Máquinas de Turing

Alan Turing

Alan Turing é um dos “pais” da Computação.

- Seu modelo computacional – a Máquina de Turing– inspirou/anteviu o computador eletrônico, que veio algumas décadas depois.
- Foi usado na quebra do sistema criptográfico Nazi Enigma na WWII
- Inventou o “Turing Test” usado em IA
- Prêmio Turing: Mais reconhecido prêmio em Teoria da Computação

Uma Máquina Pensante

Objetivo inicial da Máquina de Turing: Um modelo capaz de computar qualquer coisa que um humano possa computar. Antes da invenção do computador eletrônico, o termo "computador" de fato referia-se a uma *pessoa* cujo trabalho seria efetuar cálculos numéricos !

Como esse é um objetivo filosófico, ele de fato não pode ser provado.

Tese de Turing : Qualquer "algoritmo" pode ser executado por uma dessas máquinas.

Uma Máquina Pensante

Segundo Objetivo da Máquina de Turing: Um modelo que seja tão simples que possa ser de fato usado para provar interessantes resultados epistemológicos. Por exemplo, visava uma solução para o $10^{0.}$ problema de Hilbert.

Deixando o aspecto filosófico, o programa de Turing para quebrar o sistema de criptografia Enigma mostrou que ele era um verdadeiro *hacker*! A máquina de Turing é de fato fácil de programar, mas não muito útil na prática…

$10^{o.}$ Problema de Hilbert

Obter um algoritmo que, dada uma equação Diofantina, determina, em um número finito de operações, se existem ou não números inteiros que satisfaçam essa equação.

- Listado por Hilbert como um dos 10 problemas mais importantes da matemática em 1900, só foi resolvido em 1970, por Matijasevic, Robinson, Davis e Putnam, que mostraram que tal algoritmo não existe.
- A prova original usa Máquinas de Turing. Uma prova mais simples foi dada posteriormente (Jones and Matijasevic, Journal of Symbollic Logic, 49(1984)) usando máquinas de registradores (Minsky e Lambek).

Uma Máquina Pensante

Imagine um computador humano super-organizado e obsessivo-compulsivo. O computador quer evitar erros e, por isso, escreve tudo o que faz, uma letra/número de cada vez. O computador segue um conjunto finito de regras, que ele examina cada vez que escreve um novo símbolo. Em cada instante apenas uma regra pode ser usada, evitando assim ambiguidade. Cada regra ativa uma nova regra, dependendo da letra/número que é lido no momento. P. ex.:

Uma Máquina Pensante
EX: Programa **`Successor`**

Exemplos de regras:

If read 1, write 0, move right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

Vejamos como isso seria executado sobre um pedaço de papel que contenha o *reverso* da representação binária de 47:

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

1	1	1	1	0	1				

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	1	1	1	0	1				

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	1	1	0	1				

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

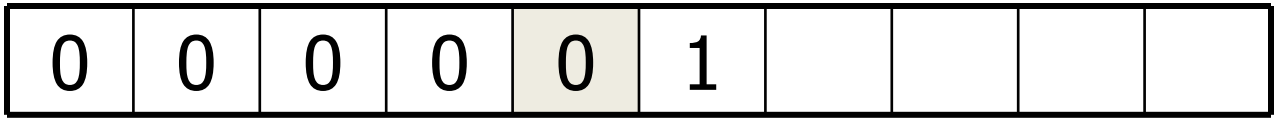
0	0	0	1	0	1				

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!



Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	0	0	1	1				

Uma Máquina Pensante
EX: Programa **`Successor`**

A saída do programa **successor** para a entrada 111101 é 000011 que é a representação binária de 48.

Analogamente, o resultado de **successor** com entrada 127 será 128:

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

1	1	1	1	1	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	1	1	1	1	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	1	1	1	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	0	1	1	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	0	0	1	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	0	0	0	1	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

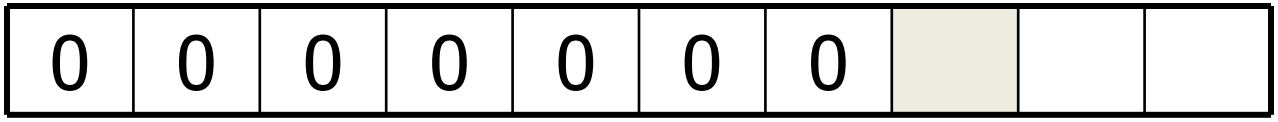
0	0	0	0	0	0	1			

Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!



Uma Máquina Pensante
EX: Programa **`Successor`**

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read , write 1, HALT!

0	0	0	0	0	0	0	1		

Uma Máquina Pensante

Era difícil para os matemáticos da época acreditarem que *qualquer* algoritmo poderia ser executado em uma máquina tão simples. Para quem já programou em *assembly*, isso é muito mais fácil!

Entretanto, diversas evidências à aceitação da Tese de Turing: entre elas, a prova da equivalência entre Máquinas de Turing e o lambda-calculus de Church (no qual são baseadas as linguagens funcionais, como Haskell, ML e Lisp)!

Máquina Turing

Uma Máquina de Turing (**TM**) é um dispositivo com uma quantidade finita de memória *"hard" read-only* (estados), e quantidade *ilimitada*[1] de memória-fita read/write. Não possui dispositivo de entrada separado. Supõe-se que os dados de entrada estão na fita, no momento em que a TM começa a executar.

Assim como um autômato, uma TM pode ser uma máquina input/output (como Transdutores de Estado Finito), ou uma máquina de decisão "yes/no". Vamos começar c/ máquinas "yes/no"

Máquina de Turing
Exemplo de Máquina de Decisão

O primeiro exemplo (adicionar 1 bit ao reverso de um string binário) é basicamente algo que pode ser feito por um Transdutor Finito (exceto quando ocorre overflow). Vejamos agora um exemplo de um nível acima na hierarquia de linguagens.

{bit-strings com mesmo número de 0's e 1's}

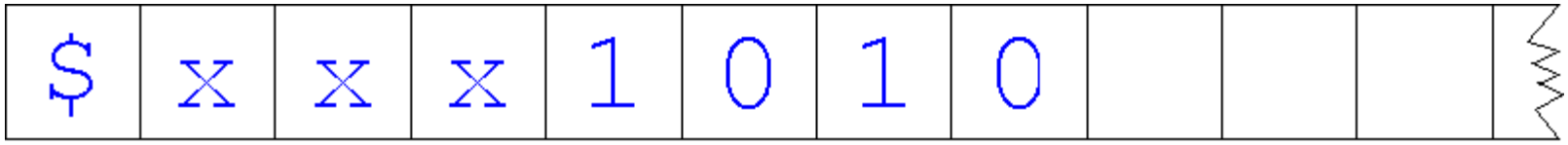
– uma linguagem livre de contexto

Máquina de Turing

Exemplo de Máquina de Decisão

Essa é uma "verdadeira" TM já que:

- A fita é semi-infinita:



- A entrada está na fita no início da execução
- Não há teste intrínsico de limite da fita à esq.
  - semelhante à detecção de pilha vazia em PDA's
  - truque similar –introduzir $ como flag
- Toda regra inclui direção de movimento (R/L)
- Situações que não podem ocorrer não são tratadas (tecnicamente indeterminadas)

Máquina de Turing

Exemplo de Máquina de Decisão

{bit-strings com mesmo número de 0's e 1's}:

Pseudocodigo:

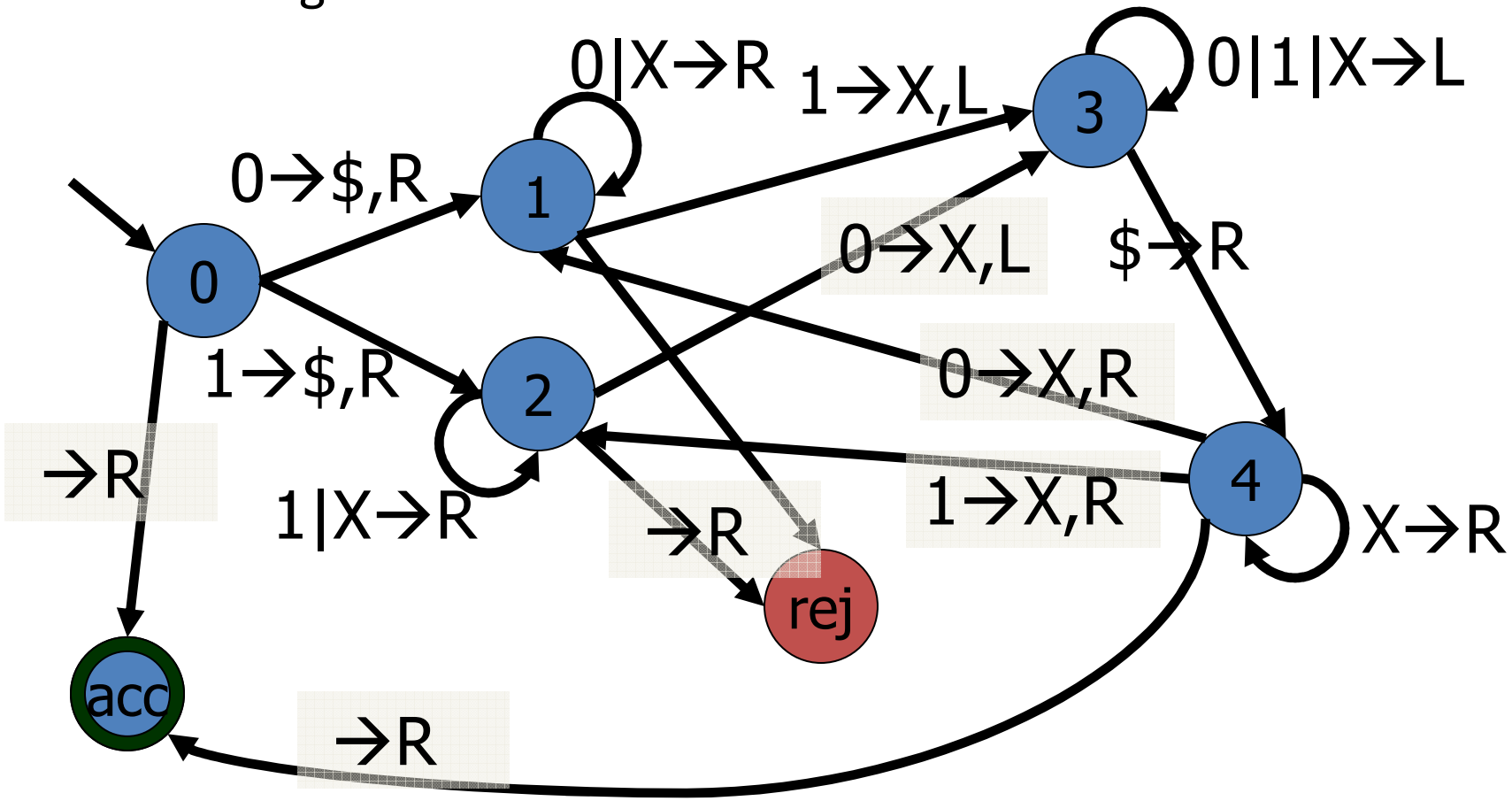
```
while (existe um 0 e um 1 na fita)
  marque esses dois símbolos
if (todo símbolo estiver marcado)
  aceita
else
  rejeita
```

Exemplo de TM
Conjunto de Instruções

0. if read , go right (*dummy move*), ACEITA
   if read 0, write $, go right, goto 1 // $ indica início da fita
   if read 1, write $, go right, goto 2
1. if read , go right, REJEITA
   if read 0 or X, go right, repeat (= *goto* 1) // pesquisa por 1
   if read 1, write X, go left, goto 3
2. if read , go right, REJECT
   if read 1 or X, go right, repeat // pesquisa por 0
   if read 0, write X, go left, goto 3
3. if read $, go right, goto 4 // pesquisa inicio da fita
   else, go left, repeat
4. if read 0, write X, go right, goto 1 // similar ao passo 0
   if read 1, write X, go right, goto 2
   if read X, go right, repeat
   if read , go right, ACEITA

Exemplo de TM
Diagrama de Estados

Essas instruções são usualmente expressas na forma de um diagrama de fluxo:



TM - Notação

Um arco do estado *p* para o estado *q* rotulado como …

- *a*→*b*,D significa que se estiver em *p* e o símbolo corrente na fita é *a*, substitua-o por *b* e mova para a direção D, e para o estado *q*
- *a*→D significa que se estiver em *p* e o símbolo corrente na fita é *a*, não o altere e mova na direção D, e para o estado *q*
- *a*|*b*|…|*z* → … significa que se o símbolo corrente na fita for qualquer das alternativas, a ação a ser realizada é a mesma.

TM – Notação de Configuração

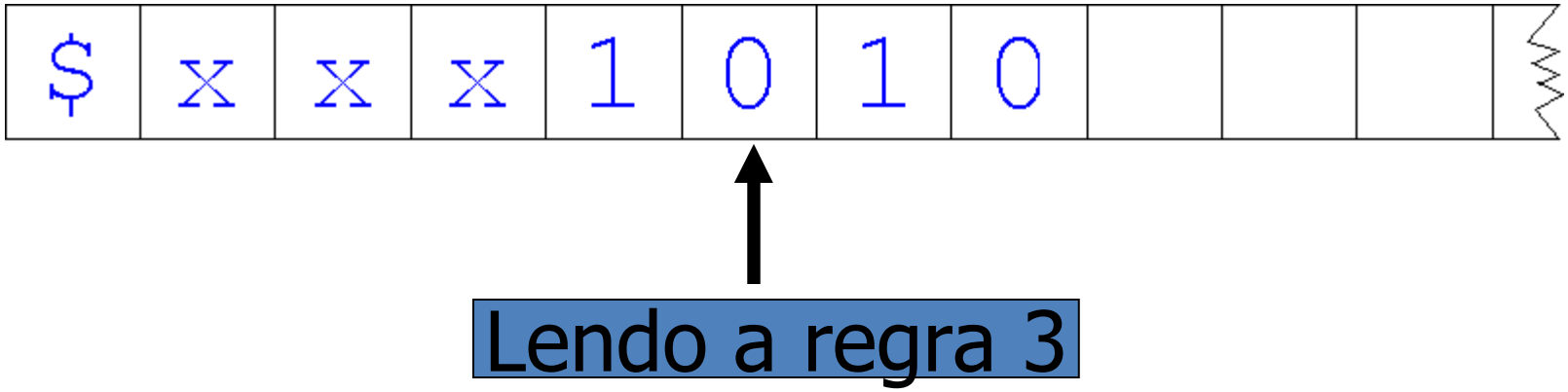
A próxima ação de uma TM é completamente determinada pelo estado corrente e pelo símbolo lido; portanto podemos prever ações futuras se sabemos:

1. o estado corrente
2. o conteúdo corrente da fita
3. a posição corrente da "cabeça" de leitura

Uma notação útil é representar essa informação na forma de um único string. O símbolo que representa o estado corrente é intercalado no conteúdo da fita, entre a porção que está à esq. da cabeça de leitura e a que está à sua dir. (incluindo o símbolo corrente).

TM – Notação de Configuração

Por exemplo



E denotada por:

$$\$xxx1q_3010$$

TM - Definição Formal
Estática

DEF: Uma ***máquina de Turing*** (TM) é uma 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$. Q, Σ, e q_0, são como em um FA. Γ é o ***alfabeto da fita***, que necessariamente inclui o símbolo branco, assim como Σ. δ é uma função:

$$\delta : (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

Portanto, dado um estado *p,* que não seja de parada, e um símbolo de fita *x*, $\delta(p,x) = (q,y,D)$ significa que a TM vai para o estado *q,* substitui *x* por *y*, e a cabeça da fita move na direção D.

TM - Dinâmica

Um string *w* é ***aceito*** por *M* se, quando dado como entrada na fita de *M*, estando a cabeça de leitura posicionada no início da fita, e sendo iniciada a execução, *M* eventualmente pára em um estado de aceitação. Nesse caso, *w* é um elemento de *L*(*M*) – a linguagem aceita por *M.*

Isso pode ser formalizado como a seguir:

TM - Definição Formal
Dinâmica

Suponha que a configuração da TM em um dado instante t é dada por $uapxv$ onde p é o estado corrente, ua é o que está à esquerda da cabeça da fita, x é o símbolo que está sendo lido, e v a porção da fita à direita de x.

Se $\delta(p,x) = (q,y,\mathrm{R})$, então escrevemos:

$$uapxv \Rightarrow uayqv$$

Sendo $uayqv$ a configuração resultante no instante t+1.

Se $\delta(p,x) = (q,y,\mathrm{L})$, escrevemos:

$$uapxv \Rightarrow uqayv$$

Existem dois casos especiais:

- cabeça da fita está sobre uma posição em branco[1]
- cabeça da fita bate na extremidade esquerda –> fica parada[2]

TM - Definição Formal
Dinâmica

Como no caso de gramáticas livres de contexto, podemos considerar o fecho reflexivo e transitivo "$\Rightarrow^*$" de "$\Rightarrow$". I.e. a relação entre strings definida recursivamente por:

- se $u = v$ então $u \Rightarrow^* v$
- se $u \Rightarrow v$ então $u \Rightarrow^* v$
- se $u \Rightarrow^* v$ e $v \Rightarrow^* w$, então $u \Rightarrow^* w$

"$\Rightarrow^*$" lê-se "***computa para***"

Um string x é ***aceito*** por M se a *configuração inicial* $q_0 x$ computa para alguma *configuração de aceitação* y –i.e., uma configuração contendo q_{acc}.

A ***linguagem aceita por*** M é o conjunto de todos os strings aceitos. I.e:

$$L(M) = \{ x \in \Sigma^* \mid \exists \text{ config. aceitação } y,\ q_0 x \Rightarrow^* y \}$$

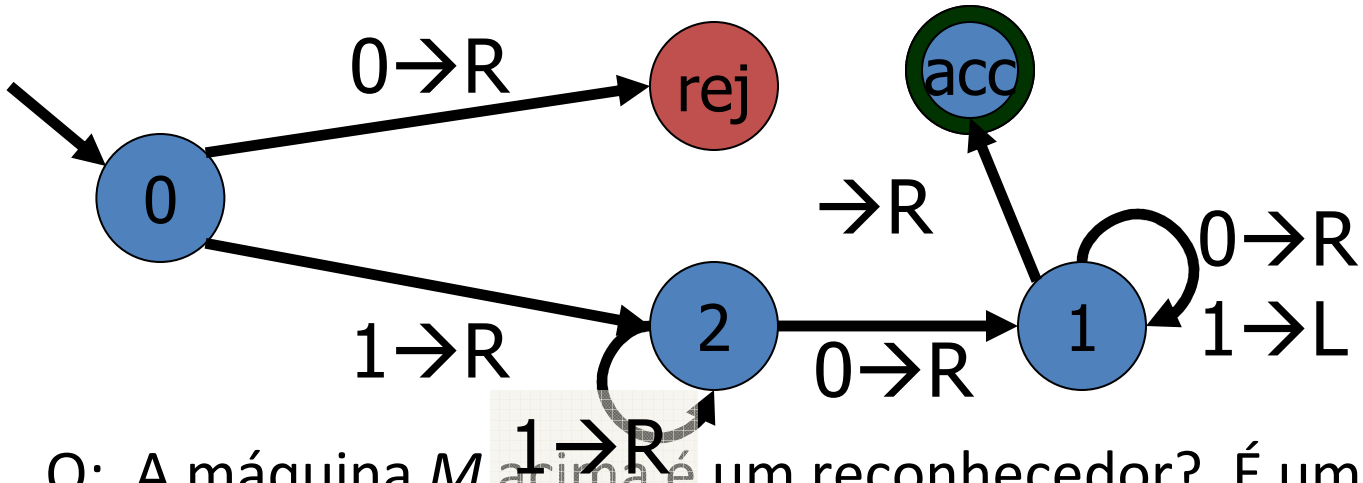
Reconhecedores vs. Decisores

Há 3 possíveis resultados para uma entrada w :

1. A TM M eventualmente entra em q_{acc} e, portanto, pára e aceita. ($w \in L(M)$)
2. A TM M eventualmente entra em q_{rej} ou *falha* em algum ponto. M ***rejeita*** w ($w \notin L(M)$)
3. Nenhum dos dois ocorre! I.e., a computação de M nunca pára, entrando em um ***loop infinito*** e nunca atingindo q_{acc} ou q_{rej}.
   Nesse caso, w não é aceito, nem rejeitado.
   Entretanto, um string que não seja aceito explicitamente não pertence à linguagem. ($w \notin L(M)$)

Reconhecedores vs. Decisores

Uma Máquina de Turing é um ***reconhecedor*** e ***reconhece*** $L(M)$. Se, além disso, *M nunca entra em loop infinito*, então M é dito um ***decisor*** e diz-se que ***decide*** $L(M)$.

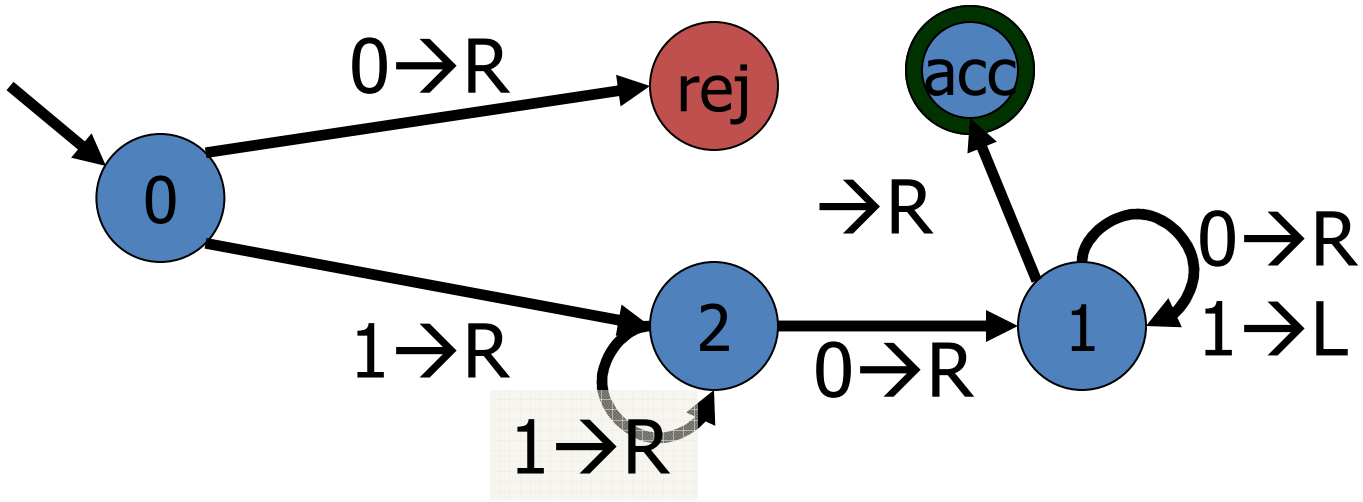


Q: A máquina M acima é um reconhecedor? É um decisor? O que é $L(M)$?

Reconhecedores vs. Decisores

R: *M* é um reconhecedor mas não um decisor, porque a entrada 101 causa um loop infinito.

$L(M) = 1^+ 0^+$



Q: A linguagem *L*(*M*) é decid***ível*** ?

Reconhecedores vs. Decisores

R: Sim. Toda regular é decidível porque sempre se pode converter um DFA para uma TM sem nenhum loop infinito.

Q: Como isso pode ser feito?

Exercício

Forneça TMs para:

$\{w \in \{a, b\} \mid |w| \text{ é par e } w = w^R\}$

$\{w \# w \mid w \in \{0,1\}^* \}$

$\{a^i b^j c^k \mid i \times j = k \text{ e } i,j,k \geq 1 \}$

$\{0^{2^n} \mid n \geq 0 \}$

Variantes de Máquinas de Turing

Máquinas de Turing Input-Output

TM's ***Input/output*** (ou ***IO*** ou ***transdutoras***) diferem de TM's reconhecedoras por terem um estado de parada neutro q_{halt}, ao invés de estados de aceitação e rejeição. Esse tipo de TM é então vista como uma função (string -> string), que mapeia o conteúdo inicial da fita *u* ao conteúdo (porção não vazia) da fita no instante em que atinge o estado q_{halt}. Se *v* é o conteúdo da fita quando a máquina pára, escrevemos $f_M(u) = v$

Se *M* falha durante a computação, ou entra em um loop infinito, *M* é dita ***indefinida*** sobre *u*.

Máquinas de Turing Input-Output

Quando f_M falha ou entra em um loop infinito para algum string de entrada, f_M é dita uma ***função parcial.***

Se *M* sempre pára (halt) para qualquer possível string de entrada, a função f_M é ***total*** (i.e. sempre definida).

TM Notação

Sipser adota três maneiras de descrever algoritmos de TM's.

1) Pseudocódigo de alto nível, que explica como o algoritmo funciona, sem entrar em detalhes técnicos da notação formal de TM's
2) Nível de implementação – descreve como a TM opera sobre sua fita, sem mencionar estados explicitamente.
3) Descrição de baixo nível:
   - Conjunto completo de instruções em estilo "goto"
   - Diagrama de Estados
   - Descrição Formal (na forma de 7-tuple)

TM: Alto Nível
Exemplo

Vamos descrever, por exemplo, uma Máquina de Turing *M* que multiplica um número por 2, em notação unária:

M = "Sobre a entrada $w = 1^n$

Para cada caractere *c* em *w*

copie *c* no próximo "espaço branco'' disponível

TM: Nível de Implementação
Exemplo

A idéia é mostrar como funciona o processo de copiar cada caractere no final da fita, mencionado na descrição de alto nível. Devemos manter informação sobre que caracteres já foram copiados, distinguindo esses caracteres. Uma maneira de fazer isso é usar um caractre distinto, digamos X.

EX: Vejamos como 11111 é transformado.

TM: Nível de Implementação
Exemplo

Então, passo a passo, o conteúdo da entrada é transformado como a seguir:

- 11111
- X1111X
- XX111XX
- XXX11XXX
- XXXX1XXXX
- XXXXXXXXXX
- 1111111111

TM: Nível de Implementação Exemplo

O ***nível de implementação*** descreve como o algoritmo opera de fato na máquina de Turing, facilitando a obtenção do diagrama de estados

Alguns métodos úteis:

- fast forward
  - move p/ a dir. enquanto uma condição é satisfeita
- rewind
  - move p/ esq. enquanto uma condição é satisfeita
- Pode requerer funcionalidade extra:
  - Adicione $ se for necessário detectar fim de fita

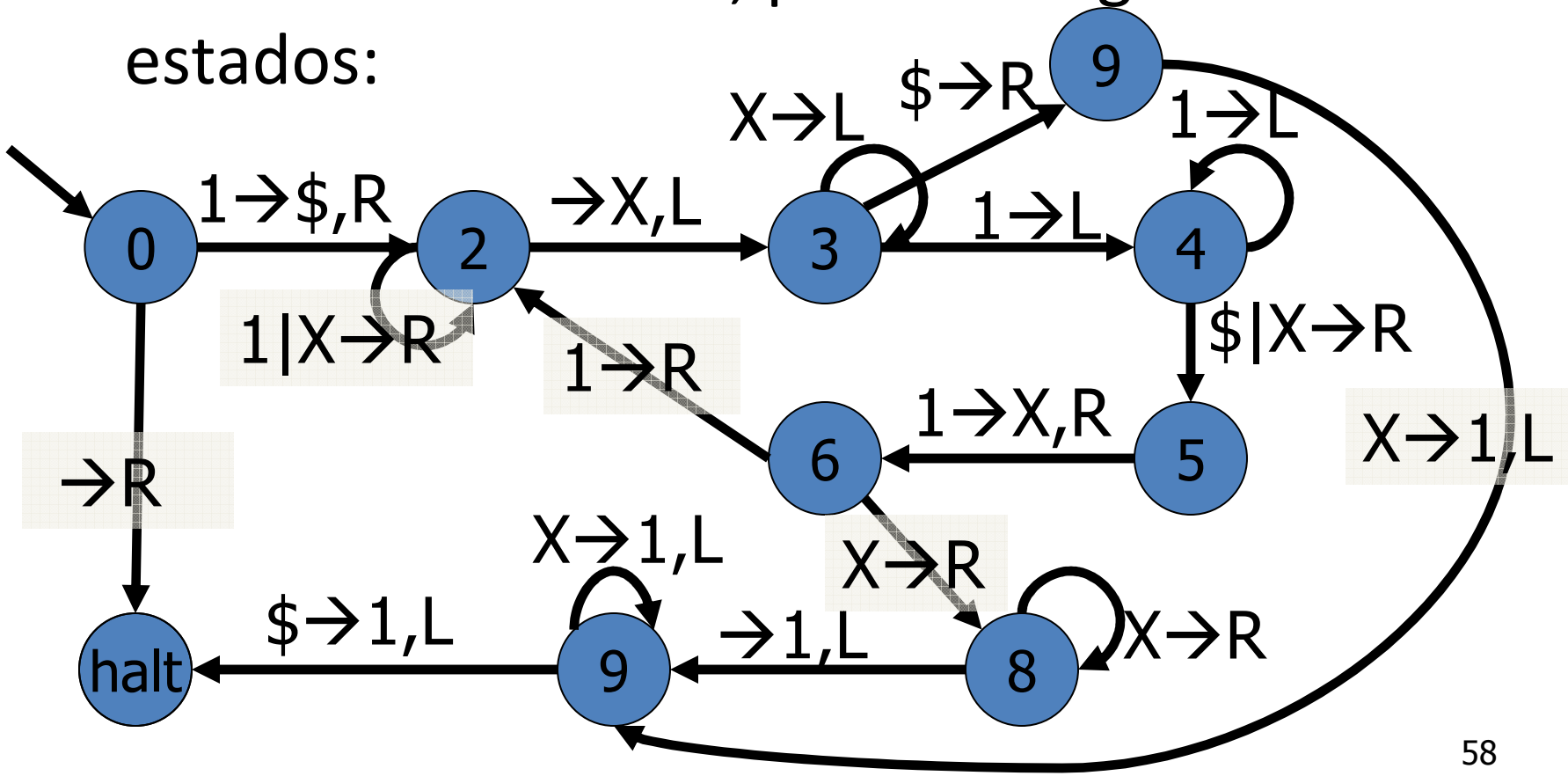
TM: Nível de Implementação
Exemplo

M = "Sobre entrada *w* = 1^n

1. HALT se entrada vazia
2. Escreva $ na posição mais à esquerda
3. Ande p/ dir. e escreva X no primeiro branco
4. Ande p/ esq. enquanto encontrar X ou 1
5. Mova para direita
6. Se ler X, mova p/ direita e vá para 9. Senão, substitua 1 por X, mova p/ direita.
7. Se lê X [[*w* original terminado]] vá para 8 Senão, vá para 3
8. Ande p/ dir. até encontrar branco e o substitua p/ X
9. Ande p/ esq. substituindo todo não branco por 1
10. HALT

TM: Baixo Nível
Exemplo

No ***baixo nível*** a Máquina de Turing é descrita em detalhe, por um diagrama de estados:



TM's Não Deterministas

Uma Máquina de Turing não Determinista *N* permite mais de uma possível ação para um dado par (estado, símbolo na fita).

Um string *w* é ***aceito*** por *N* se algum *ramo da computação* de *N* sobre esse string entra eventualmente no estado q_{acc}.

Se, por outro lado, em qq ramo da computação, *N* eventualmente entra no estado q_{rej} **ou** falha **ou** entra em um loop infinito, *w* não é aceito por aquele ramo.

Simbolicamente:

$$L(N) = \{ x \in \Sigma^* \mid \exists \text{ config.de aceitação } y,\ q_0 x \Rightarrow^* y \}$$

TM's Não Deterministas
Reconhecedores vs. Decisores

N é dita um ***reconhecedor não determinista*** e diz-se que *reconhece L*(*N*); além disso, se para qualquer string de entrada e todos os ramos de computação, *N* sempre pára, então *N* é dita um ***decisor não determinista*** e diz-se que *decide L*(*N*).

TM Não Determinista
Exemplo

Considere o seguinte método não determinista:

```
void nonDeterministicCrossOut(char c)
        while()
            if (read blank) go left
            else
                        if (read c)
                                    cross out, go right, return
                       OR go right
                       OR go left
```

Non-Deterministic TM Example

Usando nonDeterministicCrossOut() construimos o seguinte programa:

1. while(some character not crossed out)
   nonDeterministicCrossOut(‘0’)
   nonDeterministicCrossOut(‘1’)
   nonDeterministicCrossOut(‘2’)
2. ACEITA

Q: Qual é a linguagem que esse programa não determinista reconhece ?

Non-Deterministic TM Example

R: $\{x \in \{0,1,2\}^* \mid x$ tem igual no. de 0's e 1's e 2's $\}$

Q: Suponha que q seja o estado em que permanece a TM enquanto executa nonDeterministicCrossOut('1') e q' é o estado em que executa nonDeterministicCrossOut('2').

Suponha que a configuração corrente é

$u =$ 0XX1Xq12X2

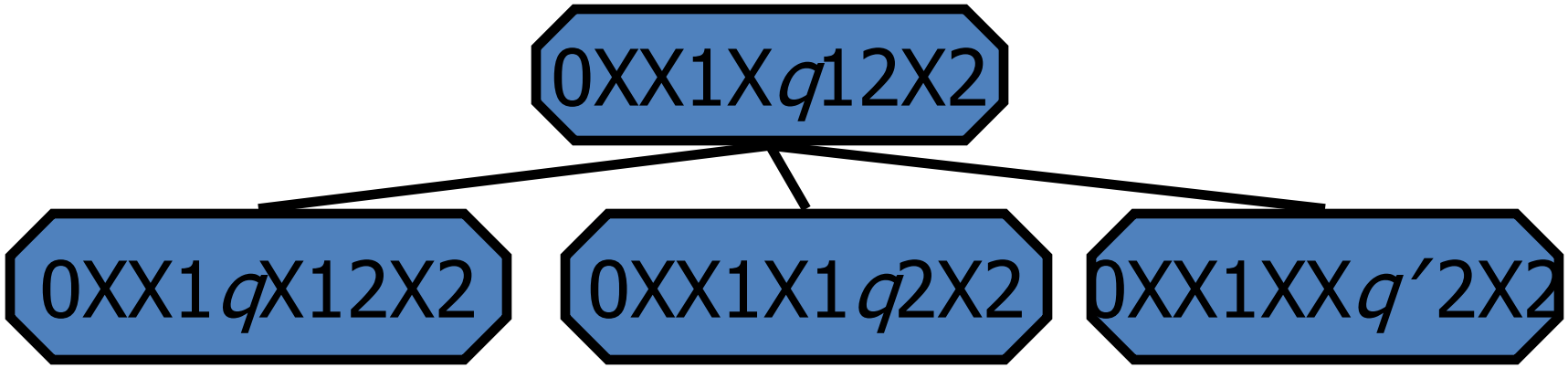
Para qual v temos que $u \Rightarrow v$?

Non-Deterministic TM Example

R: 0XX1Xq12X2 $\Rightarrow$

0XX1qX12X2 | 0XX1X1q2X2 | 0XX1XXq' 2X2

Isso define 3 ramos da árvore de computação



Q: Essa TM não determinista TM é um *decisor*?

Non-Deterministic TM Example

R: Não. Essa TM é um reconhecedor, e não um decisor. nonDeterministicCrossOut() pode entrar em um ramo de computação infinito, já que pode alternar move esq-dir ad infinitum, sem nunca marcar um símbolo. I.e., a árvore de computação é infinita!

Nota: Se você desenhar o diagrama de estados, verá a NTM é mais compacta que a versão TM, portanto há vantagens no não determinismo! Mais tarde, você verá exemplos de programas não deterministas "eficientes" para problemas práticos importantes, sem correspondentes deterministas eficientes: O problema **P** vs. **NP.**

NTM's
Lema de Konig

Para o Problema 3.3 (Sipser) o seguinte fato é importante:

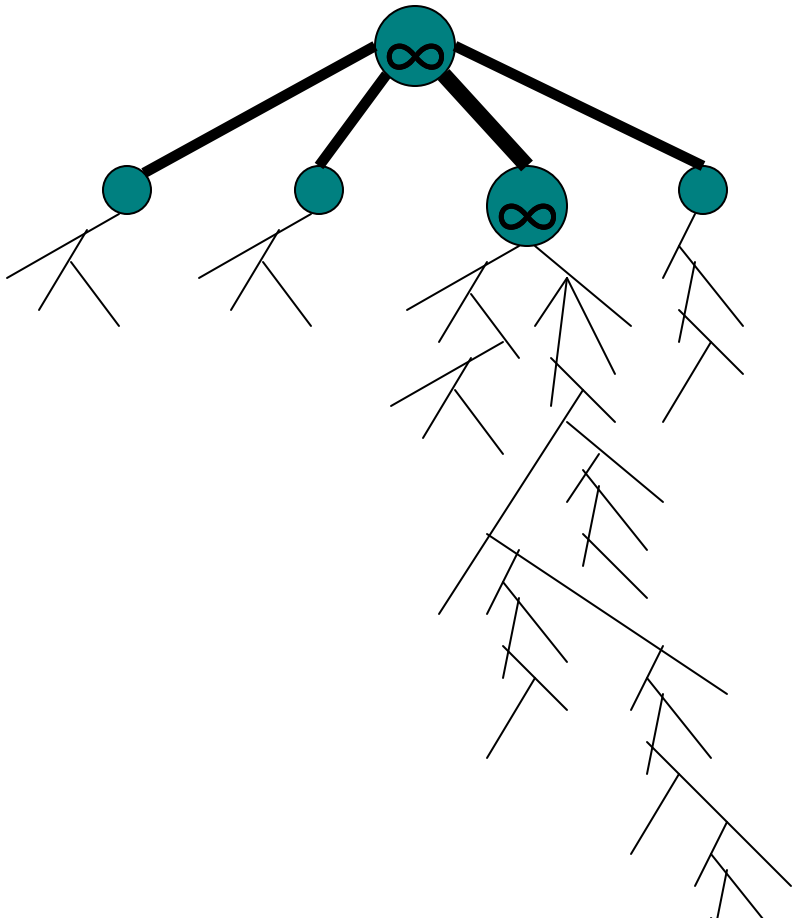
Se uma NTM é um decisor, então dada uma entrada qualquer, existe um número *h* tal que todo ramo de computação envolve no máximo *h* passos básicos, ou seja, a árvore de computação tem altura *h*. Isso segue de:

Lema de Konig: Uma árvore infinita, com número finito de ramos em cada nodo, deve conter algum caminho infinitamente longo.

De fato, usamos o contrapositivo: uma árvore sem caminho infinito e com no. finito de ramos em cada nodo deve ser finita.

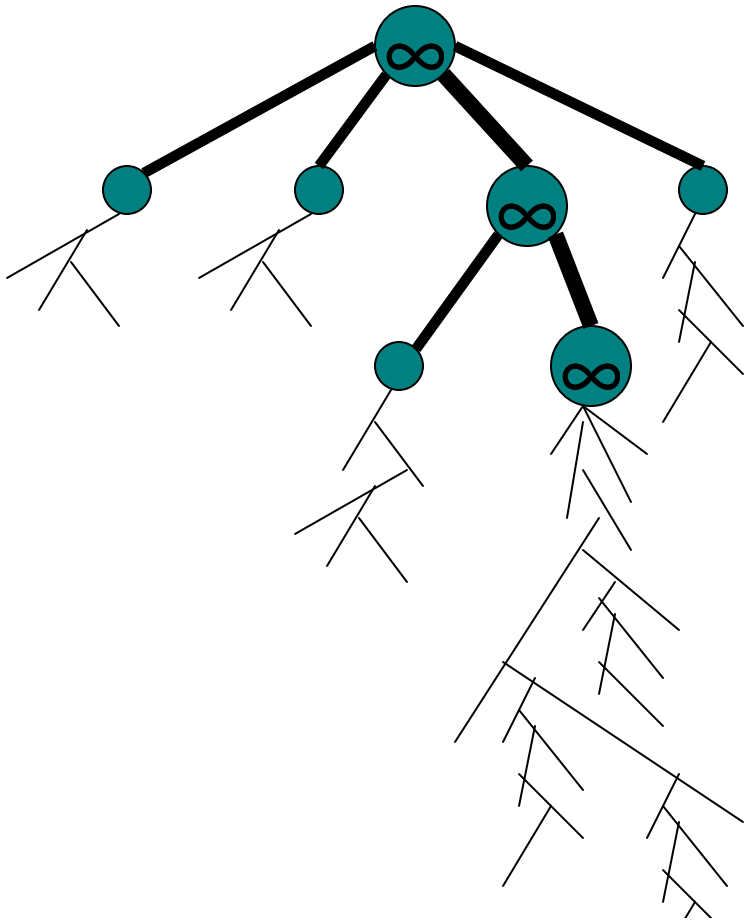
Lema de Konig
Idéia da Prova

A idéia é “advinhar” onde é a parte infinita da árvore e ir nessa direção:



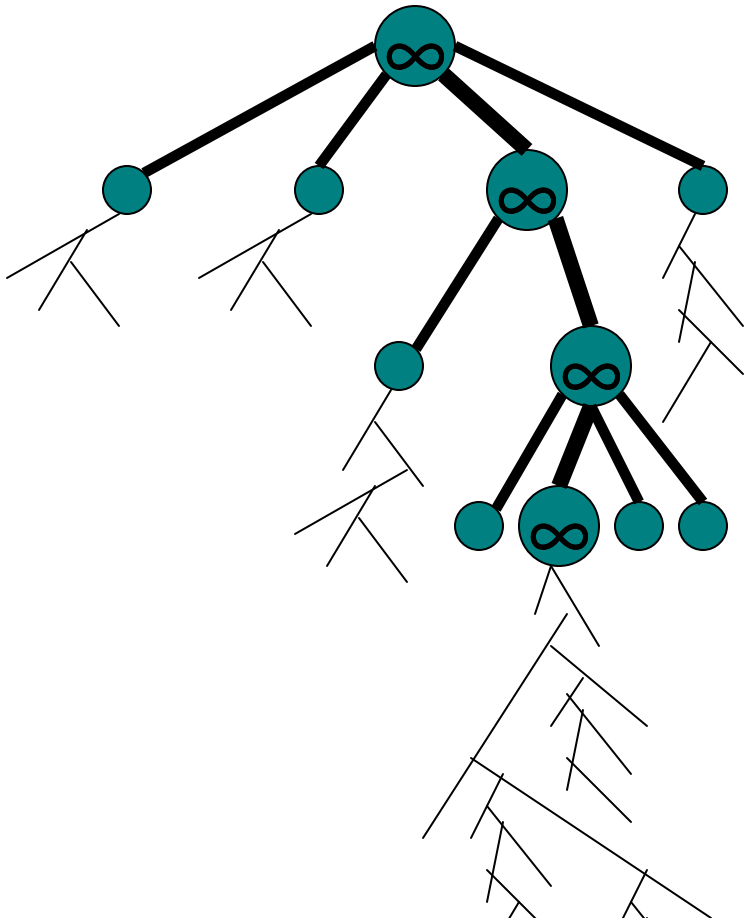
Lema de Konig
Idéia da Prova

A idéia é “advinhar” onde é a parte infinita da árvore e ir nessa direção:



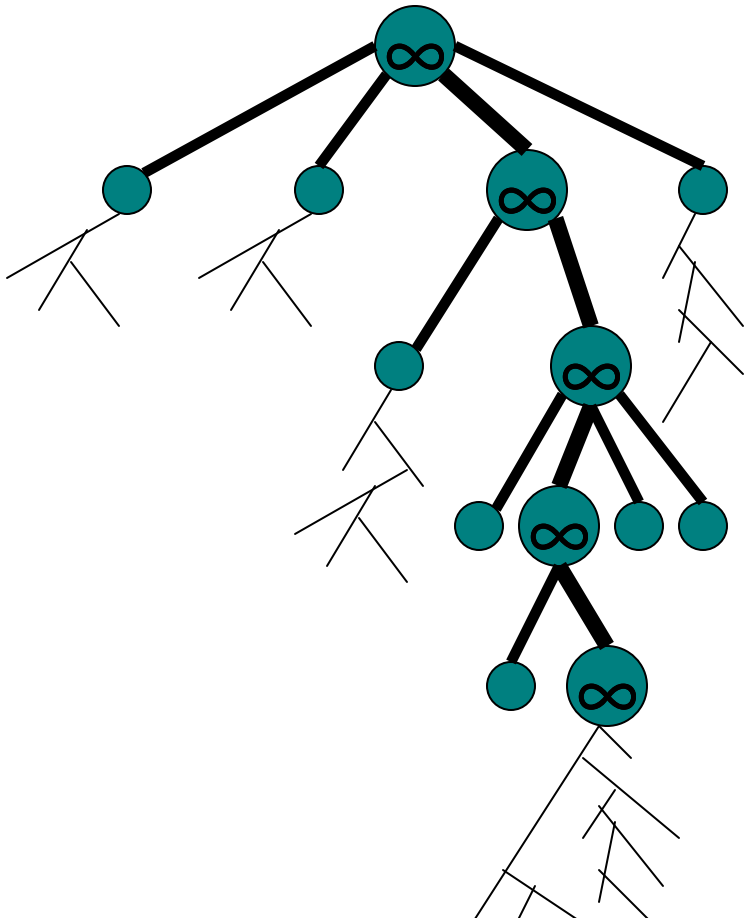
Lema de Konig
Idéia da Prova

A idéia é “advinhar” onde é a parte infinita da árvore e ir nessa direção:



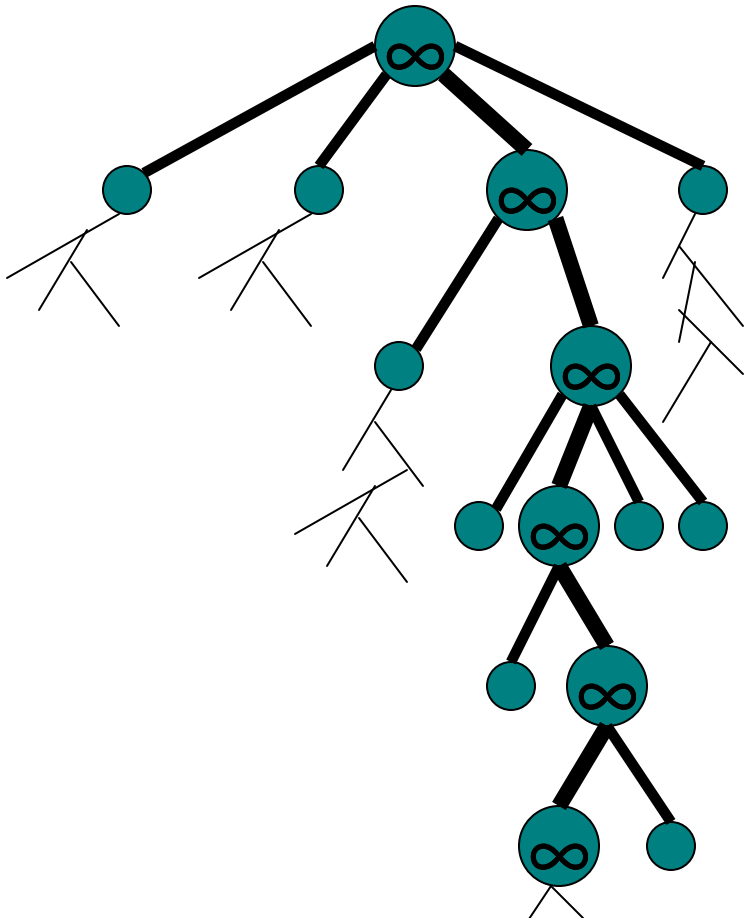
Lema de Konig
Idéia da Prova

A idéia é “advinhar” onde é a parte infinita da árvore e ir nessa direção:



Lema de Konig
Idéia da Prova

A idéia é “advinhar” onde é a parte infinita da árvore e ir nessa direção:



Lema de Konig
Prova

Prova. Dada uma árvore infinita com número finito de ramos em cada nodo, construímos um caminho infinito indutivamente:

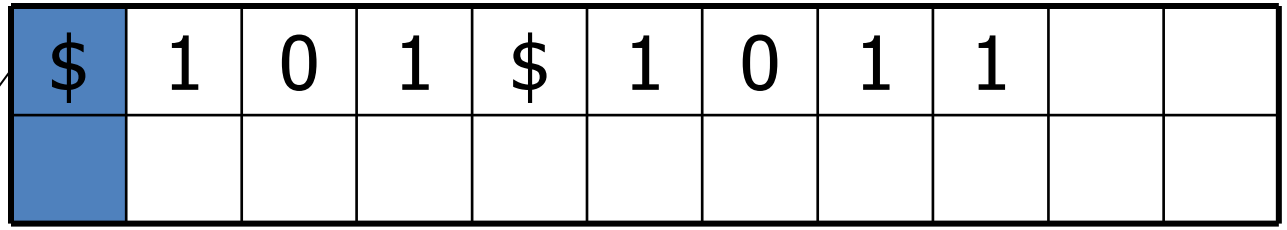
O vértice inicial v_0 é a raiz.

Arco $v_n \rightarrow v_{n+1}$: Suponha $v_0 \rightarrow v_1 \ldots v_{n-1} \rightarrow v_n$ tenha sido construído e que a subárvore a partir de v_n é infinita. Então um dos filhos (em no. finito) de v_n, seja v_{n+1}, deve ter uma subárvore infinita; então adicione o arco $v_n \rightarrow v_{n+1}$.

TM com Múltiplas Fitas

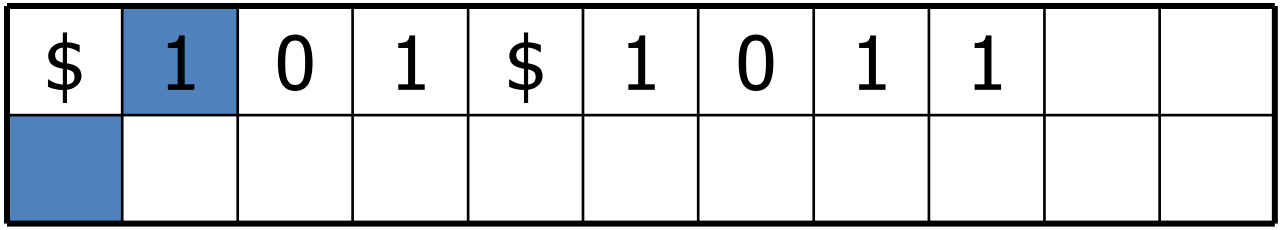
Frequentemente é útil considerar que temos várias fitas ao realizar uma computação. Por exemplo, considere uma TM com duas fitas I/O para adicionar números (vamos apenas mostrar como ela atua sobre uma entrada típica)

TM com Múltiplas Fitas
Exemplo: Adição

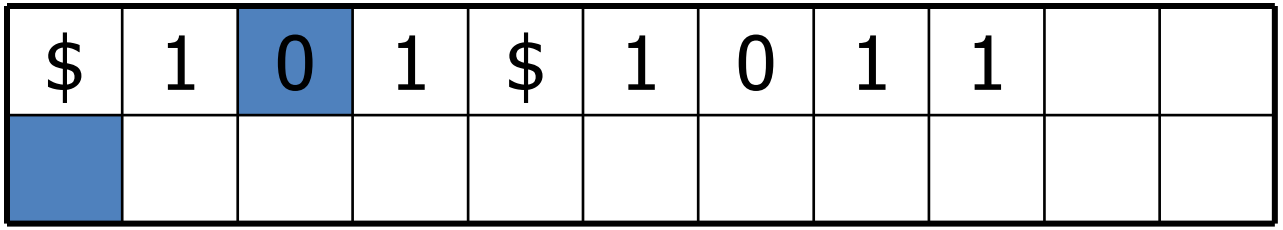


Input string

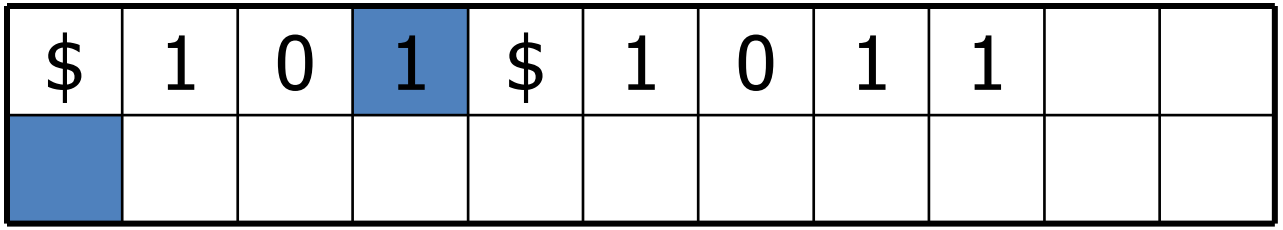
TM com Múltiplas Fitas
Exemplo: Adição



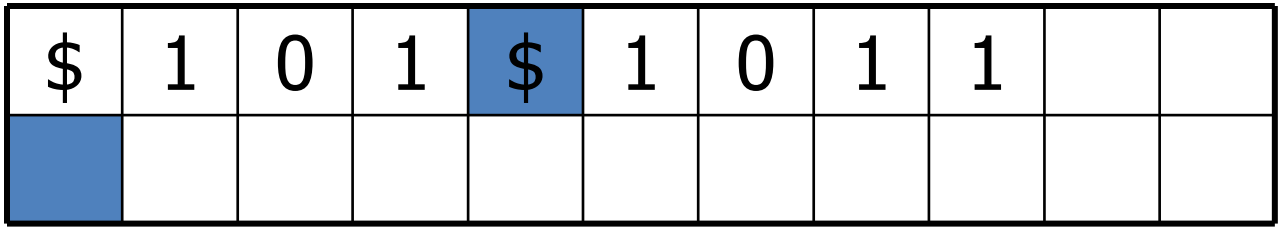
TM com Múltiplas Fitas
Exemplo: Adição



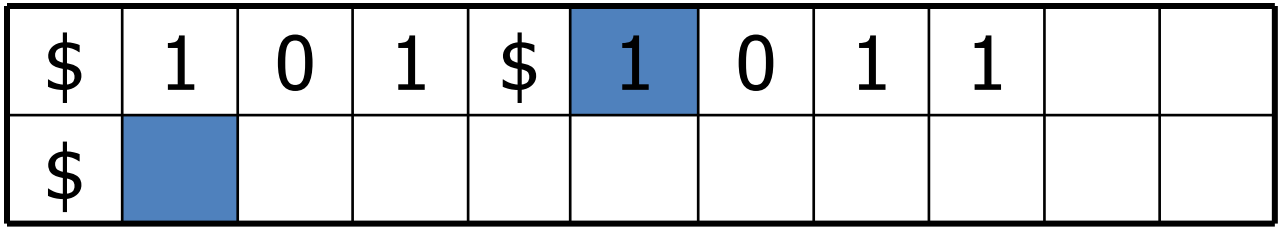
TM com Múltiplas Fitas
Exemplo: Adição



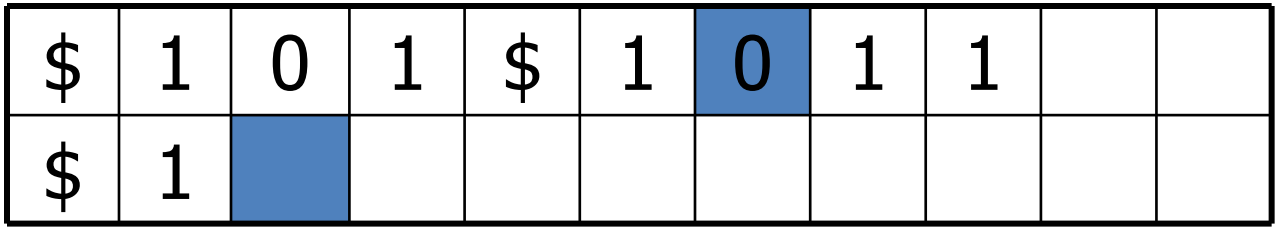
TM com Múltiplas Fitas
Exemplo: Adição



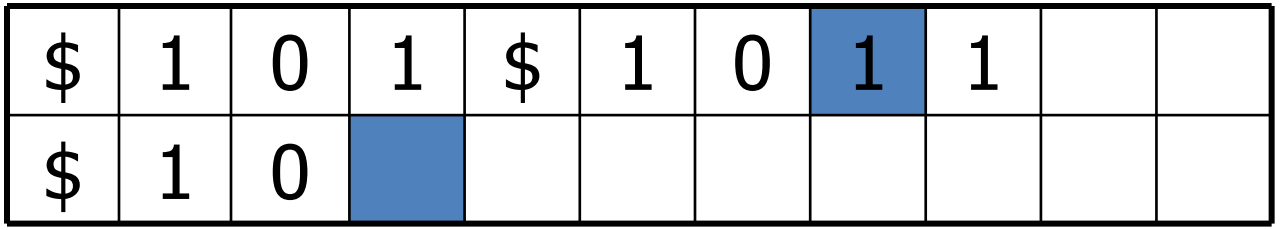
TM com Múltiplas Fitas
Exemplo: Adição



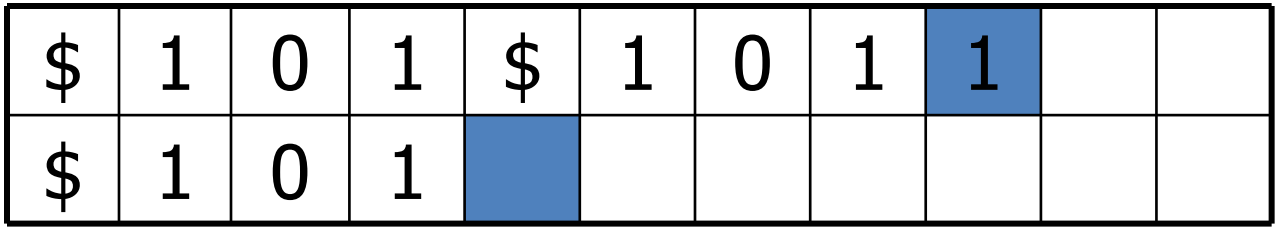
TM com Múltiplas Fitas
Exemplo: Adição



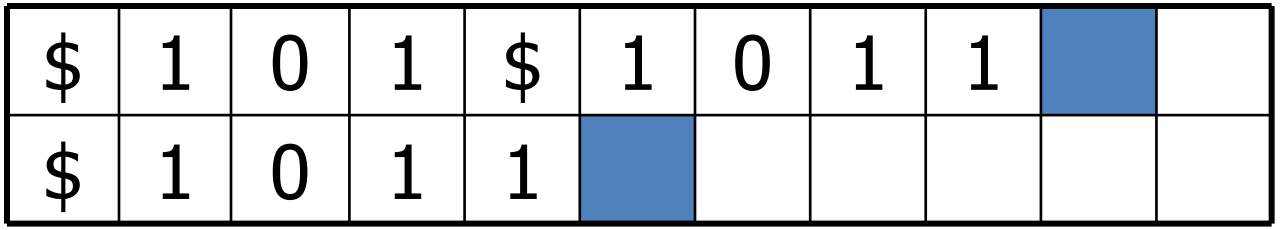
TM com Múltiplas Fitas
Exemplo: Adição



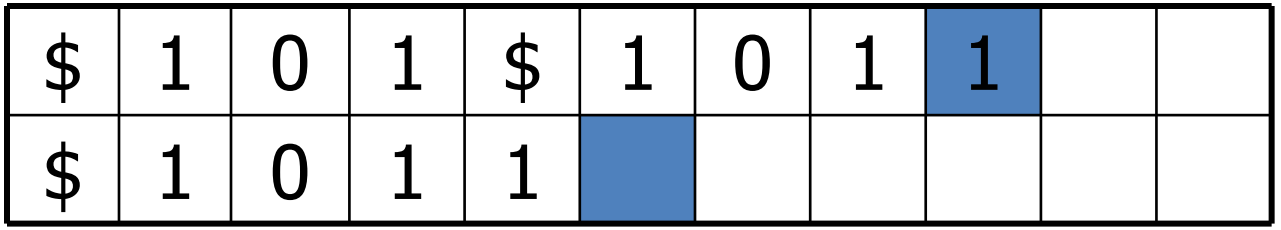
TM com Múltiplas Fitas
Exemplo: Adição



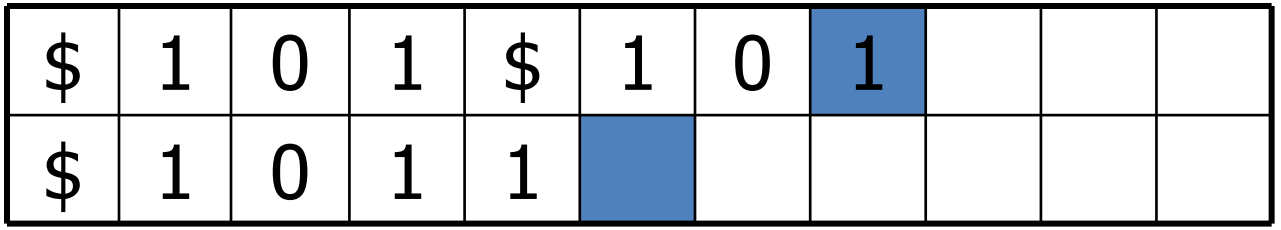
TM com Múltiplas Fitas
Exemplo: Adição



TM com Múltiplas Fitas
Exemplo: Adição



TM com Múltiplas Fitas
Exemplo: Adição



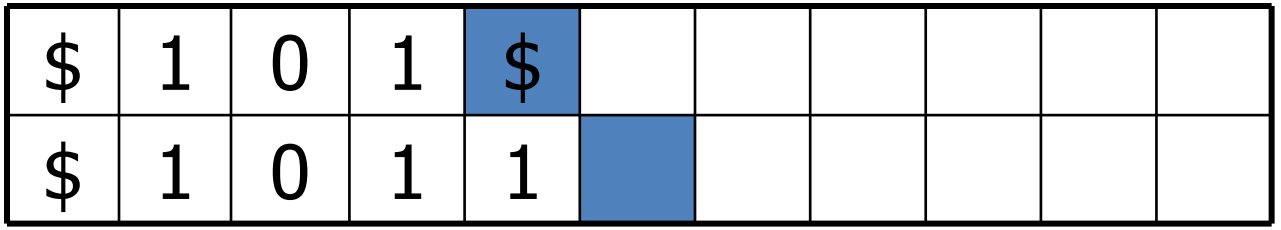
TM com Múltiplas Fitas
Exemplo: Adição

$	1	0	1	$	1	0				
$	1	0	1	1						

TM com Múltiplas Fitas
Exemplo: Adição

$	1	0	1	$	1					
$	1	0	1	1						

TM com Múltiplas Fitas
Exemplo: Adição



TM com Múltiplas Fitas
Exemplo: Adição

$	1	0	1							
$	1	0	1	1						

TM com Múltiplas Fitas
Exemplo: Adição

$	1	0	0							
$	1	0	1	0						

TM com Múltiplas Fitas
Exemplo: Adição

$	1	0	0							
$	1	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição

$	0	0	0							
$	1	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição

0	0	0	0							
$	0	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição

1	0	0	0							
1	0	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição

1	0	0	0							
1	0	0	0	0						

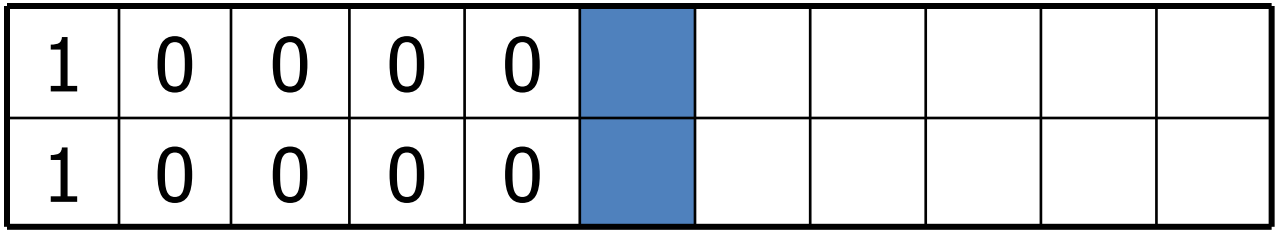
TM com Múltiplas Fitas
Exemplo: Adição

1	0	0	0							
1	0	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição

1	0	0	0							
1	0	0	0	0						

TM com Múltiplas Fitas
Exemplo: Adição



HALT!

Output string

TM com Múltiplas Fitas
Notação Formal

NOTA: A TM multifita de Sipser não pode pausar em uma das fitas como faz a do exemplo anterior. Isso não é problema pois uma pausa pode ser simulada por um "move L-R".

Formalmente, a função de transição δ de uma máquina *k*-fita é:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}\}^k$$

TM com Múltiplas Fitas
Convenção

- Entrada e saída sempre na 1$^{a.}$ fita
- Se for máquina I/O, a saída é também na primeira fita
- Essas máquinas podem ser consideradas como geradores de "vetores de string". E.x., poderíamos supor que uma máquina 4-fita produz como saída um valor em $(\Sigma^*)^4$

Exercício

- Descreva como construir máquinas de Turing Multifita para reconhecer:
  - $\{w\ w^R\ w \mid w \in \{0,1\}^*\}$
  - $\{w\ w \mid w \in \{0,1\}^*\}$
- Descreva como construir máquinas de Turing não Determinísticas para reconhecer:
  - $\{w\ w \mid w \in \{0,1\}^*\}$
  - $\{x\ x^R\ y \mid x, y \in \{0,1\}^*$ e $|x| > |y|\}$

Modelos Universais de Computação

Equivalência entre Variantes de TM

- TM's definem naturalmente uma classe.
- Toda variante razoável de TM define a mesma classe de linguagens. (reforça a Tese Church-Turing)

Recursivo e Recursivamente Enumerável

DEF: Uma linguagem é ***recursivamente enumerável*** se ela pode ser *reconhecida* por uma Máquina de Turing. Uma linguagem é ***recursiva*** se ela pode ser *decidida* por uma Máquina de Turing.

A terminologia "recursivamente enumerável" vem de outro modelo de TM: uma TM que nunca pára (quando descreve uma linguagem infinita) e escreve cada um dos strings da linguagem em uma fita auxiliar (impressora).

Enumerador Recursivo
Exemplo

Por exemplo, a saída de um enumerador para a linguagem **pal** seria, em um dado instante, como a seguir:

1. (*string vazio*)
2. a
3. b
4. aa
5. bb
6. aba
7. bab
8. aaa
9. bbb

Enumerador Recursivo
Exemplo

Enumeradores Recursivos são equivalentes a TM reconhecedoras, em razão do seguinte argumento:

1) Enumerador E → TM M:
    1) Sobre a entrada w:
        1) Rode E. Toda vez que E dá como saída uma cadeia, compare-a com w.
        2) Se w em algum momento aparece na saída de E, aceite.
2) TM M → Enumerador E:
    1) Ignore a entrada.
    2) Repita o seguinte para i=1, 2, 3, …
        1) Rode M por i passos sobre cada cadeia s_1, s_2, s_3, …
        2) Se quaisquer computações aceitam, imprima a s_j correspondente.

Equivalência de *k*-fitas vs. 1-fita

O argumento de que TM's com *k*-fitas são universais é bastante convincente. Afinal, todos os computadores atuais têm a memória dividida em diversos locais: RAM, Hard-Disc, Floppy, CD-ROM. É plausível que cada unidade de memória possa ser simulada por uma fita separada. O teorema a seguir implica em mostrar que o computador poderia manter a mesma funcionalidade (embora com alguma perda de eficiência) se toda a memória fosse juntada em uma única fita:

Equivalência de *k*-fitas vs. 1-fita

THM: Toda TM de *k*-fitas pode ser simulada por uma TM de 1-fita, e vice versa.

Prova. É claro que toda TM de 1-fita pode ser simulada por uma TM de *k*-fitas. Basta ignorar as fitas de no. 2 a *k*.

Por outro lado, para mostrar que qualquer TM de *k*-fitas pode ser simulada por uma TM de 1-fita TM, vamos provar 2 resultados:

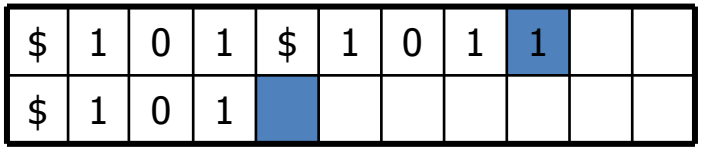
A) Qualquer TM de *k*-fitas pode ser simulada por uma TM de 2*k*-***trilhas***.

B) Qualquer TM de *k*-trilhas pode ser simulada por uma TM de 1-fita.

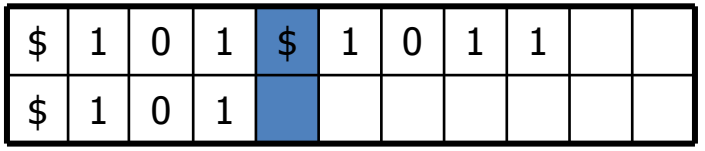
TM's de *k*-trilhas

Uma máquina de *k*-trilhas tem uma fita cujas células são divididas em *k* sub-células. Diferentemente de uma TM de *k*-fitas, ela possui apenas uma cabeça de leitura[1], que lê simultaneamente as *k* trilhas, baseando sua ação nesses valores.

EX. máquina 2-fitas:



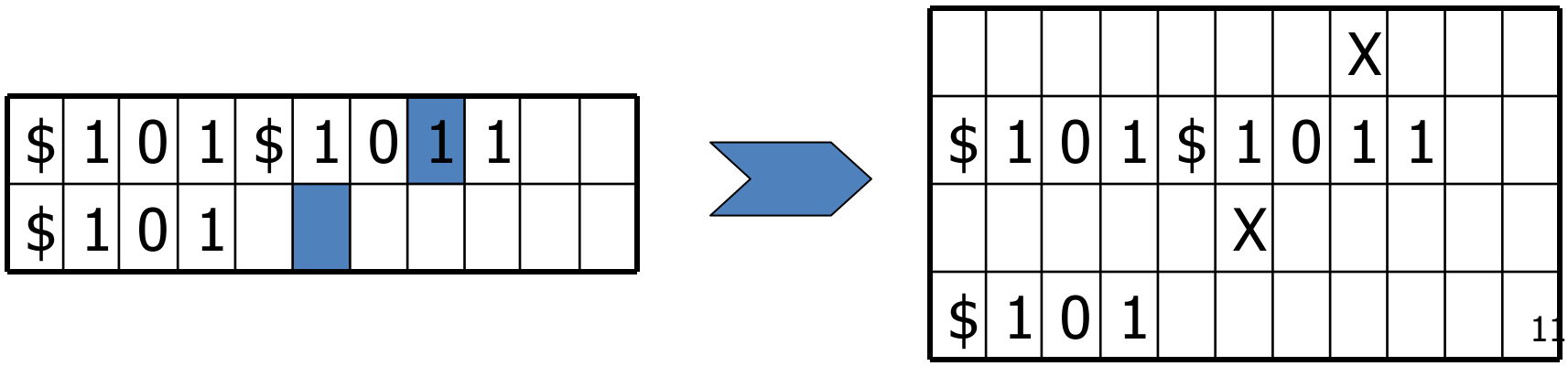
vs. máquina 2-trilhas:



2*k*-trilhas Simulando *k*-fitas

Podemos simulas qualquer TM de *k*-fitas por uma TM de *2k*-trilhas do seguinte modo:

- Cada fita da TM de *k*-fitas é representada por 2 trilhas:
  - A primeira contém o próprio conteúdo da fita.
  - A segunda é uma trilha em que todas as posições são "branco", exceto por um único X que indica qual é a célula corrente nessa fita.



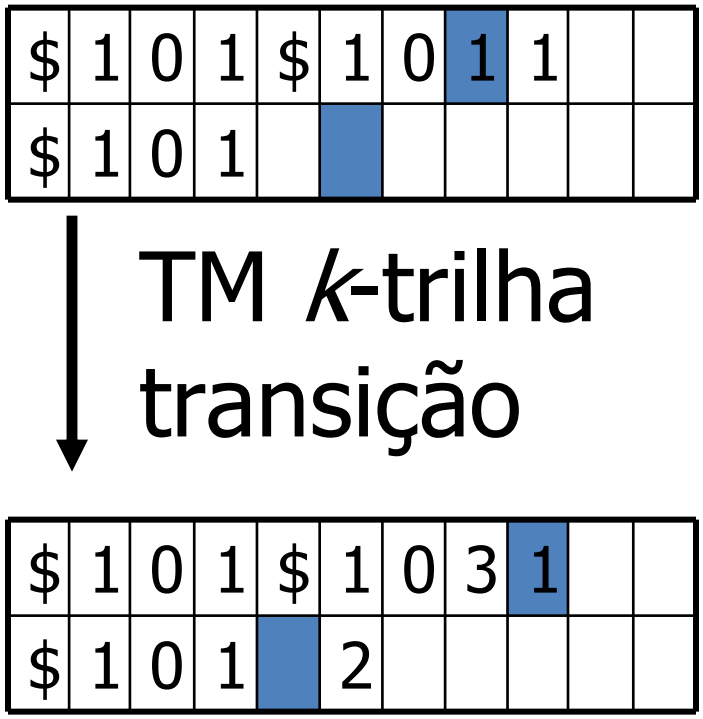
2*k*-trilhas Simulando *k*-fitas

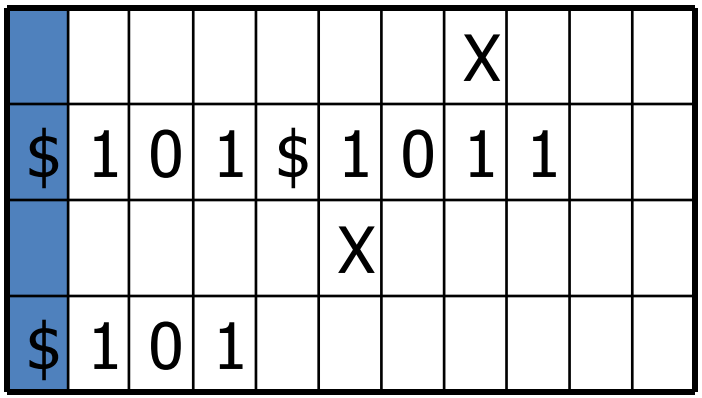
Para cada "move" da TM de *k*-fitas, a TM de 2*k*-trilhas anda p/ direita e depois p/ esquerda.

- Enquanto anda p/ direita, a TM de 2*k*-trilhas obtém a posição da cabeça de leitura de cada uma das *k*-fitas, assim como o símbolo na posição correspondente em cada fita, mantendo essa informação por meio de seus estados. Retorna depois de ler todos os X's.
- Enquanto anda p/ esquerda, para cada X encontrado a máquina modifica o conteúdo da trilha correspondente, usando o que memorizou sobre a configuração das *k*-fitas.

2*k*-trilhas Simulando *k*-fitas

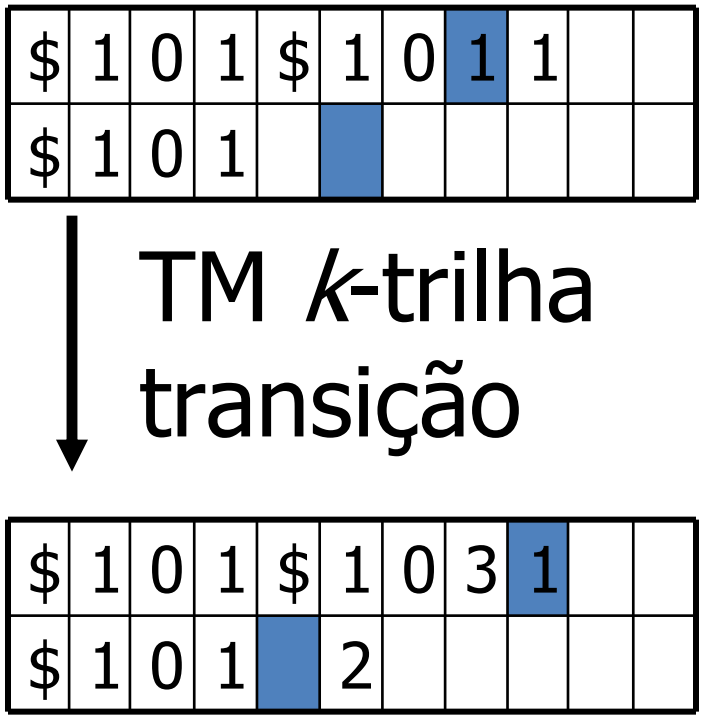
simulação por
TM de 2k-trilhas

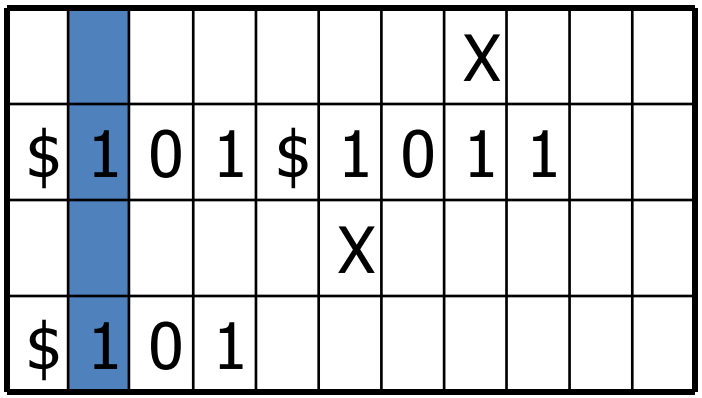




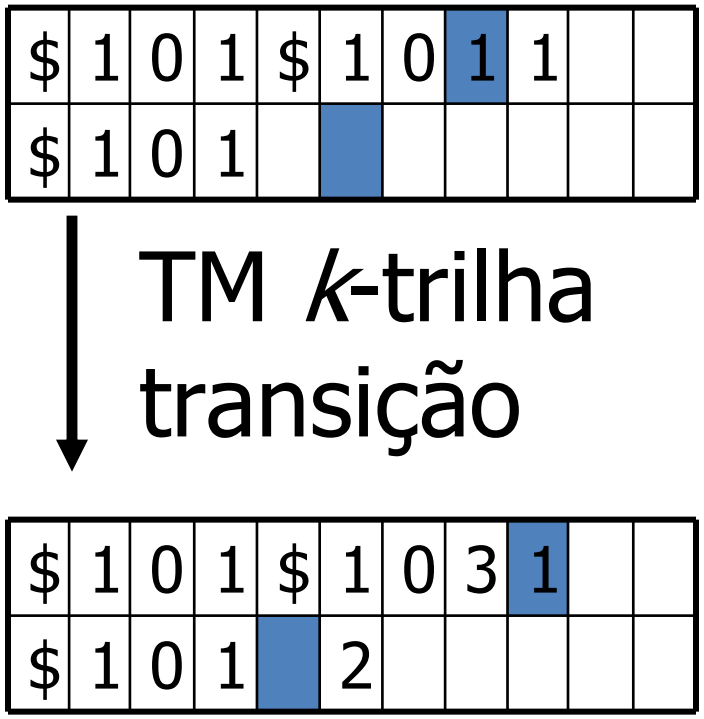
2*k*-trilhas Simulando *k*-fitas

simulação por
TM de 2k-trilhas

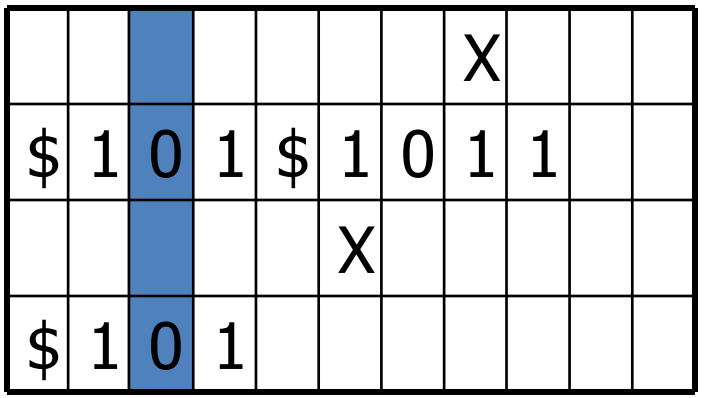




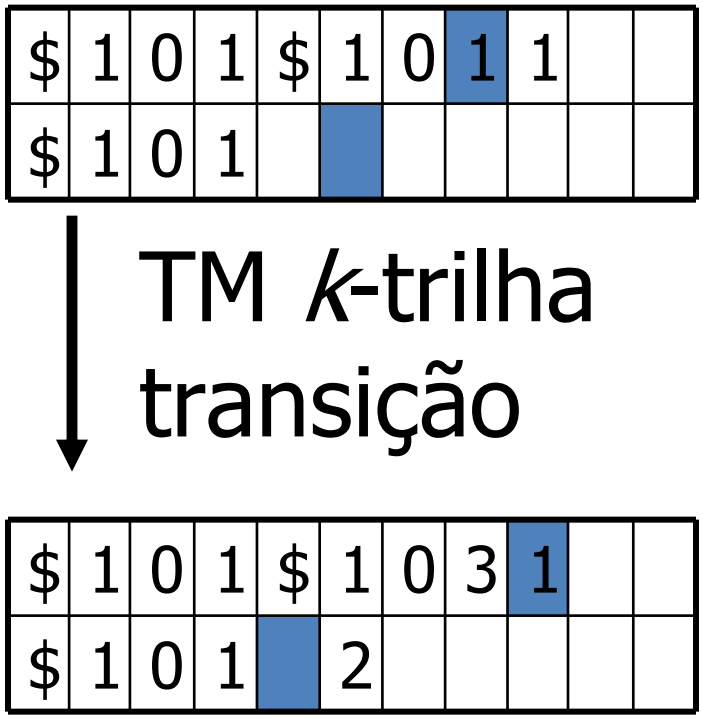
2*k*-trilhas Simulando *k*-fitas



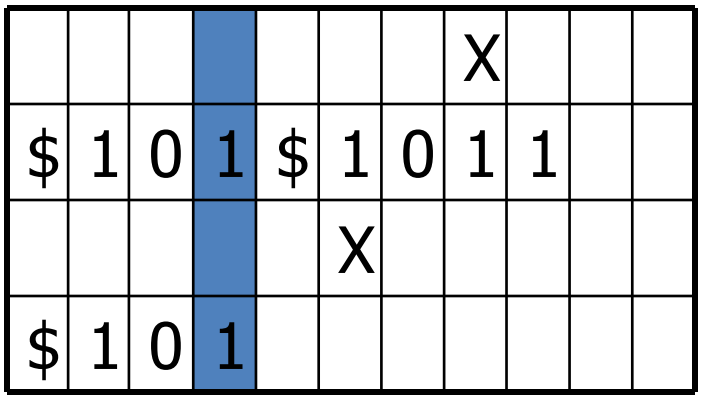
simulação por TM de 2k-trilhas



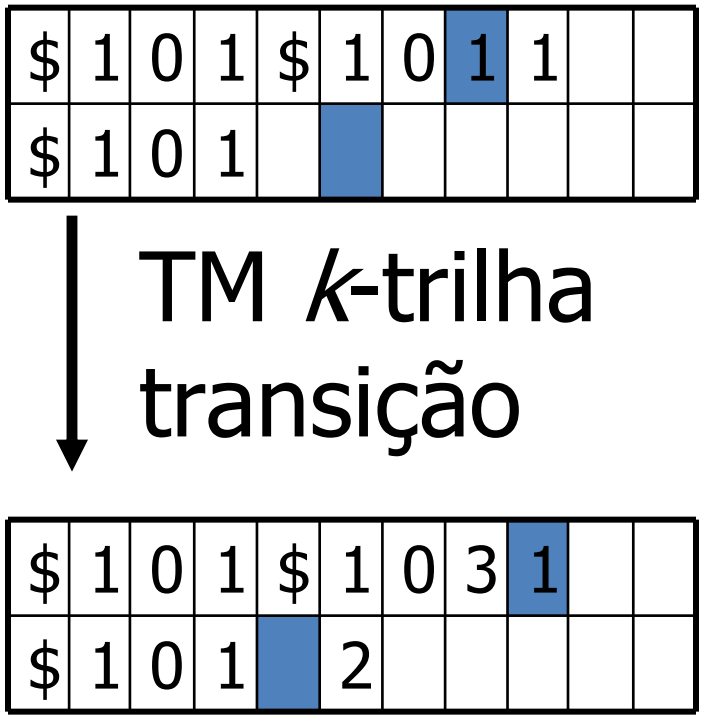
2*k*-trilhas Simulando *k*-fitas



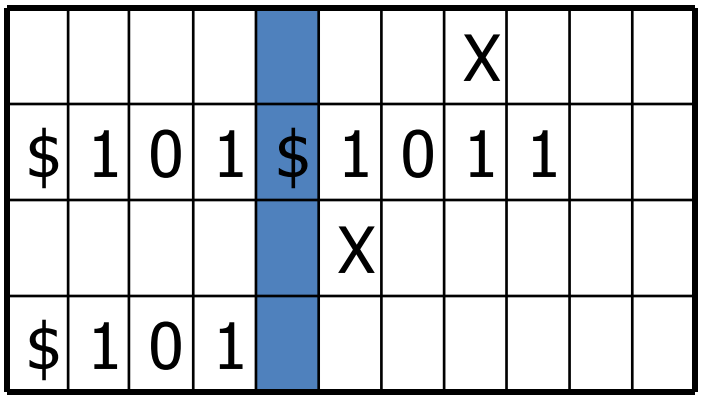
simulação por TM de 2k-trilhas



2*k*-trilhas Simulando *k*-fitas

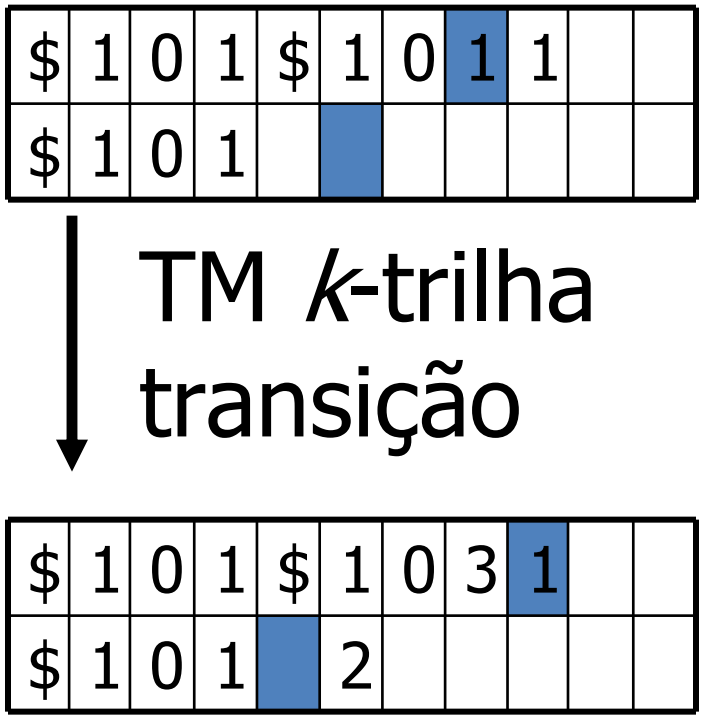


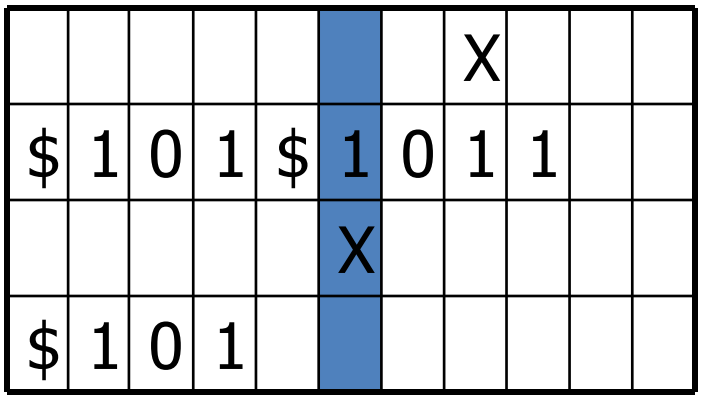
simulação por TM de 2k-trilhas



2*k*-trilhas Simulando *k*-fitas

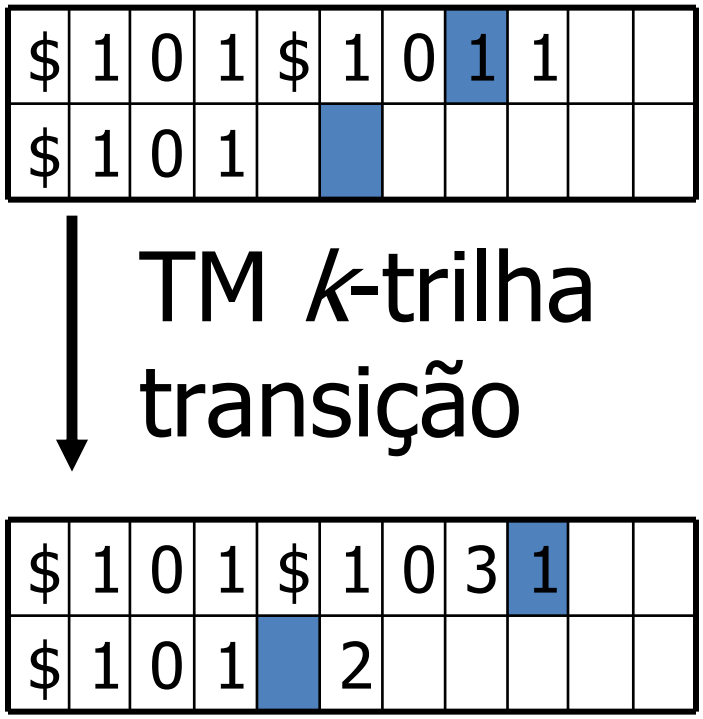
simulação por
TM de 2k-trilhas



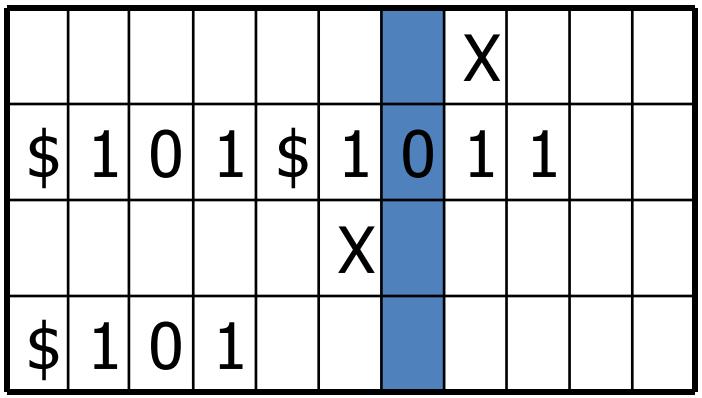


lê 2ª. fita

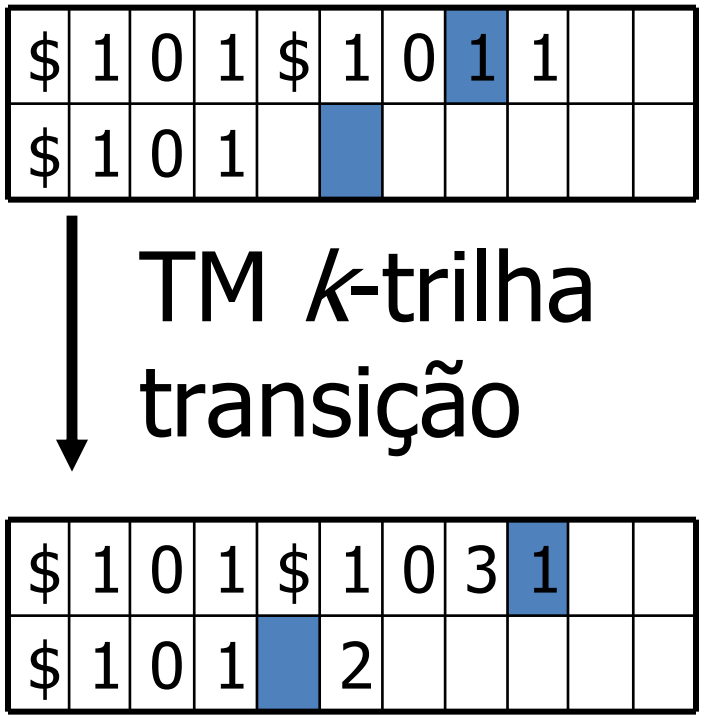
2*k*-trilhas Simulando *k*-fitas



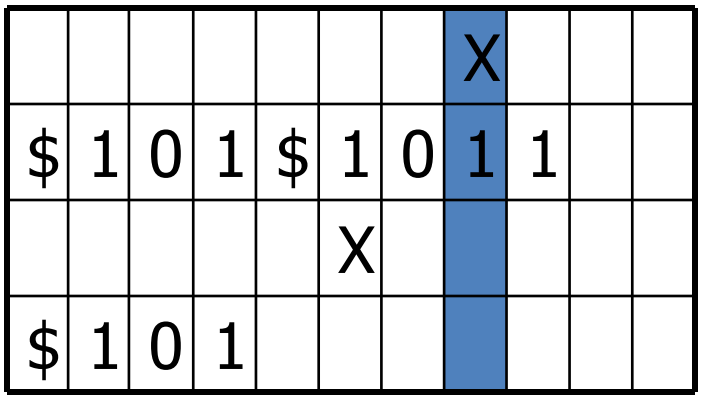
simulação por
TM de 2k-trilhas



2*k*-trilhas Simulando *k*-fitas



simulação por
TM de 2k-trilhas



lê 1ª. Fita
passou todos X's

2*k*-trilhas Simulando *k*-fitas

$	1	0	1	$	1	0	1	1		
$	1	0	1							

TM *k*-trilha
transição

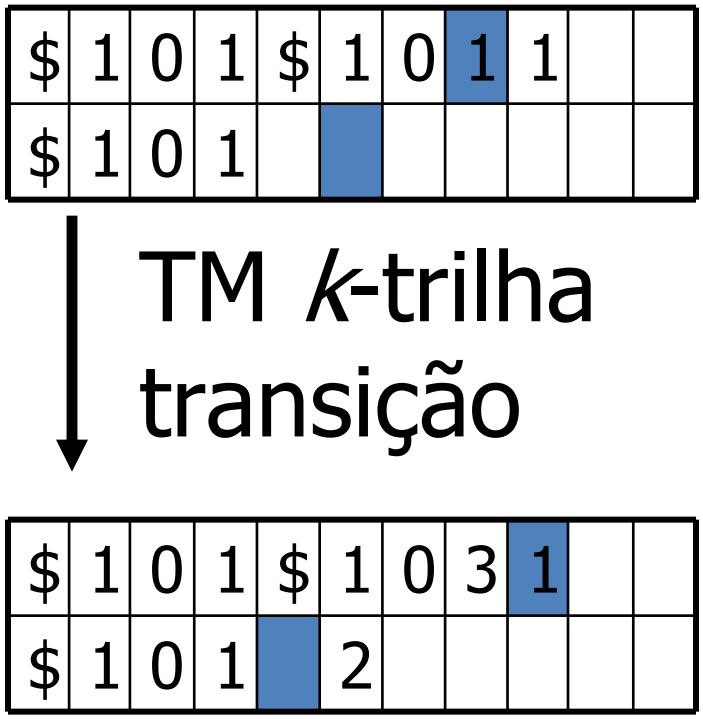
$	1	0	1	$	1	0	3	1		
$	1	0	1		2					

simulação por
TM de 2k-trilhas

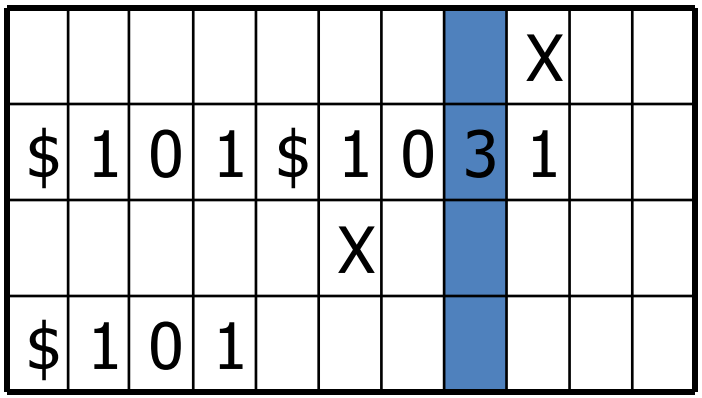
$	1	0	1	$	1	0	3	1		
					X					
$	1	0	1							

2ª. fase, na 1ª.
fita: 1→3,R

2*k*-trilhas Simulando *k*-fitas

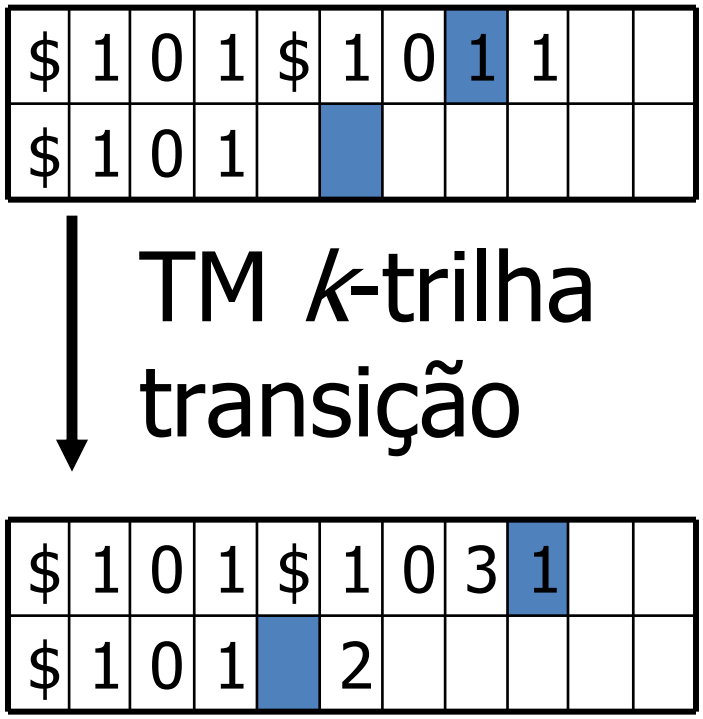


simulação por
TM de 2k-trilhas

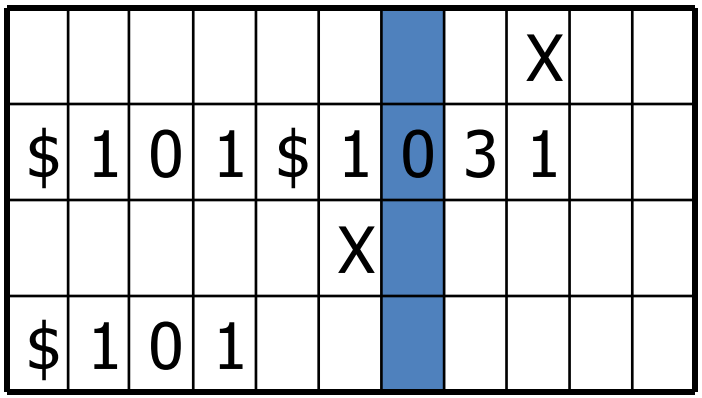


2ª. fase, na 1ª. fita:
1→3,R

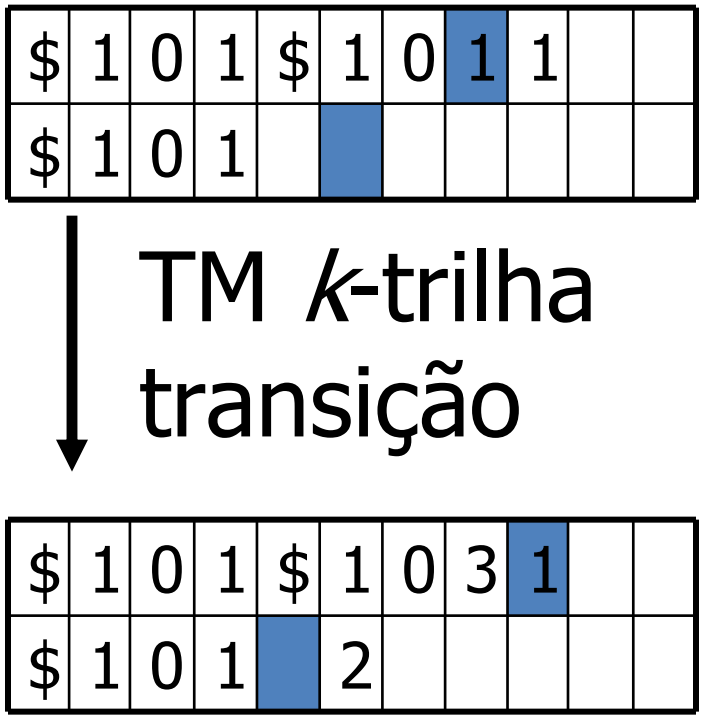
2*k*-trilhas Simulando *k*-fitas



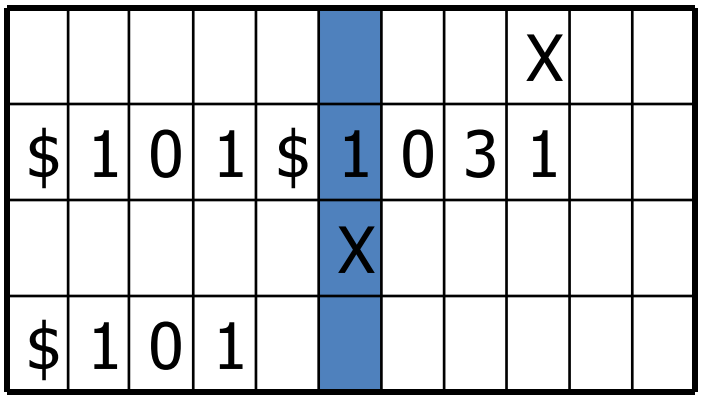
simulação por
TM de 2k-trilhas



2*k*-trilhas Simulando *k*-fitas

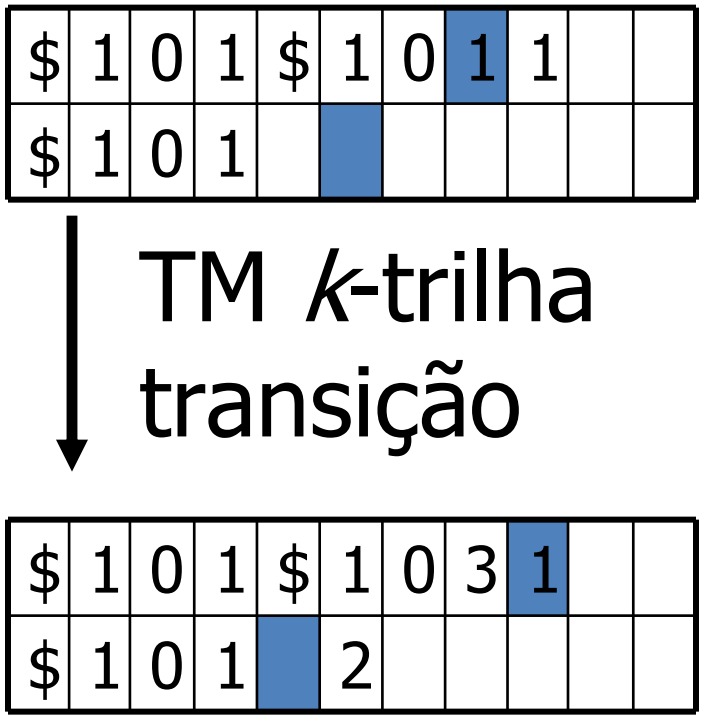


simulação por
TM de 2k-trilhas

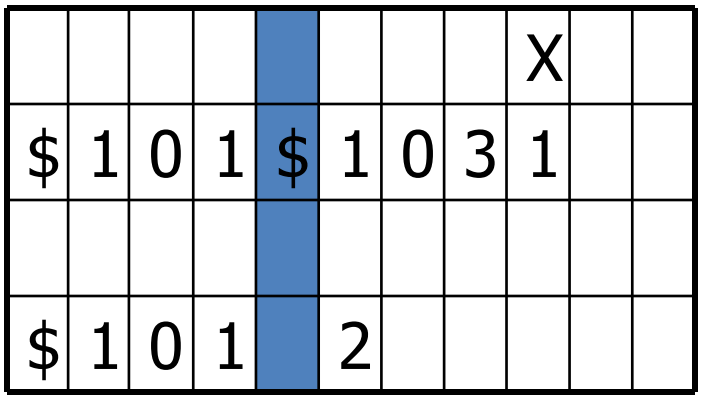


na 2nd fita: →2,L

2*k*-trilhas Simulando *k*-fitas

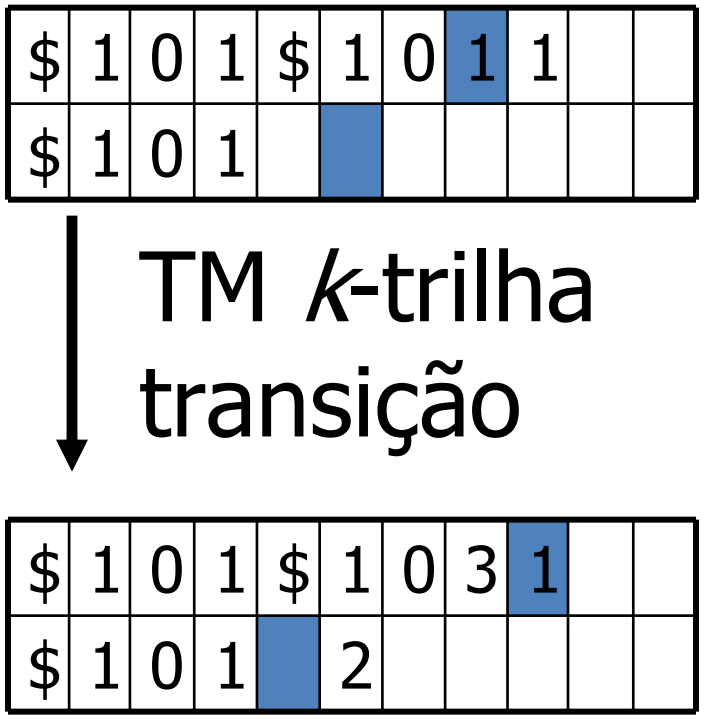


simulação por
TM de 2k-trilhas

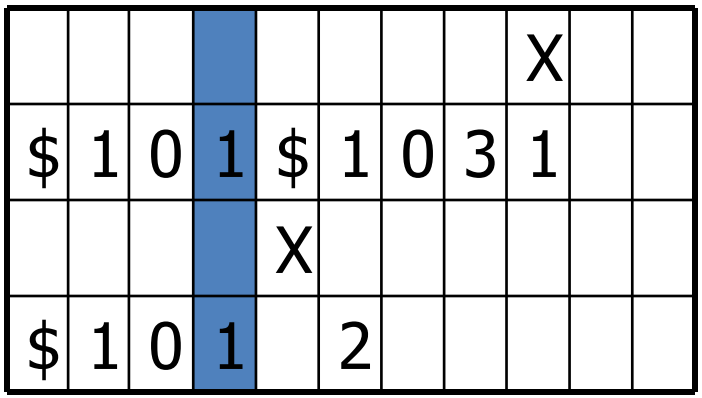


na 2nd fita: →2,L

2*k*-trilhas Simulando *k*-fitas



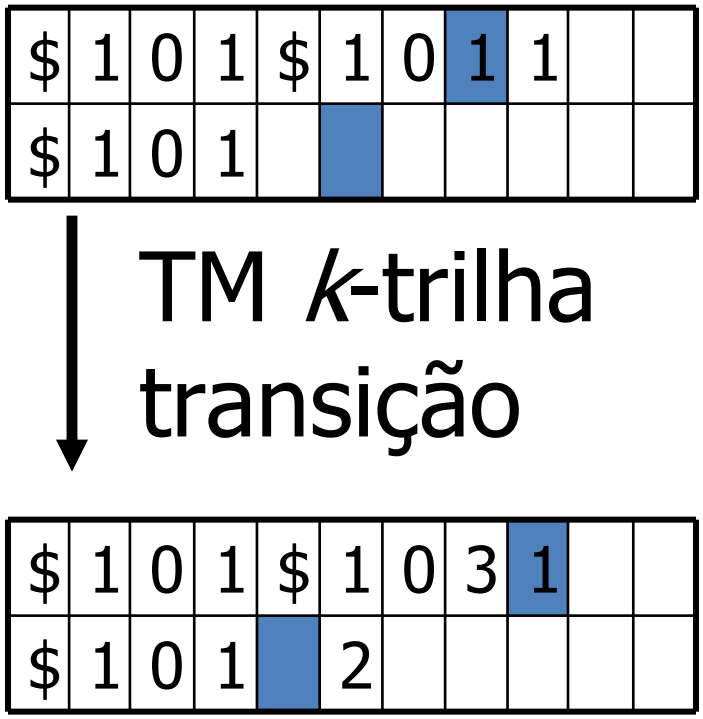
simulação por
TM de 2k-trilhas

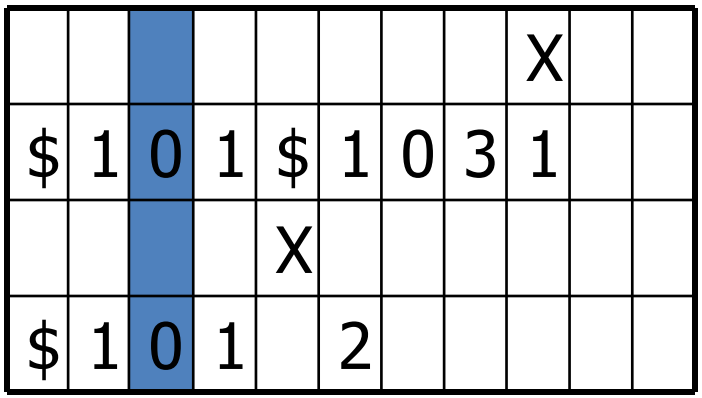


na 2nd fita: →2,L

2*k*-trilhas Simulando *k*-fitas

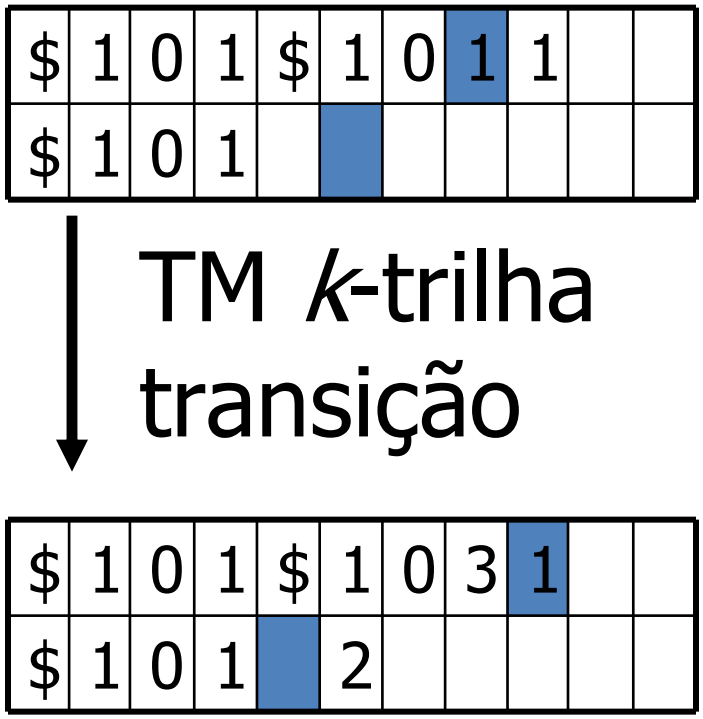
simulação por
TM de 2k-trilhas

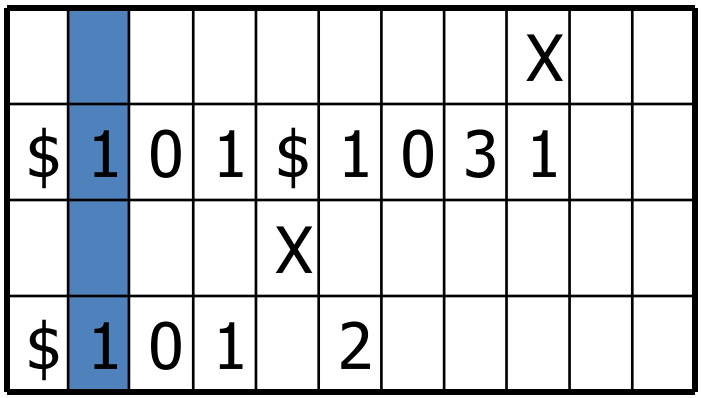




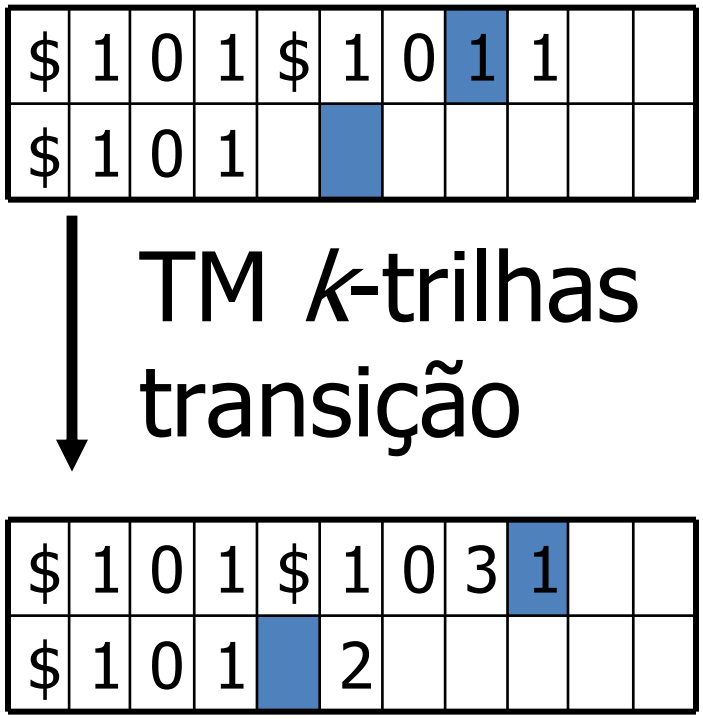
2*k*-trilhas Simulando *k*-fitas

simulação por
TM de 2k-trilhas

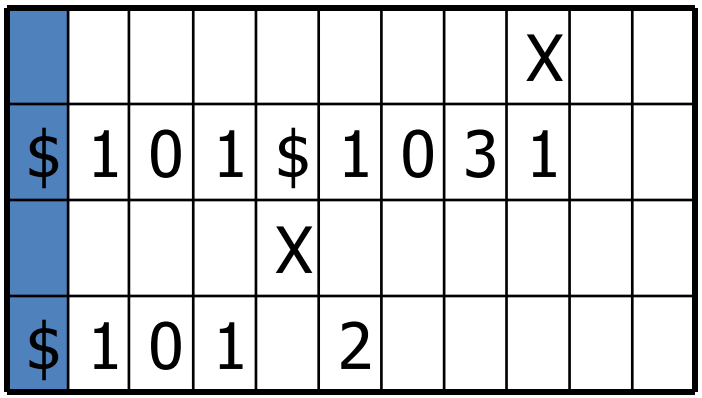




2*k*-trilhas Simulando *k*-fitas



simulação por
TM de 2k-trilhas



pronta p/ próxima
transição *k*-fita

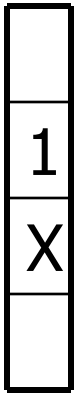
1 fita simulando *k*-trilhas

Finalmente, converter a máquina 2*k*-trilhas para uma máquina com uma única fita. De certo modo, já concluímos, já que há apenas uma cabeça de leitura.

Q: Como a informação de uma coluna de trilhas pode ser armazenada em uma única célula?

1 fita simulando *k*-trilhas

R: Basta expandir o alfabeto da fita Γ! Esse caso, o novo alfabeto consistiria de 4-tuplas, e a 4-tupla (,1,X,) representaria a coluna à direita.



Portanto, uma TM *k*-trilhas torna-se uma *TM 1-fita* se a enxergamos da maneira apropriada. Nenhuma modificação adicional é necessária.

Isso completa a simulação da M *k*-trilhas por uma TM 1-fita.

Convertendo uma NTM para Determinista

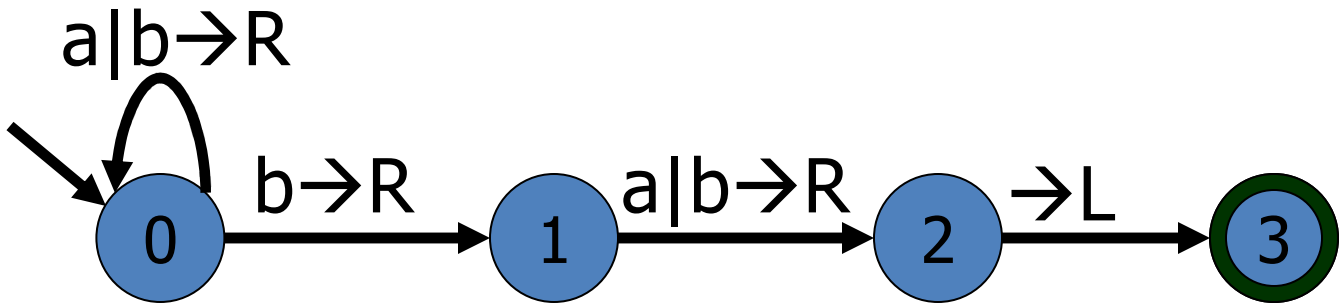
THM: Toda não determinista pode ser convertida em uma TM determinista equivalent.

A idéia da prova é a seguinte:

1) Simule a NTM por meio de uma TM 3-fitas determinista.
2) Os resultados anteriores garantem que podemos converter essa máquina em uma TM determinista de 1-fita.

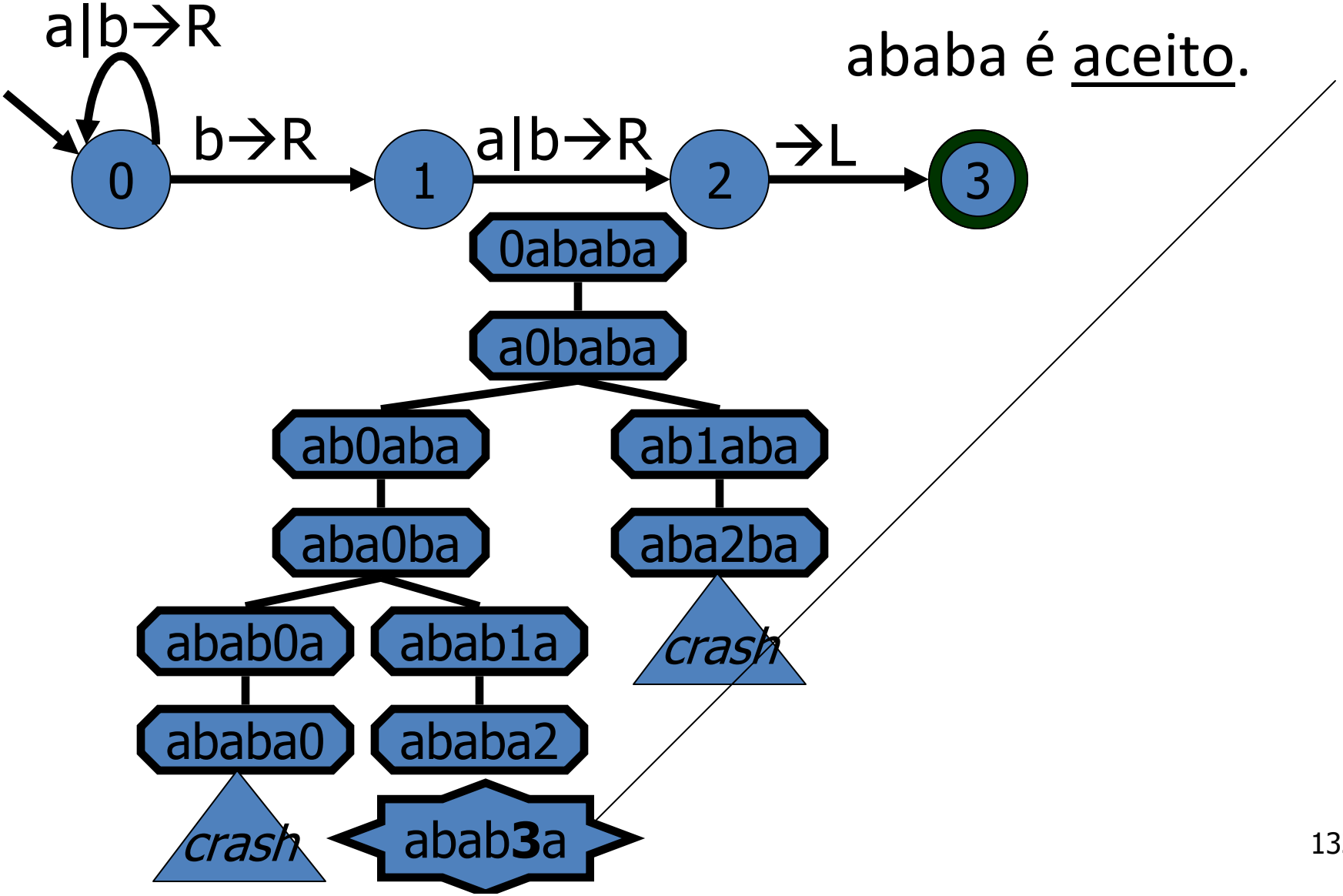
Convertendo uma NTM para Determinista

Considere a seguinte TM decisora para (a∪b)*b(a∪b):



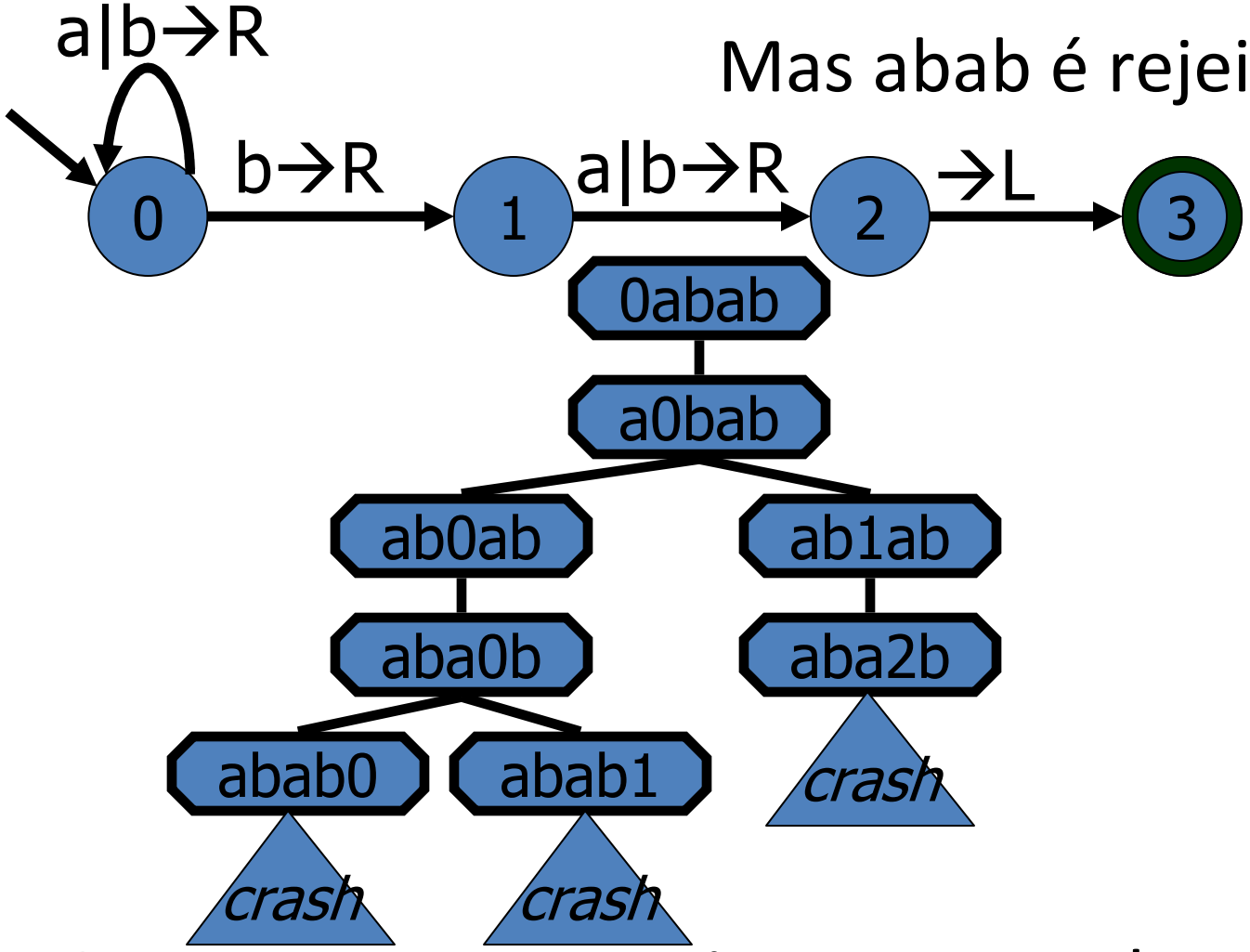
Para verificar se um string é aceito, precisamos ver se algum ramo da computação leva a uma configuração que contenha o estado “3”:

Convertendo uma NTM para Determinista



Convertendo uma NTM para Determinista

Mas abab é rejeitado:



Q: Como converter isso em um algoritmo?

Convertendo uma NTM para Determinista

R: Construa a árvore de computação da TM a partir da configuração inicial. E então verifique se ocorre alguma configuração de aceitação em algum ramo.

Q: Pesquisa em largura ou pesquisa em profundidade?

Convertendo uma NTM para Determinista

R: Pesquisa em largura! Se a NTM tem algum loop infinito, podemos prosseguir em um ramo infinito, se usarmos pesquisa em profundidade. Por outro lado, como cada nodo tem número finito de filhos, uma BFS garante que atingiremos uma configuração de aceitação, caso exista.

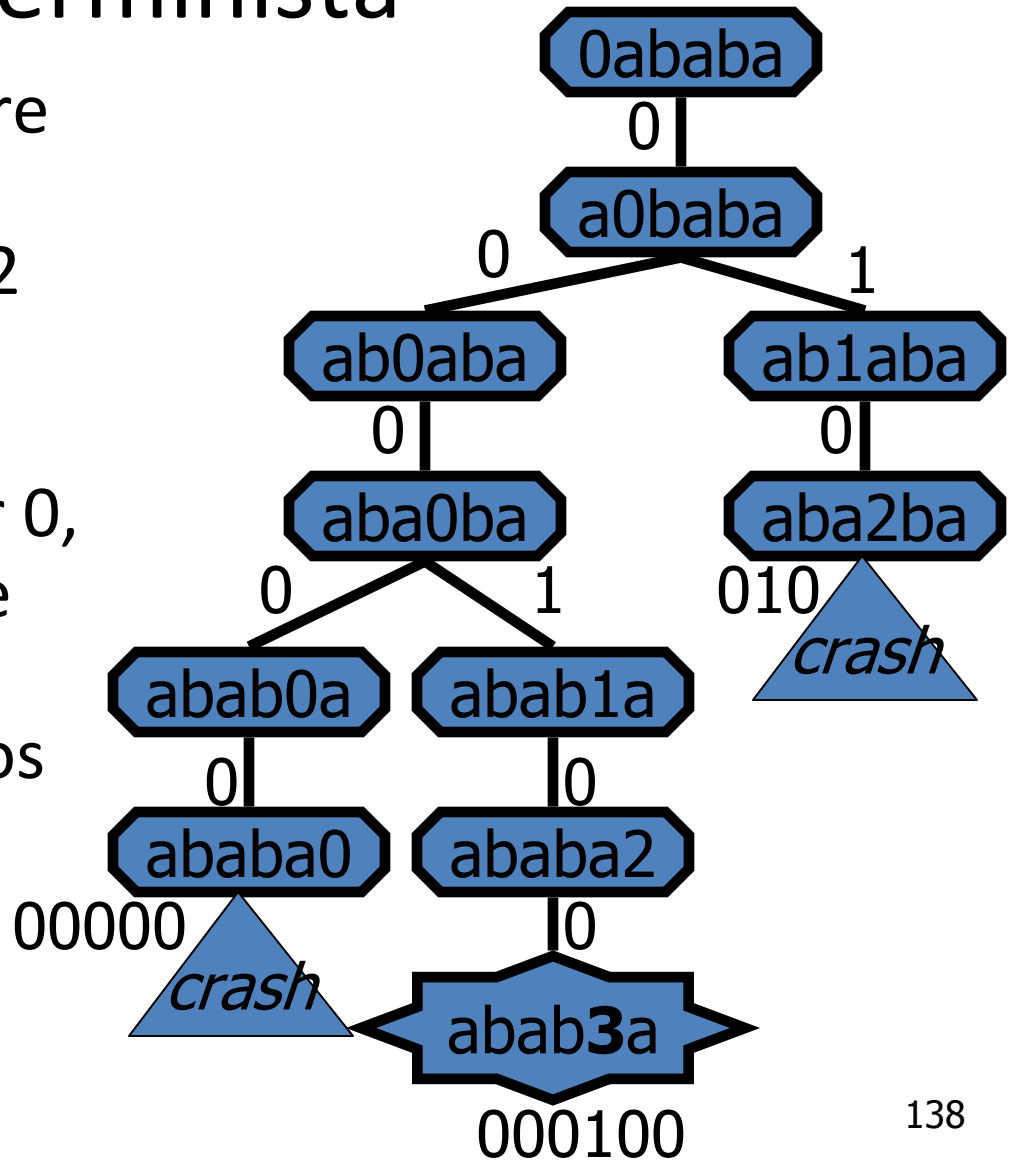
Convertendo uma NTM para Determinista

Construindo a árvore de computação BFS:

- Usamos uma TM 3-fitas
- A primeira fita apenas contém a entrada
- A segunda fita simula a computação em um ramo particular até determinada profundidade
- A última fita contém as instruções de desvio para o ramo de computação que está sendo simulado

Convertendo uma NTM para Determinista

Neste exemplo, a árvore de computação é binária: no máximo 2 escolhas em cada passo. Denote a primeira escolha por 0, e a segunda por 1 de modo a obter identificadores únicos para os ramos:



Convertendo uma NTM para Determinista

BFS prosseguiria do seguinte modo:

1. Tente 0, se configuração de aceitação, aceita.
2. Tente 1, se configuração de aceitação, aceita.
3. Tente 00, se configuração de aceitação, aceita.
4. Tente 01, se configuração de aceitação, aceita.
5. Tente 10, se configuração de aceitação, aceita.
6. Tente 11, se configuração de aceitação, aceita.
7. Tente 000, se configuração de aceitação, aceita.
8. Tente 001, se configuração de aceitação, aceita.

… No nosso caso, finalmente o string seria aceito em:

67. Tente 000100, ACEITA!

Convertendo uma NTM para Determinista

Para máquinas reconhecedoras, a conversão é desse modo.

Para máquinas decisoras, o lema de König implica que a árvore de computação é de fato finita, para qualquer entrada. Portanto poderíamos construir toda a árvore e dizer quando um string é rejeitado depois de examinar a árvore toda.