>$ 500 dígitos) *antes do sol se apagar!*
- Esse tipo de problema pode ser útil em Criptografia: requer problemas computacionais que sejam difíceis.

[2] O escalonamento de salas de aula (pode levar 1 século).

Teoria da Complexidade

A terceira parte (Teoria da Complexidade):

- Trata da classificação de problemas de acordo com a dificuldade computacional[2].
- Veremos que nem todos os problemas computáveis, podem ser resolvidos na prática: os recursos computacionais requeridos (tempo ou espaço) podem ser proibitivos.

Um exemplo é o problema de *fatoração de números grandes*:

- Atualmente nem mesmo um supercomputador podem encontrar todos os fatores de um número grande ($>$ 500 dígitos) *antes do sol se apagar!*
- Esse tipo de problema pode ser útil em Criptografia: requer problemas computacionais que sejam difíceis.

[2] O escalonamento de salas de aula (pode levar 1 século).

Teoria da Complexidade

A terceira parte (Teoria da Complexidade):

- Trata da classificação de problemas de acordo com a dificuldade computacional[2].
- Veremos que nem todos os problemas computáveis, podem ser resolvidos na prática: os recursos computacionais requeridos (tempo ou espaço) podem ser proibitivos.

Um exemplo é o problema de *fatoração de números grandes*:

- Atualmente nem mesmo um supercomputador podem encontrar todos os fatores de um número grande ($>$ 500 dígitos) *antes do sol se apagar!*
- Esse tipo de problema pode ser útil em Criptografia: requer problemas computacionais que sejam difíceis.

[2] O escalonamento de salas de aula (pode levar 1 século).

Objetivos gerais:

- Determinar o que pode e o que não pode *ser computado*, quão rapidamente, com quanto de memória e sobre que tipo de **modelo computacional**.

Roteiro

Roteiro

Símbolos

- Um símbolo (ou caractere) é uma entidade abstrata que não definimos formalmente (como um ponto em geometria).

Alfabeto

Definição

Um alfabeto é um conjunto finito de símbolos ou caracteres.

Portanto:

- Um conjunto infinito não é um alfabeto
- Um conjunto vazio ($\varnothing$) é um alfabeto

Alfabeto

Definição

Um alfabeto é um conjunto finito de símbolos ou caracteres.

Portanto:

- Um conjunto infinito não é um alfabeto
- Um conjunto vazio ($\varnothing$) é um alfabeto

Exemplos

São exemplos de alfabetos:

- $\{a, b, c\}$
- $\{\}$
- $A = \{\forall x \in \mathbb{N} \mid x \leq 10\}$

Não são exemplos de alfabetos:

- $\mathbb{N}$
- $\{a, ab, aa, ab, bb, aaa, \ldots\}$

Representação

Σ é normalmente usado para representar um alfabeto.

Exemplos

São exemplos de alfabetos:

- $\{a, b, c\}$
- $\{\}$
- $A = \{\forall x \in \mathbb{N} \mid x \leq 10\}$

Não são exemplos de alfabetos:

- $\mathbb{N}$
- $\{a, ab, aa, ab, bb, aaa, \ldots\}$

Representação

Σ é normalmente usado para representar um alfabeto.

Exemplos

São exemplos de alfabetos:

- $\{a, b, c\}$
- $\{\}$
- $A = \{\forall x \in \mathbb{N} \mid x \leq 10\}$

Não são exemplos de alfabetos:

- $\mathbb{N}$
- $\{a, ab, aa, ab, bb, aaa, \ldots\}$

Representação

Σ é normalmente usado para representar um alfabeto.

Palavra

Definição

Uma palavra (ou cadeia de caracteres) é uma sequência finita de símbolos (de um alfabeto) justapostos.

Exemplos:

- a, b e c são símbolos e abc é uma palavra.
- Notem que, pela definição acima, uma cadeia sem símbolos é uma palavra válida.
  - Notação: $\mathcal{E}$ = palavra vazia ou cadeia vazia

Palavra

Definição

Uma palavra (ou cadeia de caracteres) é uma sequência finita de símbolos (de um alfabeto) justapostos.

Exemplos:

- a, b e c são símbolos e abc é uma palavra.
- Notem que, pela definição acima, uma cadeia sem símbolos é uma palavra válida.
  - Notação: $\mathcal{E}$ = palavra vazia ou cadeia vazia

Palavra

Definição

Uma palavra (ou cadeia de caracteres) é uma sequência finita de símbolos (de um alfabeto) justapostos.

Exemplos:

- a, b e c são símbolos e abc é uma palavra.
- Notem que, pela definição acima, uma cadeia sem símbolos é uma palavra válida.
  - Notação: $\mathcal{E}$ = palavra vazia ou cadeia vazia

Comprimento de uma palavra

Definição

O comprimento, ou tamanho, de uma palavra w, representado por $|w|$, é o número de símbolos que compõem a palavra.

Exemplos:

- $|abcd| = 4$
- $|\mathcal{E}| = 0$

Prefixo e sufixo

Definição

Um prefixo de uma palavra é qualquer sequência inicial de símbolos da palavra.

Definição

Um sufixo de uma palavra é qualquer sequência final de símbolos da palavra.

Exemplos:

- abcd é uma palavra sobre $\Sigma = \{a, b, c, d\}$
- $\mathcal{E}, a, ab, abc, abcd$ são prefixos na palavra $abcd$
- $\mathcal{E}, d, cd, bcd, abcd$ são sufixos na palavra $abcd$

Exemplos

Definição

Uma subpalavra de uma palavra é qualquer sequência de símbolos contíguos da palavra.

Portanto, qualquer prefixo ou sufixo é uma subpalavra.
Exemplos:

- abcd é uma palavra sobre $\Sigma = \{a, b, c, d\}$
- $\mathcal{E}, a, b, c, d, ab, bc, cd, abc, bcd, abcd$ são subpalavra $abcd$

Concatenação de palavras

Definição

A concatenação de duas palavras v e w é uma operação binária em que uma nova palavra é formada pela justaposição de v e w

Exemplos: sejam as palavras $v = baaaa$ e $w = bb$

- $vw = baaaabb$
- $v\mathcal{E} = v = baaaa$

Usamos aqui duas propriedades da concatenação:

Elemento neutro $\mathcal{E}w = w\mathcal{E} = w$

Associatividade $v(wt) = (vw)t$

Concatenação de palavras

Definição

A concatenação de duas palavras v e w é uma operação binária em que uma nova palavra é formada pela justaposição de v e w

Exemplos: sejam as palavras $v = baaaa$ e $w = bb$

- $vw = baaaabb$
- $v\mathcal{E} = v = baaaa$

Usamos aqui duas propriedades da concatenação:

Elemento neutro $\mathcal{E}w = w\mathcal{E} = w$

Associatividade $v(wt) = (vw)t$

Concatenação sucessiva

Definição

A concatenação sucessiva de uma palavra (e.g.: 'w'), representada na forma de um expoente numa palavra, ou seja:

w^n, tal que, n é o número de concatenações sucessivas,

é definida indutivamente como:

- $w^0 = \mathcal{E}$
- $w^n = ww^{n-1}$, para $n > 0$

Exemplo

Exemplos:

- $w^3 = www$
- $w^1 = w$
- $a^n = \underbrace{aaaa\ldots a}_{n}$

Cadeia invertida

Definição

O *inverso* de uma cadeia $w = w_1 w_2 \ldots w_n$ é denotado por $w^R = w_n w_{n-1} \ldots w_1$.

Exemplo

Exemplos:

- $w = abcde$ e $w^R = edcba$
- $(w^R)^R = w$
- $(a^n)^R = \underbrace{aaaa\ldots a}_{n}$

Importante!

Notações importantes

Se Σ representa um alfabeto, então:

- Σ^* denota o conjunto de todas as palavras possíveis em Σ; e
- $\Sigma^+ = \Sigma^* - \{\mathcal{E}\}$

Considerando $\Sigma = \{a, b\}$, então:

- $\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, \ldots\}$
- $\Sigma^* = \{\mathcal{E}, a, b, aa, ab, ba, bb, aaa, \ldots\}$

Definição alternativa

Definição de palavra

Podemos definir uma palavra sobre um alfabeto Σ como qualquer elemento w de Σ^*, ou seja,

$$w \in \Sigma^*$$

Linguagem formal

Definição

Uma linguagem formal, ou simplesmente uma linguagem L definida sobre um alfabeto Σ, é um **conjunto de palavras** sobre Σ, ou seja,

$$L \subseteq \Sigma^*$$

Lembrando... Σ^* são todas as palavras possíveis, incluindo $\mathcal{E}$.

Linguagens e Cadeias

Em outras palavras, uma linguagem L é um *conjunto de palavras*.

Linguagens e Cadeias

Exemplos:

1. A linguagem de todas as cadeias que consistem em n 0's seguidos por n 1's, para algum $n \geq 0$:

$$\{\mathcal{E}, 01, 0011, 000111, \dots\}$$

2. O conjunto de cadeias de 0's e 1's com um número igual de cada um deles:

$$\{\mathcal{E}, 01, 10, 0011, 0101, 1001 \dots\}$$

Linguagens e Cadeias

Exemplos:

1. A linguagem de todas as cadeias que consistem em n 0's seguidos por n 1's, para algum $n \geq 0$:

$$\{\mathcal{E}, 01, 0011, 000111, \dots\}$$

2. O conjunto de cadeias de 0's e 1's com um número igual de cada um deles:

$$\{\mathcal{E}, 01, 10, 0011, 0101, 1001 \dots\}$$

Linguagens e Cadeias

Outros exemplos:

3. O conjunto de números na forma binária cujo valor é um número primo:

$$\{10, 11, 101, 111, 1011, \ldots\}$$

4. Σ^* é uma linguagem para qualquer alfabeto Σ
5. $\varnothing$ e $\{\mathcal{E}\}$ são linguagens sobre qualquer alfabeto. Lembrando que:

$$\varnothing \neq \{\mathcal{E}\}$$

Linguagens e Cadeias

Outros exemplos:

3. O conjunto de números na forma binária cujo valor é um número primo:
$$\{10, 11, 101, 111, 1011, \ldots\}$$
4. Σ^* é uma linguagem para qualquer alfabeto Σ
5. $\varnothing$ e $\{\mathcal{E}\}$ são linguagens sobre qualquer alfabeto. Lembrando que:
$$\varnothing \neq \{\mathcal{E}\}$$

Linguagens e Cadeias

Outros exemplos:

3. O conjunto de números na forma binária cujo valor é um número primo:

$$\{10, 11, 101, 111, 1011, \dots\}$$

4. Σ^* é uma linguagem para qualquer alfabeto Σ
5. $\varnothing$ e $\{\mathcal{E}\}$ são linguagens sobre qualquer alfabeto. Lembrando que:

$$\varnothing \neq \{\mathcal{E}\}$$

$\varnothing$ e $\{\mathcal{E}\}$

Diferença conceitual entre $\varnothing$ e $\{\mathcal{E}\}$:

- $\varnothing$ é o *conjunto vazio*, que significa a ausência de palavra
- $w = \mathcal{E}$ é a *palavra vazia*, mas é uma palavra
  - Se a palavra $w_2 = abab$, temos que $w_2 w = abab\mathcal{E}$

$\varnothing$ e $\{\mathcal{E}\}$

Diferença conceitual entre $\varnothing$ e $\{\mathcal{E}\}$:

- $\varnothing$ é o *conjunto vazio*, que significa a ausência de palavra
- $w = \mathcal{E}$ é a *palavra vazia*, mas é uma palavra
  - Se a palavra $w_2 = abab$, temos que $w_2 w = abab\mathcal{E}$

$\varnothing$ e $\{\mathcal{E}\}$

Diferença conceitual entre $\varnothing$ e $\{\mathcal{E}\}$:

- $\varnothing$ é o *conjunto vazio*, que significa a ausência de palavra
- $w = \mathcal{E}$ é a *palavra vazia*, mas é uma palavra
  - Se a palavra $w_2 = abab$, temos que $w_2 w = abab\mathcal{E}$

Linguagens e Cadeias

Podemos definir uma linguagem usando um *"formador de conjuntos"*:

$$L = \{w | \text{ algo sobre } w\}$$

Exemplos:

1. $L_1 = \{w \mid w \text{ possui um número igual de 0's e 1's e } \Sigma = \{0, 1\}\}$
2. $L_2 = \{w \mid w \text{ é uma cadeia sobre } \Sigma = \{a, b\} \text{ e } w \text{ termina com } a\}$

Linguagens e Cadeias

Podemos definir uma linguagem usando um *"formador de conjuntos"*:

$$L = \{w |\ \text{algo sobre } w\}$$

Exemplos:

1. $L_1 = \{w \mid w \text{ possui um número igual de 0's e 1's e } \Sigma = \{0, 1\}\}$
2. $L_2 = \{w \mid w \text{ é uma cadeia sobre } \Sigma = \{a, b\} \text{ e } w \text{ termina com } a\}$

Outros exemplos:

3. $L_3 = \{ww^R \mid$ w é uma cadeia sobre $\Sigma = \{0, 1\}\}$
4. $L_4 = \{w \mid w = w^R$, e w é uma cadeia sobre $\Sigma = \{a, b\}\}$

Linguagens e Cadeias

Outros exemplos:

3. $L_3 = \{ww^R \mid$ w é uma cadeia sobre $\Sigma = \{0, 1\}\}$
4. $L_4 = \{w \mid w = w^R$, e w é uma cadeia sobre $\Sigma = \{a, b\}\}$

Linguagens e Cadeias

Também podemos utilizar expressões com parâmetros:

5. $L_5 = \{0^n 1^n \mid n \geq 1\}$
6. $L_6 = \{0^i 1^j \mid 0 \geq i \geq j\}$

Linguagens e Cadeias

Também podemos utilizar expressões com parâmetros:

5. $L_5 = \{0^n 1^n \mid n \geq 1\}$
6. $L_6 = \{0^i 1^j \mid 0 \geq i \geq j\}$

Importante!

Observações:

- Linguagens podem ser infinitas.
- A única restrição importante sobre o que pode ser uma linguagem é que todos os alfabetos são finitos.

Exemplos de linguagens

- Σ^* e Σ^+ são *linguagens* sobre Σ. Lembrando que

$$\Sigma^* \neq \Sigma^+$$

- Seja $\Sigma = \{a, b\}$, então o *conjunto de palíndromos* sobre Σ é um exemplo de linguagem infinita.
  São palavras desta linguagem $\mathcal{E}, a, b, aa, bb, aba, bab, aaa, \ldots$

Exemplos de linguagens

- Σ^* e Σ^+ são *linguagens* sobre Σ. Lembrando que

$$\Sigma^* \neq \Sigma^+$$

- Seja $\Sigma = \{a, b\}$, então o *conjunto de palíndromos* sobre Σ é um exemplo de linguagem infinita.
  São palavras desta linguagem $\mathcal{E}, a, b, aa, bb, aba, bab, aaa, \ldots$

Roteiro

1. Introdução
   - O que é a Computação?
   - Um pouco de história
   - O que vamos aprender neste curso
2. Conceitos Básicos
   - Alfabetos, Palavras e Linguagens
3. **Sistemas de Estados Finitos**
4. Referências

Sistemas de estados finitos

Vamos começar com um *modelo computacional* simples:

- Um **sistema de estados finitos** ou **autômato finito** (AF).

Definição

Um sistema de estados finitos é um modelo matemático que descreve um sistema (de estados finitos) com entrada e saída.

Sistemas de estados finitos

Vamos começar com um *modelo computacional* simples:

- Um **sistema de estados finitos** ou **autômato finito** (AF).

Definição

Um sistema de estados finitos é um modelo matemático que descreve um sistema (de estados finitos) com entrada e saída.

Sistemas de estados finitos

Algumas características de um **sistema de estados finitos**:

- Número finito e pré-definido de estados
  - Estado: mostra a situação atual e passada do sistema
  - Ação: determina a transição de um estado para outro

Sistemas de estados finitos

Algumas características de um **sistema de estados finitos**:

- Número finito e pré-definido de estados
  - Estado: mostra a situação atual e passada do sistema
  - Ação: determina a transição de um estado para outro

Sistemas de estados finitos

Um **sistema de estados finitos** possui uma quantidade de memória limitada.

- Apesar disso, interagimos diariamente com esses computadores:
  - Controlador de porta automática, catraca de ônibus, controle de um elevador, semáforos, ...

Sistemas de estados finitos

Um **sistema de estados finitos** possui uma quantidade de memória limitada.

- Apesar disso, interagimos diariamente com esses computadores:
  - Controlador de porta automática, catraca de ônibus, controle de um elevador, semáforos, ...

Ilustração de porta automática

Esta porta automática possui um controlador que possibilita sua abertura assim que alguém pisa no tapete frontal (sentido único).

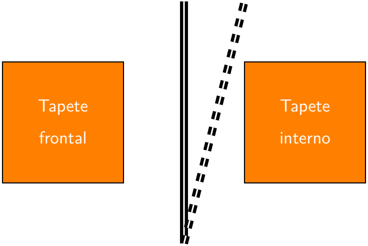
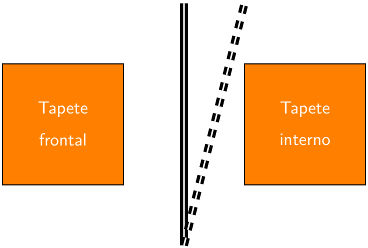


Figura: Ilustração de uma porta automática.

Sinais de porta automática

Existem 4 condições possíveis

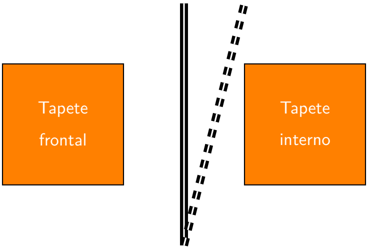
Nenhum	Ninguém pisa sobre qualquer tapete
Frontal	Alguém pisa no tapete frontal
Interno	Alguém pisa no tapete interno
Ambos	Pessoas pisam sobre os dois tapetes



Como o controlador opera?

Preencha os espaços com *"aberta"* ou *"fechada"*:

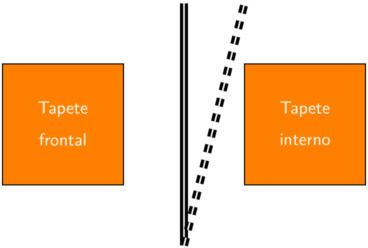
Estado	**Sinal de entrada**			
	Nenhum	**Frontal**	**Interno**	**Ambos**
Fechada				
Aberta				



Como o controlador opera?

Resposta:

Estado	**Sinal de entrada**			
	Nenhum	**Frontal**	**Interno**	**Ambos**
Fechada	*fechada*	*aberta*	*fechada*	*fechada*
Aberta	*fechada*	*aberta*	*aberta*	*aberta*



Porta automática: diagrama de estados

Tabela de transições:

Estado	Sinal de entrada			
	Nenhum	**Frontal**	**Interno**	**Ambos**
Fechada	*fechada*	*aberta*	*fechada*	*fechada*
Aberta	*fechada*	*aberta*	*aberta*	*aberta*

Figura: Considerando só os estados possíveis de uma porta automática.

Porta automática: diagrama de estados

Tabela de transições:

Estado	Sinal de entrada			
	Nenhum	**Frontal**	**Interno**	**Ambos**
Fechada	*fechada*	*aberta*	*fechada*	*fechada*
Aberta	*fechada*	*aberta*	*aberta*	*aberta*

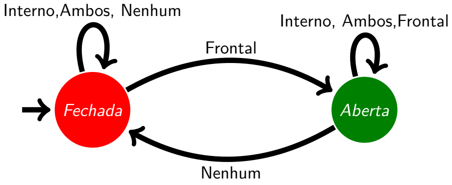


Figura: Diagrama de estados para um controlador de uma porta automática.

Este é um autômato finito de dois estados.

Porta automática: diagrama de estados

Tabela de transições:

Estado	Sinal de entrada			
	Nenhum	**Frontal**	**Interno**	**Ambos**
Fechada	*fechada*	*aberta*	*fechada*	*fechada*
Aberta	*fechada*	*aberta*	*aberta*	*aberta*

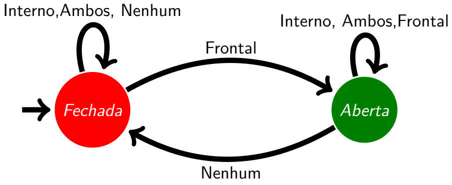


Figura: Diagrama de estados para um controlador de uma porta automática.

Este é um autômato finito de dois estados.

Autômatos Finitos

Esse controlador é um bom exemplo de **sistema de estados finitos**:

- Não existe memória de estados anteriores, apenas sabe-se o estado atual, e o conjunto de possíveis transições (para outros estados).
- Podemos dizer que é um computador que tem apenas *1 bit de memória*.

Vamos ver outros exemplos com memórias maiores.

Autômatos Finitos

Esse controlador é um bom exemplo de **sistema de estados finitos**:

- Não existe memória de estados anteriores, apenas sabe-se o estado atual, e o conjunto de possíveis transições (para outros estados).
- Podemos dizer que é um computador que tem apenas *1 bit de memória*.

Vamos ver outros exemplos com memórias maiores.

Fim

Dúvidas?

Roteiro

Referências

Referências:

1. *"Introdução à Teoria da Computação"* de M. Sipser, 2007.
2. *"Introdução à Teoria de Autômatos, Linguagens e Computação"* de J. E. Hopcroft, R. Motwani, e J. D. Ullman, 2003.
3. Materiais adaptados dos slides do Prof. Evandro E. S. Ruiz, da USP.