# Linguagens Formais e Autômatos

## Aula 23 - Máquinas de Turing restritas

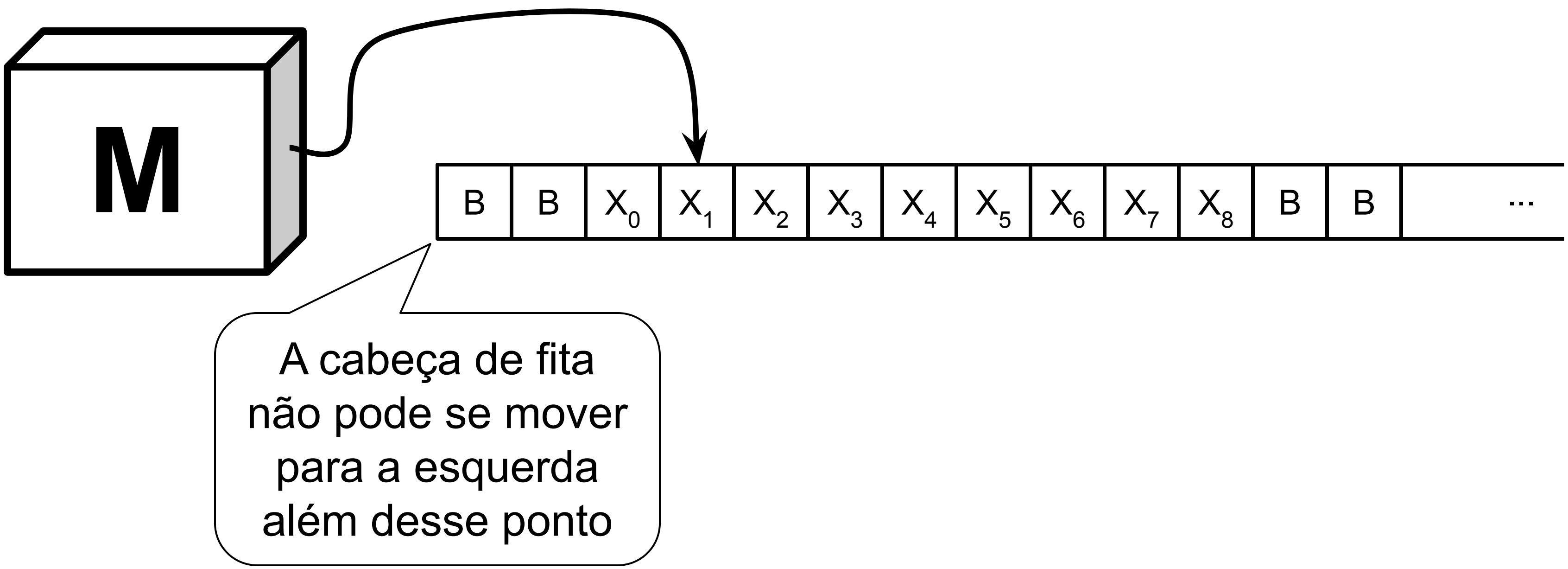
Prof. Dr. Daniel Lucrédio
Departamento de Computação / UFSCar
Última revisão: ago/2015

# Referências bibliográficas


- **Introdução à teoria dos autômatos, linguagens e computação / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman** ; tradução da 2.ed. original de Vandenberg D. de Souza. - Rio de Janeiro : Elsevier, 2002 (Tradução de: Introduction to automata theory, languages, and computation - ISBN 85-352-1072-5)
    - Capítulo 8 - Seção 8.5

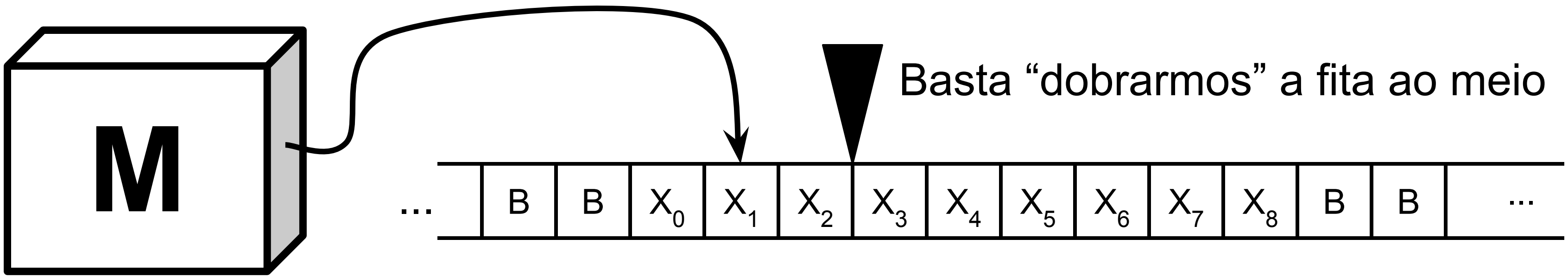
# MT com fita semi-infinita

- Ou seja, a fita tem um início
    - A cabeça não pode se mover além desse ponto
- Veremos que a capacidade de processamento de linguagens é a mesma que as MTs com fita duplamente infinita

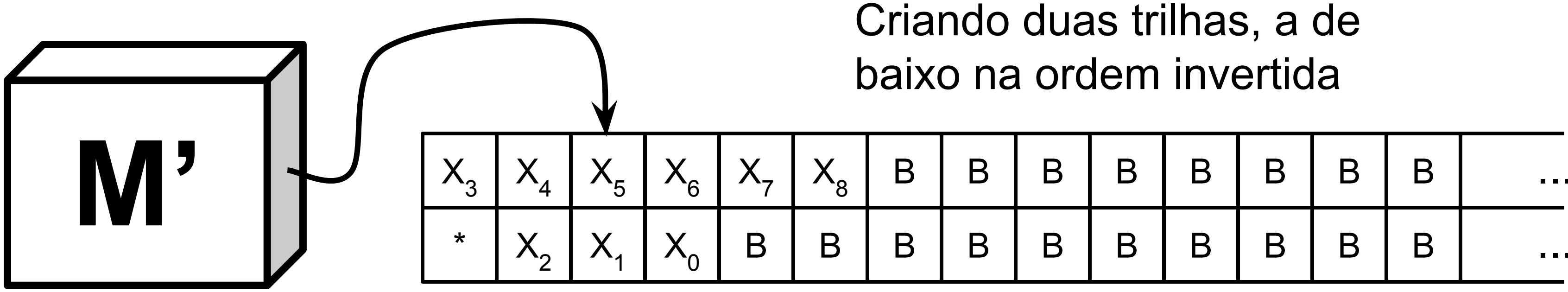


# MT com fita semi-infinita

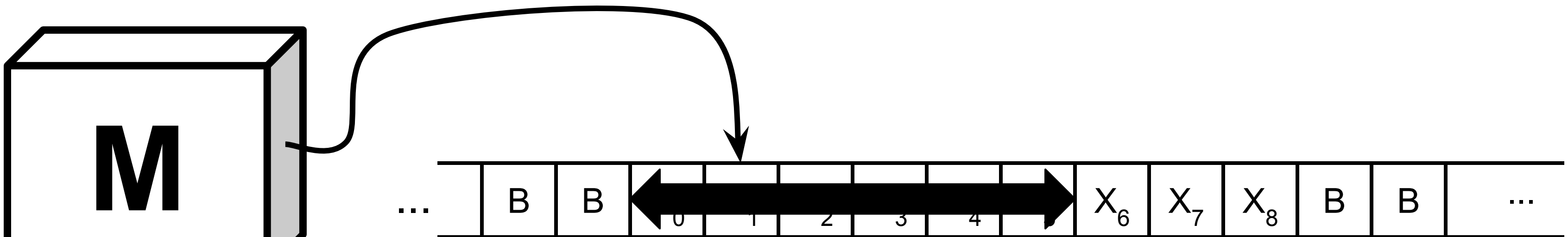
Basta “dobrarmos” a fita ao meio



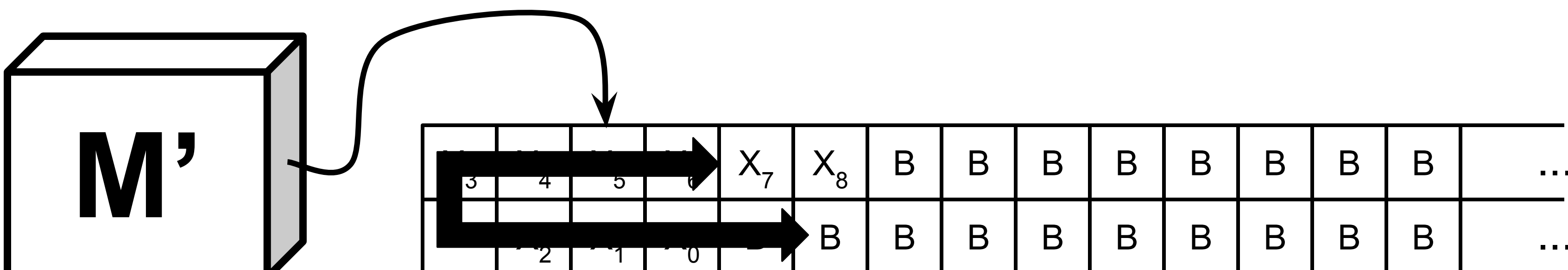

Criando duas trilhas, a de baixo na ordem invertida



# MT com fita semi-infinita




O movimento original da cabeça, da esquerda para a direita, pode ser simulado

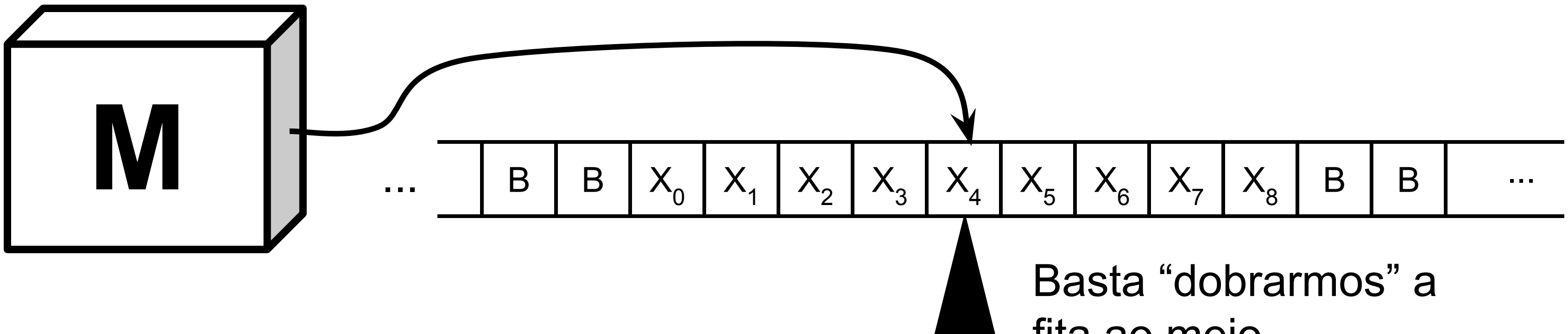



Sempre que o início da fita é alcançado, inverte-se a ordem de movimento, e troca-se de trilha

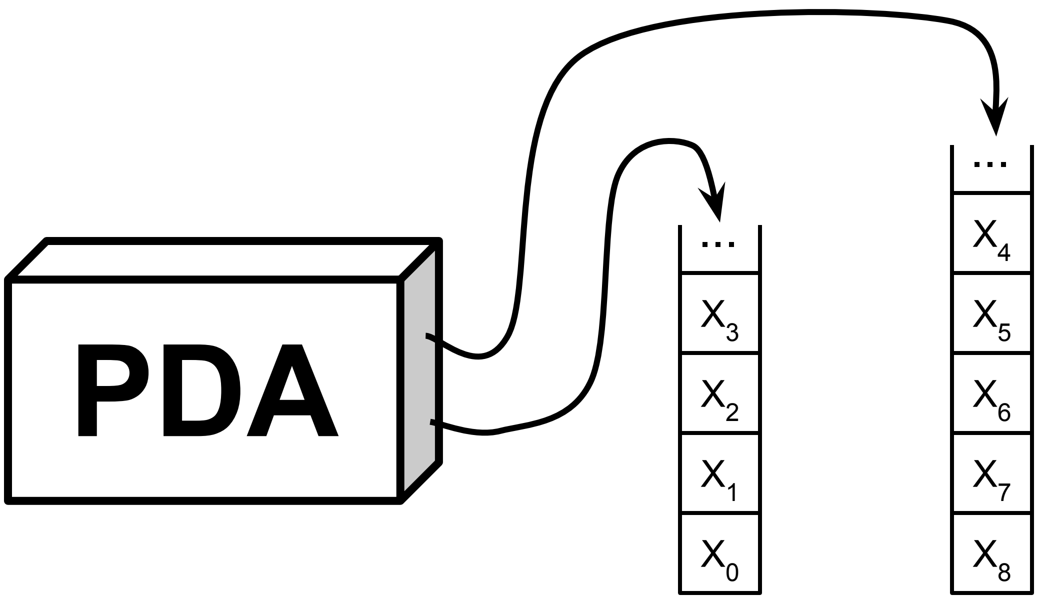
# PDA com duas pilhas

- Um PDA com uma única pilha aceita as linguagens livres de contexto (Tipo-2)
- Mas acrescentando uma segunda pilha, é possível simular completamente uma Máquina de Turing!
  - Passando a aceitar as linguagens recursivamente enumeráveis (Tipo-0)
- Veremos essa construção a seguir

# PDA com duas pilhas



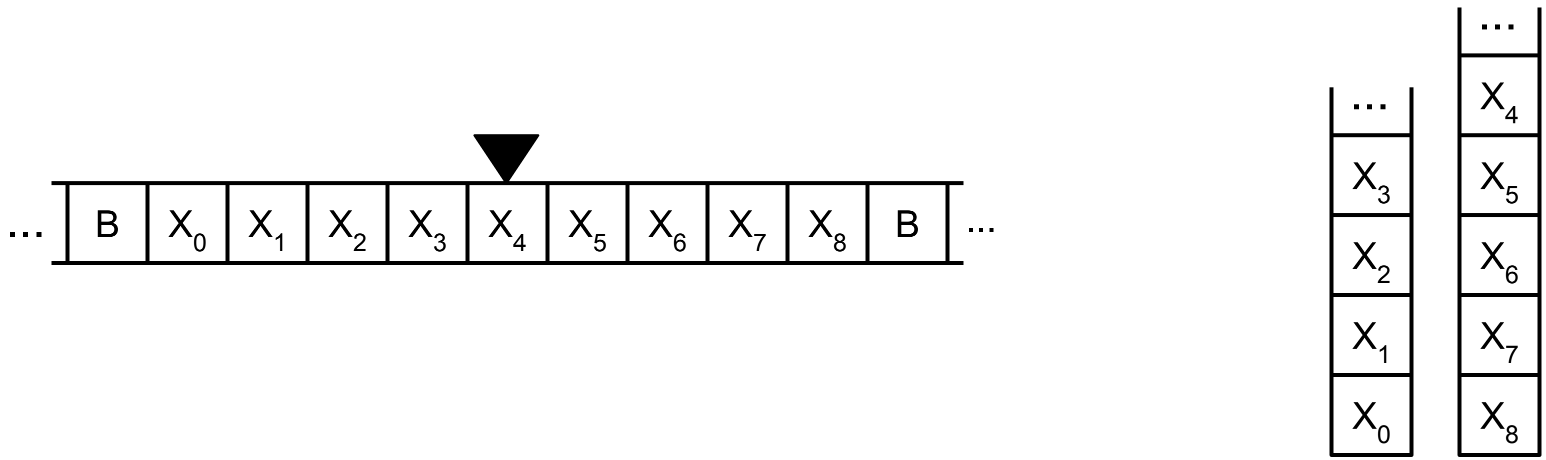

Basta “dobrarmos” a fita ao meio




E criar duas pilhas:
- Uma contendo os símbolos à esquerda da cabeça
- Outra contendo os símbolos à direita da cabeça

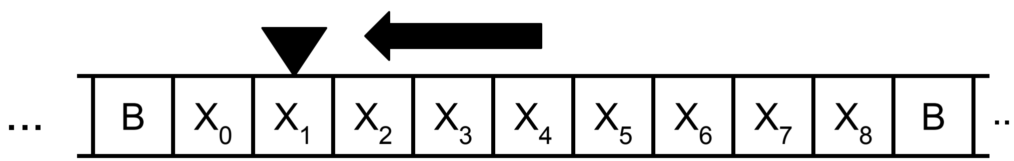
# PDA com duas pilhas

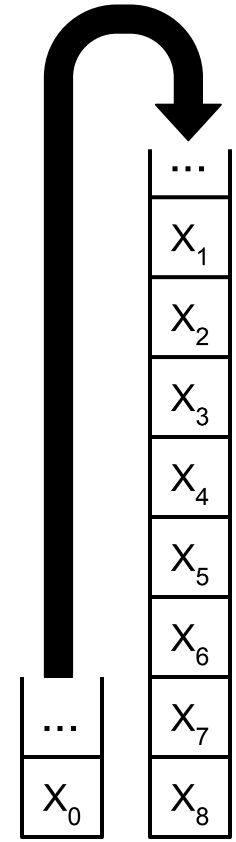
Como simular o movimento da cabeça na fita?



# PDA com duas pilhas

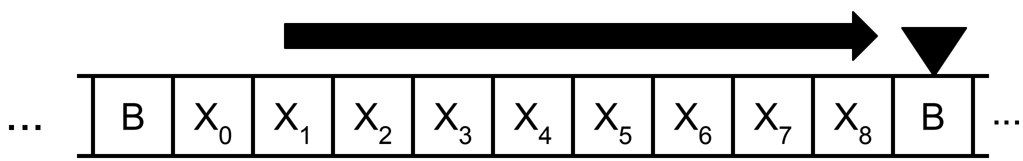
À medida que a cabeça se move para a esquerda, deslocamos os símbolos da pilha da esquerda para a pilha da direita

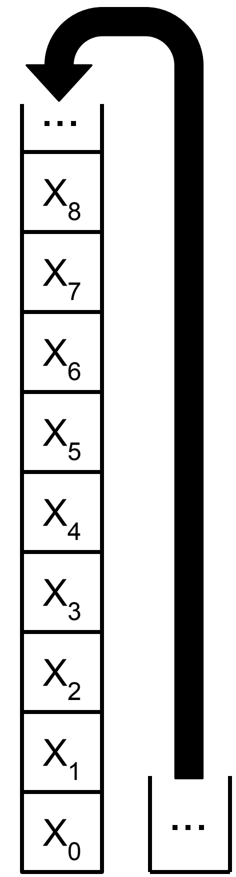


# PDA com duas pilhas

À medida que a cabeça se move para direita, deslocamos os símbolos da pilha da direita para a pilha da esquerda




O fundo das pilhas representa a posição em que os infinitos brancos começam

# Máquinas de Turing e computadores reais

# Máquinas de Turing e computadores reais

- Agora iremos comparar a máquina de Turing e computadores reais, que usamos no dia-a-dia
  - Iremos mostrar que reconhecem as mesmas linguagens: as linguagens recursivamente enumeráveis
- Ou seja:
  - Um computador pode simular uma máquina de Turing
  - Uma máquina de Turing pode simular um computador
    - E mais: pode fazê-lo em um tempo polinomial em relação ao tempo de execução do computador
    - * Lembrando que a noção de tempo = movimentos

# Simulação de uma MT por um computador

- Deve ser bastante óbvio que é relativamente fácil implementar uma MT usando uma linguagem de programação:
  - Uma variável armazena estados
  - Tabela de transições para os movimentos
  - A fita pode ser armazenada em um array
  - Um ponteiro aponta o local da cabeça de leitura
  - Existem, de fato, muitos simuladores de MT disponíveis, o que demonstra essa possibilidade

# Simulação de uma MT por um computador

- Mas existe um aspecto no qual a MT “ganha”
  - Sua fita é infinita!
  - A memória do computador (memória principal, disco rígido, pen drive, etc) é finita!!
- Alguém pode argumentar que é sempre possível implementar um sistema de troca de discos em tempo de execução
  - Sempre que for necessário um novo pedaço de fita, troque o disco mais distante da cabeça por um disco novo
  - Ou seja, assumimos que conseguimos fabricar quantos discos o computador necessita
  - Mas aí estamos assumindo que a quantidade de matéria no Universo é infinita
    - E isso é outra discussão...
- Na prática, podemos confiar que a quantidade de memória não é infinita, mas é suficientemente grande para armazenar dados sobre todos os possíveis problemas que iremos encontrar

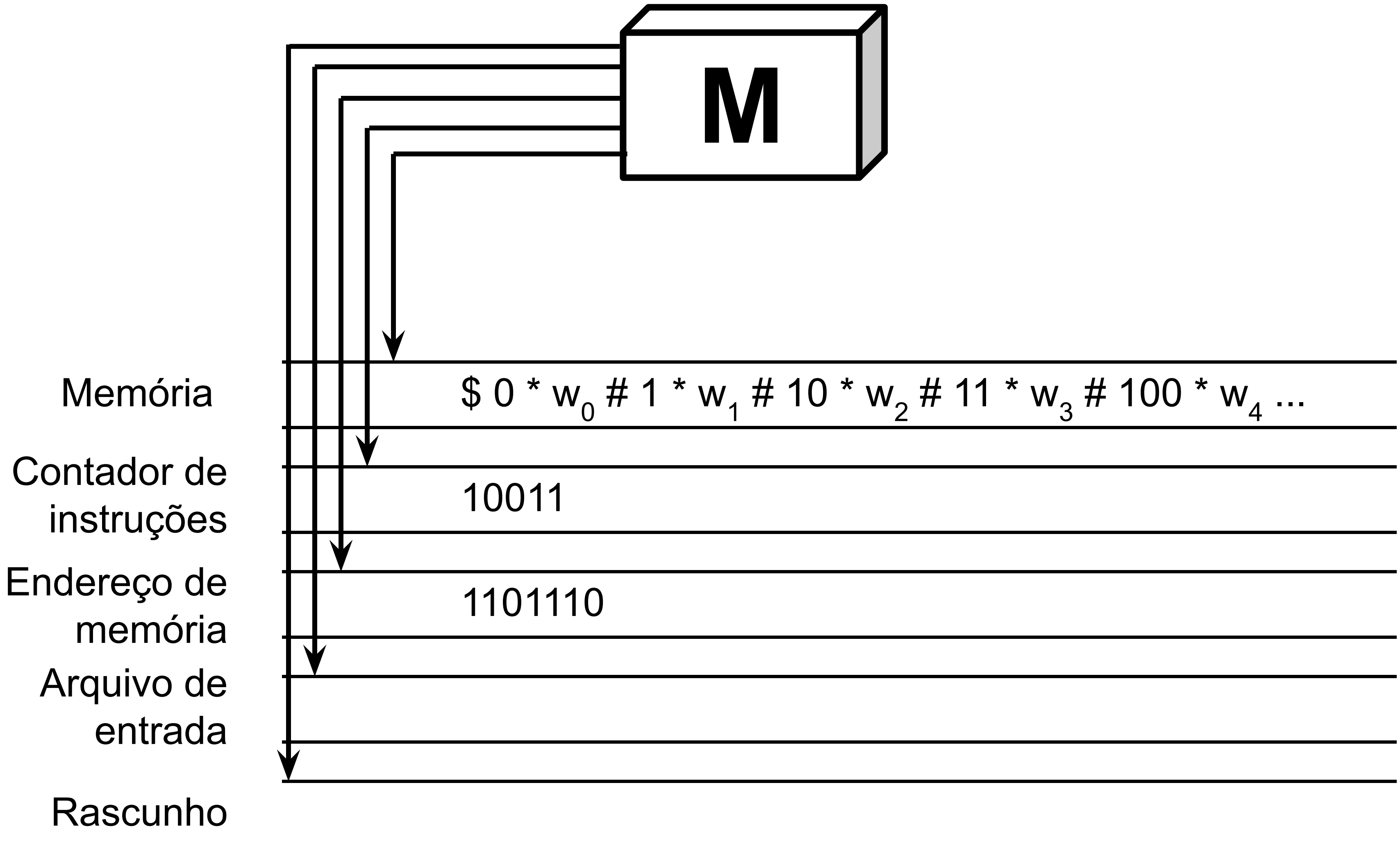
# Simulação de um computador por uma MT

- A demonstração anterior não é tão surpreendente
- Mais interessante é o fato de que uma MT consegue fazer tudo que um computador faz (mais lentamente, claro)
- Iremos demonstrar como é possível simular a execução de um programa de computador em uma MT
- Para isso, vamos primeiro definir um modelo realista de um computador típico

# Modelo de um computador

- Memória
  - Sequência indefinidamente longa de palavras
  - Cada uma com um endereço
- Programa
  - Está armazenado em algumas das palavras
  - Cada palavra representa uma instrução simples
    - Como linguagem de máquina, por exemplo
      - Movimentam uma palavra para outra
      - Adicionam uma palavra a outra
      - Endereçamento indireto (ponteiros)
  - Cada instrução envolve um número limitado (finito) de palavras
  - Cada instrução altera o valor de no máximo uma palavra

# MT que simula um computador



# MT que simula um computador

- A cada instrução:
  - A MT procura na fita 1 o local apontado pelo conteúdo da fita 2
  - A MT interpreta o conteúdo da instrução
  - Se a instrução exigir o valor de um endereço (ponteiro), a MT o copia para a fita 3. Em seguida, ela busca na fita 1 o endereço apontado pela fita 3, e copia seu conteúdo na fita 3
  - A MT então executa a instrução: cópia, soma, “salto”, etc, usando o rascunho opcionalmente como auxílio
  - A MT então incrementa o conteúdo da fita 2 e recomeça o ciclo

# Tempo de simulação

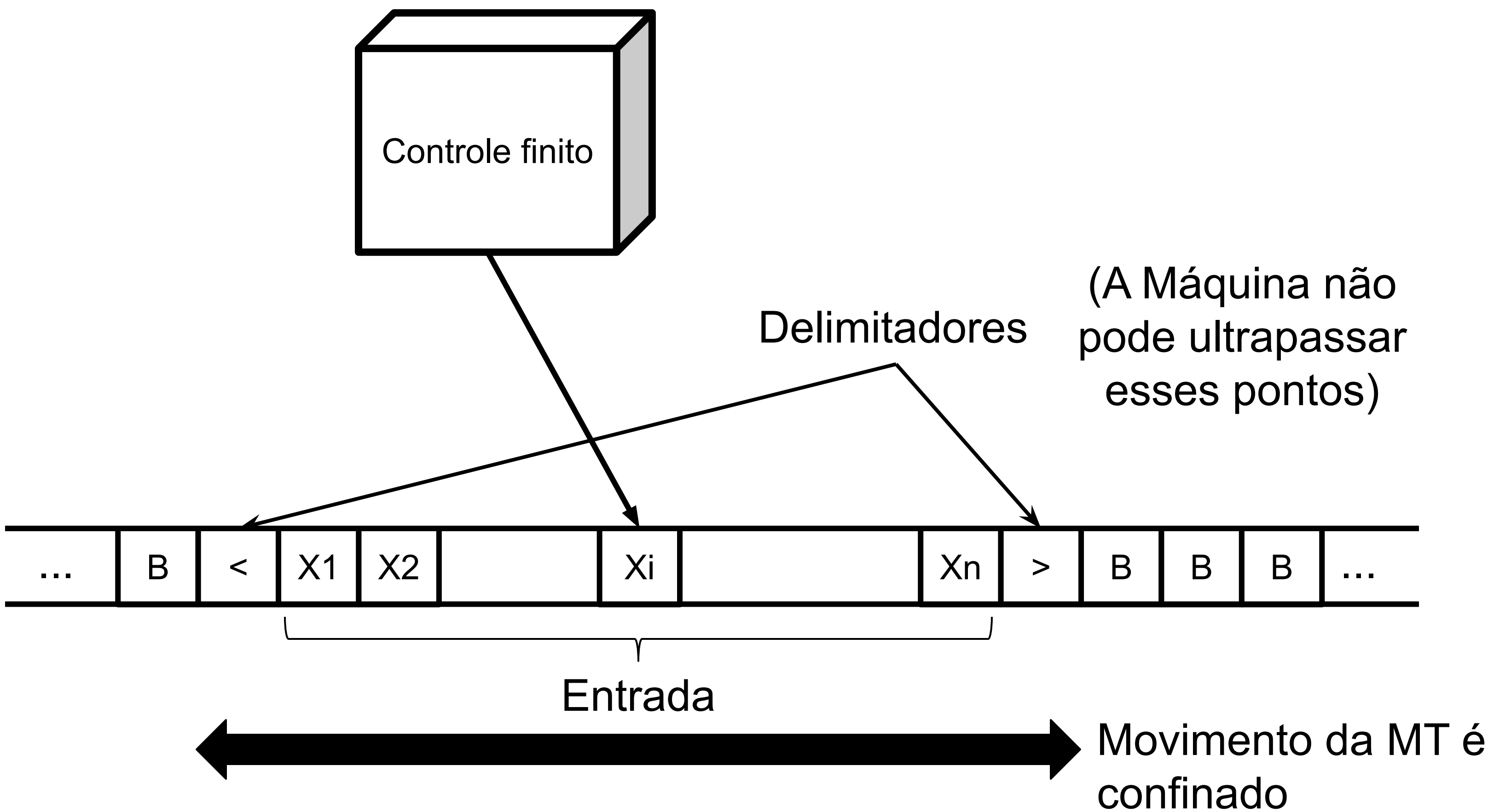
- Estamos em busca de um modelo matemático para analisar a capacidade de um computador em resolver problemas
    - Encontramos um: a MT, que tem a mesma capacidade de reconhecimento que um computador comum
    - Mas ainda falta analisar se ela é eficiente
        - Ou seja, ela executa em tempo “similar”?
        - Ou seja (2), se eu estudar o comportamento e eficiência das MTs para um determinado problema, os estudos serão também válidos para os computadores reais?
- A linha divisória é polinomial X não-polinomial!
    - Polinomial = tratável
    - Não-polinomial (exponencial, por exemplo) = intratável

# Tempo de simulação

- Primeiro. A MT que simula um computador tem várias fitas, ou seja, iremos no mínimo “gastar” um tempo quadrático para converter para uma fita
  - Conforme vimos anteriormente
  - Quadrático = polinômio, então até agora, OK!
- Uma instrução do computador leva no máximo $O(n^2)$ movimentos da MT
  - Incluindo a busca de endereços, execução de instruções, etc
- Portanto: n instruções do computador leva no máximo $O(n^3)$ movimentos da MT
- Combinado à conversão para uma única fita, uma MT de uma fita pode simular n etapas de um computador em no máximo $O(n^6)$ movimentos

# Máquinas de Turing com Fita Limitada

# Máquinas de Turing com Fita Limitada



# Máquinas de Turing com Fita Limitada

- Funcionamento é igual à MT com fita infinita
- Porém existem dois delimitadores, um à esquerda e um à direita
  - Imediatamente ao redor da entrada
- Ou seja, se a entrada tem n símbolos
  - A fita terá n+2 posições
- A MT pode ler/escrever na fita
  - Mas não pode se mover além dos delimitadores

# A linguagem de uma MT com fita limitada

- O conjunto de linguagens aceitas pelas máquinas de Turing com fita limitada é chamado de
    - Linguagens sensíveis ao contexto
- Na hierarquia de Chomsky

| Hierarquia | Gramáticas | Linguagens | Autômato mínimo |
|---|---|---|---|
| Tipo-0 | Recursivamente Enumeráveis | Recursivamente Enumeráveis | Máquinas de Turing |
| **Tipo-1** | **Sensíveis ao contexto** | **Sensíveis ao contexto** | **MT com fita limitada** |
| Tipo-2 | Livres de contexto | Livres de contexto | Autômatos de pilha |
| Tipo-3 | Regulares (Expressões regulares) | Regulares | Autômatos finitos |

# Linguagens sensíveis ao contexto

- Lembre-se da restrição das gramáticas livres de contexto (lema do bombeamento):

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

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

  - Em algumas LP, variáveis precisam ser declaradas antes de serem utilizadas
    - É o mesmo caso da linguagem {ww | w em um alfabeto com mais de um símbolo}
  - Essas linguagens não são livres do contexto

# O que é “livre de contexto”?

- Considere a gramática (livre de contexto) abaixo:
  - E → E + E
  - E → E * E
  - E → E – E
  - E → E / E
  - E → ( E )
  - E → a
  - E → b
- Considere o próximo passo de derivação para a seguinte cadeia:
  - E + E * (E - E)
- Iremos substituir o E mais à esquerda

# O que é “livre de contexto”?

- E + E * (E - E)
- A princípio, é possível usar qualquer regra que comece com E → ...
  - Claro, depende da cadeia de terminais sendo reconhecida
- Mas o importante aqui é: as regras não impõem restrição à substituição!
  - Todas as regras podem ser usadas em qualquer ocorrência de E!
  - Não importa se o E aparece no começo da cadeia, após os parêntesis, depois do +, etc
  - Ou seja, não importa o CONTEXTO, qualquer regra pode ser usada

# O que é “livre de contexto”?

- Suponha agora que queiramos fazer uma restrição adicional:
  - Sempre que aparecer logo após a abertura de parêntesis, uma expressão deve ser uma soma ou uma subtração, ou seja:
    - a + b * (a – b * c) faz parte da linguagem
    - a + b * (a / b + c) não faz parte da linguagem
    - a + b * (a * c) não faz parte da linguagem
- Poderíamos modificar a gramática para acrescentar essa restrição
  - E → E + E
  - E → E * E
  - E → E – E
  - E → E / E
  - E → a
  - E → b
  - E → ( E’ )
  - E’ → E + E | E - E

Exemplo de dependência do contexto. Quando abre parêntesis, E “muda” um pouquniho

# O que é “livre de contexto”?

- Suponha agora que queiramos fazer outras restrições:
  - Sempre que aparecer logo após o sinal de +, uma expressão não pode ser soma
  - Sempre que aparecer antes de fechar o parêntesis, uma expressão deve ser uma divisão
  - Etc...
- Poderíamos modificar a gramática para acrescentar essas restrições
  - Iríamos criar regras adicionais, replicando símbolos para cobrir todos os casos
- Mas existem casos onde não é possível adicionar essa restrição em uma CFG
  - Justamente aqueles cobertos pelo lema do bombeamento para linguagens livres de contexto

# Linguagens livres de contexto

- Ex: $\{a^n b^n c^n \mid n \geq 1\}$
- Nesses casos, é necessário adicionar sensibilidade ao contexto à gramática de forma explícita
- Uma gramática sensível ao contexto é similar a uma gramática livre de contexto, porém as regras obedecem ao formato $\alpha \rightarrow \beta$, onde:
  - $\alpha$ pode ter qualquer quantidade de símbolos terminais e/ou não-terminais
    - Mas pelo menos um não-terminal
  - $\beta$ é composto de terminais e/ou não-terminais (assim como nas CFGs)
  - $|\beta| \geq |\alpha|$ ou seja, o lado direito possui uma quantidade de símbolos não-inferior à do lado esquerdo da mesma regra

# Gramáticas sensíveis ao contexto

- Ex:
  - $E \rightarrow E + E$
  - $E \rightarrow E * E$
  - $E \rightarrow E – E$
  - $E \rightarrow E / E$
  - $E \rightarrow ( E )$
  - $E \rightarrow a$
  - $E \rightarrow b$
  - $(E \rightarrow E + E$
  - $(E \rightarrow E - E$

# Gramáticas sensíveis ao contexto

- Outro exemplo:
  - $S \rightarrow aSBC$
  - $S \rightarrow aBC$
  - $CB \rightarrow BC$
  - $aB \rightarrow ab$
  - $bB \rightarrow bb$
  - $bC \rightarrow bc$
  - $cC \rightarrow cc$
- Faça algumas derivações e veja o resultado:
  - $\{a^n b^n c^n \mid n \geq 1\}$

# Gramáticas sensíveis ao contexto

- Outro exemplo:
  - $S \rightarrow aSBC$
  - $S \rightarrow aBC$
  - $CB \rightarrow BC$
  - $aB \rightarrow ab$
  - $bB \rightarrow bb$
  - $bC \rightarrow bc$
  - $cC \rightarrow cc$

Recursão simples, do centro para fora. Produz formas sentenciais onde para cada “a” existe um “BC”: aaBCBC, aaaaBCBCBCBC

“Inverte” os BCs, colocando-os em ordem, mas sem alterar sua quantidade: aaaBCBCBC ⇒ aaaBBCCBC ⇒ aaaBBCBCC ⇒ aaaBBBCCC

Adiciona os terminais, mas somente se estiverem na ordem correta!

# Gramáticas sensíveis ao contexto

- São ditas monotônicas
  - O comprimento das formas sentenciais durante o processo de derivação nunca diminui
  - Devido à restrição $|\beta| \geq |\alpha|$
  - Consequência imediata:
    - não existe regra do tipo $S \rightarrow \varepsilon$
    - Ou seja, estritamente falando, a cadeia vazia não faz parte de nenhuma linguagem sensível ao contexto
    - Mas essa restrição normalmente é desconsiderada na prática
- Algumas gramáticas sensíveis ao contexto podem ser convertidas em gramáticas livres de contexto
  - Aquelas que não podem são gramáticas estritamente sensíveis ao contexto
  - Exemplo anterior

# Gramáticas sensíveis ao contexto

- Resumindo
  - São gramáticas mais genéricas do que as livres de contexto
  - Denotam as linguagens livres de contexto
  - São reconhecidas por Máquinas de Turing com fita limitada
  - Toda gramática livre de contexto é sensível ao contexto
    - Desconsiderando o caso da cadeia vazia, obviamente

# Finalizando

- E se “relaxarmos” as restrições das gramáticas sensíveis de contexto?
- Ou seja, as regras obedecem ao formato $\alpha \rightarrow \beta$, onde:
  - $\alpha$ pode ter qualquer quantidade de símbolos terminais e/ou não-terminais
  - $\beta$ pode ter qualquer quantidade de símbolos terminais e/ou não-terminais
- Essas gramáticas são chamadas de gramáticas irrestritas
  - Ou gramáticas recursivamente enumeráveis
  - Que são as gramáticas Tipo-0

# Finalizando

- Exemplo:
  - $S \rightarrow aAbcC$
  - $bc \rightarrow B$
  - $ABC \rightarrow b$

Lado esquerdo não possui não-terminal

Lado direito tem comprimento menor que o esquerdo (gramática não-monotônica)

# Finalizando

- Hierarquia de Chomsky completa

| Hierarquia | Gramáticas | Linguagens | Autômato mínimo |
| --- | --- | --- | --- |
| Tipo-0 | Recursivamente Enumeráveis ou irrestritas | Recursivamente Enumeráveis | Máquinas de Turing |
| Tipo-1 | Sensíveis ao contexto | Sensíveis ao contexto | MT com fita limitada |
| Tipo-2 | Livres de contexto | Livres de contexto | Autômatos de pilha |
| Tipo-3 | Regulares (Expressões regulares) | Regulares | Autômatos finitos |

# Fim

Aula 23 - Máquinas de Turing restritas