CC1004 - Modelos de Computação
Teóricas 24 e 25

Ana Paula Tomás

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Junho 2021

Simulador de MTs – https://turingmachinesimulator.com

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

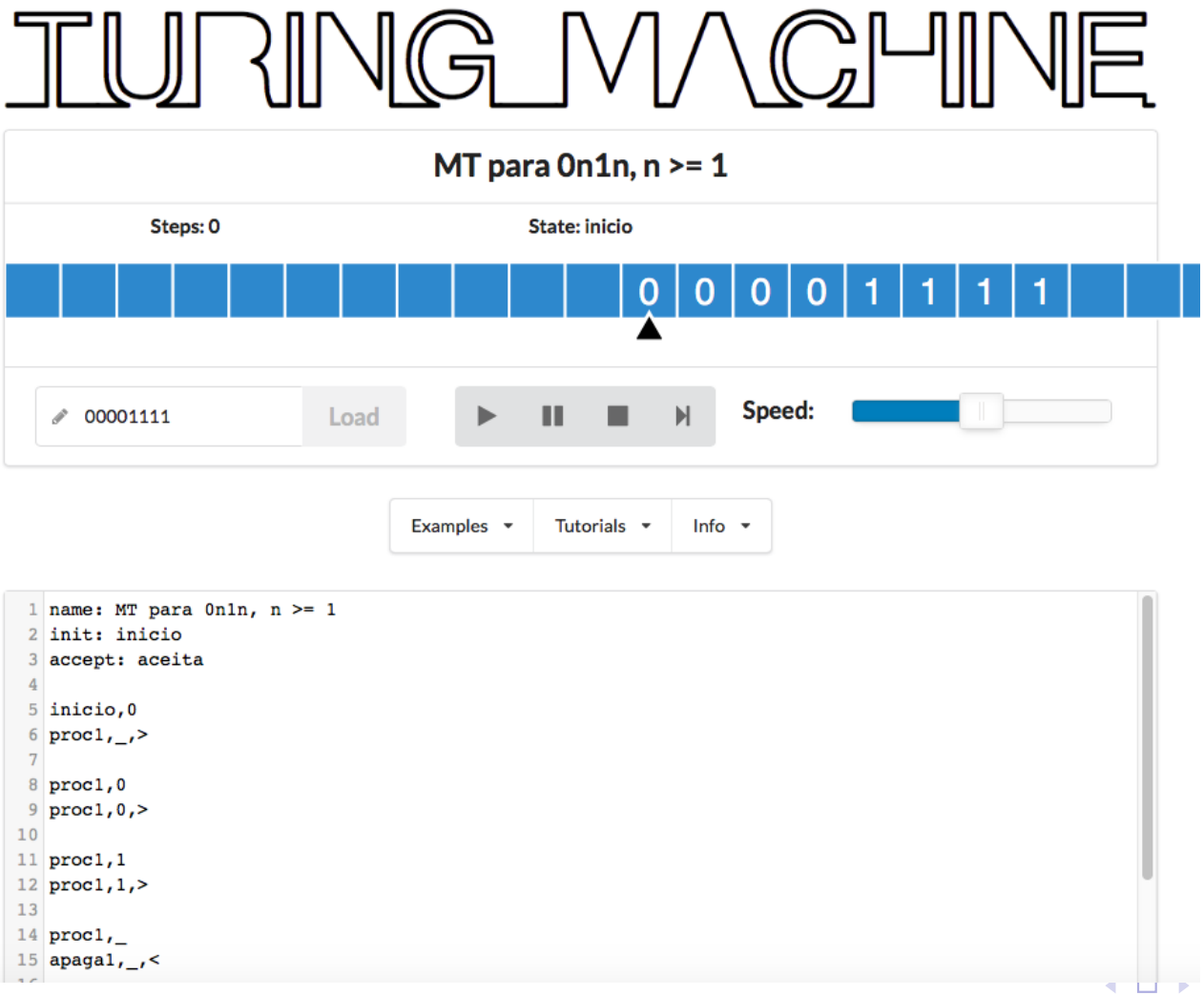
MT que aceita $\{0^n 1^n \mid n \geq 1\}$

Estado inicial `inicio` e conjunto de estados finais $F = \{\texttt{aceita}\}$; Branco $\bullet$.

$(inicio, 0, proc1, \bullet, d)$	apaga 0; procura o par, i.e., o 1 mais à direita
$(proc1, 0, proc1, 0, d)$	
$(proc1, 1, proc1, 1, d)$	
$(proc1, \bullet, apaga1, \bullet, e)$	encontra $\bullet$; à esquerda está 1?
$(apaga1, 1, proc0, \bullet, e)$	apaga o 1 mais à direita; procura 0 mais à esquerda
$(proc0, 0, proc0, 0, e)$	
$(proc0, 1, proc0, 1, e)$	
$(proc0, \bullet, apaga0, \bullet, d)$	encontra $\bullet$; à direita está 0?
$(apaga0, 0, proc1, \bullet, d)$	tem 0; procura o par
$(apaga0, \bullet, aceita, \bullet, e)$	não tem mais nada; pára em `aceita`

Simulador de MTs – https://turingmachinesimulator.com

Exemplos de MTs codificadas para este simulador estão disponíveis no Sigarra.



Tradução da MT:

O estado inicial e os finais são declarados no início.
Símbolo branco é _ (*underscore*).
Deslocamentos *e*, *d* e – são $<$, $>$ e $-$.

Cada transição é definida em duas linhas.

Código de MT para o simulador

```
name: MT para 0n1n, n >= 1
init: inicio
accept: aceita

inicio,0
proc1,_,>

proc1,0
proc1,0,>

proc1,1
proc1,1,>

proc1,_
apaga1,_,<

apaga1,1
proc0,_,<

proc0,0
proc0,0,<

proc0,1
proc0,1,<

proc0,_
apaga0,_,>

apaga0,0
proc1,_,>
apaga0,_
aceita,_,<
```

Lema da Repetição para LICs

Exemplos de linguagens que não são LICs:

$$\{a^n b^n c^n \mid n \in \mathbb{N}\} \qquad \{ww \mid w \in \{a, b\}^\star\}$$

$$\{a^p \mid p \text{ primo}\} \qquad \{a^{n^2} \mid n \in \mathbb{N}\}$$

Não satisfazem a condição do Lema da Repetição para LICs.

Lema da Repetição para LICs

Se L é uma LIC então existe uma constante $n \in \mathbb{Z}^+$, só dependente de L, tal que qualquer que seja $z \in L$, se $|z| \geq n$ então podemos escrever z como $uvwxy$ de forma que $|vx| \geq 1$, $|vwx| \leq n$ e, para todo $i \in \mathbb{N}$, se tem $uv^i wx^i y \in L$.

Aplicação do Lema da Repetição

Exemplo: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, escolhemos $z = a^n b^n c^n$.
- Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, pois como $|vwx| \leq n$, não temos simultaneamente a's, b's e c's em vwx.
- $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{a, b, c\}^*$ tais que

$$a^n b^n c^n = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq n \quad \wedge \quad \forall_{i \in \mathbb{N}}\, uv^i wx^i y \in L$$

- Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultaneamente a's, b's e c's.
- Assim, **qualquer** que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*i.e., se cortarmos* v e x), $uv^0wx^0y \notin L$ porque não tem igual número de a's, b's e c's. □

Aplicação do Lema da Repetição

Exemplo: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, escolhemos $z = a^n b^n c^n$.
- Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, pois como $|vwx| \leq n$, não temos simultaneamente a's, b's e c's em vwx.
- $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{a, b, c\}^\star$ tais que

$$a^n b^n c^n = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq n \quad \wedge \quad \forall_{i \in \mathbb{N}}\, uv^i wx^i y \in L$$

- Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultaneamente a's, b's e c's.
- Assim, **qualquer** que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*i.e., se cortarmos* v e x), $uv^0 wx^0 y \notin L$ porque não tem igual número de a's, b's e c's. □

Aplicação do Lema da Repetição

Exemplo: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, escolhemos $z = a^n b^n c^n$.
- Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, pois como $|vwx| \leq n$, não temos simultaneamente a's, b's e c's em vwx.
- $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{a, b, c\}^\star$ tais que

$$a^n b^n c^n = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq n \quad \wedge \quad \forall_{i \in \mathbb{N}}\, uv^i wx^i y \in L$$

- Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultanemente a's, b's e c's.
- Assim, **qualquer** que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*i.e., se cortarmos* v e x), $uv^0 wx^0 y \notin L$ porque não tem igual número de a's, b's e c's. □

Aplicação do Lema da Repetição

Exemplo: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, escolhemos $z = a^n b^n c^n$.
- Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, pois como $|vwx| \leq n$, não temos simultaneamente a's, b's e c's em vwx.
- $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{a, b, c\}^\star$ tais que

$$a^n b^n c^n = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq n \quad \wedge \quad \forall_{i \in \mathbb{N}}\, uv^i wx^i y \in L$$

- Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultanemente a's, b's e c's.
- Assim, **qualquer** que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*i.e., se cortarmos* v e x), $uv^0 wx^0 y \notin L$ porque não tem igual número de a's, b's e c's. □

Aplicação do Lema da Repetição

Exemplo: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, escolhemos $z = a^n b^n c^n$.
- Esta escolha simplifica o tipo de subpalavras que vamos ter que analisar, pois como $|vwx| \leq n$, não temos simultaneamente a's, b's e c's em vwx.
- $z \in L$ e $|z| = 3n \geq n$. Não existem $u, v, w, x, y \in \{a, b, c\}^\star$ tais que

$$a^n b^n c^n = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq n \quad \wedge \quad \forall_{i \in \mathbb{N}}\, uv^i wx^i y \in L$$

- Como já observámos, para se ter $|vwx| \leq n$, em vwx não podem existir simultanemente a's, b's e c's.
- Assim, **qualquer** que seja a decomposição de z como $uvwxy$ com $|vx| \neq 0$ e $|vwx| \leq n$, para $i = 0$ (*i.e., se cortarmos* v e x), $uv^0wx^0y \notin L$ porque não tem igual número de a's, b's e c's. □

Ideia da Prova do Lema

Prova

- Seja $\mathcal{G}$ uma GIC na FN Chomsky (estendida) que gere L. Tome-se $n = 2^{|V|}$, para mostrar que qualquer sequência $z \in L$, com $|z| \geq n$, satisfaz as condições indicadas no lema.
- As árvores de derivação em G são árvores binárias. A altura de uma árvore binária com k nós é maior ou igual a $\lfloor \log_2 k \rfloor$.
- Como os terminais são introduzidos por regras $A \to a$, a árvore de derivação de z terá de ter pelo menos n nós internos se $|z| \geq n$. Isso implica que a altura da árvore seja pelo menos $\lfloor \log_2 n \rfloor + 1 \geq |V| + 1$.
- Como a **altura** é o comprimento máximo que os caminhos desde a raíz até às folhas podem ter, se o caminho máximo tem comprimento $\geq |V| + 1$ então envolve pelo menos $|V| + 1$ variáveis até ao pai da folha. Portanto, há uma variável que se repete.

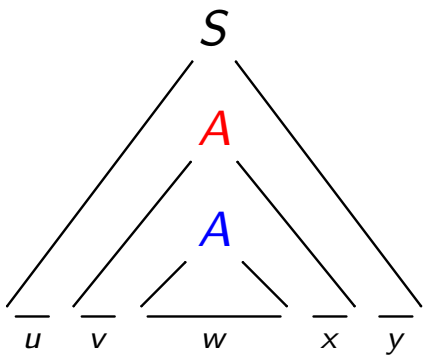
Observação: A FN Chomsky estendida permite ter a regra $S \to \varepsilon$, se o símbolo inicial S não ocorrer no lado direito de regras.

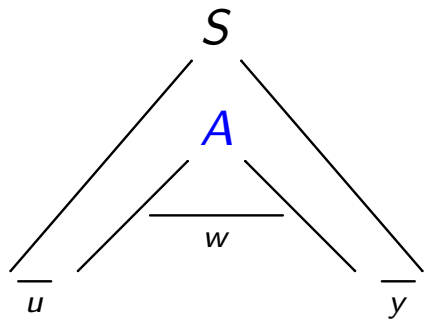
Ideia da Prova do Lema

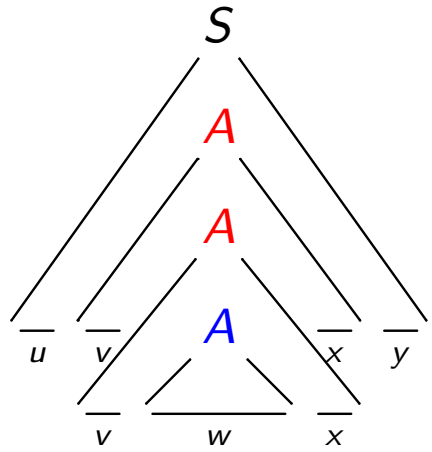
- Seja A a última variável que se repete no caminho (i.e., a que se repete em níveis mais profundos). Dessa forma, podemos concluir que árvore para z inclui uma subárvore A_1 com raíz A, que gera vwx com $|vwx| \leq n$ e $vx \neq \varepsilon$, e em que w é gerada por uma subárvore A_2 também com raíz A. Se não se verificasse $|vwx| \leq n$, a árvore A_1 continha outras repetições mais profundas. Podemos usar A_1 e A_2 para mostrar $uv^i wx^i y \in L$, para todo $i \geq 0$. Se $i = 0$, transforma-se a árvore substituindo A_1 por A_2, o que dá origem a uma árvore de derivação para $uv^0 wx^0 y$. Se $i \geq 2$, substitui-se A_2 por A_1, sucessivamente, obtendo $uv^i wx^i y$ após i substituições.

Ideia da Prova do Lema

- Seja A a última variável que se repete no caminho (i.e., a que se repete em níveis mais profundos). Dessa forma, podemos concluir que árvore para z inclui uma subárvore A_1 com raíz A, que gera vwx com $|vwx| \leq n$ e $vx \neq \varepsilon$, e em que w é gerada por uma subárvore A_2 também com raíz A. Se não se verificasse $|vwx| \leq n$, a árvore A_1 continha outras repetições mais profundas. Podemos usar A_1 e A_2 para mostrar $uv^i wx^i y \in L$, para todo $i \geq 0$. Se $i = 0$, transforma-se a árvore substituindo A_1 por A_2, o que dá origem a uma árvore de derivação para $uv^0 wx^0 y$. Se $i \geq 2$, substitui-se A_2 por A_1, sucessivamente, obtendo $uv^i wx^i y$ após i substituições.







Aplicação do Lema da Repetição

Exemplo: $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. Para esta palavra, vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx, pois $|vwx| \leq n$.
- Em todos os casos, para $i = 0$, tem-se $uv^i wx^i y \notin L$.
- Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \quad \text{e} \quad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Aplicação do Lema da Repetição

Exemplo: $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. Para esta palavra, vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx, pois $|vwx| \leq n$.
- Em todos os casos, para $i = 0$, tem-se $uv^i wx^i y \notin L$.
- Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \quad \text{e} \quad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Aplicação do Lema da Repetição

Exemplo: $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. Para esta palavra, vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx, pois $|vwx| \leq n$.
- Em todos os casos, para $i = 0$, tem-se $uv^iwx^iy \notin L$.
- Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v}w\underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \quad \text{e} \quad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v}w\underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Aplicação do Lema da Repetição

Exemplo: $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. Para esta palavra, vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx, pois $|vwx| \leq n$.
- Em todos os casos, para $i = 0$, tem-se $uv^i wx^i y \notin L$.
- Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \quad \text{e} \quad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v} w \underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Aplicação do Lema da Repetição

Exemplo: $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

Prova: Vamos ver que L não satisfaz a condição do lema da repetição para LICs.

- Dado $n \geq 1$, tomamos $z = 0^{2n}1^{2n}0^{2n}1^{2n}$. Para esta palavra, vwx abrange no máximo dois blocos da palavra, qualquer que seja vwx, pois $|vwx| \leq n$.
- Em todos os casos, para $i = 0$, tem-se $uv^iwx^iy \notin L$.
- Por exemplo, se v for subpalavra do primeiro bloco de 0's, temos duas possibilidades:

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v}w\underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y} \quad \text{e} \quad z = \underbrace{0^{|u|}}_{u}\underbrace{0^{|v|}}_{v}w\underbrace{0^{|x|}}_{x}\underbrace{y'1^{2n}0^{2n}1^{2n}}_{y}$$

No primeiro caso, $uv^0wx^0y = 0^{|u|}wy'0^{2n}1^{2n} = 0^{2n-|v|}1^{2n-|x|}0^{2n}1^{2n} \notin L$ porque, se dividirmos a palavra ao meio, a primeira metade termina em 0 e segunda termina em 1. No segundo, $uv^0wx^0y = 0^{2n-|v|-|x|}1^{2n}0^{2n}1^{2n} \notin L$, porque depois do meio da palavra há mais 0's do que na primeira metade.

Aplicação do Lema da Repetição

Exemplo (cont): $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

- Do mesmo modo, se v tiver 0's do primeiro bloco e algum 1 do segundo, então x só poderá ter 1's do segundo bloco, sendo

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^k 1^{|v|-k}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y' 0^{2n} 1^{2n}}_{y}$$

  para algum k, e $|u| > n$. Se cortarmos v e x, a palavra uv^0wx^0y tem mais 0's no segundo bloco de 0's do que no primeiro.

- Os restantes casos podem ser analisados de forma análoga para concluir que, qualquer que seja a decomposição de z na forma $uvwxy$, com $|vwx| \leq n$ e $vx \neq \varepsilon$, tem-se $uv^0wx^0y \notin L$. □

Aplicação do Lema da Repetição

Exemplo (cont): $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

- Do mesmo modo, se v tiver 0's do primeiro bloco e algum 1 do segundo, então x só poderá ter 1's do segundo bloco, sendo

$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^k 1^{|v|-k}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y}$$

  para algum k, e $|u| > n$. Se cortarmos v e x, a palavra uv^0wx^0y tem mais 0's no segundo bloco de 0's do que no primeiro.

- Os restantes casos podem ser analisados de forma análoga para concluir que, qualquer que seja a decomposição de z na forma $uvwxy$, com $|vwx| \leq n$ e $vx \neq \varepsilon$, tem-se $uv^0wx^0y \notin L$. □

Aplicação do Lema da Repetição

Exemplo (cont): $L = \{ww \mid w \in \{0,1\}^*\}$ não é LIC.

- Do mesmo modo, se v tiver 0's do primeiro bloco e algum 1 do segundo, então x só poderá ter 1's do segundo bloco, sendo

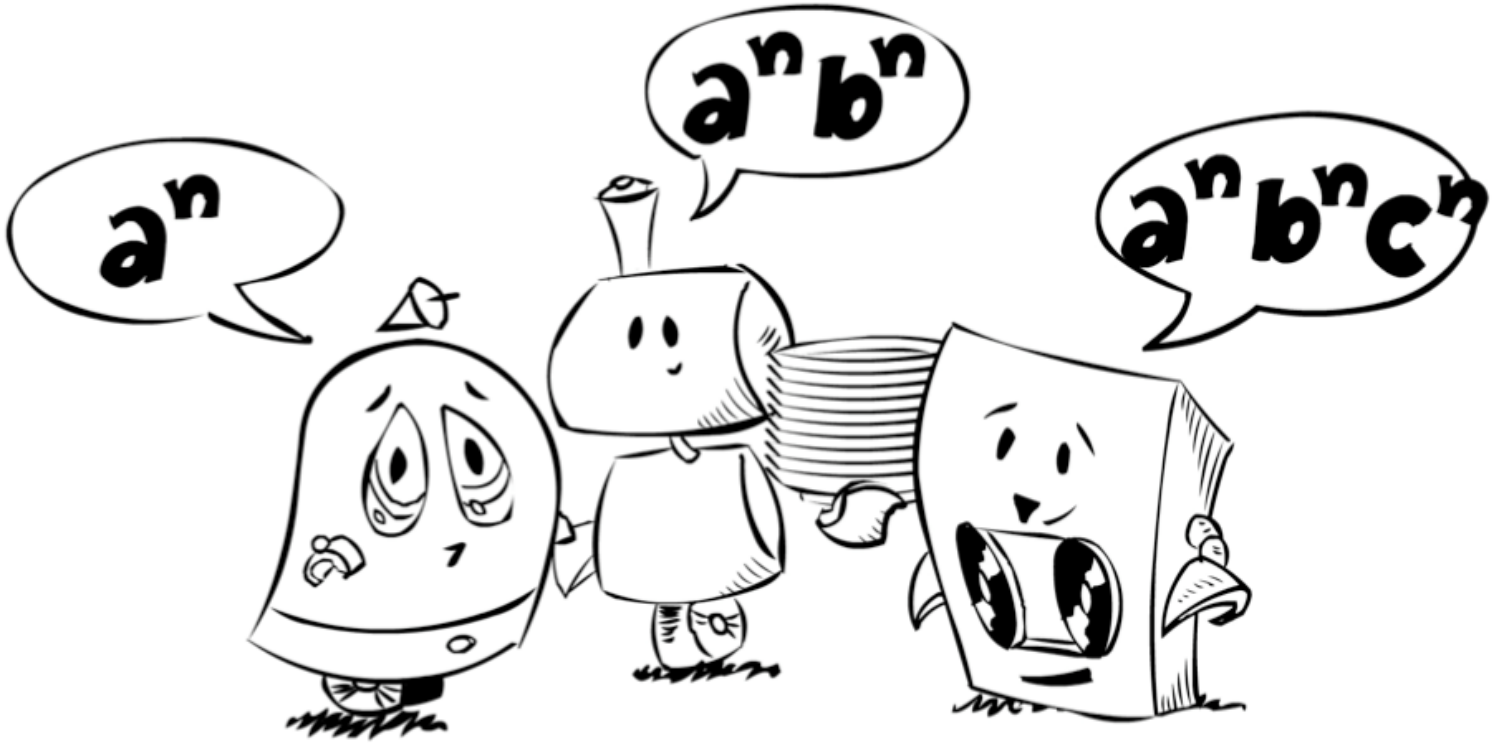
$$z = \underbrace{0^{|u|}}_{u}\underbrace{0^k 1^{|v|-k}}_{v} w \underbrace{1^{|x|}}_{x}\underbrace{y'0^{2n}1^{2n}}_{y}$$

  para algum k, e $|u| > n$. Se cortarmos v e x, a palavra uv^0wx^0y tem mais 0's no segundo bloco de 0's do que no primeiro.

- Os restantes casos podem ser analisados de forma análoga para concluir que, qualquer que seja a decomposição de z na forma $uvwxy$, com $|vwx| \leq n$ e $vx \neq \varepsilon$, tem-se $uv^0wx^0y \notin L$. □

Linguagens Formais e Computabilidade

Área de TCS. Limites da computação. Computabilidade: **existe algoritmo para resolução do problema?** Complexidade: **Que recursos requer** (tempo e espaço)? Como se **descreve o problema**? Que **tipo de máquina** usará?



Fonte:http://www.ic.uff.br/~ueverton/files/LF/aula09.pdf

Linguagens reconhecidas por máquinas de Turing

Máquina de Turing como reconhecedor

- Uma **uma máquina de Turing aceita uma palavra x de $\Sigma^\star$** se, partindo do estado inicial, com x na fita e a cabeça de leitura/escrita posicionada no símbolo de x mais à esquerda, pode **parar num estado final**.

  A linguagem $\mathcal{L}(M)$ reconhecida pela máquina M é o conjunto das palavras de $\Sigma^\star$ que M aceita.

- Uma linguagem $L \subseteq \Sigma^\star$ diz-se **decidível** (ou **recursiva**) sse existir uma máquina de Turing $\mathcal{M}$ tal que $L = \mathcal{L}(\mathcal{M})$ e $\mathcal{M}$ **pára**, para todo $x \in \Sigma^\star$, e o estado em que pára é final se e só se $x \in L$.

- L diz-se **semi-decidível** ou **recursivamente enumerável** se e só se existir uma máquina de Turing $\mathcal{M}$ tal que $L = \mathcal{L}(\mathcal{M})$. Dado $x \in \Sigma^\star \setminus L$, a máquina pode parar ou não parar, mas pára para todo $x \in L$.

Linguagens decidíveis e recursivamente enumeráveis

- A classe das **linguagens independentes de contexto** está propriamente contida na classe das **linguagens decidíveis**.

  O algoritmo CYK é um **algoritmo de decisão** para "$x \in \mathcal{L}(G)$?", se G está na FN Chomsky.

  Exemplos de linguagens decidíveis que não são LICs

  $$\{0^p \mid p \text{ primo}\}$$

  $$\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$$

- A classe das **linguagens decidíveis** está propriamente contida na classe das **linguagens recursivamente enumeráveis**.

Linguagens decidíveis e recursivamente enumeráveis

NB: Slide não apresentado nas aulas.

Seja Σ um alfabeto. *Verdade* ou *Falso*?

1. $\forall L \subseteq \Sigma^\star$, se L é decidível então $\overline{L}$ é decidível. V
2. $\forall L \subseteq \Sigma^\star$, se L e $\overline{L}$ são recursivamente enumeráveis, L é decidível. V
3. $\forall L, M \subseteq \Sigma^\star$, se L e M são decidíveis, $L \cup M$ é decidível. V

Funções calculadas por máquinas de Turing

Máquina de Turing como computador

- As funções parciais dos naturais nos naturais que podem ser **calculadas** por máquinas de Turing designam-se por **funções parcialmente recursivas**.
- Se $f(n)$ puder ser calculado por uma máquina de Turing que **pára**, para todo o *input* n, a função f diz-se **função recursiva** ou **computável**.

Exemplos de problemas indecidíveis

Existem muitos problemas que não podem ser resolvidos por MTs. Se aceitarmos a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver, isso significa que não podem ser resolvidos computacionalmente.

Exemplos:

- Existe um algoritmo para determinar o AFD mínimo que é equivalente a um dado autómato finito. Consequentemente, existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

  Mas, não existe um algoritmo que determine se duas GICs $\mathcal{G}$ e $\mathcal{G}'$ quaisquer são ou não são equivalentes, ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

- Não existe um algoritmo para decidir se dois autómatos de pilha quaisquer reconhecem a mesma linguagem.

- Também, o problema de decidir se uma GIC $\mathcal{G}$ arbitrária gera $\Sigma^\star$ indecidível.

Exemplos de problemas indecidíveis

Existem muitos problemas que não podem ser resolvidos por MTs. Se aceitarmos a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver, isso significa que não podem ser resolvidos computacionalmente.

Exemplos:

- Existe um algoritmo para determinar o AFD mínimo que é equivalente a um dado autómato finito. Consequentemente, existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

  Mas, não existe um algoritmo que determine se duas GICs $\mathcal{G}$ e $\mathcal{G}'$ quaisquer são ou não são equivalentes, ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

- Não existe um algoritmo para decidir se dois autómatos de pilha quaisquer reconhecem a mesma linguagem.

- Também, o problema de decidir se uma GIC $\mathcal{G}$ arbitrária gera $\Sigma^\star$ indecidível.

Exemplos de problemas indecidíveis

Existem muitos problemas que não podem ser resolvidos por MTs. Se aceitarmos a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver, isso significa que não podem ser resolvidos computacionalmente.

Exemplos:

- Existe um algoritmo para determinar o AFD mínimo que é equivalente a um dado autómato finito. Consequentemente, existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

  Mas, não existe um algoritmo que determine se duas GICs $\mathcal{G}$ e $\mathcal{G}'$ quaisquer são ou não são equivalentes, ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

- Não existe um algoritmo para decidir se dois autómatos de pilha quaisquer reconhecem a mesma linguagem.

- Também, o problema de decidir se uma GIC $\mathcal{G}$ arbitrária gera $\Sigma^\star$ indecidível.

Exemplos de problemas indecidíveis

Existem muitos problemas que não podem ser resolvidos por MTs. Se aceitarmos a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver, isso significa que não podem ser resolvidos computacionalmente.

Exemplos:

- Existe um algoritmo para determinar o AFD mínimo que é equivalente a um dado autómato finito. Consequentemente, existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

  Mas, não existe um algoritmo que determine se duas GICs $\mathcal{G}$ e $\mathcal{G}'$ quaisquer são ou não são equivalentes, ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

- Não existe um algoritmo para decidir se dois autómatos de pilha quaisquer reconhecem a mesma linguagem.

- Também, o problema de decidir se uma GIC $\mathcal{G}$ arbitrária gera $\Sigma^\star$ indecidível.

Exemplos de problemas indecidíveis

Existem muitos problemas que não podem ser resolvidos por MTs. Se aceitarmos a **conjetura de Church–Turing** que diz que existe um método algorítmico para resolver um dado problema se e só se existe uma máquina de Turing para o resolver, isso significa que não podem ser resolvidos computacionalmente.

Exemplos:

- Existe um algoritmo para determinar o AFD mínimo que é equivalente a um dado autómato finito. Consequentemente, existe um algoritmo para verificar se dois autómatos finitos quaisquer são equivalentes.

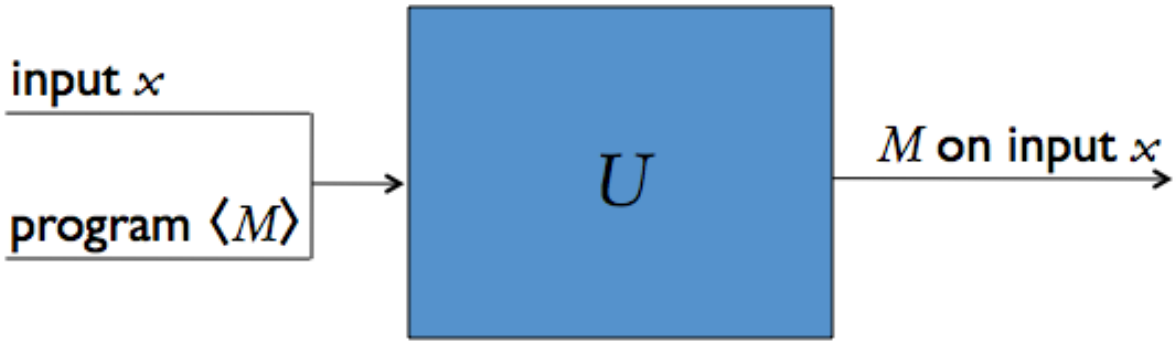
  Mas, não existe um algoritmo que determine se duas GICs $\mathcal{G}$ e $\mathcal{G}'$ quaisquer são ou não são equivalentes, ou seja, se $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

- Não existe um algoritmo para decidir se dois autómatos de pilha quaisquer reconhecem a mesma linguagem.

- Também, o problema de decidir se uma GIC $\mathcal{G}$ arbitrária gera $\Sigma^\star$ indecidível.

Máquina de Turing Universal

Uma Máquina de Turing Universal U é uma máquina de Turing que consegue **simular** qualquer máquina de Turing M arbitrária, dado $(\langle M \rangle, x)$, sendo $\langle M \rangle$ uma sequência que descreve M e x a sequência que M deve analisar.



Observação:

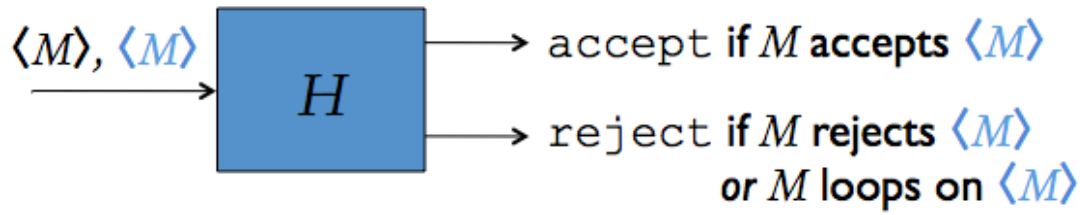
A prova de que a linguagem $A_{\mathsf{TM}} = \{(\langle M \rangle, x) \mid M \text{ é uma MT e } M \text{ aceita } x\}$ é indecidível, que apresentamos a seguir, não foi dada nas aulas. Mas, pode permitir perceber melhor a justificação da indecidiblidade do **Problema da paragem**.

Exemplos de provas de indecidibilidade

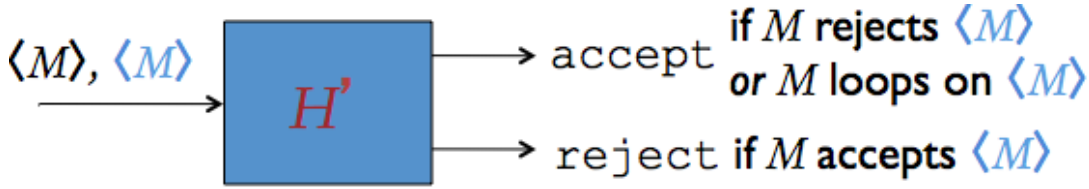
Teorema (A.Turing): $A_{\text{TM}} = \{(\langle M \rangle, x) \mid M \text{ é MT e } M \text{ aceita } x\}$ é indecidível.

Prova (por redução ao absurdo):
Suponhamos que A_{TM} é decidível. Seja H uma MT que, dado $(\langle M \rangle, x)$, pára no estado `accept`, se M aceita x. Caso contrário, pára no estado `reject`, rejeitando $(\langle M \rangle, x)$.

O que acontece se $x = \langle M \rangle$?



Podemos definir uma máquina H' que faz o contrário:

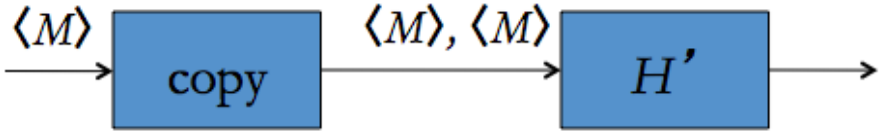


Exemplos de provas de indecidibilidade

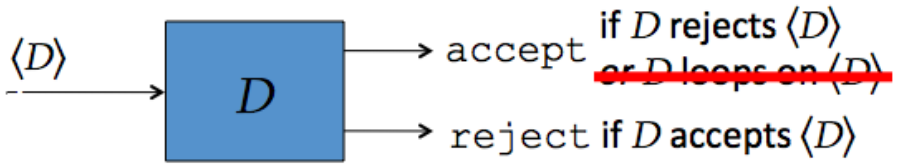
Teorema (A.Turing): $A_{\text{TM}} = \{(\langle M \rangle, x) \mid M \text{ é MT e } M \text{ aceita } x\}$ é indecidível.

Prova (cont):

Podemos construir uma MT D que efetua uma cópia do seu input e a seguir executa H':



Então, quando D recebe $\langle D \rangle$, teriamos uma **inconsistência**:



Notar que, por construção, H' pára sempre e, portanto, D também.

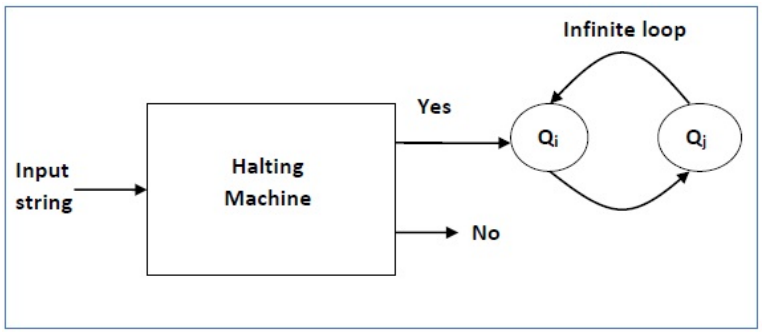
O absurdo resultou de se ter suposto que H existia. Portanto, H não existe. □

Problema da Paragem (*halting problem*)

Não existe um programa de computador, que analise o código de um qualquer programa de computador arbitrário e diga se este pára ou não.

Problema de paragem:
Não existe uma máquina de Turing que receba o código de uma qualquer máquina de Turing e diga se esta pára sempre ou não.

Ideia da prova (por redução ao absurdo): Suponhamos que existia uma MT H que resolvia o *halting problem*. Construimos uma MT D que, sempre que H dá resposta `Yes`, entra num *loop* infinito e, portanto, não pára. E, se H dá resposta `No`, D pára.



Quando D recebe $\langle D \rangle$, pára ou não pára. Mas:

- se D pára, $H(\langle D \rangle, \langle D \rangle)$ responde `Yes`, pelo que D entra num ciclo infinito, o que é absurdo, se D pára;
- se D não pára, $H(\langle D \rangle, \langle D \rangle)$ responde `No` e, por construção, D pára, o que é absurdo, se D não pára.

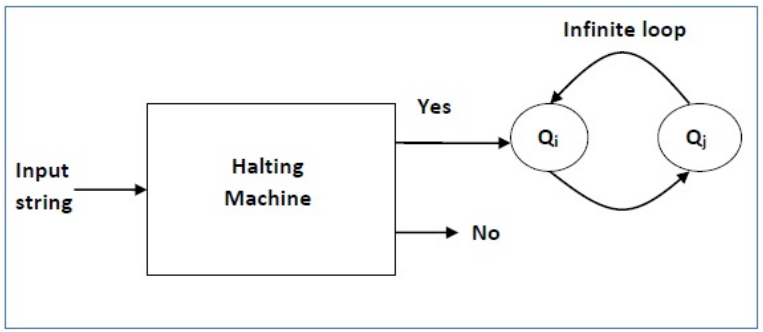
O absurdo resultou de se ter suposto que H existia. Portanto, H não existe.

Problema da Paragem (*halting problem*)

Não existe um programa de computador, que analise o código de um qualquer programa de computador arbitrário e diga se este pára ou não.

Problema de paragem:
Não existe uma máquina de Turing que receba o código de uma qualquer máquina de Turing e diga se esta pára sempre ou não.

Ideia da prova (por redução ao absurdo): Suponhamos que existia uma MT H que resolvia o *halting problem*. Construimos uma MT D que, sempre que H dá resposta `Yes`, entra num *loop* infinito e, portanto, não pára. E, se H dá resposta `No`, D pára.



Quando D recebe $\langle D \rangle$, pára ou não pára. Mas:

- se D pára, $H(\langle D \rangle, \langle D \rangle)$ responde `Yes`, pelo que D entra num ciclo infinito, o que é absurdo, se D pára;
- se D não pára, $H(\langle D \rangle, \langle D \rangle)$ responde `No` e, por construção, D pára, o que é absurdo, se D não pára.

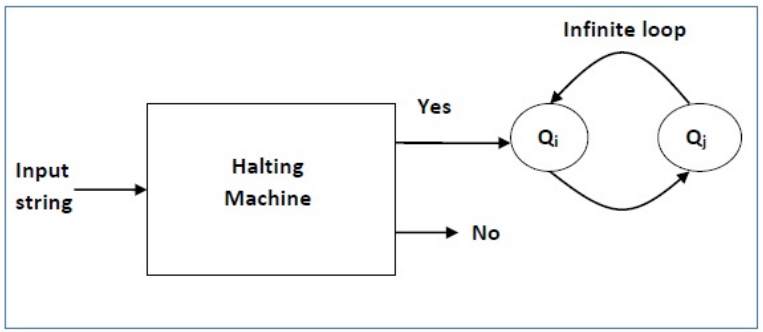
O absurdo resultou de se ter suposto que H existia. Portanto, H não existe.

Problema da Paragem (*halting problem*)

Não existe um programa de computador, que analise o código de um qualquer programa de computador arbitrário e diga se este pára ou não.

Problema de paragem:
Não existe uma máquina de Turing que receba o código de uma qualquer máquina de Turing e diga se esta pára sempre ou não.

Ideia da prova (por redução ao absurdo): Suponhamos que existia uma MT H que resolvia o *halting problem*. Construimos uma MT D que, sempre que H dá resposta `Yes`, entra num *loop* infinito e, portanto, não pára. E, se H dá resposta `No`, D pára.



Quando D recebe $\langle D \rangle$, pára ou não pára. Mas:

- se D pára, $H(\langle D \rangle, \langle D \rangle)$ responde `Yes`, pelo que D entra num ciclo infinito, o que é absurdo, se D pára;
- se D não pára, $H(\langle D \rangle, \langle D \rangle)$ responde `No` e, por construção, D pára, o que é absurdo, se D não pára.

O absurdo resultou de se ter suposto que H existia. Portanto, H não existe.

UCs de continuação...

Lógica Computacional

Computabilidade e Complexidade

Desenho e Análise de Algoritmos

Semânticas de Linguagens de Programação ...

Compiladores ...