Cap1: Histórico

- Foco: Modelos matemáticos para descrição de computadores
  - Máquinas
  - Linguagens

Cap1: Histórico

- Alan Turing (década 30)
  - Turing Machine: Modelo de computador
  - Existem questões fundamentais que podem ser precisamente definidas mas que não podem ser decididas.
  - The halting Problem: Dado uma entrada W e uma máquina T não é possível saber se T para para W.

Cap1: Histórico

- Alonzo Church (década 30)
  - Tese de Church: Não há nenhuma função que possa ser definida por humanos e cujo calculo possa ser descrito por um algoritmo que não possa ser calculada por uma máquina de Turing.

Cap1: Histórico

- Décadas 40 e 50
  - Vários modelos de automatos
- Chomsky (década 50)
  - Estudos de gramáticas formais
- Cook (década 60)
  - Estudo Computabilidade
  - Problemas NP, NP-completos

Cap1: Histórico

- Alan Turing
  - The halting Problem: Dado uma entrada W e uma máquina T não é possível saber se T para para W.
- Alonzo Church
  - Tese de Church: Não há nenhuma função que possa ser definida por humanos e cujo calculo possa ser descrito por um algoritmo que não possa ser calculada por uma máquina de Turing.

Cap1: Tópicos

- Autômatos
- Linguagens
- Máquinas de Turing

Cap2: Linguagens

- Definições:
  - Alfabeto (Σ): Conjunto finito de símbolos .
  - Sentença ou palavra (w): Cadeia finita de símbolos de um alfabeto.
  - Linguagem (L): Conjunto específico de sentenças sobre um alfabeto.

Cap2: Linguagens

- Definições:
  - Palavra vazia (ε): Palavra sem símbolos. O alfabeto, por definição, nunca inclui a palavra vazia.
  - Linguagem vazia (ϕ): Linguagem sem palavras, inclusive a vazia.
- Resultados:
  - $\varepsilon \not\subset L \Rightarrow L + \{\varepsilon\} \neq L$
  - $L + \phi = L$

Cap2: Linguagens

- Expressões regulares: Gerador recursivo de palavras pertencentes a uma linguagem.
  - Definição Recursiva de ER
    - Dado Σ = alfabeto
    - Todo símbolo $\alpha \in \Sigma$ é uma ER
    - Se β_1 é uma ER e β_2 é uma ER então:
      - $\beta_1\beta_2$ é uma ER (concatenação)
      - $\beta_1 + \beta_2$ é uma ER (alternativa)

Cap2: Linguagens

- Definição Recursiva de ER (continuação)
  - Se β_1 é uma ER e β_2 é uma ER então:
    - β_1^* é uma ER (fechamento de Kleene/repetição)
    - (β_1) é uma ER
- Notações alternativas:
  - β_1^+ : uma ou mais repetições
  - $\beta_1?$: zero ou uma repetição
  - $\beta_1 \mid \beta_2$: alternativa

Cap2: Linguagens

- Expressões Regulares:
  - Fechamento de Kleene: zero ou mais repetições
    - Ex: $\Sigma = \{0\} \Rightarrow \Sigma^* = \{\varepsilon, 0, 00, 000, ...\}$
  - Extensão para Linguagens: L^* é o conjunto de todas as palavras formadas por palavras de L, inclusive a palavra vazia.

Cap2: Linguagens

- Outras representações:
  - Σ^3
  - Σ^n
- Resultados:
  - não é verdade que se $w_1 \in L$ e $w_2 \in L$ então $w_1w_2 \in L$
  - $\phi^* = \{\varepsilon\}$
  - $L^* = L^{**}$

Cap2: Expressões Regulares

- Exemplos:
  - ER de todas as sentenças de *as* e *bês* com pelo menos 2 *as*.
    - (a+b)*a(a+b)*a(a+b)* ou
    - b*ab*a(a+b)* ou
    - b*a(a+b)*ab*
  - Importante:
    - (a+b)*a(a+b)*a(a+b)* ≡ b*a(a+b)*ab* porque definem a mesma linguagem mas isto não implica que (a+b)* ≡ b*

Cap2: Expressões Regulares

- Note que:
  - (a*)* = a*
  - (a*b*)* = (a+b)* = (a* + b*)* ≠ a* + b*
  - (a + ab)* ≠ (a* + ab*)*

Cap2: Expressões Regulares

- Teorema:
  - Toda linguagens finita é uma linguagem regular (pode ser definida por uma ER)
  - Alternativamente:
    - Toda Linguagem que pode ser definida por uma expressão regular é uma linguagem regular

Cap2: Autômatos Finitos

- Autômatos Finitos e Expressões regulares representam o mesmo conjunto de linguagens
- Autômatos Finitos são reconhecedores e Expressões Regulares são geradores

Cap2: Autômatos Finitos

- Motivação
  - Processos determinísticos onde existam finitos estados e onde a passagem de um estado para o outro é determinada apenas pelo estado atual e uma entrada
  - $Estado_{n+1}$ = função($Estado_n$, Entrada)
  - Computador (stand alone) é um destes processos, onde o estado atual é o conteúdo de toda a memória e a função de transição é dada pelo conjunto de todos os programas

Cap2: Autômatos Finitos

- Definição:
  - Um autômato finito é uma quíntupla $<S,I,F,\Sigma,\delta>$ onde:
    - S é um conjunto finito de estados
    - I é o estado inicial tal que $I \in S$
    - F é o conjunto de estados finais tal que $F \subset S$
    - Σ é o alfabeto
    - δ é a função de transição que relaciona cada par <Estado, Símbolo> com um novo estado. Onde Símbolo $\in \Sigma$

Cap2: Autômatos Finitos

- Ex: A linguagem de todas as palavras formadas por a e b onde existe pelo menos um b, (a+b)*b(a + b)*, pode ser representada pelo autômato A abaixo.
  - A = $<S,I,F,\Sigma,\delta>$
    - S = {0,1,2}
    - I = 0
    - F = {2}
    - Σ = {a, b}
    - $\delta(0,a) = 1$, $\delta(0,b) = 2$, $\delta(1,a) = 0$, $\delta(1,b) = 2$,
    - $\delta(2,a) = 2$, $\delta(2,b) = 2$

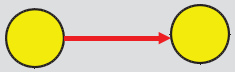
Cap2: Autômatos Finitos

- A função de transição pode ser representada através de uma tabela de transição.
  - Ex: a função do exemplo anterior pode ser representada por:

		a	b
inicial	0	1	2
	1	0	2
final	2	2	2

Cap2: Autômatos Finitos

– Alternativamente todo o autômato pode ser representado por um grafo onde:

-  representam os estados
-  representam transição entre estados
- 

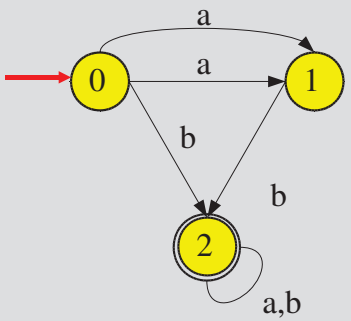
representam a entrada
- 

ou representam o estado inicial
- 

ou representam os estados finais

Cap2: Autômatos Finitos

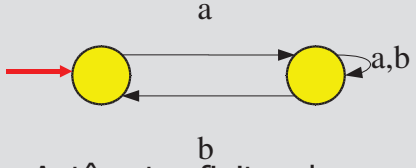
– O autômato visto anteriormente fica então:



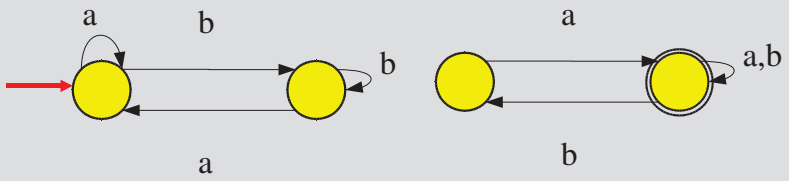
Cap2: Autômatos Finitos

– Autômatos que não definem linguagens:

- Autômatos finitos sem estado final



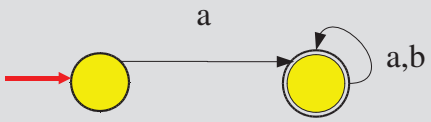
- Autômatos finitos desconectados



Cap2: Autômatos Finitos

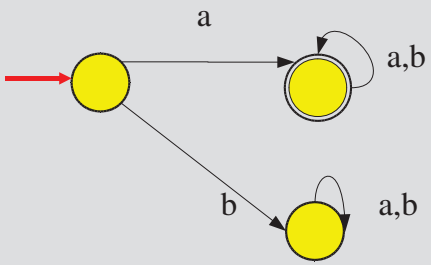
– Estado morto:

- Formalmente cada estado num AF tem que ter arcos para cada símbolo da linguagem
- Como representar a ER a(a+b)*



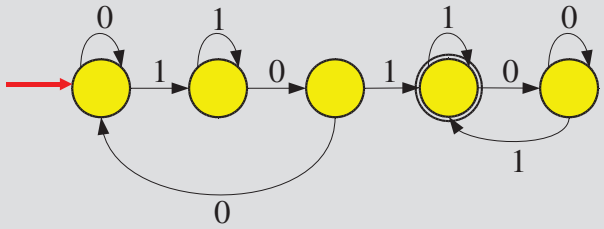
Cap2: Autômatos Finitos

– Estado morto:
  - a(a+b)*



Cap2: Autômatos Finitos

– Exercícios em aula:
  - Dê os autômatos para as linguagens abaixo:
    – Linguagem sobre o alfabeto {0,1,2,3,4,5,6,7,8,9,+,-,.} dos números reais com sinal
    – $L=\{w \mid w \in \{0,1\}^* \wedge w\ contem\ 101 \wedge w\ termina\ por\ 1\}$

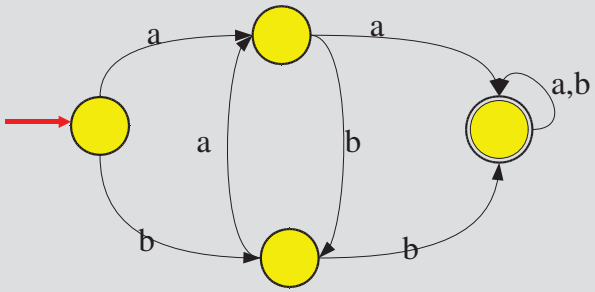


Cap2: Autômatos Finitos

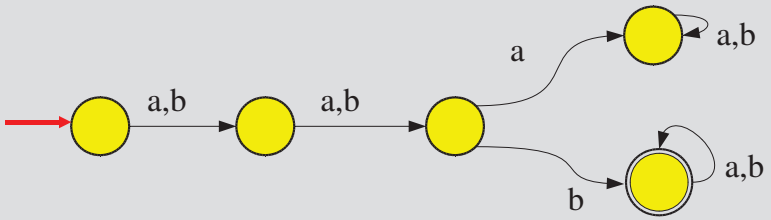
– Exercícios em aula:
  - Dê os autômatos para as linguagens abaixo:
    – Linguagem para o conjunto de (0+1)* com no máximo um par de zeros consecutivos e um par de uns consecutivos.
  - Dê os autômatos para as expressões regulares abaixo:
    – 10 + (0 + 11) 0* 1
    – 01 (((10)* + 111)* + 0)* 1

Cap2: Autômatos Finitos

– Exercícios em aula:
  - Dê a expressão regular para os autômatos:



Cap2: Autômatos Finitos



Cap2: Autômatos finitos

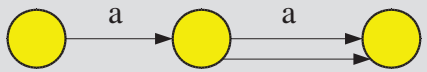
- Existem dois tipos de autômatos finitos:
  - Determinísticos:
    - Para cada entrada existe um e somente um estado para qual o automato pode transitar
    - Abreviatura: AFD ou DFA (em Inglês)
  - Não determinísticos:
    - AFND ou NFA (em Inglês)
  - Embora muitas vezes seja mais fácil representar uma linguagem através de um AFND, as linguagens definidas por AFND e AFD são exatamente as mesmas e são as linguagens regulares

Cap2: Automatos finitos

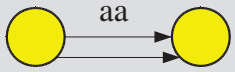
- Flexibilizações:
  - Para facilitar a definição e desenho de AF normalmente se permite algumas flexibilizações na sua representação.
  - Alguns autores preferem chamar o resultado destas alterações de Grafo de Transição.

Cap2: Automatos finitos

- Flexibilizações:
  - Múltiplos símbolos por transição:
    - Num GT pode-se ter mais de um símbolo associado a cada transição.
    - AF:

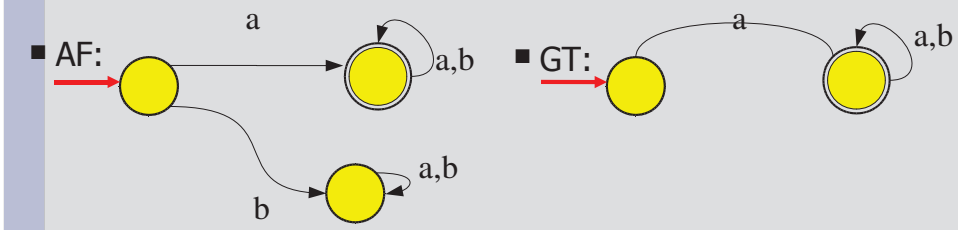


    - GT:



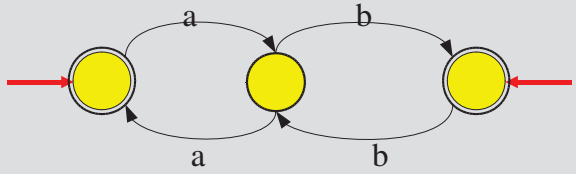
Cap2: Automatos finitos

- Flexibilizações:
  - Ausência de algumas transições:
    - Num GT não é obrigatório que todos os estados tenham transições para todos os símbolos do alfabeto. Isto elimina os estados mortos e cria mais um tipo de rejeição de uma palavra.



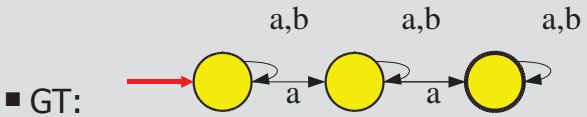
Cap2: Automatos finitos

- Flexibilizações:
  - Múltiplos estados iniciais:
    - Num GT pode-se ter mais de um estado inicial.



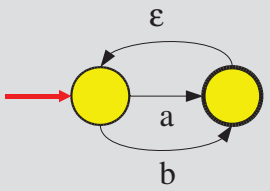
Cap2: Automatos finitos

- Flexibilizações:
  - Não determinismo:
    - Num GT pode-se ter transições aonde é necessário fazer uma escolha mas onde não se tem regra para saber qual é a escolha correta.



Cap2: Automatos finitos

- Flexibilizações:
  - Transições vazias:
    - Num GT pode-se ter transições onde nenhum símbolo é consumido.

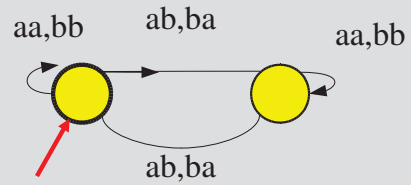


Cap2: Automatos finitos

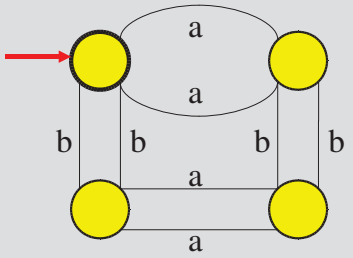
- Conseqüências do ε :
  - Não determinismo
  - Infinitas transições possíveis para cada estado
  - Infinitos caminhos diferentes
  - Caminhos infinitos

Cap2: Automatos finitos

- Exemplo de GT x AF: Número par de as e bês
  - GT:



  - AF:



Cap2: Automatos finitos

- Exercícios:

I. Mostre um autômato finito para a linguagem de todas as sentenças em {a,b} que terminem em número par de as.

II. Dê uma expressão regular para a linguagem de todos os números reais com sinal.

Cap2: Automatos finitos

- Teorema de Kleene
  - Qualquer linguagem que pode ser definida por:
    - Uma Expressão Regular ou
    - Um Autômato Finito ou
    - Um Grafo de Transição;

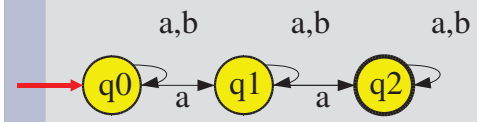
    pode ser definida por qualquer dos três métodos.
  - Em outras palavras os três métodos são equivalentes.

Cap2: Automatos finitos

- AFND → AFD:
  - O método consiste em criar os todos os subconjuntos necessários para os conjuntos de estados do automato inicial.
  - Importante: Caso algum estado do automato não possua saída para algum símbolo um estado terminal extra, erro, deve ser criado e todas as transições que faltam devem levar a este estado.

Cap2: Automatos finitos

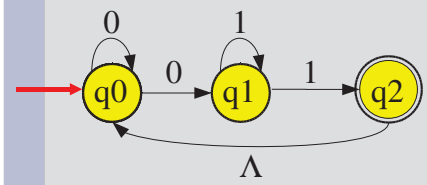
- AFND → AFD:



Estado	Entrada	
	a	b
q0	{q0,q1}	q0
{q0,q1}	{q0,q1,q2}	{q0,q1}
{q0,q1,q2}	{q0,q1,q2}	{q0,q1,q2}

Cap2: Automatos finitos

- AFND → AFD:



Estado	Entrada	
	0	1

Cap3: Gramáticas regulares

- Uma gramática é um mecanismo para gerar sentenças/palavras de uma Linguagem.
- Uma Gramática é definida por uma quadrupla G={N,T,S,P} onde:
  - N é o conjunto de símbolos não Terminais
  - T é o conjunto de símbolos terminais do alfabeto (Σ) da linguagem
  - S é o símbolo inicial (onde $S \in N$)
  - P é o conjunto de regras de produção que descreve como os não-terminais devem ser derivados.

Cap3: Gramáticas regulares

- Exemplo: G_1 = { {S,B}, {0,1}, S, P}

P:	S	$\rightarrow$	B
	B	$\rightarrow$	0B1
	B	$\rightarrow$	1B0
	B	$\rightarrow$	Λ

- Exercício:
  - Dê 5 sentenças geradas pela gramática.
  - Diga qual a linguagem representada pela

Cap3: Gramáticas regulares

- Pode-se representar produções de forma reduzida unindo-se o lado direito das mesmas por um símbolo "|" (desde que o mesmo não faça parte da linguagem).
  - Ex: P: S $\rightarrow$ B | 0B1 | 1B0 | ε é equivalente as 4 produções precedentes.

Cap3: Gramáticas regulares

- Uma gramática é dita regular se as suas produções são do tipo: $\alpha \rightarrow w\beta$ e $\alpha \rightarrow w$, onde α e $\beta \in N$ $w \in T^*$. (linear à direita)
- Alternativamente uma gramática regular pode ser definida por produções do tipo: $\alpha \rightarrow \beta w$ e $\alpha \rightarrow w$, onde α e $\beta \in N$ $w \in T^*$. (linear à esquerda)
- Gramáticas regulares geram linguagens regulares que podem ser representadas também por expressões regulares e autômatos finitos.

Cap3: Gramáticas regulares

- Exercício: Quais das gramáticas abaixo é regular?

1-	S	$\rightarrow$	B
	B	$\rightarrow$	0B1
	B	$\rightarrow$	1B0
	B	$\rightarrow$	ε
2-	S	$\rightarrow$	0B
	B	$\rightarrow$	1B
	B	$\rightarrow$	0
	B	$\rightarrow$	1

Cap3: Gramáticas regulares

- Obs.:
  - Uma gramática não regular pode representar uma linguagem regular
  - Neste caso existe uma gramática regular que também representa a mesma linguagem.

Cap3: Gramáticas regulares

- Derivação gramatical
  - Uma gramática deriva sentenças a partir do seu símbolo inicial e repetidamente substituindo não-terminais pelo corpo de sua produção
  - Todas as possíveis sequências de terminais (sentenças) produzidas desta forma formam a linguagem definida pela gramática
  - Exemplos para a gramática 2 do slide anterior: 0, 1, 00, 01, 10, 11, ...

Cap3: Gramáticas regulares

- Exemplo de derivação:
  - Dada a gramática abaixo, derive a sentença:
    - id – 5 + 7.3

	E	→	I | I O E
	I	→	id | N
	O	→	+ | - | * | /
	N	→	D | D.D
	D	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | DD

Cap3: Gramáticas regulares

	E	⇒	I O E ⇒ I – E ⇒ I – I O E ⇒ I – I + E ⇒ I – I + I
		⇒	id - I + I ⇒ id – N + I ⇒ id – N + N ⇒ id – D + N
		⇒	id – 5 + N ⇒ id – 5 + D.D ⇒ id – 5 + 7.D
		⇒	id – 5 + 7.3

Cap3: Gramáticas regulares

- Derivação mais à esquerda
  - Quando numa derivação o símbolo não terminal derivado é sempre o não terminal mais à esquerda a derivação é chamada de derivação mais à esquerda.
  - Equivalentemente existe a derivação mais à direita

Cap3: Gramáticas regulares

E	$\Rightarrow$	I O E $\Rightarrow$ I – E $\Rightarrow$ I – I O E $\Rightarrow$ I – I + E $\Rightarrow$ I – I + I
	$\Rightarrow$	id - I + I $\Rightarrow$ id – N + I $\Rightarrow$ id – N + N $\Rightarrow$ id – D + N
	$\Rightarrow$	id – 5 + N $\Rightarrow$ id – 5 + D.D $\Rightarrow$ id – 5 + 7.D
	$\Rightarrow$	id – 5 + 7.3

Derivação mais à esquerda:

E	$\Rightarrow$	I O E $\Rightarrow$ id O E $\Rightarrow$ id – E $\Rightarrow$ id – I O E $\Rightarrow$ id – N O E
	$\Rightarrow$	id - D O E $\Rightarrow$ id – 5 O E $\Rightarrow$ id – 5 + E $\Rightarrow$ id – 5 + I
	$\Rightarrow$	id – 5 + N $\Rightarrow$ id – 5 + D.D $\Rightarrow$ id – 5 + 7.D
	$\Rightarrow$	id – 5 + 7.3

Cap3: Linguagens regulares

- Def: Linguagens Regulares são linguagens que podem ser definidas por uma Expressão Regular (ou Autômato Finito ou Gramática Regular)
- Teoremas:
  - Se L_1 e L_2 são LR então:
    - $L_1 + L_2$ é LR
    - L_1L_2 é LR
    - L_1^* é LR

Trabalho 1

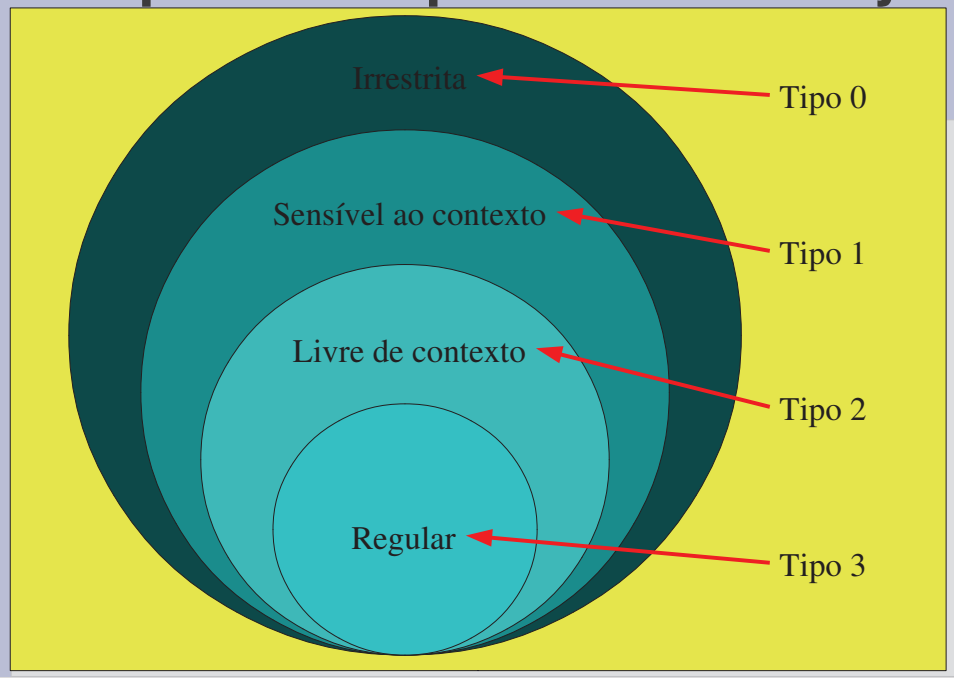
- Entrega 07/05/2012
- Grupos de até 3.
- 1- Faça autômatos determinísticos para as linguagens abaixo
  - Linguagens L1 de {0,1,2,3,4,5,6,7,8,9,'-','+'} que representa números inteiros. Sinal é opcional e tem que ter pelo menos 1 digito
  - Linguagens L2 de {0,1,2,3,4,5,6,7,8,9,'-','+','.',E} que representa números em ponto-flutuante. Sinal é opcional, tem que ter pelo menos 1 digito depois da virgula e se tiver expoente este deve vir logo após o número.

Trabalho 1

- Linguagem L3 de {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} dos números em hexadecimal.

- 4- Implemente e teste L3 no editor de autômatos JFlap

Cap8:Hierarquia de Chomsky



Cap4: Gramáticas Livres de Contexto

- Uma das principais aplicações do estudo de Linguagens é a criação de compiladores para linguagens de programação.
- Linguagens de programação não são linguagens regulares e portanto necessitam de uma forma de representação mais poderosa que gramáticas regulares (ou autômatos finitos ou expressões regulares)
- Obs.: Mesmo gramáticas livres de contexto não são suficientes para representar **completamente** uma Linguagem de Programação.

Cap4: Gramáticas Livres de Contexto

- Uma gramática é dita livre de contexto (GLC ou CFG) se as suas produções são do tipo: $\alpha \rightarrow \beta$, onde $\alpha \in N$ e $\beta \in (N \cup T)^*$.

– Exemplo:

List	$\rightarrow$	List + Digit
List	$\rightarrow$	List - Digit
List	$\rightarrow$	Digit
Digit	$\rightarrow$	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

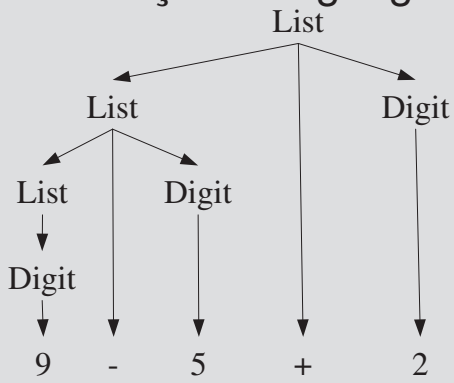
Cap4: Gramáticas Livres de Contexto

- Derivação: mostra como o símbolo inicial da gramática deriva uma sentença da linguagem.
  - Ex: 9 – 5 + 2

	List	⇒	List + Digit ⇒ List – Digit + Digit
		⇒	Digit - Digit + Digit ⇒ 9 - Digit + Digit
		⇒	9 - 5 + Digit ⇒ 9 – 5 + 2

Cap4: Gramáticas Livres de Contexto

- Árvore de derivação: mostra graficamente como o símbolo inicial da gramática deriva uma sentença da linguagem.



Cap4: Gramáticas Livres de Contexto

- Uma árvore de derivação gera uma única sentença, mas uma sentença pode possuir várias árvores de derivações diferentes.
- Se uma sentença tem mais de uma árvore de derivação mais à esquerda a gramatica/linguagem da sentença é ambígua.
- Solução: Usar sempre linguagens não ambíguas ou usar informações adicionais para desambiguá-las
- Tipo de informação capaz de retirar ambiguidade de uma linguagem/gramática.
  - Precedência de operadores
  - Associatividade de operadores

Cap4: Gramáticas Livres de Contexto

- Exercícios:
  - Dê GLC para as seguintes linguagens.
    - Palindromes sobre o alfabeto {a,b}
    - O conjunto de todos os strings sobre o alfabeto {a,b} com duas vezes mais as que bês.
  - Diga se a gramática abaixo é ambígua

    S → aB | bA

    A → a | aS | bAA

    B → b | bS | aBB|
  - Ache a forma normal de Chomsky para a GLC abaixo
    - S → ~S | [S > S] | p | q

Cap5: Autômato de Pilha

- Autômato de pilha AP (PDA - Pushdown automata) são máquinas que reconhecem palavras de Linguagens Livres de Contexto.
- Um AP é semelhante a um AF com estados e transições, mas com modificações.
  - Uma fita de entrada
  - Uma pilha
  - Um alfabeto da pilha
  - As operações de transições editam a pilha

Cap5: Autômato de Pilha

- A fita de entrada:
  - Composta de infinitas células, cada uma contendo um dos símbolos do alfabeto ou branco (Δ)
  - Limitada à esquerda e infinita à direita
  - Cada célula é lida uma só vez.
  - É assumido que após a primeira célula em branco a fita está em branco

Cap5: Autômato de Pilha

- A pilha:
  - É uma pilha, ou seja, a entrada e saída de elementos se dá sempre pelo topo
  - Capaz de armazenar um número infinito de elementos
  - Inicialmente cada elemento da pilha contém branco
  - Cada elemento armazenado na pilha deve pertencer ao alfabeto da pilha que pode ser igual ou diferente do alfabeto da linguagem

Cap5: Autômato de Pilha

- Definição: Um Autômato de Pilha (AP) é uma tupla de sete elementos <Q, Σ, Γ, I, F, P>:
  - Q : Conjunto finito de estados
  - Σ : Alfabeto finito da fita de entrada
  - Γ : Alfabeto finito da Pilha
  - I: Estado inicial
  - F: Conjunto de estados finais ($F \subseteq Q$)
  - P: Conjunto finito de transições

Cap5: Autômato de Pilha

- P: Conjunto finito de transições
  - As operações são representadas por transições de um conjunto (de estado, símbolo de entrada, e símbolo no topo da pilha) para um novo conjunto de estados com a inserção de zero ou mais símbolos na pilha

Cap5: Autômato de Pilha

- Algoritmo de execução:
  - Começa no estado inicial
  - Executa as operações de transição enquanto altera a pilha
  - Se não for possível continuar, rejeita a sentença
  - Se chegar a um estado final, aceita a sentença

Cap5: Autômato de Pilha

- Exercício:
  - L = {(a+b)* | o número de as e bês é igual}

$\delta(q0,a,\Delta) = \{(q0,A)\}$
$\delta(q0,b,\Delta) = \{(q0,B)\}$
$\delta(q0,a,A) = \{(q0,AA)\}$
$\delta(q0,b,B) = \{(q0,BB)\}$
$\delta(q0,a,B) = \{(q0,\Lambda)\}$
$\delta(q0,b,A) = \{(q0,\Lambda)\}$
$\delta(q0,\Delta,\Delta) = \{(q1,\Lambda)\}$

- Estado inicial: q0
- Estado final: q1

Cap5: Autômato de Pilha

- Determinismo x Não determinismo:
  - Um PDA é dito não determinista quando mais de uma operação pode ser feita para o mesmo conjunto de estado, símbolo de entrada e símbolo no topo da pilha.
  - Diferente de autômatos finitos os autômatos de pilha deterministas e não deterministas não representam o mesmo conjunto de linguagens.
  - O Conjunto das linguagens livres de contexto é representado por automatos de pilha não deterministas. Ex: $\{ww^{-1} \mid w \in \{0,1\}^*\}$

Cap5: Autômato de Pilha

- Exercícios:
  - Faça um PDA para a gramática abaixo:

    $S \rightarrow aAA$

    $A \rightarrow aS \mid bS \mid a$
  - Dê a gramática para a linguagem aceita pelo PDA abaixo:

    Inicial:q0, final: q2

    $\delta(q0,a,\Delta) = \{(q0,A)\}$ $\quad \delta(q1,b,A) = \{(q1,\Lambda)\}$

    $\delta(q0,a,A) = \{(q0,AA)\}$ $\quad \delta(q1,\Delta,A) = \{(q2,\Lambda)\}$

    $\delta(q0,b,A) = \{(q1,\Lambda)\}$ $\quad \delta(q2,\Delta,\Delta) = \{(q2,\Lambda)\}$

Cap5: Autômato de Pilha

- Exercícios:
  - Faça um PDA para a aceitar as linguagens abaixo:
    - Conjunto de todos os strings de {a,b} que não tenham a forma ww.
    - $\left\{a^i b^{2i} \middle| i \geq 1\right\}$
    - $\left\{wcw \middle| w \in \{a,b\}^*\right\}$
    - $\left\{aa^* b^i c^j \middle| i \geq 1 \wedge j \geq i\right\}$

Cap5: Autômato de Pilha

- Exercícios:
  - Faça um PDA para a aceitar as linguagens abaixo:
    - Conjunto de todos os strings de {a,b} que não tenham a forma ww.
    - $\left\{a^i b^{2i} \middle| i \geq 1\right\}$
    - $\left\{wcw \middle| w \in \{a,b\}^*\right\}$
    - $\left\{aa^* b^i c^j \middle| i \geq 1 \wedge j \geq i\right\}$

Cap4: Gramáticas Livres de Contexto

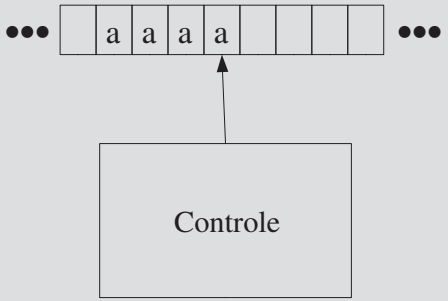
- Uma das principais aplicações do estudo de Linguagens é a criação de compiladores para linguagens de programação.
- Linguagens de programação não são linguagens regulares e portanto necessitam de uma forma de representação mais poderosa que gramáticas regulares (ou autômatos finitos ou expressões regulares)
- Obs.: Mesmo gramáticas livres de contexto não são suficientes para representar **completamente** uma Linguagem de Programação.

Cap6:Ling. Sensíveis ao contexto

- Linguagens sensíveis ao contexto são linguagens cujas sentenças exibem dependência entre trechos distintos das mesmas.
  - No entanto é importante notar que algumas LLC também possuem dependência entre trechos distintos. Portanto, LSC são aquelas com dependência e que não podem ser representadas por GLC

Cap7:Máquinas de Turing

- Máquinas de Turing com fita limitada (ou Linear Bounded Automata – LBA) são reconhecedores de Linguagens sensíveis ao contexto.
- Turing Machine Ilimitada



Cap7:Máquinas de Turing

- Máquinas de Turing:
  - $TM = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Q: Conjunto finito de Estados
  - Σ: Conjunto de símbolos de entrada
  - Γ: Conjunto de símbolos da fita ($\Sigma \supseteq \Gamma$)
  - δ: Função de transição $(Q, \Gamma) \rightarrow (Q, \Gamma, \{E, D\})$
  - q_0: Estado inicial
  - F: Conjunto de estados finais

Cap7:Máquinas de Turing

- Na Máquina de Turing ilimitada é assumido que as posições da fita que não contém símbolos de entrada contém um simbolo especial, normalmente branco (representaremos este símbolo por Δ)

Cap7:Máquinas de Turing

- Função de transição
  $\delta(q, X)=(p, Y, M)$
  $q \, e \, p \in Q$
  $X \, e \, Y \in \Gamma$
  $M \in \{E, D\}$
- Interpretação:
  – Dado que a máquina está no estado q e tem sob o cabeçote da fita o símbolo X, passa para o estado p escreve o símbolo Y na fita e move o cabeçote de leitura para a esquerda (E) ou para a direita (D)

Cap7: Maquina de Turing

- Editor de Automatos: Jflap
- Exercício:
  – Faça uma MT que ache o complemento de 2 de ur número
  – Implemente-o usando JFlap

Cap7:Máquinas de Turing

- O modelo de maquinas de Turing permite uma serie de variações, no entanto nenhuma destas variações aumenta o poder de computação do modelo original
- Variações de Maquinas de Turing:
  - Transição sem movimentação: Podemos incrementar a maquina de Turing para permitir que o cabecote de leitura que parado sobre uma celula após a escrita. Isto pode ser facilmente simulado por uma MT normal. Como?
  - Multi-fita: Aparentemente isto daria um maior poder de computação, mas uma MT multi-ta pode ser simulada por uma MT normal. Como?

Cap7:Máquinas de Turing

- Variações de Maquinas de Turing:
  - Fita infinita para ambos os lados: Pode ser simulada facilmente por uma máquina de duas fitas.
  - Não Determinismo: O não determinismo, no caso de MT, não aumenta o poder de representação do modelo, porque o não determinismo pode ser representado por uma MT determinista as alternativas como ramos de uma arvore e tentando todas as alternativas ate achar a solução.
  - Fita Multidimensional: Neste caso em vez de uma fita de entrada tem-se um array de entrada. Pode ser simulada por uma máquina multi-fita.

Cap7:Máquinas de Turing

- Variações de Maquinas de Turing:
  - Multi-cabecote: Vários cabeçotes movendo-se sobre a mesma fita.
  - Off-line: E uma MT multi-fita onde a fita de entrada é somente de leitura. Gravações sõo feitas apenas sobre as outras fitas. Normalmente a fita de entrada é delimitada entre ¢ e $. O cabeçote de leitura não pode passar destes símbolos.

Cap7:Máquinas de Turing

- Linguagem representada por uma máquina de Turing: linguagem recursivamente enumerável (RE)
  - Linguagem para as quais os strings podem ser enumerados por uma máquina de Turing
  - Nestas linguagens uma máquina de Turing sempre para se a sentença pertence a linguagem, mas nem sempre para se a sentença não pertence a linguagem
  - Portanto enquanto a MT não para não é possível afirmar se a sentença pertence ou não a linguagem

Cap7:Máquinas de Turing

- Linguagem recursivas: Nestas linguagens uma máquina de Turing sempre para, não importando se a sentença pertence ou não a linguagem

Cap7:Hierarquia de Chomsky

Chomsky hierarchy	Grammar	Language	Machine/Automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
—	—	Recursive	—
Type-1	Context-sensitive	Context-sensitive	Linear bounded automaton
—	Indexed	Indexed	Nested stack automaton
—	Tree-adjoining	Mildly context-sensitive	Embedded pushdown automaton
Type-2	Context-free	Context-free	Pushdown automaton
—	Deterministic context-free	Deterministic context-free	Deterministic pushdown automaton
—	Visibly pushdown	Visibly pushdown	Visibly pushdown automaton
Type-3	Regular	Regular	Finite automaton
—	—	Star-free language	Aperiodic finite state automaton