Máquinas de Turing

Agenda

- Máquinas de Turing (TM)
  - Alan Turing
  - Motivação
    - Tese de Church-Turing
  - Definições
    - Computação
    - Configuração de TM
    - Reconhecedores vs. Decisores

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...

10º. 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!

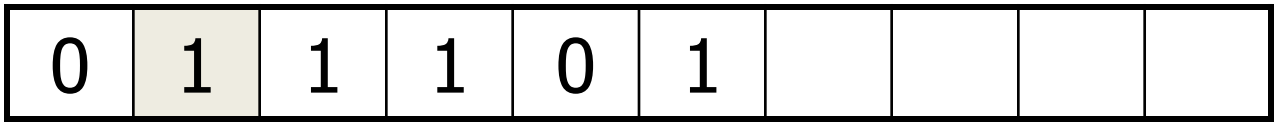
1	1	1	1	0	1				

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!

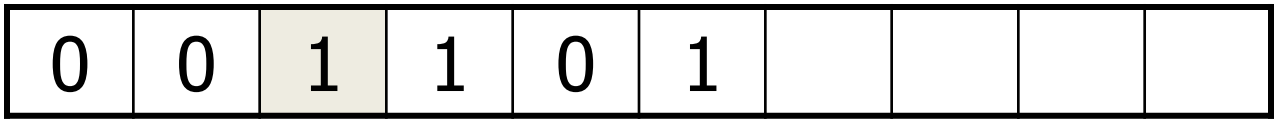


Uma Máquina Pensante
EX: Programa **`Successor`**

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!

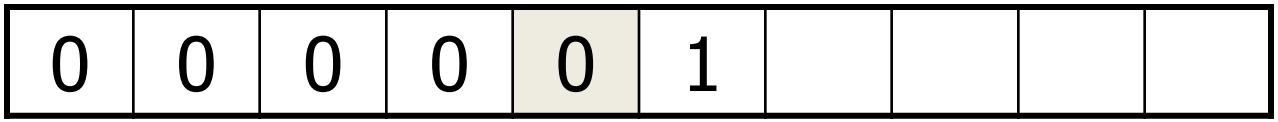
0	0	0	1	0	1				

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!

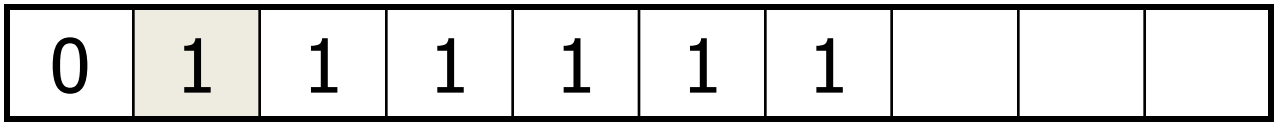
1	1	1	1	1	1	1			

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!



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!

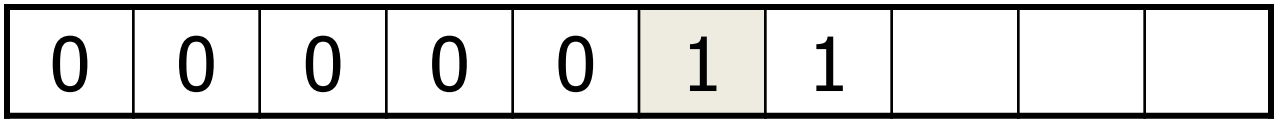
0	0	0	0	1	1	1			

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!



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!

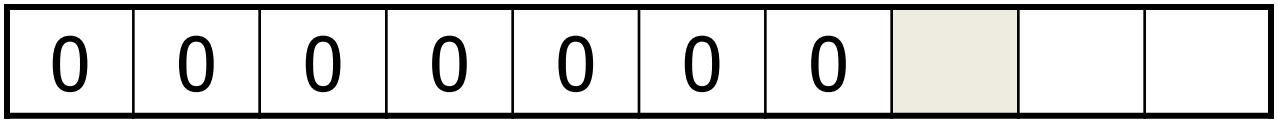
0	0	0	0	0	0	1			

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}

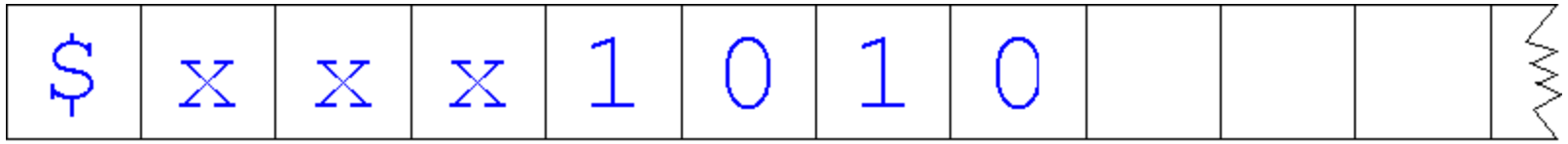
– uma linguagem livre de contexto

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:

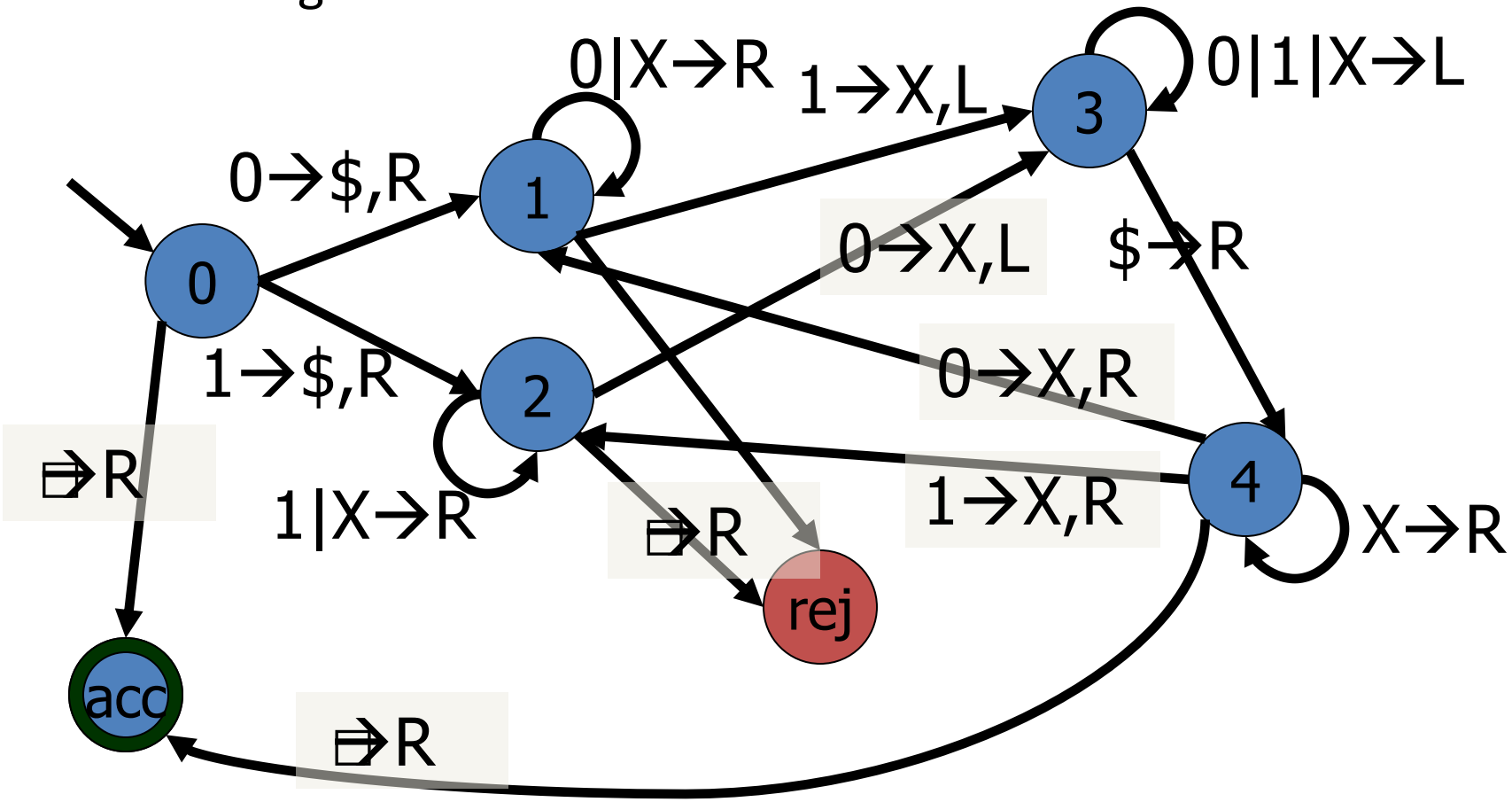
```
while (existe um 0 e um 1 na fita)
  marque esses dois símbolos
if (todo símbolo estiver marcado)
  aceita
else
  rejeita
```

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

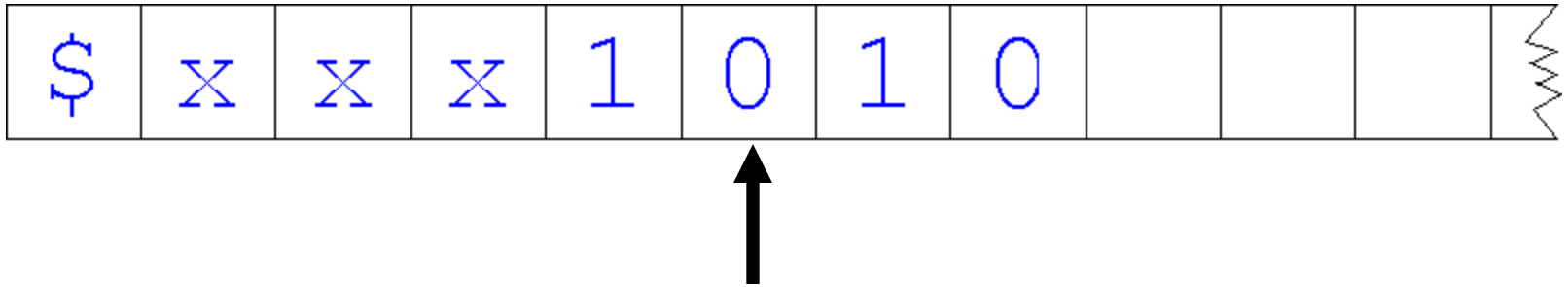
A próxima ação de uma TM é completamente determinada pelo estado corrente e pelo símbolo lido; portanto podemos prever ações futuras se sabemos:

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:

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

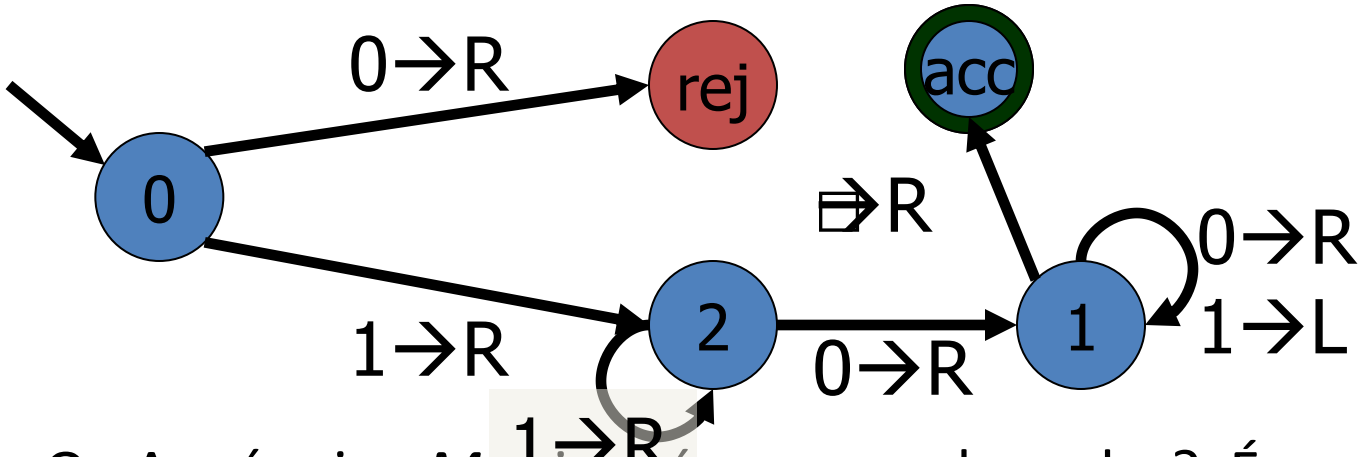
Reconhecedores vs. Decisores

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*** $L(M)$.

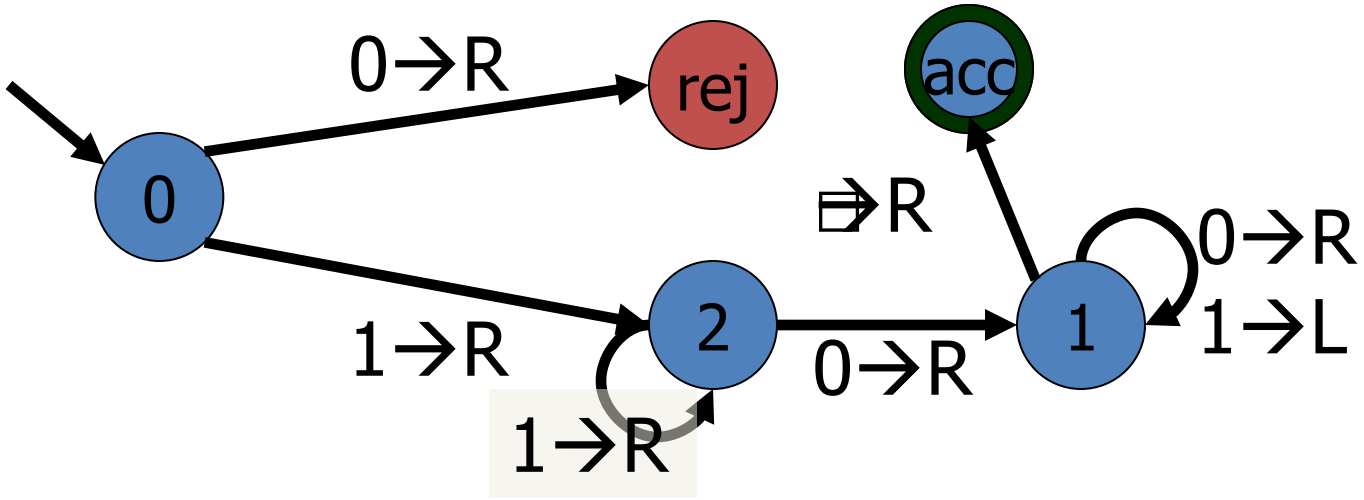


Q: A máquina M acima é um reconhecedor? É um decisor? O que é $L(M)$?

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:

- $\{w \in \{a, b\} \mid |w| \text{ é par e } w = w^R\}$
- $\{w \# w \mid w \in \{0,1\}^* \}$
- $\{a^i b^j c^k \mid i \times j = k \text{ e } i,j,k \geq 1 \}$
- $\{0^{2^n} \mid n \geq 0 \}$