

FUNDAMENTOS TEÓRICOS DA COMPUTAÇÃO

--- MT LINEARMENTE LIMITADA ---

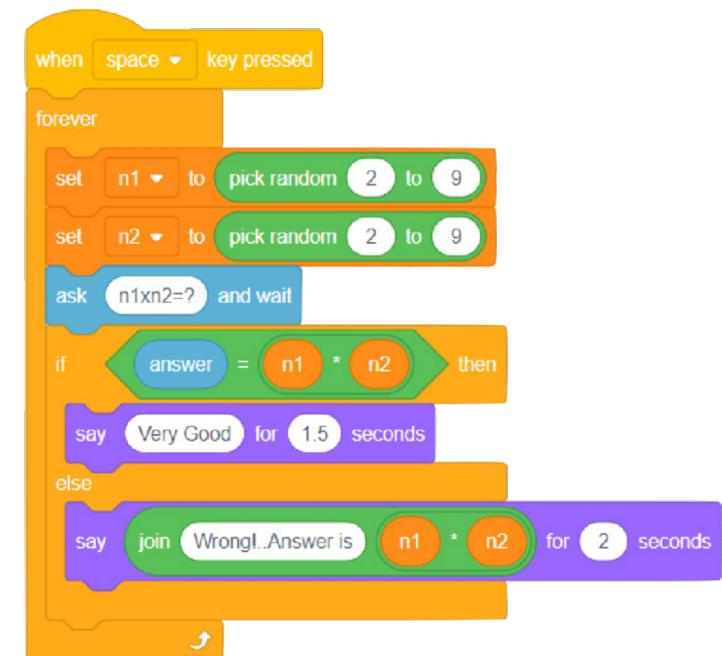
Linguagem Sensível ao Contexto

Linguagens Livre de Contexto

Chomsky introduziu GSC para descrever a sintaxe da **linguagem natural** onde, muitas vezes, uma palavra pode ou não ser apropriada em um determinado lugar, **dependendo do contexto**

Entretanto, é uma questão em aberto **quanto do poder expressivo das GSC** é necessário para capturar a sensibilidade ao contexto encontrada em linguagens naturais

As sintaxes de algumas linguagens de programação visual podem ser descritas por **gramáticas gráficas sensíveis ao contexto**



Linguagens Livre de Contexto

Nas **linguagens livres de contexto** (LLC) se você tiver uma regra, por exemplo, $A \rightarrow \gamma$, isto implica que a variável A pode ser **substituída** por γ , onde γ pode ser qualquer combinação de símbolos terminais e variáveis

$$\begin{array}{c} \dots\dots A \dots\dots \\ \Downarrow \\ \dots\dots \gamma \dots\dots \end{array}$$

É chamada de “**livre**” de contexto porque não importa o que vem antes, nem depois, da variável A , mas sempre é possível substituir A por γ

Linguagens Sensíveis ao Contexto

Entretanto, pode ser útil **somente** permitir que a regra seja aplicada se A estiver **cercada** de certos símbolos (**contexto**)

Às vezes é importante saber o **contexto** onde a variável A ocorre antes de aplicar a regra. É o caso das **linguagens sensíveis ao contexto (LSC)**, que são geradas por uma **gramática sensível ao contexto (GSC)**

De acordo com **Noam Chomsky**, a forma geral de uma GSC é:

$$\alpha A \beta \rightarrow \alpha \omega \beta$$

Onde α e β são o contexto onde A aparece

Importante notar que não é possível alterar o contexto de A

Isto implica que a aplicação da regra só pode ser feita naquele contexto específico, isto é, α e β têm que estar em ambos os lados

Linguagens Sensíveis ao Contexto

Neste caso, α e β podem ser qualquer coisa, incluindo a cadeia vazia (ε), mas é requerido que ω seja não-vazia

Se $\alpha = \beta = \varepsilon$ nós voltamos a definição de gramática livre de contexto. Por isso é que qualquer linguagem livre de contexto é **também** uma linguagem sensível ao contexto ($LLC \rightarrow LSC$)

Entretanto, nem toda linguagem sensível ao contexto é uma linguagem livre de contexto ($LSC \not\rightarrow LLC$)

Como é requerido que ω seja não-vazia, a única forma de gerar a cadeia vazia é através da variável inicial ($S \rightarrow \varepsilon$), desde que S **não apareça** do lado direito de **nenhuma** regra de produção

Linguagens Sensíveis ao Contexto

De acordo com a definição de **Hopcroft & Ullman (1979)**, uma definição alternativa, e equivalente à definição de Chomsky, é

$$\alpha \rightarrow \beta, \text{ onde } |\alpha| \leq |\beta|$$

As produções geram *formas sentenciais* que nunca diminuem

A definição formal é a mesa $G = (V, \Sigma, P, S)$ sendo:

- V é um conjunto de símbolos não terminais (variáveis)
- Σ é um conjunto de símbolos terminais
- P é o conjunto de regras de produção
- $S (S \in V)$ é o símbolo não terminal inicial

A linguagem $L(G) = \{\omega \in \Sigma^* : S \Rightarrow^+ \omega\}$ gerada por G é chamada de **Linguagem Sensível ao Contexto** ou Linguagem Tipo 1 na Hierarquia de Chomsky

Exemplo

$$L_2 = \{a^n b^n c^n : n \geq 0\}$$

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

$$P : \quad S \rightarrow aSBC \mid aBC$$

$$CB \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

É uma gramática sensível a o contexto (Hopcroft & Ullman)

Exemplo

$$L_2 = \{a^n b^n c^n : n \geq 0\}$$

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

$$P : \quad S \rightarrow aSBC \mid aBC$$

$$CB \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

$$\begin{aligned} S &\Rightarrow aSBC \Rightarrow aaBCBC \\ &\Rightarrow aabCBC \Rightarrow aabBCC \\ &\Rightarrow aabbCC \Rightarrow aabbcC \\ &\Rightarrow aabbcc \end{aligned}$$

É uma gramática sensível a o contexto (Hopcroft & Ullman)

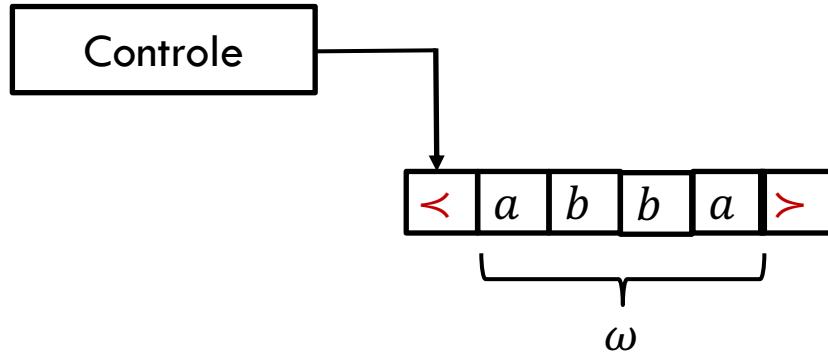
Constraste com a gramática irrestrita para a mesma linguagem dos
dos slides anteriores

Hierarquia de Chomsky

| Hierarquia de Chomsky | Gramática | Linguagem | Reconhecedor |
|-----------------------|----------------------|----------------------------|-------------------------------------|
| Tipo 0 | Irrestrita | Recursivamente Enumeráveis | Máquina de Turing |
| Tipo 1 | Sensível ao Contexto | Sensíveis ao Contexto | Máquina de Turing com Fita Limitada |
| Tipo 2 | Livre de Contexto | Livres de Contexto | Autômato de Pilha |
| Tipo 3 | Linear | Regulares | Autômato Finito |

* Toda Linguagem Sensível ao Contexto é Turing-reconhecível e Turing-decidível

Autômatos Limitados Linearmente



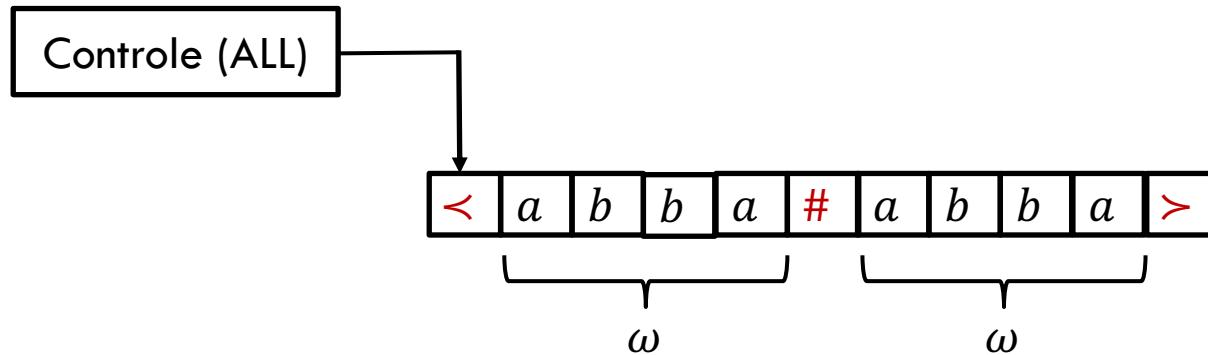
Um **Autômato Limitado Linearmente** (ALL) é uma MT com fita limitada ao tamanho da entrada $|\omega| + 2$

O funcionamento é igual a uma MT

Os dois delimitadores (< e >) são adicionados na fita entre o início e fim da cadeia

O cursor não é permitido mover para fora dos delimitadores

Autômatos Limitados Linearmente



Um ALL só pode resolver problemas que requerem memória proporcional à usada para a entrada
Em outras palavras, é permitido que a memória seja incrementada em no máximo um fator constante linear em $|\omega|$, por exemplo, $2 * |\omega| + 3$

Autômatos Limitados Linearmente

Um ALL é uma 6-upla $M = (Q, \Sigma, \Gamma, s, F, \delta)$, onde:

Q é um conjunto finito de estados

Σ é um alfabeto finito de símbolos (sem o símbolo \sqcup)

Γ é o alfabeto finito de símbolos da fita ($\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$)

$s \in Q$ é o estado inicial

$F \subseteq Q$ é o conjunto dos estados de aceitação

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{E, D\}$ é uma função parcial chamada função de transição, onde E é o movimento para a esquerda e D é o movimento para a direita

* Algumas definições também incluem estados de rejeição

** Nas funções de transição acrescentaremos os delimitadores da fita

Autômatos Limitados Linearmente

δ é a **função de transição redefinida**, agora incluindo o não-determinismo. Entretanto, ainda não foi provado se o não-determinismo aumenta ou não o poder computacional das ALLs

$$Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{E, D\}}$$

$$Q \times \{\langle\} \rightarrow 2^{Q \times \{\langle\} \times \{D\}}$$

$$Q \times \{\rangle\} \rightarrow 2^{Q \times \{\rangle\} \times \{E\}}$$

Os símbolos delimitadores não podem mudar de posição

Uma linguagem L é uma LSC sse L é decidida por um ALL, isto é, podemos ter dois resultados ao processar ω : aceita ou rejeita

MTs que decidem linguagens do Tipo 3, Tipo 2, ou Tipo 1 são, na verdade, ALLs. O livro do **Sipser** mostra um exemplo de linguagem que é decidível, mas não por um ALL

Exemplo

:: Descrição de alto-nível

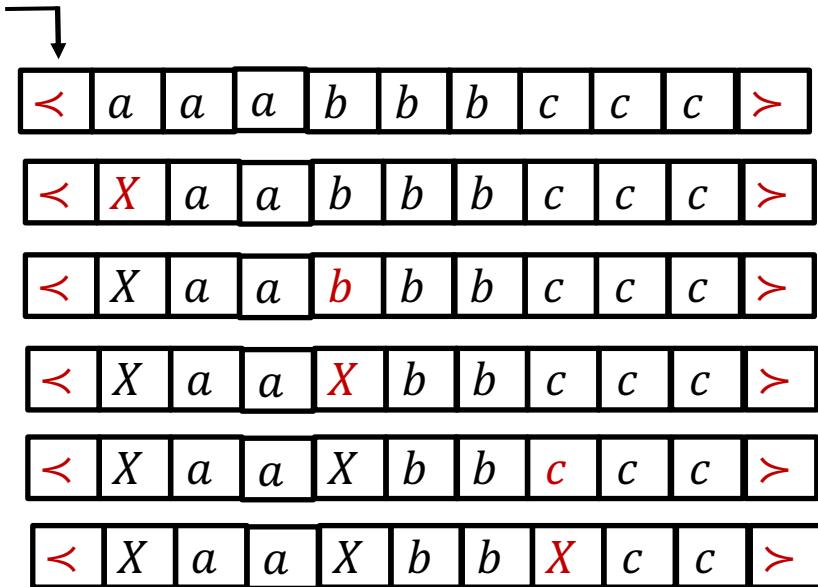
$$L = \{a^n b^n c^n : n \geq 0\}$$

Sobre a cadeia de entrada ω :

1. Mova o cursor para a direita;
2. Marque um a com X e move para a direita, ignorando a's e X's até encontrar um b;
3. Marque um b com X e move para a direita, ignorando b's e X's até encontrar um c;
4. Marque um c com X e move para a esquerda até encontrar <;
5. Mova o cursor para a direita;
6. Enquanto houver X, move o cursor para a direita. Em seguida:
 - a) Se encontrar a vá para o passo 2;
 - b) Se encontrar > aceite a cadeia
 - c) Senão, rejeite a cadeia

Exemplo

:: Descrição de alto-nível



Marque um **a** com X e move para a **direita**

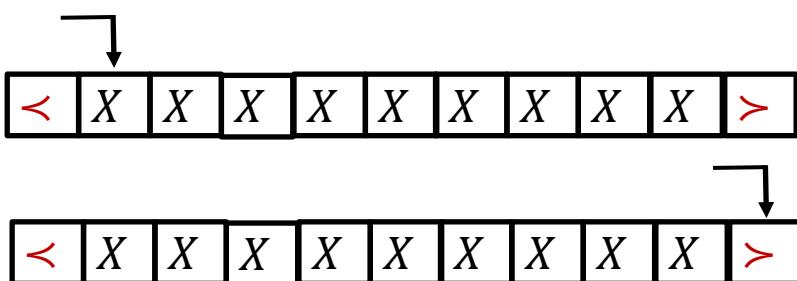
Ignorando a's e X's até encontrar um b

Marque um **b** com X e move para a **direita**

Ignorando b's e X's até encontrar um c

Marque um **c** com X e move para a **esquerda**

...

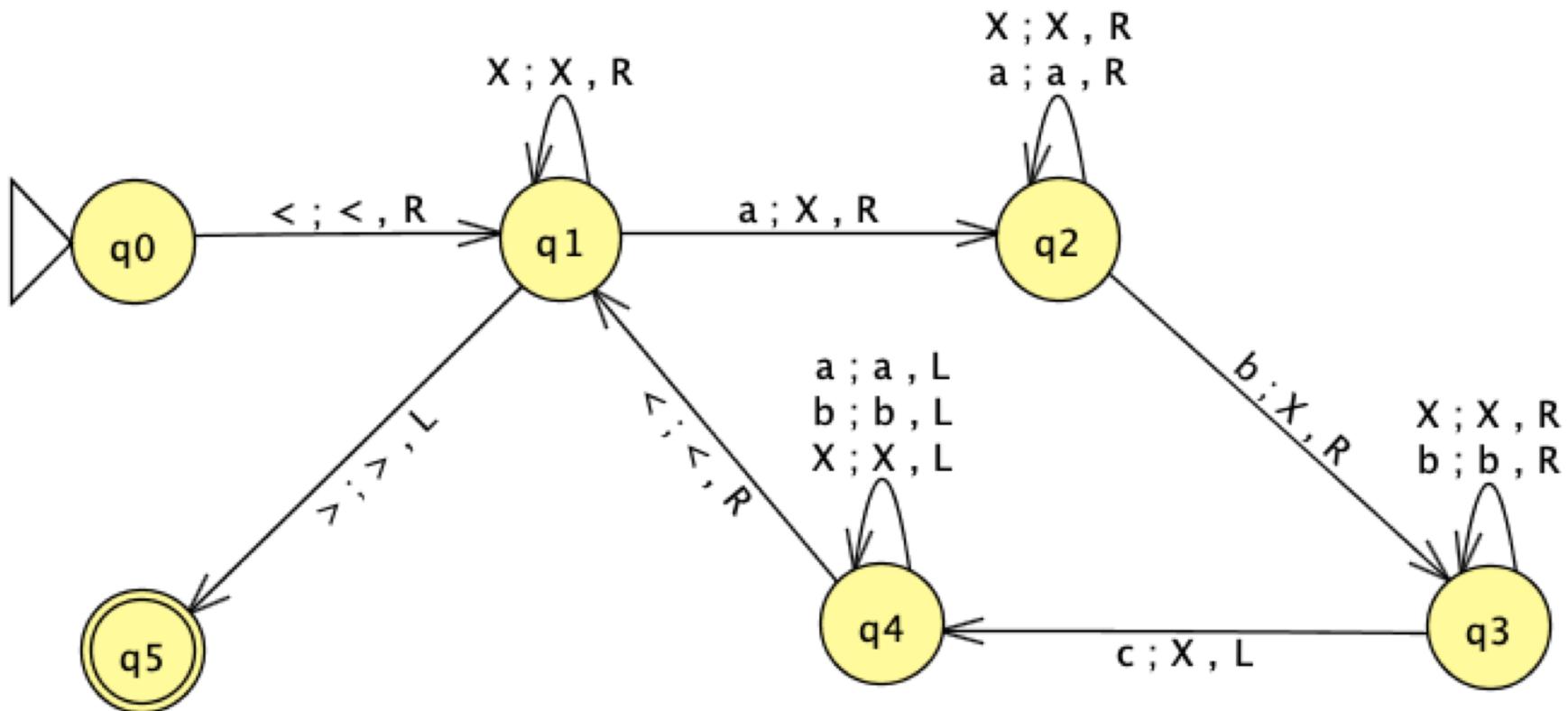


Enquanto houver X, move o cursor para a direita

Se encontrar > aceite a cadeia.

Exemplo

:: Descrição de alto-nível



Exemplo

:: Descrição de alto-nível

JFLAP : (lsc.jff)

File Input Test View Convert Help

Editor Multiple Run

Table Text Size

| Input | Result |
|----------------|--------|
| <> | Accept |
| <abc> | Accept |
| <aabbcc> | Accept |
| <aaabbbccc> | Accept |
| <aaaabbbbcccc> | Accept |
| <a> | Reject |
| | Reject |
| <c> | Reject |
| <ab> | Reject |
| <ac> | Reject |
| <abbc> | Reject |
| <abcc> | Reject |
| <aabbc> | Reject |

Diagram description:

This NFA diagram is defined in JFLAP. It has six states: q0 (start), q1, q2, q3, q4, and q5 (accept). Transitions are as follows:

- From q0 to q1 on input <;<, R.
- From q1 to q2 on input a;X, R.
- From q1 to q4 on input a;a, L; b;b, L; X;X, L.
- From q2 to q3 on input b;X, R.
- From q3 to q4 on input c;X, L.
- From q4 to q1 on input X;X, R.
- From q5 to q1 on input <;>, L.

Buttons at the bottom:

Load Inputs Run Inputs Clear Enter Lambda View Trace



Conte-me mais sobre linguagens
sensíveis ao contexto