Complexidade computacional, lógica e teoria da prova

Tópicos em teoria de complexidade

Prof. Edward Hermann Haeusler

Lab. Tec$\mathcal{MF}$, DI, PUC-Rio

03/2017

Sumário

Sumário

Crivo de Eratóstenes

function $\mathrm{E}(n)$
    $v \coloneqq \langle \underbrace{\textbf{false}, \textbf{true}, \textbf{true}, \ldots, \textbf{true}}_{n} \rangle$
    for $i \coloneqq 2, 3, 4, \ldots \leqslant \sqrt{n}$ **do**
        if $v[i] = \textbf{true}$ **then**
            for $j \coloneqq i^2, i^2 + i, i^2 + 2i, i^2 + 3i, \ldots \leqslant n$ **do**
                $v[j] \coloneqq \textbf{false}$
            end
        end
    end
    return v
end

É mais fácil verificar uma solução do que calculá-la?

$$f(x) = 0^{2^{|x|}}$$

- Escrever $0^{2^{|x|}}$, dado x: "tempo" exponencial
- Verificar se $f(x) = y$, dados $\langle x, y \rangle$: "tempo" polinomial

$$\varphi(x_1, \ldots, x_n)$$

- Encontrar uma valoração que satisfaça φ: "tempo" exponencial (?)
- Verificar se uma dada valoração satisfaz φ: "tempo" polinomial

Problema verificável vs. calculável

$$\mathsf{PV}_{\mathsf{poli}} = \{f : f(a_1, \ldots, a_n) = b \text{ é } \textit{verificável} \text{ em tempo poli}\}$$
$$\mathsf{PC}_{\mathsf{poli}} = \{f : f(a_1, \ldots, a_n) \text{ é } \textit{calculável} \text{ em tempo poli}\}$$

$$\mathsf{PV}_{\mathsf{poli}} \subseteq \mathsf{PC}_{\mathsf{poli}} \Leftrightarrow \mathbf{P} = \mathbf{NP}$$

Lógica e computação

O que é (teoria da) computação?

(Tentativa de) conceituação do *computável*

O que é lógica?

(Tentativa de) conceituação do *razoável*

Lógica

Razoável

Todo *evento* que é passível de explicação na forma argumentativa, construída sobre fatos iniciais *inquestionáveis*

Lógica antes de 1879

- Lógica Aristotélica e Escolástica (c. 300 a.C.)
- Álgebras Booleanas (Boole, 1847)
- Álgebra relacional (DeMorgan, Schroeder, C. S. Peirce, séc. XIX)

Lógica como assunto matemático

1830, DeMorgan observa que a álgebra não necessita lidar apenas com conceitos numéricos

1854, Boole descreve uma álgebra a partir de operações entre conjuntos e relações lógicas, confirmando DeMorgan

1879, Frege estabelece a lógica como um sistema formal com uma linguagem particular, distinta da natural; conceito formal de *prova* matemática

1884, Frege busca fundamentar a aritmética em bases puramente lógicas: pertinência ($\in$) como conceito primitivo; paradoxos (Russell, Banach-Tarski, etc.)

Lógica e matemática na primeira metade do século XX

1903, Russell introduz a teoria dos tipos para resolver o paradoxo de Russell

1910, Russell e Whitehead publicam *Principia Mathematica*

1929, Presburger prova que a aritmética sem $\times$ é decidível

1930, Traski formaliza a semântica da lógica de primeira ordem

1930, Gödel prova a completude da lógica de primeira ordem

1931, Skolem prova que a aritmética sem $+$ e S é decidível

1931, Herbrand prova a consistência de um fragmento da aritmética (só S)

1931, Gödel introduz a ideia de aritmetizar (codificar na forma numérica) a linguagem de um sistema formal de forma que metateoremas do sistema possam ser vistos como teoremas aritméticos e prova o seu famoso teorema da incompletude

1931, Gödel prova a não-provabilidade da consistência

1936, Gentzen prova a consistência da aritmética (*Haupstaz* para o cálculo de sequentes)

Computação

Computável

Toda tarefa que pode ser realizada por um ser *burro* com um mínimo de *conhecimento* e capacidade

$$\text{burro} = \text{incapaz de aprender}$$
$$\text{conhecimento} = ?$$

Computação antes de 1900

- Máquina de raciocinar (Leibniz, 1667)
- Máquina de calcular de Pascal (séc. XVII)
- Máquina de Babbage (séc. XIX)

Computação do ponto de vista das funções recursivas

1927/8, Ackermann define uma função que necessita de recursão simultânea

1931, Gödel define a classe das funções primitivas recursivas associando-as a provas em aritmética

1934, Rózsa Péter prova que a classe das funções primitivas recursivas pode ser definida por recursão simples e *nested* a partir de funções constantes iniciais, identidade e sucessor; prova que a função de Ackermann não é primitiva recursiva (apesar de computável)

1936, Turing define uma máquina formal a partir de princípios simples (ler, escrever e apagar símbolos numa fita) e define o conceito de *máquina universal*; prova que não existe uma máquina capaz de verificar se outra para ou não; desde o início sua máquina possui versão não-determinística

1936, Church define o λ-calculus e mostra que este é capaz de definir todas as funções para as quais existe uma máquina de Turing

1938, Kleene aceitando que computável inclui parcialidade funcional, define as funções parcialmente recursivas e lança a *Tese de Church*

1954, Markov estabelece o conceito de computável com base em identificação de palavras e símbolos (algoritmos de Markov) e justifica o ponto de vista finitista da computação

Computação do ponto de vista das funções recursivas

1927/8, Ackermann define uma função que necessita de recursão simultânea

1931, Gödel define a classe das funções primitivas recursivas associando-as a provas em aritmética

1934, Rózsa Péter prova que a classe das funções primitivas recursivas pode ser definida por recursão simples e *nested* a partir de funções constantes iniciais, identidade e sucessor; prova que a função de Ackermann não é primitiva recursiva (apesar de computável)

1936, Turing define uma máquina formal a partir de princípios simples (ler, escrever e apagar símbol*ersal*; prova que não existe ão; desde o início sua máquina possui versão não-determinística

Máquina programável

1936, Church define o λ-calculus e mostra que este é capaz de definir todas as funções para as quais existe uma máquina de Turing

1938, Kleene aceitando que computável inclui parcialidade funcional, define as funções parcialmente recursivas e lança a *Tese de Church*

1954, Markov estabelece o conceito de computável com base em identificação de palavras e símbolos (algoritmos de Markov) e justifica o ponto de vista finitista da computação

Computação do ponto de vista das funções recursivas

1927/8, Ackermann define uma função que necessita de recursão simultânea

1931, Gödel provas em

Programação lógica, Prolog

1934, Rózsa Péter prova que a classe das funções primitivas recursivas pode ser definida por recursão simples e *nested* a partir de funções constantes iniciais, identidade e sucessor; prova que a função de Ackermann não é primitiva recursiva (apesar de computável)

1936, Turing define uma máquina formal a partir de princípios simples (ler, escrever e apagar símbol *ersal*; prova que não existe ão; desde o início sua máquina possui versão não-determinística

Máquina programável

1936, Church define o λ-calculus e mostra que este é capaz de definir todas as funções para as quais existe uma máquina de Turing

1938, Kleene aceitando que computável inclui parcialidade funcional, define as funções parcialmente recursivas e lança a *Tese de Church*

1954, Markov estabelece o conceito de computável com base em identificação de palavras e símbolos (algoritmos de Markov) e justifica o ponto de vista finitista da computação

Computação do ponto de vista das funções recursivas

1927/8, Ackermann define uma função que necessita de recursão simultânea

1931, Gödel provas em

Programação lógica, Prolog

1934, Rózsa Péter prova que a classe das funções primitivas recursivas pode ser definida por recursão simples e *nested* a partir de funções constantes iniciais, identidade e sucessor; prova que a função de Ackermann não é primitiva recursiva (apesar de computável)

1936, Turing define uma máquina formal a partir de princípios simples (ler, escrever e apagar símbol *ersal*; prova que não existe ão; desde o início sua máquina possui versão não-determinística

Máquina programável

1936, Church define todas as funções para as quais

Lisp, ling. funcionais

1938, Kleene aceitando que computável inclui parcialidade funcional, define as funções parcialmente recursivas e lança a *Tese de Church*

1954, Markov estabelece o conceito de computável com base em identificação de palavras e símbolos (algoritmos de Markov) e justifica o ponto de vista finitista da computação

Computação do ponto de vista das funções recursivas

1927/8, Ackermann define uma função que necessita de recursão simultânea

1931, Gödel provas em

Programação lógica, Prolog

1934, Rózsa Péter prova que a classe das funções primitivas recursivas pode ser definida por recursão simples e *nested* a partir de funções constantes iniciais, identidade e sucessor; prova que a função de Ackermann não é primitiva recursiva (apesar de computável)

1936, Turing define uma máquina formal a partir de princípios simples (ler, escrever e apagar símbol *ersal*; prova que não existe ão; desde o início sua máquina possui versão não-determinística

Máquina programável

1936, Church define todas as funções para as quais

Lisp, ling. funcionais

1938, Kleene aceitando que computável inclui parcialidade funcional, define as funções parcialmente recursivas e lança a *Tese de Church*

1954, Markov estabelece o conceito de computável com base em identificação de palavras e símbo ta finitista da comp

SNOBOL, ling. transf.

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \quad (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \quad (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Tese de Church

Uma função $f\colon \mathbb{N} \to \mathbb{N}$ é computável sse:

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \quad (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Tese de Church

Uma função $f\colon \mathbb{N} \to \mathbb{N}$ é computável sse:

(i) existe um combinador $\mathsf{F} = C_1 C_2 \ldots C_n$ tal que, para todo $n \in \mathbb{N}$,

$$(\mathsf{F}:n: \triangleright :m:) \Leftrightarrow f(n) = m$$

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \;\; (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Tese de Church

Uma função $f\colon \mathbb{N} \to \mathbb{N}$ é computável sse:

(ii) f é recursiva

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \;\; (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Tese de Church

Uma função $f\colon \mathbb{N} \to \mathbb{N}$ é computável sse:

(iii) existe uma máquina de Turing M tal que

$$M \text{ com entrada } \frac{n}{11\ldots111} \text{ para com saída } \frac{11\ldots111}{m} \Leftrightarrow f(n) = m$$

Lógica combinatória

$$\mathbf{S}xyz \triangleright (xy)xz \qquad \mathbf{K}xy \triangleright x \qquad \mathbf{I}x \triangleright x \quad (\equiv \mathbf{SKK})$$

$$:0: \equiv \mathbf{I} \quad :1: \equiv \mathsf{P}:0:\mathbf{K} \quad :2: \equiv \mathsf{P}:1:\mathbf{K} \quad \ldots \quad :n: \equiv \mathsf{P}:n-1:\mathbf{K}$$

$$\mathsf{P} \equiv \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{SK}))\mathbf{K}))))(\mathbf{KK})$$

Tese de Church

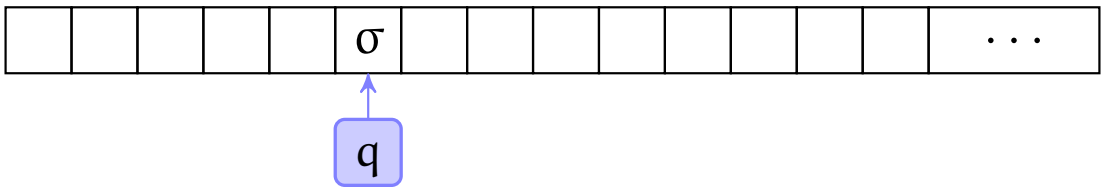
Uma função $f\colon \mathbb{N} \to \mathbb{N}$ é computável sse:

(iv) existe um algoritmo de Markov A tal que

$$A \text{ lendo } \frac{n}{11\ldots111} \text{ para e imprime } \frac{11\ldots111}{m} \Leftrightarrow f(n) = m$$

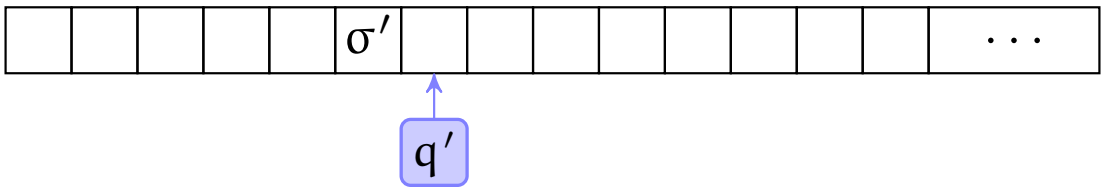
Máquina de Turing

Modelo determinístico



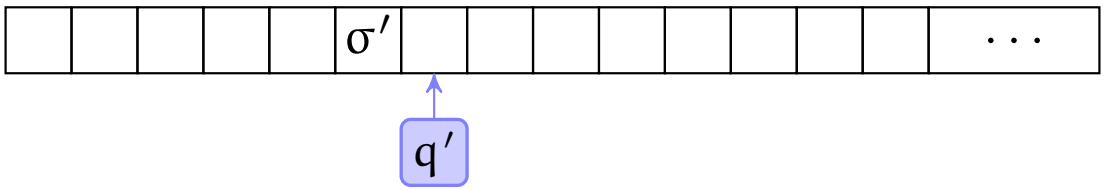
Máquina de Turing

Modelo determinístico



Máquina de Turing

Modelo determinístico

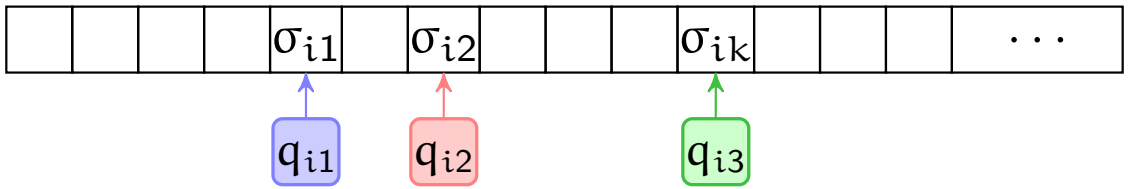


Máquina universal

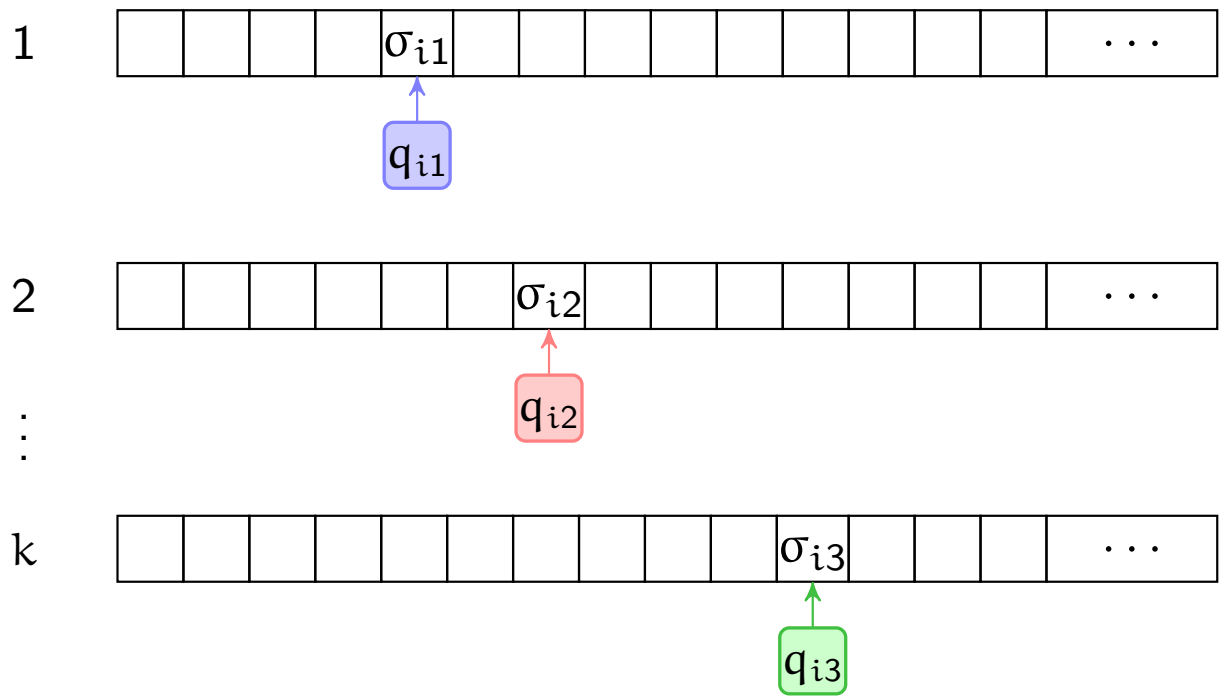
$$U(i, a) = T_i(a)$$

Máquina de Turing determinística: Variações

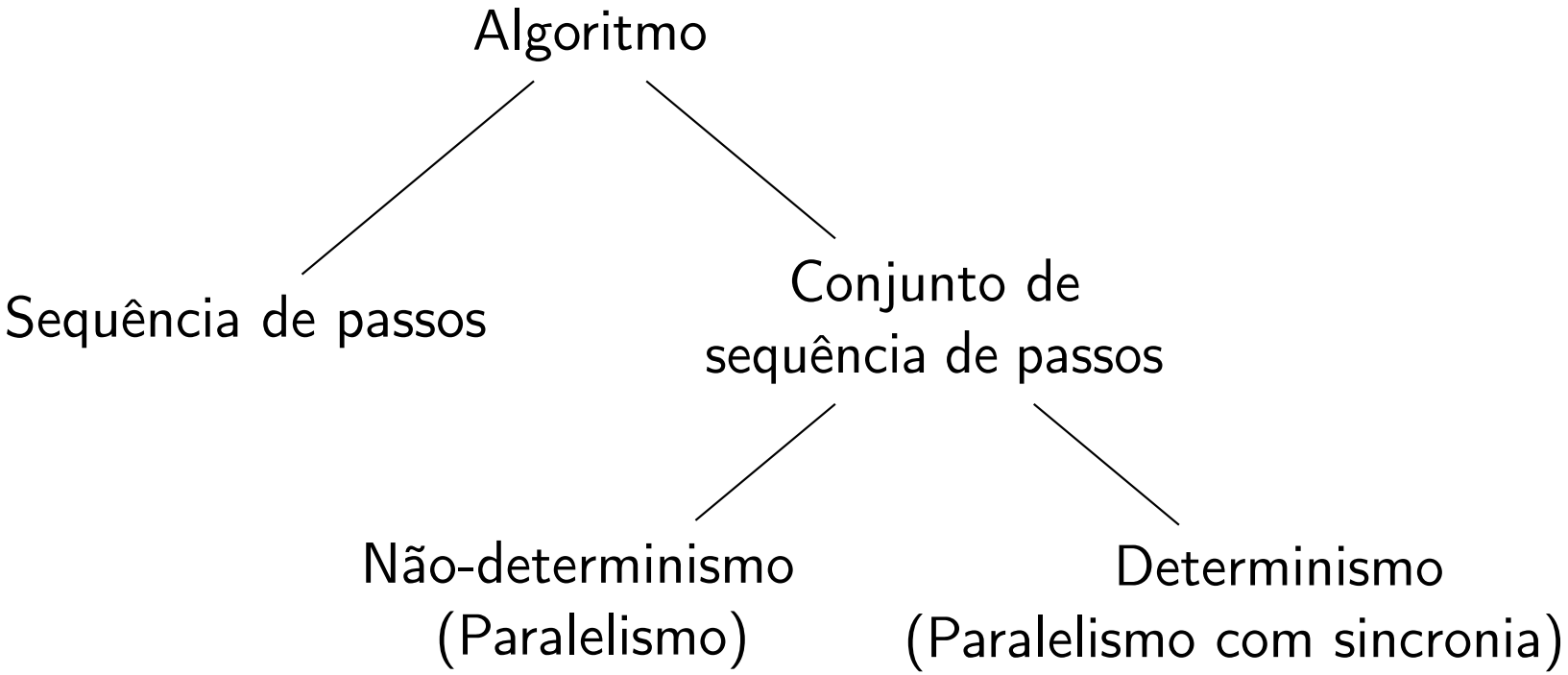
Modelo multi-cabeça



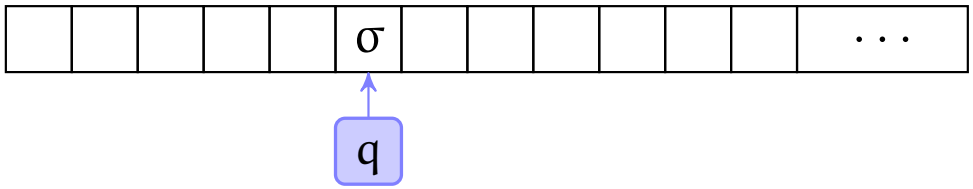
Modelo multi-fita



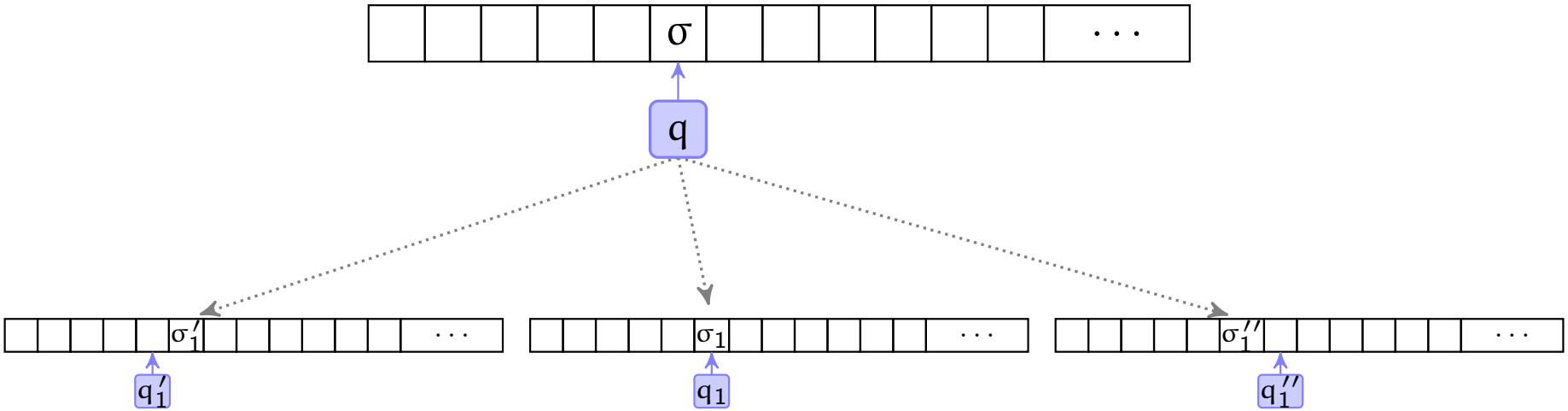
A partir da década de 1950...



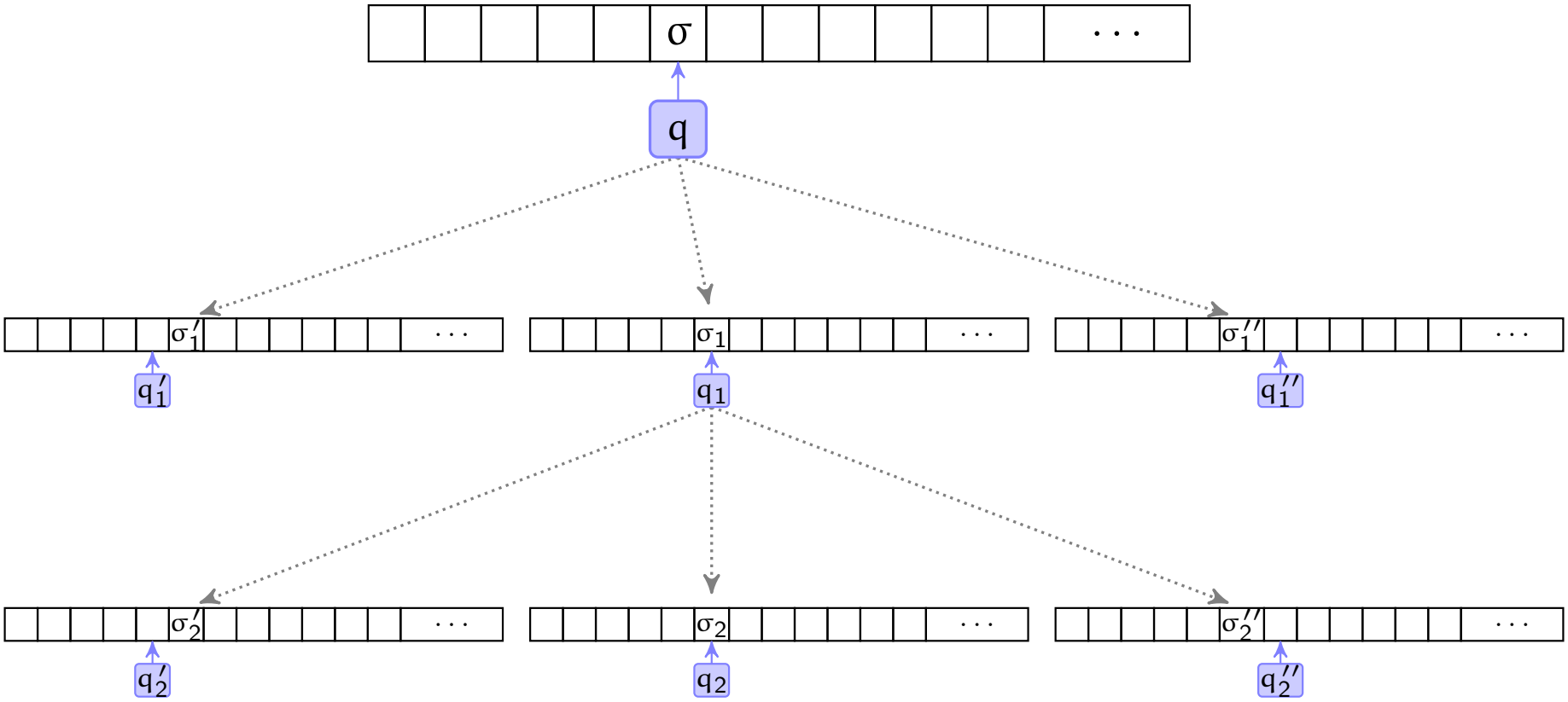
Máquina de Turing não-determinística



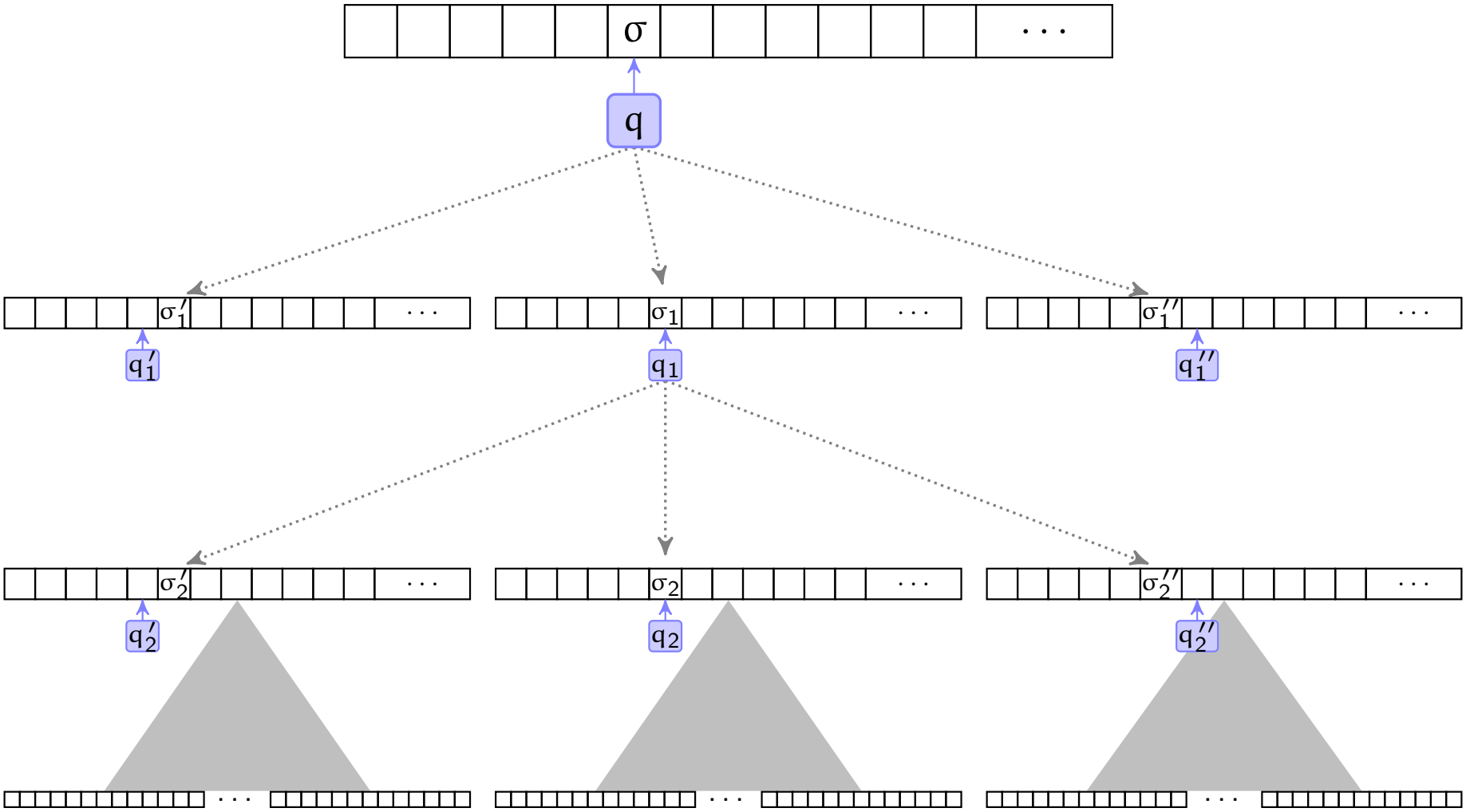
Máquina de Turing não-determinística



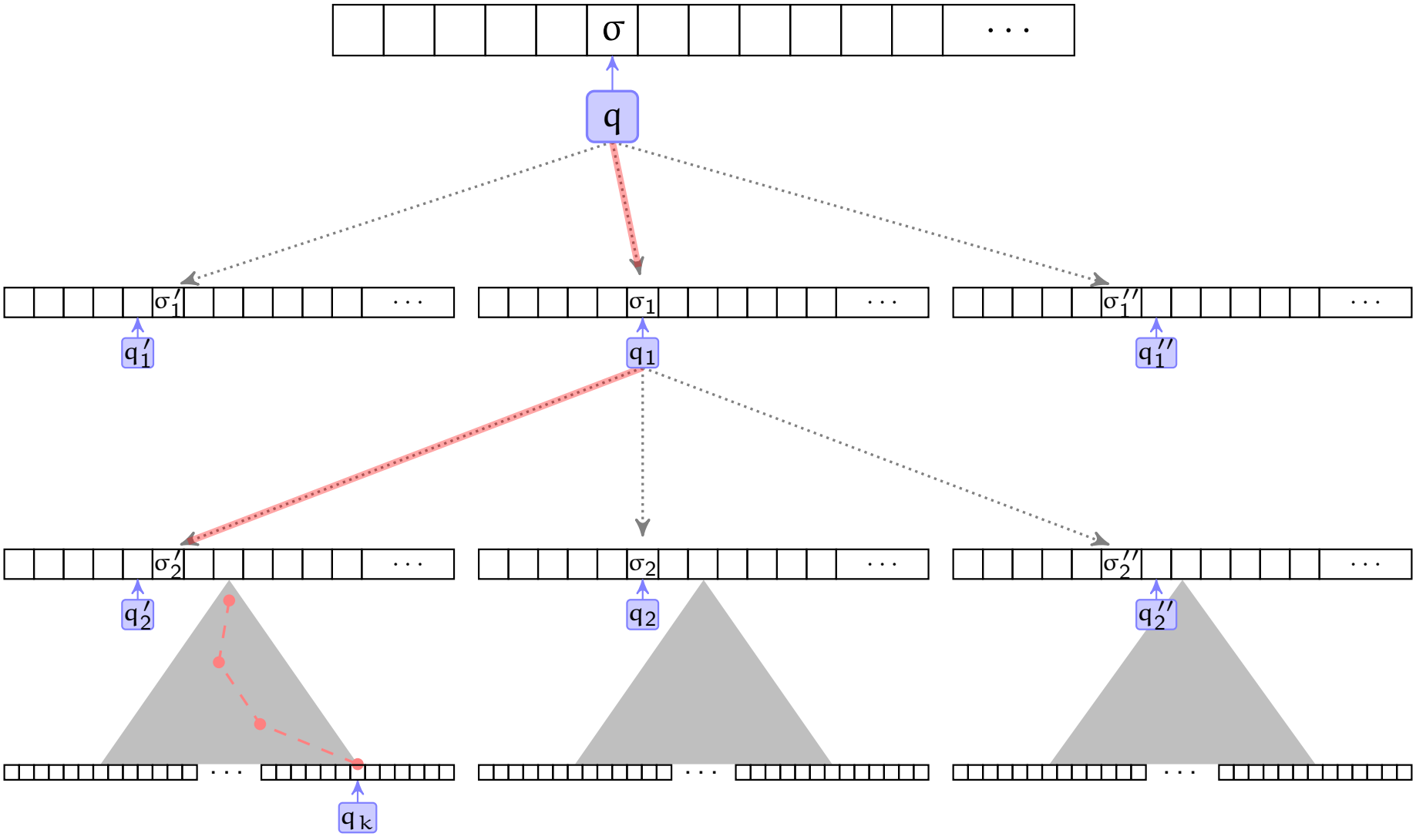
Máquina de Turing não-determinística



Máquina de Turing não-determinística



Máquina de Turing não-determinística

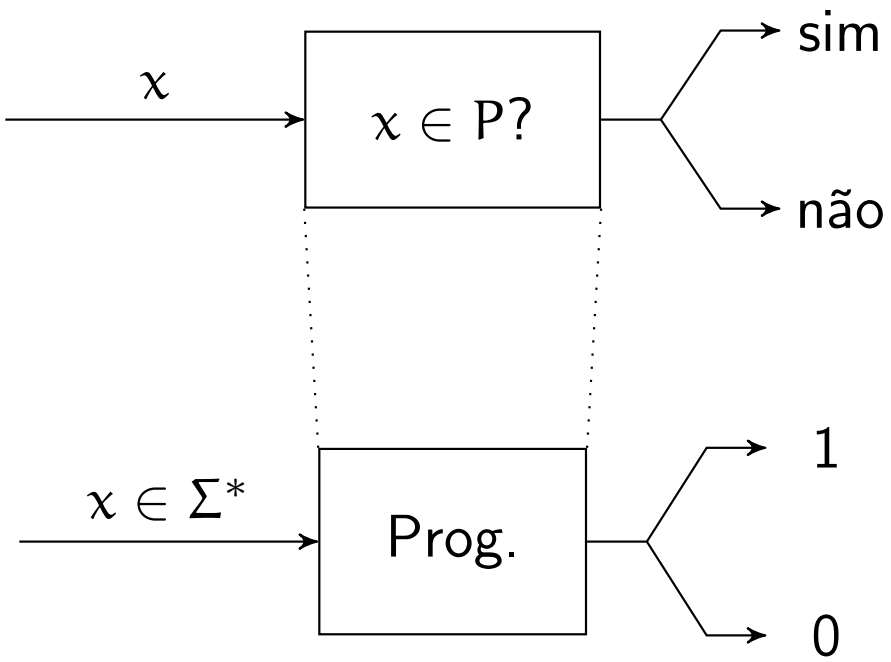


Sumário

Medindo a eficiência de algoritmos

- Modelo de computação
- Utilização de recursos: Tempo vs. memória
- Problemas que resolvem
  - Computação de funções
  - Problemas de otimização
  - Problemas de decisão
  - Linguagens
- Classes de complexidade (como defini-las?)

Problema de decisão vs. linguagens



Problemas de decisão $\approx$ Linguagens formais

Definições

$$L \in \mathbf{TIME}(f) \Leftrightarrow \exists M \in \text{TuringDet que decide } L \text{ e}$$
$$\exists c,\ \forall x \in \text{Strings},\ \text{steps}(M, x) \leqslant c \cdot f(|x|)$$

$$L \in \mathbf{SPACE}(f) \Leftrightarrow \exists M \in \text{TuringDet que decide } L \text{ e}$$
$$\exists c,\ \forall x \in \text{Strings},\ \text{space}(M, x) \leqslant c \cdot f(|x|)$$

- Como medir espaço (memória)?
- Qualquer tipo de função serve como parâmetro de medida?

Por que classes assintóticas de funções?

Teorema (*Speedup* linear)

Se uma linguagem L *é decidida em tempo* $\mathrm{f}(\mathrm{n})$ *então, para qualquer* $\epsilon > 0$, *existe uma máquina de Turing* M_ϵ *que decide* L *em tempo* $\epsilon \mathrm{f}(\mathrm{n}) + \mathrm{n} + 2$

Prova

Via modificação do tamanho da "palavra" de memória

Por que classes assintóticas de funções?

Teorema (*Speedup* linear)

Se uma linguagem L *é decidida em tempo* $f(n)$ *então, para qualquer* $\epsilon > 0$, *existe uma máquina de Turing* M_ϵ *que decide* L *em tempo* $\epsilon f(n) + n + 2$

Prova

Via modificação do tamanho da "palavra" de memória

Consequência

Se L é decidida em tempo $f(n) = 165n^k + \ldots + 54n + 657$
Então L também é decidida em tempo $f'(n) = n^k$

Obs. O mesmo teorema e técnica de prova valem para funções de medida de memória e uso de espaço (número máximo de células visitadas).

Funções steps e space, e a robustez do modelo

- steps deve considerar o tempo de leitura da entrada?
- space deve considerar o espaço utilizado pela entrada?
- O modelo de máquina de Turing é robusto em relação às medidas?

Fato (Máquina com $k > 1$ fitas)

Se L *é reconhecida em tempo* $\mathcal{O}(f(|x|))$ *por uma máquina multi-fita então* L *é reconhecida em tempo* $\mathcal{O}(f(|x|)^2)$ *por uma máquina com uma fita*

Sumário

Funções de tempo construtivas

Teorema (*Gap*)

Existe f *recursiva tal que* $\mathbf{TIME}(\mathrm{f}(\mathrm{n})) = \mathbf{TIME}(2^{\mathrm{f}(\mathrm{n})})$

Definição

Uma função de tempo f é construtível sse existe uma máquina de Turing M tal que, para todo n, $\mathrm{M}(1^{\mathrm{n}}) = 1^{\mathrm{f}(\mathrm{n})}$ e $\mathrm{steps}(\mathrm{M}, 1^{\mathrm{n}}) \leqslant \mathrm{c} \cdot \mathrm{f}(\mathrm{n})$

Propriedades

- Para qualquer função g computável existe uma função de tempo construtível f tal que $\mathrm{g} < \mathrm{f}$
- Funções polinomiais, exponenciais e logaritmos (inteiros) são funções de tempo construtíveis

Propriedades e definições

Se f é uma função de tempo construtível então

$$\mathbf{coTIME}(f) = \{L : \exists M \in \text{TuringDet que decide } \bar{L} \text{ e } \forall x \in \text{Strings}, \text{steps}(M, x) \in \mathcal{O}(f(|x|))\}$$

Fatos

- $\mathbf{TIME}(f) = \mathbf{coTIME}(f)$
- *Se* $n \leqslant f(n)$ *e* $|L_1 \Delta L_2|$ *é finito então* $L_1 \in \mathbf{TIME}(f)$ *sse* $L_2 \in \mathbf{TIME}(f)$
- $\mathbf{TIME}(f)$ *é construtivamente enumerável, i.e., existe uma máquina* T *tal que* $T(i, x) = T_i(x)$ *e* $\mathbf{TIME}(f) = \{L_i : T_i \text{ decide } L_i\}$

Definições

$$\mathbf{P} = \bigcup_{i \in \mathbb{N}} \mathbf{TIME}(n^i) \qquad \mathbf{EXP} = \bigcup_{i \in \mathbb{N}} \mathbf{TIME}(2^{n^i})$$

Teorema de Cantor

Teorema

Para todo conjunto B, $|B| < |2^B|$ $\qquad (2^B = \{A : A \subseteq B\})$

Prova

Suponha que $|B| = |2^B|$. Então existe $f\colon B \to 2^B$. Seja $S = \{x : x \notin f(x)\}$. Temos que

$$f^{-1}(S) \in S \Leftrightarrow f^{-1}(S) \notin S,$$

o que é absurdo. Logo, a hipótese inicial é falsa. □

Paradoxo do barbeiro. Em uma cidade existe um barbeiro que barbeia todos os homens que não barbeiam a si próprios e somente esses.

O método da diagonal de Cantor

Suponha que $|(0,1)| = |\mathbb{N}|$,

$$
\begin{array}{lllllllll}
a_0 & = & 0, & a_{00} & a_{01} & a_{02} & a_{03} & \ldots & a_{0n} \ldots \\
a_1 & = & 0, & a_{10} & a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \ldots \\
a_2 & = & 0, & a_{20} & a_{21} & a_{22} & a_{13} & \ldots & a_{1n} \ldots \\
 & \vdots & & & & & & & \\
a_n & = & 0, & a_{n0} & a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \ldots \\
 & \vdots & & & & & & &
\end{array}
$$

Seja $b = 0, b_0 b_1 b_2 \ldots b_n \ldots$ tal que

$$
b_j = \begin{cases} 5 & \text{se } a_{jj} = 9 \\ 9 & \text{senão} \end{cases}
$$

Temos que $\forall i (a_i \neq b)$. Logo, $|(0,1)| \neq |\mathbb{N}|$.

Hierarquia própria de funções construtivas

$$\text{Para}_f = \{\langle T, x\rangle : T(x) \text{ para em no máximo } f(|x|) \text{ passos}\}$$

Fatos

- $\text{Para}_f \in \mathbf{TIME}((f(n))^2)$
- $\text{Para}_f \notin \mathbf{TIME}(f\lfloor \frac{n}{2} \rfloor)$

Corolários

- $\mathbf{TIME}(f(n)) \subsetneq \mathbf{TIME}((f(2n+1))^3)$
- $\mathbf{P} \subsetneq \mathbf{EXP}$

Sumário

Classes de complexidade e algumas relações

$$\mathbf{PSPACE} = \bigcup_{i \in \mathbb{N}} \mathbf{SPACE}(n^i) \qquad \mathbf{NPSPACE} = \bigcup_{i \in \mathbb{N}} \mathbf{NSPACE}(n^i)$$

$$\mathbf{NP} = \bigcup_{i \in \mathbb{N}} \mathbf{NTIME}(n^i)$$

$$\mathbf{L} = \mathbf{SPACE}(\log n)$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \begin{array}{c} \mathbf{PSPACE} \\ \| \\ \mathbf{NPSPACE} \end{array} \subseteq \mathbf{EXP}$$

Classes de complexidade e algumas relações (cont.)

- $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$ e $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$
- $\mathbf{NTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$
- $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{TIME}(k^{\log n + f(n)})$ (num. conf. + Reachability)
- Reachability $\in \mathbf{SPACE}(\log^2 n)$ e $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{SPACE}((f(n))^2)$
- núm. nós alcançáveis. $\in \mathbf{NSPACE}(\log n)$

$$\Rightarrow \mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$$

Sumário

A ciência da computação hoje: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (Cook, 1971)

P Encontra solução em tempo polinomial

NP Verifica solução em tempo polinomial

coNP Verifica que não é solução em tempo polinomial

$$\underbrace{\textsc{Sat}}_{\text{verificação de modelos}} \in \mathbf{NP} \qquad \underbrace{\textsc{Taut}}_{\text{prova de teoremas}} \in \mathbf{coNP}$$

A ciência da computação hoje: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (Cook, 1971)

P Encontra solução em tempo polinomial

NP Verifica solução em tempo polinomial

coNP Verifica que não é solução em tempo polinomial

$$\underbrace{\textsc{Sat}}_{\text{verificação de modelos}} \in \mathbf{NP} \qquad \underbrace{\textsc{Taut}}_{\text{prova de teoremas}} \in \mathbf{coNP}$$

Obs. Se $\mathbf{coNP} \neq \mathbf{NP}$ então $\mathbf{NP} \neq \mathbf{P}$.

Completude

Definição

Seja P um problema (linguagem) e seja C uma classe de problemas. Então

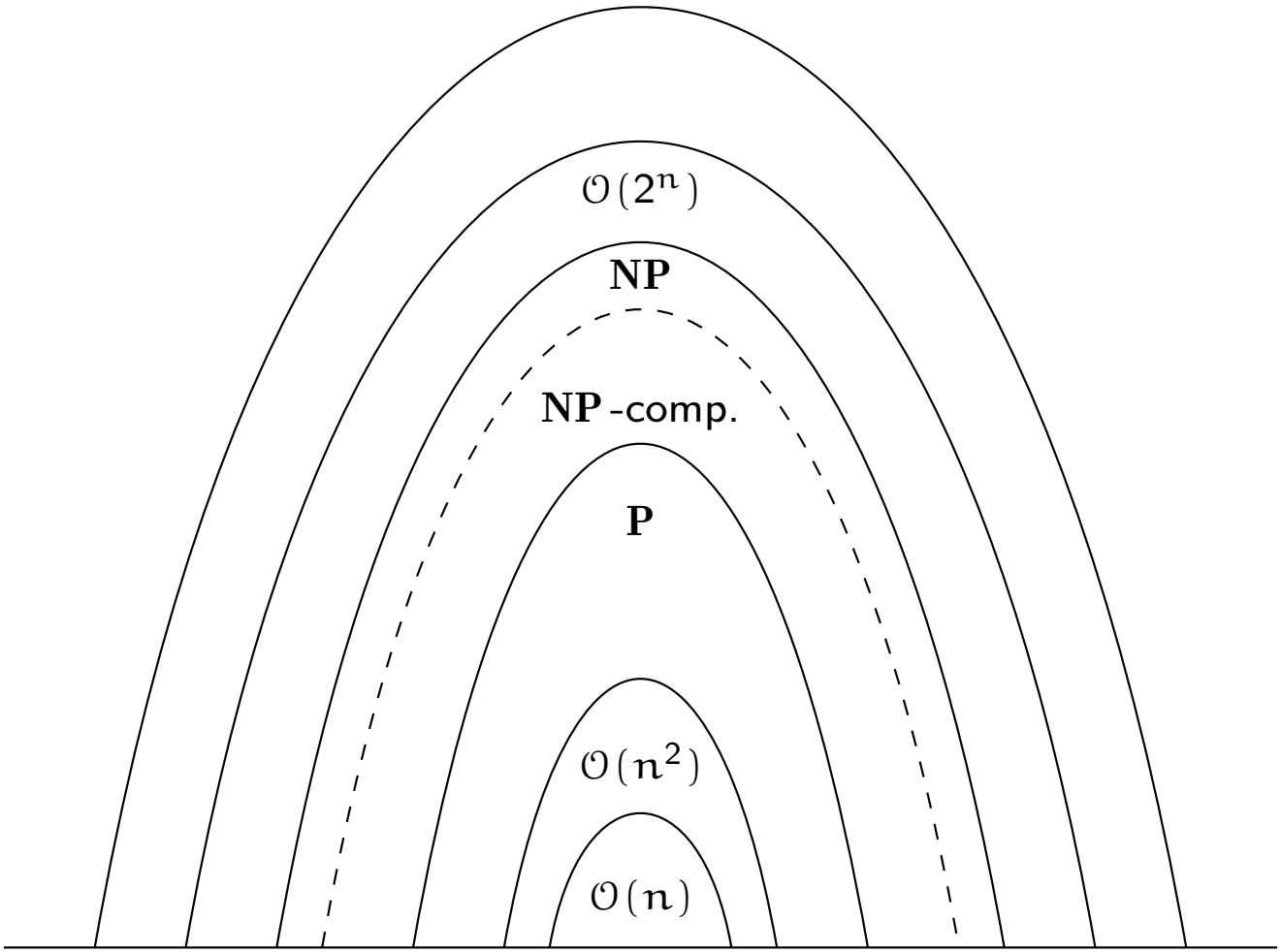
$$\mathrm{P} \text{ é } \underline{\mathrm{C}\text{-completo}} \Leftrightarrow \text{todo problema de } \mathrm{C} \text{ é redutível a } \mathrm{P},$$

i.e., resolver P é tão difícil quanto resolver qualquer outro problema de C

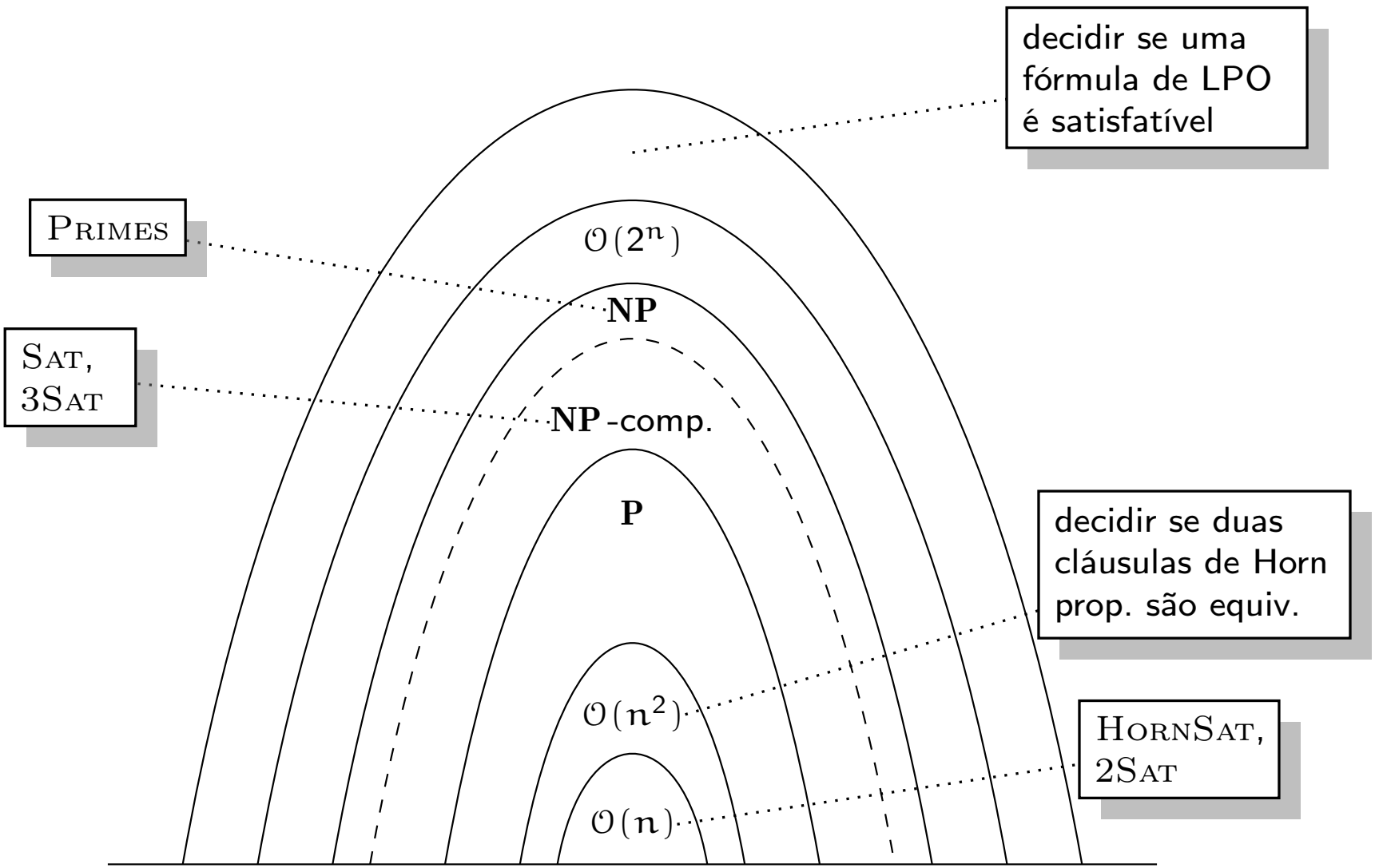
Exemplos

- Saber se um programa para (via outro programa) é $\mathbf{R}$-completo—$\mathbf{R}$ é o conjunto dos problemas (ling.) recursivos
- Saber se uma solução para um problema é verificável em tempo polinomial é tão difícil quanto decidir se uma sentença da lógica proposicional é "verdadeira"—$\textsc{Sat}$ é $\mathbf{NP}$-completo

Hierarquia de classes de complexidade, supondo $\mathbf{P} \neq \mathbf{NP}$



Hierarquia de classes de complexidade, supondo $\mathbf{P} \neq \mathbf{NP}$



Sumário

Importante

Fato

Existe um oráculo B *tal que* $\mathbf{P}^{\mathrm{B}} = \mathbf{NP}^{\mathrm{B}}$

Prova

$\mathbf{NPSPACE} = \mathbf{PSPACE}$ e B um problema $\mathbf{NPSPACE}$-completo ▶ □

Fato

Existe um oráculo C *tal que* $\mathbf{P}^{\mathrm{C}} \neq \mathbf{NP}^{\mathrm{C}}$

Prova

Via diagonalização ▶ □

Importante

Fato

Existe um oráculo B *tal que* $\mathbf{P}^{\mathrm{B}} = \mathbf{NP}^{\mathrm{B}}$

Prova

$\mathbf{NPSPACE} = \mathbf{PSPACE}$ e B um problema $\mathbf{NPSPACE}$-completo ▶ □

Fato

Existe um oráculo C *tal que* $\mathbf{P}^{\mathrm{C}} \neq \mathbf{NP}^{\mathrm{C}}$

Prova

Via diagonalização ▶ □

Discussão. Uso de simulação e diagonalização para provar $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$.

Sumário

Importante

Fato

Se $\mathbf{P} \neq \mathbf{NP}$ *então* $\mathbf{NP} - \mathbf{NP}$*-completo* $\neq \varnothing$

Prova

Via diagonalização uniforme (Ladner, 1975) □

Lema

Sejam duas classes de linguagens C_1 *e* C_2 *tais que*

1. *Ambas são construtivamente enumeráveis*
2. *Ambas são fechadas para variação finita*
3. *Existe* $\mathrm{L}_1 \notin \mathrm{C}_1$ *e* $\mathrm{L}_2 \notin \mathrm{C}_2$

Então, existe L *tal que*

$$\mathrm{L} \notin \mathrm{C}_1 \cup \mathrm{C}_2 \quad e \quad \mathrm{L} \leqslant \mathrm{L}_1 \oplus \mathrm{L}_2$$

Prova

Via diagonalização

□

Teorema

Se $\mathbf{P} \neq \mathbf{NP}$, *então*

$$\textsc{Sat} \notin \mathbf{P} \;\; e \;\; \varnothing \notin \mathbf{NP}\text{-}completo \;\Rightarrow\; \exists \mathrm{L}(\mathrm{L} \notin \mathbf{P} \cup \mathbf{NP}\text{-}completo)$$

Representação vs. complexidade

Pergunta. A escolha do formato de representação de dados pode alterar a complexidade de um problema?

Teorema

Se alguma linguagem unária for $\mathbf{NP}$*-completa, então* $\mathbf{P} = \mathbf{NP}$

Prova

□

Sumário

Hierarquia de Kleene

Hierarquia polinomial

Sumário

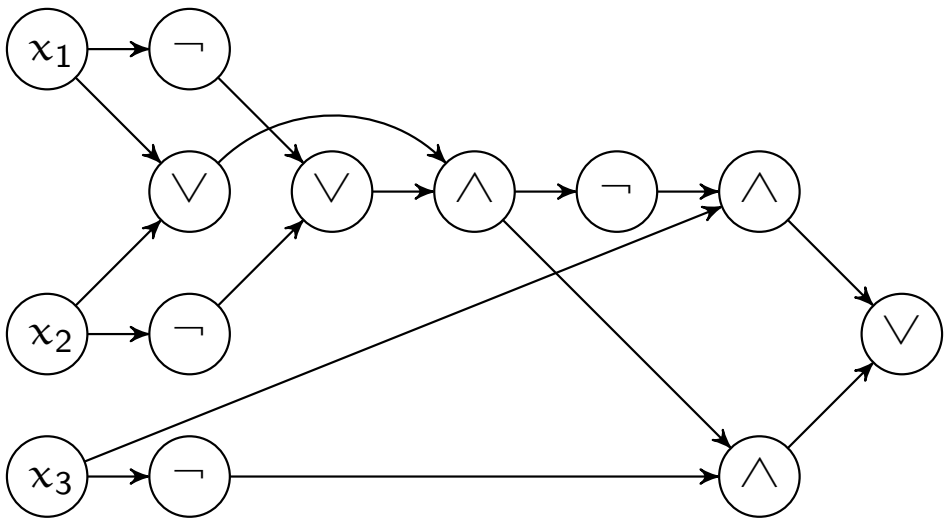
Circuitos booleanos

Definição

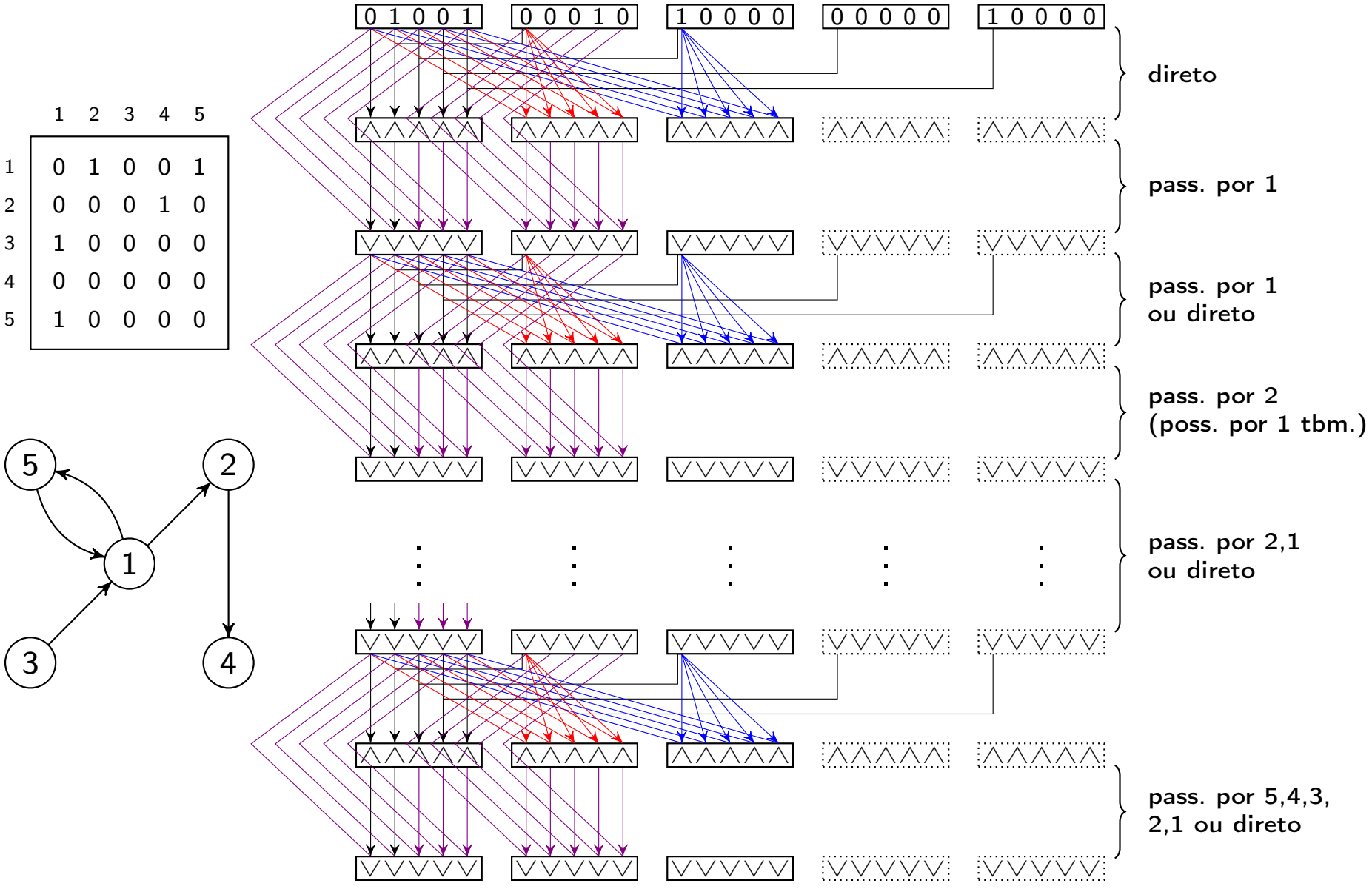
Um circuito booleano é um dígrafo acíclico com nós AND, OR, NOT, nós iniciais (sem arco entrante) e apenas um nó terminal (sem saída)

Exemplo

$$(x_3 \wedge \neg((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))) \vee (\neg x_3 \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$$



Circuito booleano para REACHABILITY



Famílias de circuitos booleanos

Definição

Uma linguagem L é decidida por uma família de circuitos booleanos $(C_i)_{i \in \mathbb{N}}$ sse, para todo $s \in \text{String}$ tal que $|s| = n$,

$$C_n \text{ aceita } s \Leftrightarrow s \in L$$

Pergunta

O tamanho de um circuito depende da complexidade (em MT) do problema de decisão associado? (*Resposta no próximo slide*)

Obs. REACHABILITY tem circuitos de tamanho $\mathcal{O}(n^3)$ e profundidade $\mathcal{O}(n)$.

Conjectura

Todo problema de decisão com famílias de circuitos de tamanho polinomial é um problema que está em $\mathbf{P}$ *(?)*

Pergunta (anterior)

O tamanho de um circuito depende da complexidade (em MT) do problema de decisão associado?

Resposta

Não. Problemas indecidíveis de famílias polinomiais de circuitos booleanos

Exemplo

Seja $D \subset \{1\}^*$ uma linguagem indecidível e seja $(A_i)_{i \in \mathbb{N}}$ uma família de circuitos tal que

1. se $1^k \in D$ então A_k é um circuito só com portas AND e k fontes
2. se $1^k \notin D$ então A_k é um circuito só com portas AND e uma porta final NOT

Famílias uniformes de circuitos booleanos

Fato

$\mathbf{NLSPACE} \subseteq \mathbf{P} \qquad (k^{\log n} = n)$

Definição

Uma família de circuitos booleanos $(C_i)_{i \in \mathbb{N}}$ é uniforme sse existe $M \in$ TuringNDet que dada a entrada 1^n gera o circuito C_n usando $\log n$ células da fita

Exemplo

REACHABILITY possui família uniforme de circuitos booleanos

- Dado n, existe uma MT para gerar gerar C_n usando somente $\log n$ células da fita
- Gerar todos os circuitos de profundidade n com n^2 nós fontes e verificar se a forma é a requerida

Conjectura (Nova)

$$\mathbf{P} = \{\mathrm{L} : \mathrm{L} \text{ é aceita por famílias uniformes de circuitos booleanos de tamanho polinomial}\} \textit{ (?)}$$

Prova

Via a mesma construção usada na prova de que $\textsc{Sat}$ é $\mathbf{NP}$-completo e a definição de família de circuitos de tamanho polinomial.

- $\supseteq$ Trivial.
- $\subseteq$ Seja $\mathrm{L} \in \mathbf{P}$. Existe $\mathrm{M} \in \text{TuringDet}$ que decide L em tempo n^k. Seja $w \in \mathrm{L}$ tal que $|w| = n$. Dada a tabela de computação de M constrói-se um circuito C_n.

□

Tabela de computação de M para a entrada w

[TODO]

Observações

1. O tamanho do circuito [TODO] só depende de M
   Se $\Gamma = \mathsf{Est}(M) \cup \mathsf{Alfa}(M) \cup \{\diamond, \Rightarrow\}$ e $c = |\Gamma|$ então $|\mathsf{Pad}| = 3c$
2. $w \in \Gamma$ sse o circuito tem valor "true" quando alimentado com $\mathsf{cod}(w)$
3. A construção de um circuito $C_{|w|}$ é feita a partir de M e $|w|$ usando-se espaço de ordem $\log|w|$ na fita; o algoritmo imprime na fita de saída as diversas cópias de Pad (de tamanho independente da entrada) com as respectivas junções de E/S que usam espaço de ordem $\log|w|$

Proposição

Toda linguagem $\mathrm{L} \in \mathbf{P}$ *possui família uniforme de circuitos booleanos de tamanho polinomial*

Corolário

Se existe $\mathrm{L} \in \mathbf{NP}$ *tal que toda família uniforme de circuitos que decide* L *não tem tamanho limitada por nenhum polinômio, então* $\mathbf{P} \neq \mathbf{NP}$

Um caminho para provar $\mathbf{P} \neq \mathbf{NP}$

Provar que algum problema $\mathbf{NP}$-completo possui cota inferior super-polinomial

Circuitos vs. circuitos monotônicos

Definições

- Um circuito é monotônico se não possui porta NOT
- Uma função booleana f é monotônica sse

$$a \leqslant b \Rightarrow f(x_1, \ldots, a, \ldots, x_n) \leqslant f(x_1, \ldots, b, \ldots x_n)$$

Observações

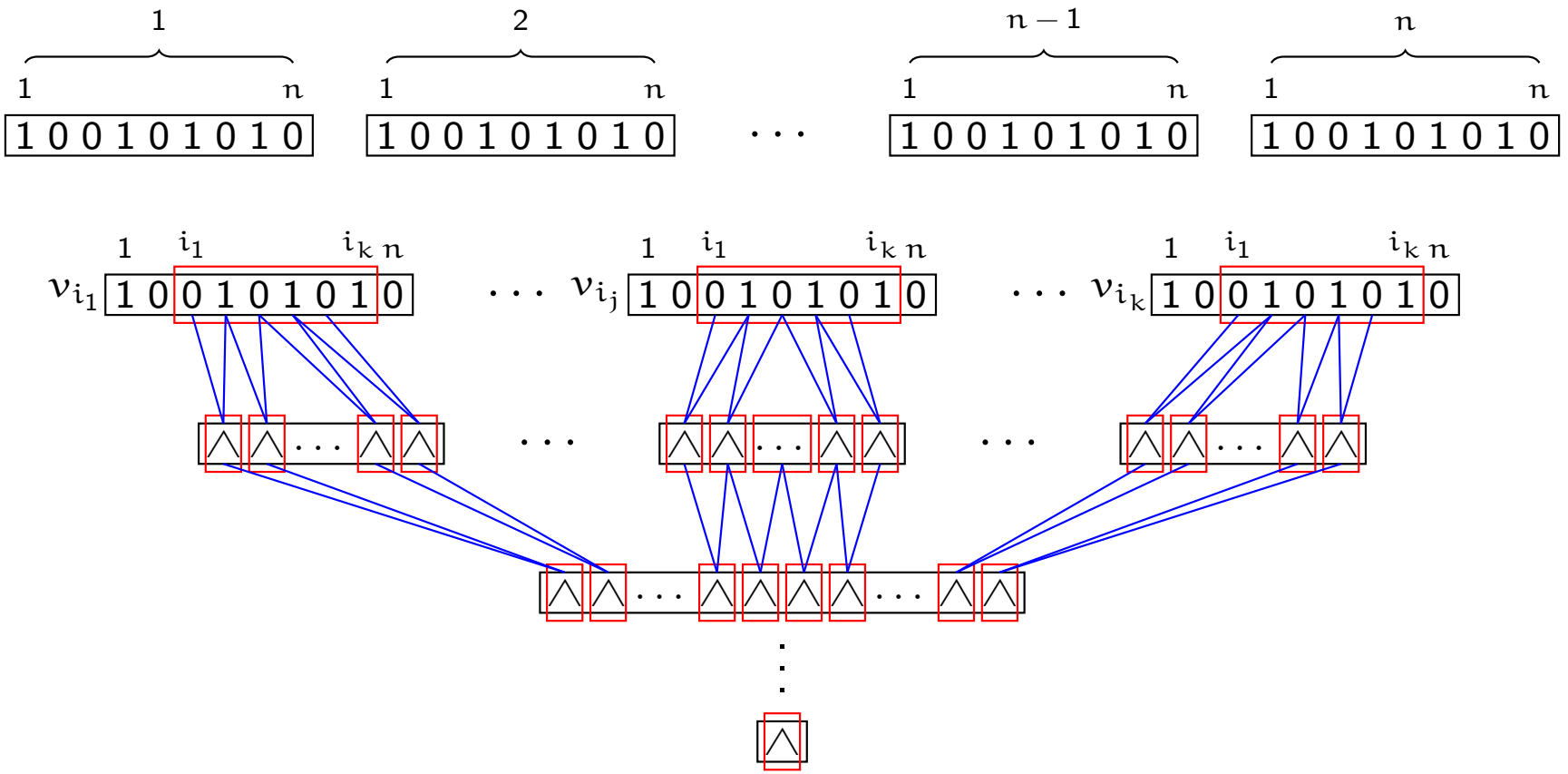
- Todo circuito monotônico computa uma função monotônica
- O circuito utilizado na demostração da conjectura é monotônico, i.e., o problema ($\mathbf{P}$-completo) de avaliar um circuito é redutível via circuitos monotônicos (expressividade!)

Exemplos

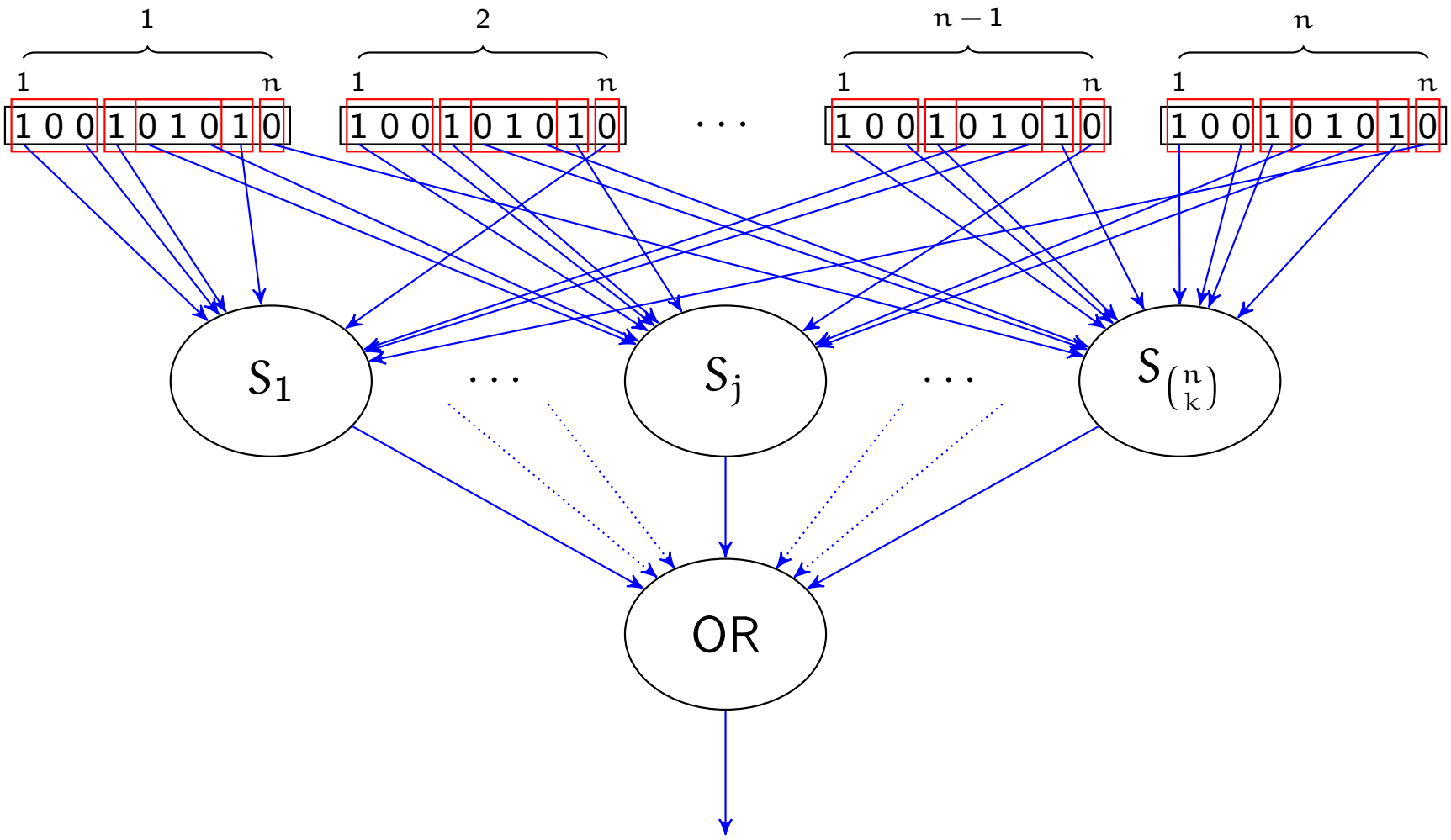
- Monotônicos: Reachability, Hamilton Cycle, Clique
- Não-monotônicos: Knapsack, cobertura euleriana

Circuitos ingênuos monotônicos para computar $\textsc{Clique}_{n,k}$

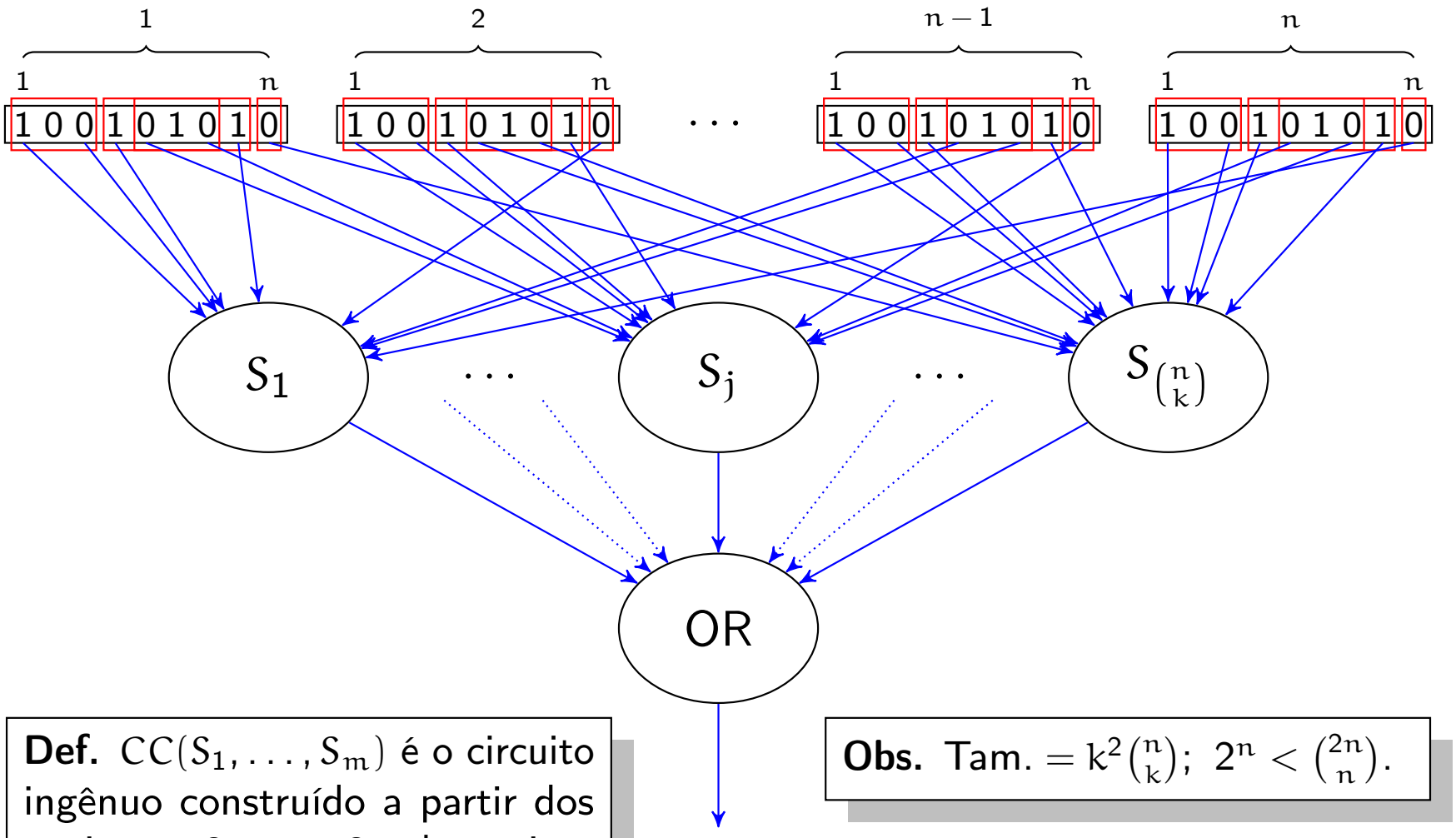
Circuitos ingênuos monotônicos para computar $\textsc{Clique}_{n,k}$



Circuitos ingênuos monotônicos para computar $\textsc{Clique}_{n,k}$



Circuitos ingênuos monotônicos para computar $\textsc{Clique}_{n,k}$



Def. $CC(S_1, \ldots, S_m)$ é o circuito ingênuo construído a partir dos conjuntos $S_1, \ldots, S_m$ de vértices.

Obs. Tam. $= k^2\binom{n}{k}$; $2^n < \binom{2n}{n}$.

Sumário

Teorema de Razborov

Teorema

Existe uma constante c *tal que, para qualquer* n *suficientemente grande, todos os circuitos monotônicos para* $\textsc{Clique}_{n,k}$ *em que* $k = \sqrt[4]{n}$ *tem cota inferior a* $\Omega(2^{c\sqrt[8]{n}})$

Razborov: Estratégia de prova

1. Considerar um circuito monotônico C qualquer
   - Considerar tipos particulares de exemplos negativos e positivos para teste de clique (exemplos em quantidade exponencial)
2. Aproximar C via um circuito ingênuo $CC(S_1, \ldots, S_m)$ introduzindo um número pequeno (poli) de *falsos positivos* e *falsos negativos*
3. Cada aproximação é executada sobre uma porta (*gate*) do circuito
4. Demonstra-se que o circuito ingênuo resultante da aplicação de "3" a todas as portas de C tem uma quantidade exponencial de falsos positivos e falos negativos
5. Conclui-se que C só pode ter uma quantidade exponencial de portas

Positivos e negativos

Exemplo positivo

Grafo simples com n vértices contendo um subgrafo completo de tamanho k

- Existem $\binom{n}{k}$ exemplos positivos

Exemplo negativo

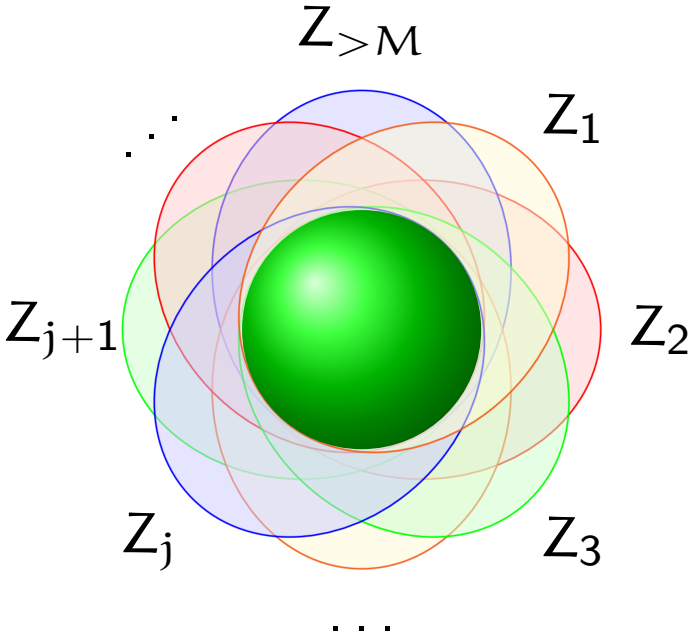
Um grafo com n vértices e $(k-1)$-colorável

- Existem $(k-1)^n$ exemplos negativos

Resultado importante para o teorema de Razborov

Lema de Erdös-Rado

Seja F *uma família de conjuntos com mais que* $\mathrm{M} = (\mathrm{p}-1)^{\mathrm{L}} \cdot \mathrm{L}!$ *elementos, todos não vazios e todos com cardinalidade máxima* L. *Então* F *contém um "girassol",*



$$Z_i \cap Z_j = Z_k \cap Z_\nu = \textit{Núcleo}$$

Prova

Via indução em L.

Construção indutiva do circuito aproximante para C (1)

Base: $C = g_{ij}$

Porta de entrada g_{ij}, $\text{Aprox}(C) = \text{CC}(\{g_{ij}\})$

Passo indutivo: $C = C_1$ OR C_2

$$\text{Aprox}(C_1) = \text{CC}(F_1)$$
$$\text{Aprox}(C_2) = \text{CC}(F_2)$$
$$\text{Aprox}(C) = \text{CC}(F_1 \cup F_2)$$

Obs. Manter a cardinalidade dos geradores do circuito ingênuo.

Se $|F_1 \cup F_2| > M$, substituir os elementos do "girassol" pelo seu núcleo

$$\text{Aprox}(C) = \text{CC}(\text{Nu}(F_1 \cup F_2))$$

Se usarmos $M = (p-1)^L \cdot L!$, forçamos a existência de "girassóis" com pétalas de tamanho L

Construção indutiva do circuito aproximante para C (2)

Passo indutivo: $C = C_1$ AND C_2

$$\text{Aprox}(C_1) = CC(F_1)$$
$$\text{Aprox}(C_2) = CC(F_2)$$
$$\text{Aprox}(C) = CC(\text{Nu}(\{X \cup Y : X \in F_1 \text{ e } |X \cup Y| \leqslant L\}))$$

□

Lemas

1. Cada aproximação introduz no máximo,

$$M^2 2^{-p} (k-1)^n \quad \text{falsos positivos}$$

$$M^2 \binom{n-L-1}{k-L-1} \quad \text{falsos negativos}$$

2. Todo circuito aproximante ou é idêntico ao circuito "falso" ou resulta em "verdadeiro" em pelo menos metade dos exemplos negativos

O que é introduzir falsos positivos ou falsos negativos?

Obs. No caso base da construção do circuito aproximante $\mathrm{Aprox}(\{g_{ij}\}) = \mathrm{CC}(\{g_{ij}\})$, a aproximação é exata!

- Se $C = C_1$ OR C_2, $\mathrm{Aprox}(C_i) = \mathrm{CC}(F_i)$, $\mathrm{CC}(F_i)$ retorna "verdadeiro" para algum i quando alimentado com um exemplo positivo G, e $\mathrm{Aprox}(C)$ retorna "falso" quando alimentado com G

$$\Rightarrow \mathrm{Aprox}(C) \text{ introduziu um falso negativo}$$

- Se $C = C_1$ OR C_2, $\mathrm{Aprox}(C_i) = \mathrm{CC}(F_i)$, $\mathrm{CC}(F_i)$ retorna "falso" para algum i quando alimentado com um exemplo negativo G, e $\mathrm{Aprox}(C)$ retorna "verdadeiro" quando alimentado com G

$$\Rightarrow \mathrm{Aprox}(C) \text{ introduziu um falso positivo}$$

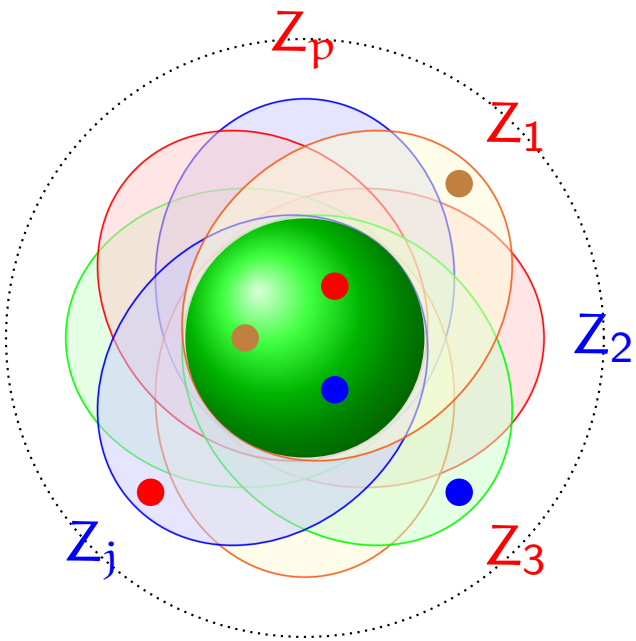
- $C = C_1$ AND C_2, similar aos casos anteriores

Lema: Máx. $M^2 2^{-p}(k-1)^n$ falsos positivos por aprox. (1)

Caso $C = C_1$ OR C_2

Então $\text{Aprox}(C_i) = CC(F_i)$ e $\text{Aprox}(C) = CC(\text{Nu}(F_1 \cup F_2))$.

- A introdução de falsos positivos é "culpa" do operador Nu
- Substitui-se $(Z_1, \ldots, Z_p)$, um "girassol", por Z, seu núcleo em $F_1 \cup F_2$
  - Se G é falso positivo, então...



$$Z_2, Z \in F_1$$

$$Z_1, Z_3, Z_p \in F_2$$

Perguntas.
(i) Quantas cores diferentes podem haver no núcleo?
(ii) Qual a probabilidade de se colorir todos os vértices em G com repetição e o núcleo não?

Lema: Máx. $M^2 2^{-p}(k-1)^n$ falsos positivos por aprox. (2)

Seja $\mathrm{Rep}(X) =$ "Há cores repetidas em X". Então

$$\mathrm{prob}(\mathrm{Rep}(Z_1) \wedge \cdots \wedge \mathrm{Rep}(Z_p) \wedge \neg\,\mathrm{Rep}(Z)) \leqslant \mathrm{prob}(\mathrm{Rep}(Z_1) \wedge \cdots \wedge \mathrm{Rep}(Z_p)) \mid \neg\,\mathrm{Rep}(Z)$$

$$\frac{\mathrm{prob}(\mathrm{Rep}(Z_1) \wedge \cdots \wedge \mathrm{Rep}(Z_p) \wedge \neg\,\mathrm{Rep}(Z))}{\neg\,\mathrm{Rep}(Z)} = \mathrm{prob}(\mathrm{Rep}(Z_1) \wedge \cdots \wedge \mathrm{Rep}(Z_p)) \mid \neg\,\mathrm{Rep}(Z)$$

$$= \prod_{i=1}^{p} = \mathrm{prob}(\mathrm{Rep}(Z_i) \mid \neg\,\mathrm{Rep}(Z)) \leqslant \prod_{i=1}^{p} \mathrm{prob}(\mathrm{Rep}(Z_i)) \leqslant 2^{-p}$$

$$\Rightarrow \mathrm{Rep}(Z_i) = \frac{\binom{|Z_i|}{2}}{k-1} \leqslant \frac{\binom{L}{2}}{k-1} \leqslant \frac{\frac{L!}{(L-2)!\cdot 2!}}{k-1} = \frac{\frac{(\sqrt[8]{n})!}{(\sqrt[8]{n}-2)!\cdot 2!}}{\sqrt[4]{n}-1} \leqslant \frac{1}{2}$$

Estimativa de falsos positivos por operação Nu na OR-aproximação:

$$\leqslant 2^{-1} \cdot \underbrace{(k-1)^n}_{\substack{\text{colorações} \\ \text{possíveis}}} \cdot 2 \underbrace{M/(p-1)}_{\text{operações Nu}}$$

Lema: Máx. $M^2 2^{-p}(k-1)^n$ falsos positivos por aprox. (3)

Caso $C = C_1$ AND C_2

$$\mathsf{Aprox}(C) = \mathsf{CC}(\,\mathsf{Nu}\,(\{X \cup Y : X \in F_1 \text{ e } |X \cup Y| \leqslant L\}))$$

Lema: Máx. $M^2 2^{-p}(k-1)^n$ falsos positivos por aprox. (3)

Caso $C = C_1$ AND C_2

$$\text{Aprox}(C) = CC(\underbrace{\text{Nu}}(\{\underbrace{X \cup Y}_{\text{Não inclui falsos positivos}} : X \in F_1 \text{ e } \underbrace{|X \cup Y| \leqslant L}_{\text{Pode retirar positivos mas nunca inclui falsos positivos}}\}))$$

Inclui no máximo 2^{-p} falsos positivos por Nu

Lema: Máx. $M^2 2^{-p}(k-1)^n$ falsos positivos por aprox. (3)

Caso $C = C_1$ AND C_2

$$\text{Aprox}(C) = CC(\underbrace{\text{Nu}}(\{\underbrace{X \cup Y}_{\text{Não inclui falsos positivos}} : X \in F_1 \text{ e } \underbrace{|X \cup Y| \leqslant L}_{\text{Pode retirar positivos mas nunca inclui falsos positivos}}\}))$$

Inclui no máximo 2^{-p} falsos positivos por Nu

Logo, estimativa de falsos positivos por operação Nu na AND-aproximação:

$$\leqslant 2^{-p}(k-1)^n M^2$$

□

Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (1)

1. Nu não introduz falsos negativos

   $\Rightarrow$ A aproximação para OR não introduz falsos negativos

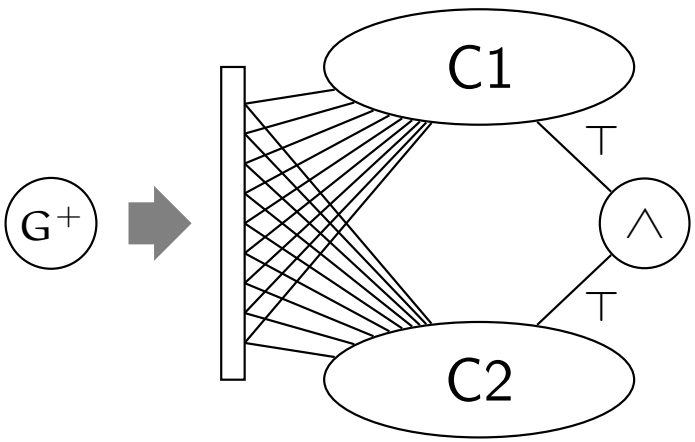
2. $C = C_1$ AND C_2

Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (1)

1. Nu não introduz falsos negativos

   $\Rightarrow$ A aproximação para OR não introduz falsos negativos

2. $C = C_1$ AND C_2



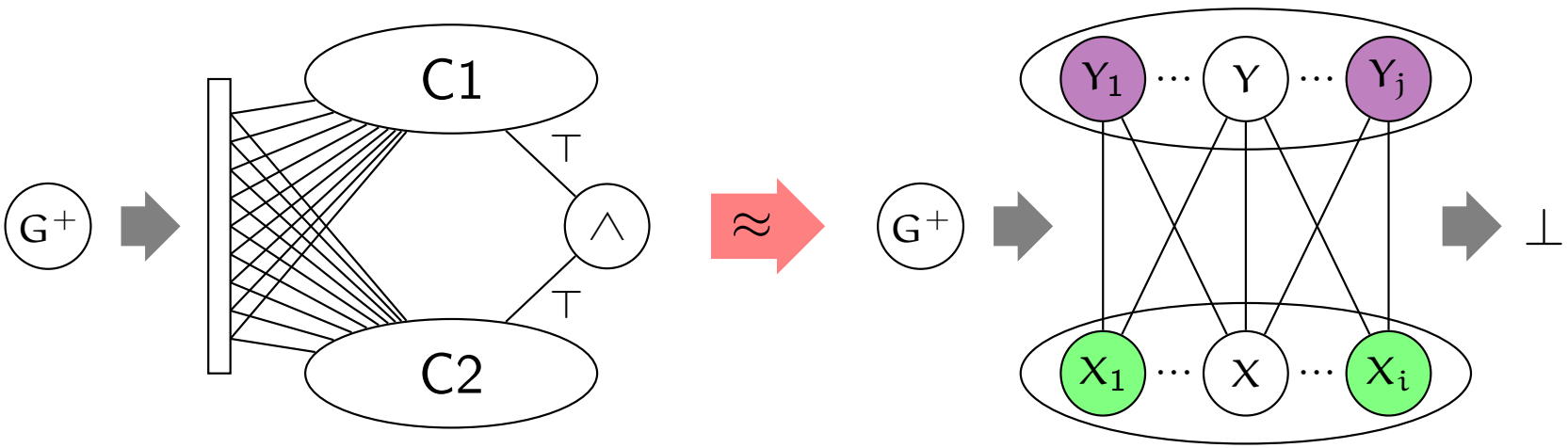
Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (1)

1. Nu não introduz falsos negativos

   $\Rightarrow$ A aproximação para OR não introduz falsos negativos

2. $C = C_1$ AND C_2

$$CC(\{X \cup Y : X \in F_2, Y \in F_1 \text{ e } |X \cup Y| \leqslant L\})$$



Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (1)

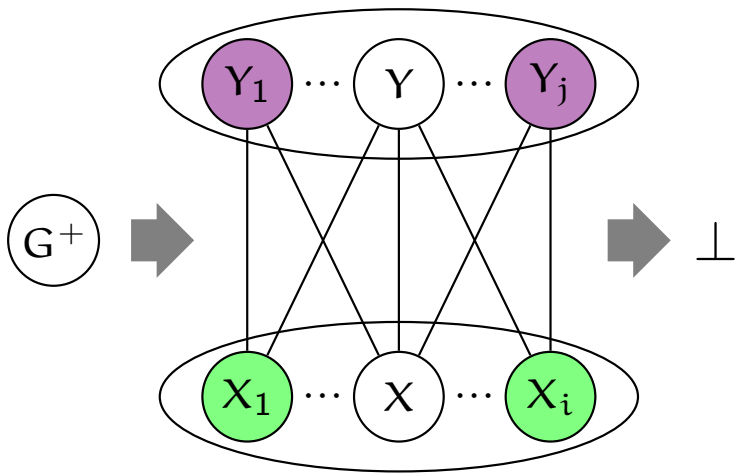
1. Nu não introduz falsos negativos

   $\Rightarrow$ A aproximação para OR não introduz falsos negativos

2. $C = C_1$ AND C_2

$$CC(\{X \cup Y : X \in F_2, Y \in F_1 \text{ e } |X \cup Y| \leqslant L\})$$

$$M^2\binom{n-L-1}{k-L-1} \approx$$



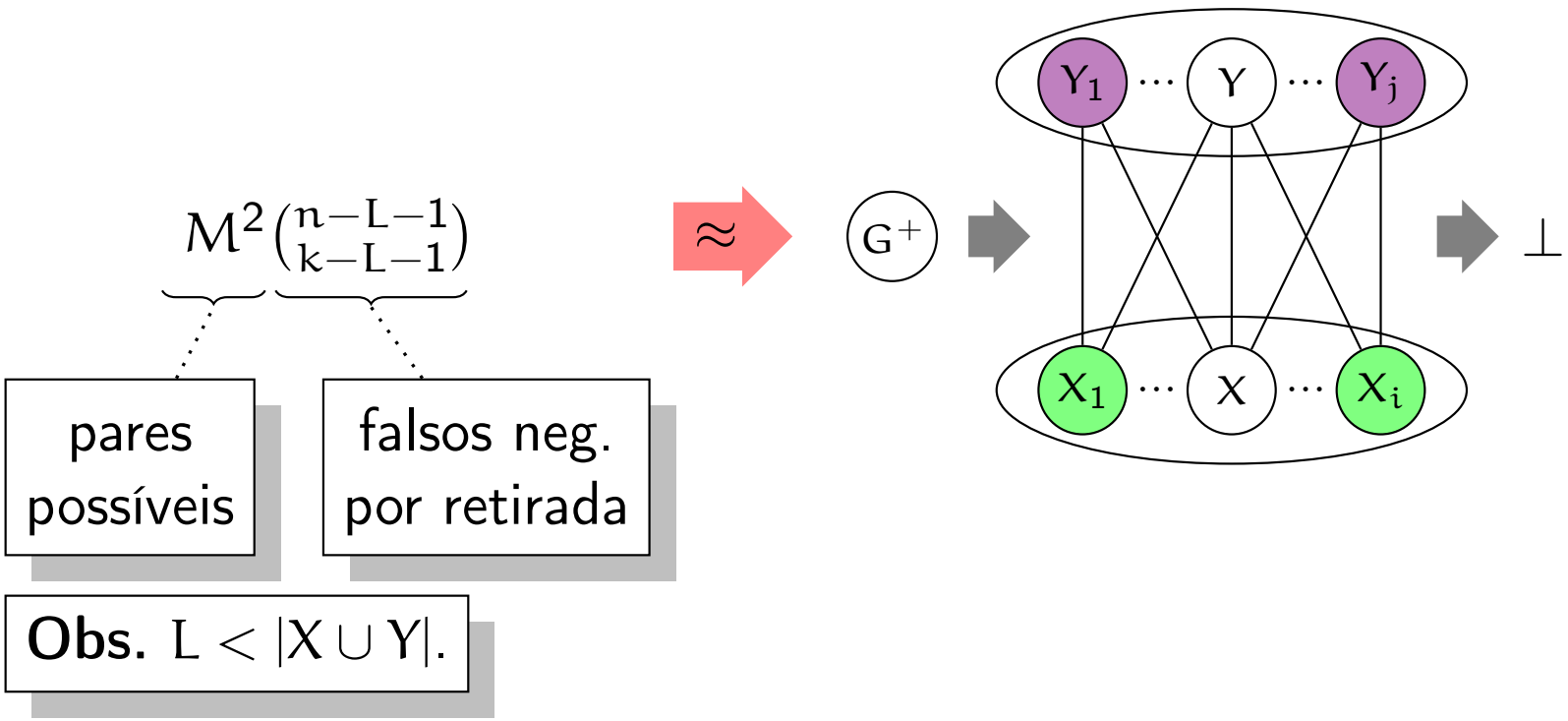
Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (1)

1. Nu não introduz falsos negativos

   $\Rightarrow$ A aproximação para OR não introduz falsos negativos

2. $C = C_1$ AND C_2

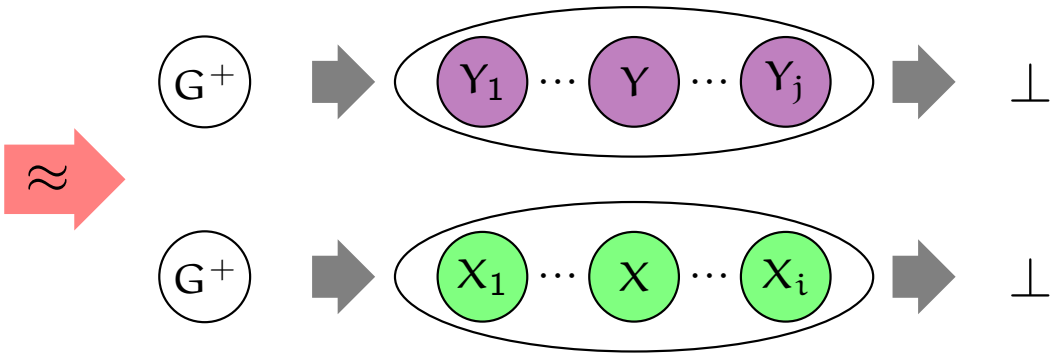
$$CC(\{X \cup Y : X \in F_2, Y \in F_1 \text{ e } |X \cup Y| \leqslant L\})$$



Obs. $L < |X \cup Y|$.

Lema: Máx. $M^2\binom{n-L-1}{k-L-1}$ falsos negativos por aprox. (2)

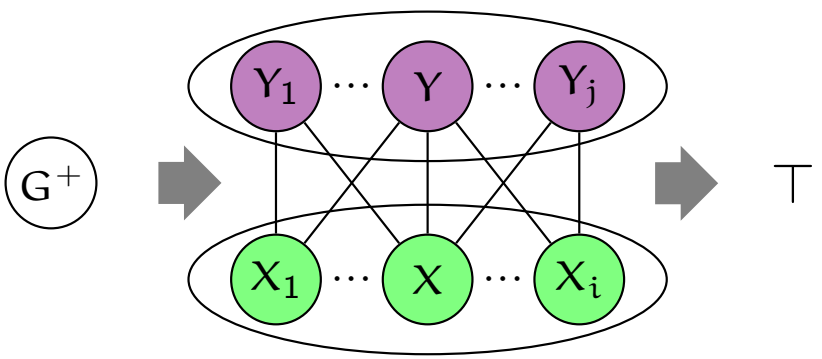
$\mathsf{Aprox}(C_1) = CC(F_1)$



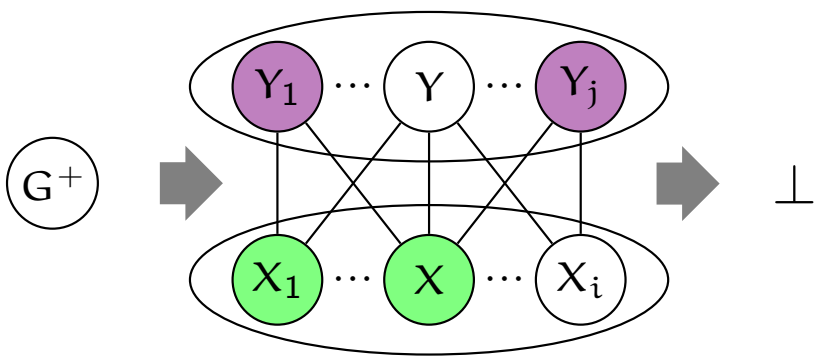
$\mathsf{Aprox}(C_2) = CC(F_2)$

$\approx$

$CC(\{X \cup Y : X \in F_2 \text{ e } Y \in F_1)$



$CC(\{X \cup Y : X \in F_2, Y \in F_1 \text{ e } |X \cup Y| \leqslant L)$



Lema: Todo circuito aproximante ou é idêntico ao circuito "falso" ou resulta em "verdadeiro" em pelo menos metade dos exemplos negativos

1. A operação de aproximação de um AND pode deletar todos os pares
2. Se o circuito aproximante, ingênuo $CC(X_1, \ldots, X_p)$, aceita algum circuito, então algum X_i aceita os exemplos negativos (têm cliques)

Logo, existem

$$\frac{\binom{|Z_i|}{2}}{k-1} \leqslant \frac{\binom{L}{2}}{k-1} \leqslant \frac{\frac{L!}{(L-2)!\cdot 2!}}{k-1} = \frac{\frac{(\sqrt[8]{n})!}{(\sqrt[8]{n}-2)!\cdot 2!}}{\sqrt[4]{n}-1} \leqslant \frac{1}{2}$$

exemplos de circuitos com $k-1$ colorações num circuito de L vértices

Sumário

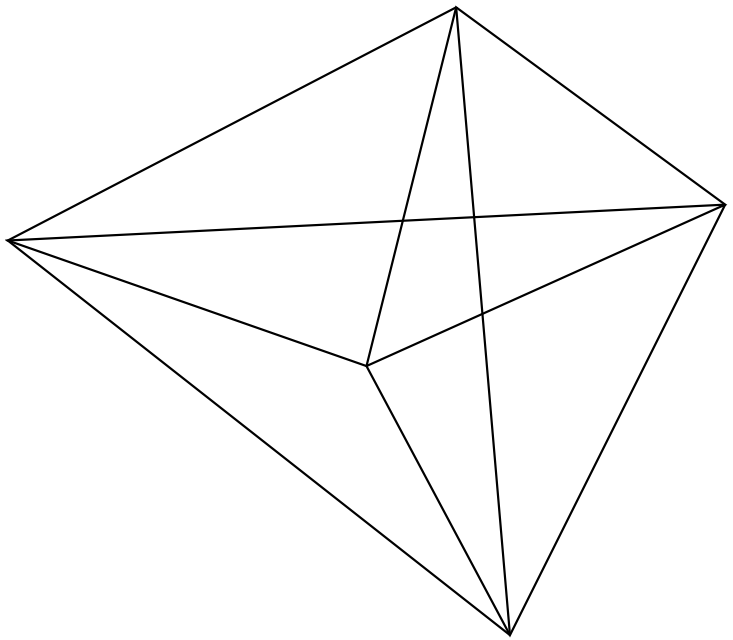
Colapso da hierarquia polinomial de complexidade

Definição

$\mathbf{P/poly}$ é a classe de linguagens aceitas por circuitos booleanos de tamanho polinomial

Exemplo positivo de tamanho n

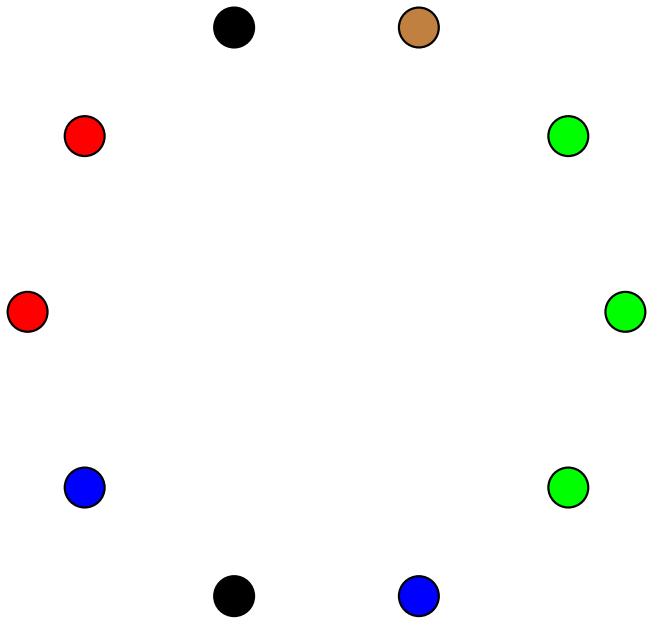
Um grafo com $\binom{k}{2}$ arestas conectando k vértices de todas as formas possíveis



Obs. Existem $\binom{n}{k}$ exemplos positivos.

Exemplo negativo de tamanho n

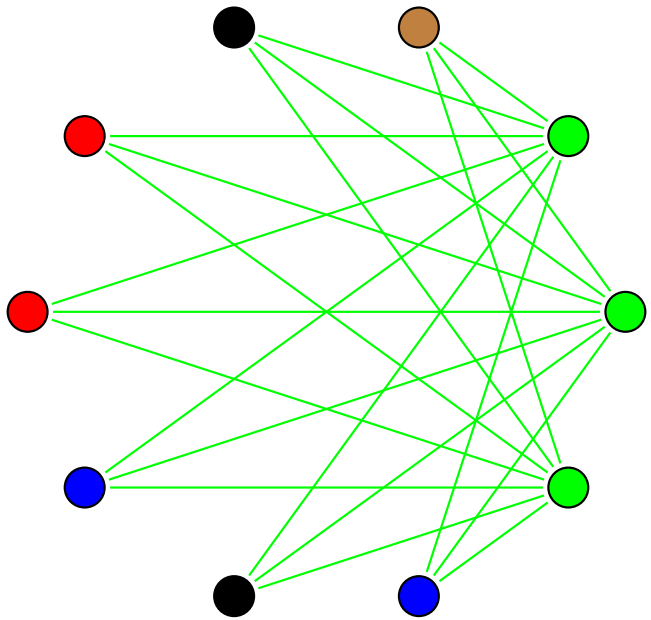
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Exemplo negativo de tamanho n

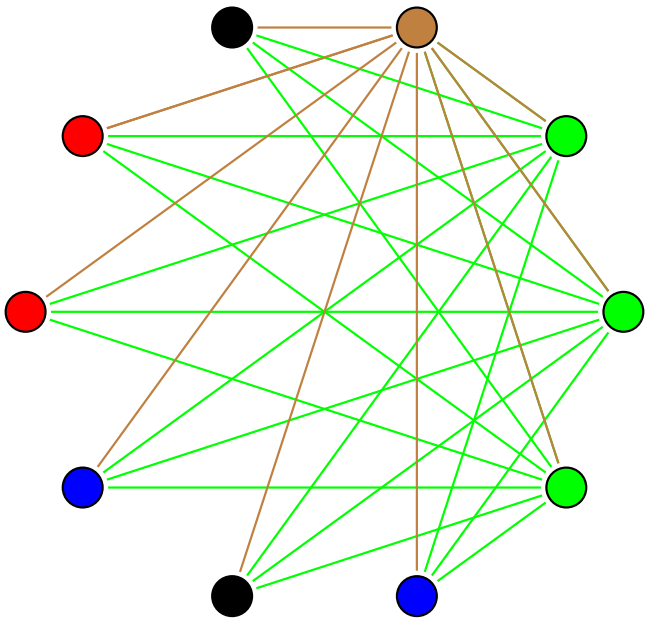
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Exemplo negativo de tamanho n

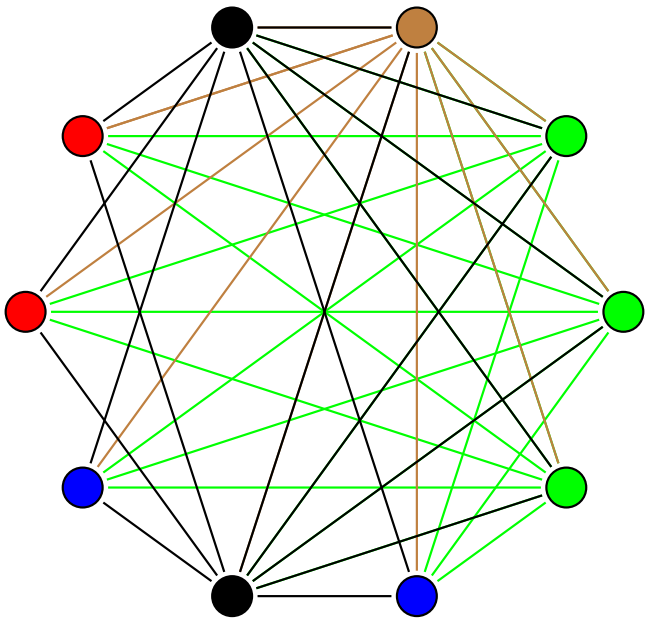
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Exemplo negativo de tamanho n

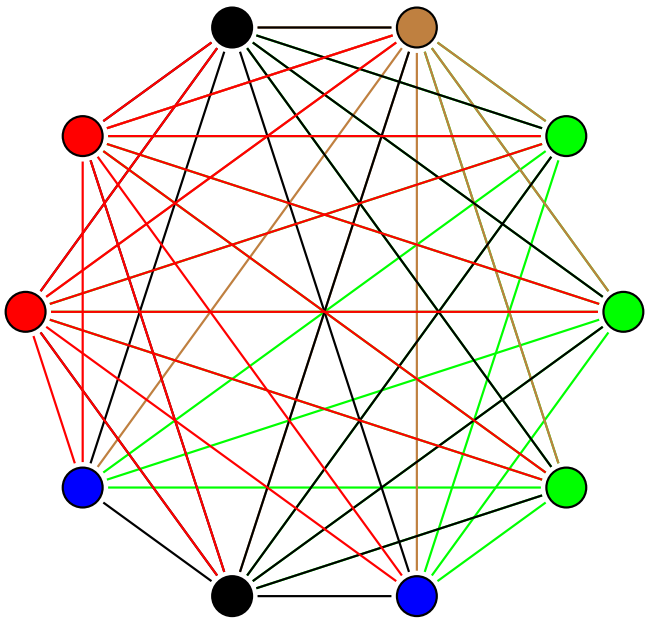
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Exemplo negativo de tamanho n

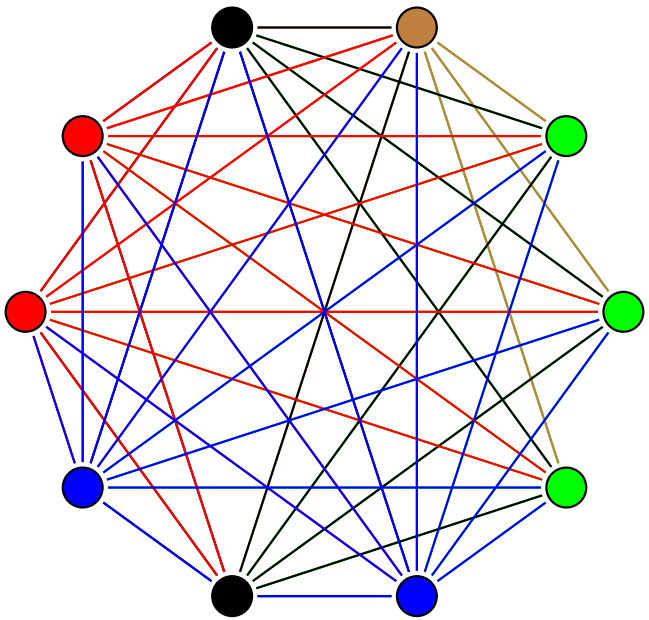
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Exemplo negativo de tamanho n

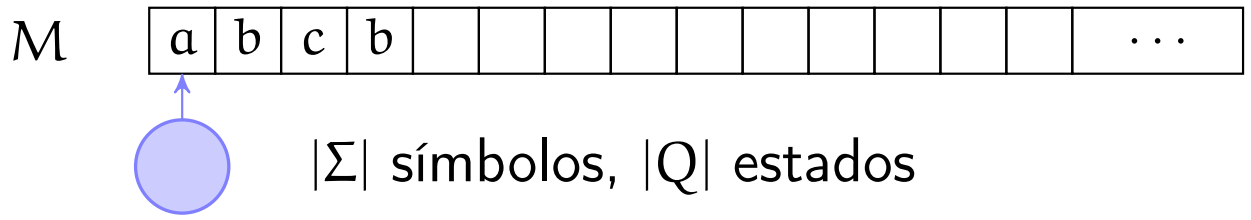
Colorir os vértices com $(k-1)$ cores diferentes e unir por uma aresta todos os pares de nós que são coloridos com cores diferentes



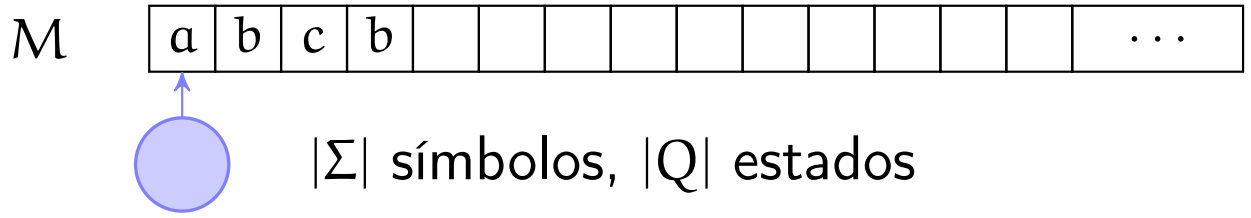
Obs. Existem $(k-1)^n$ exemplos negativos (contando isomorfismo).

Fim

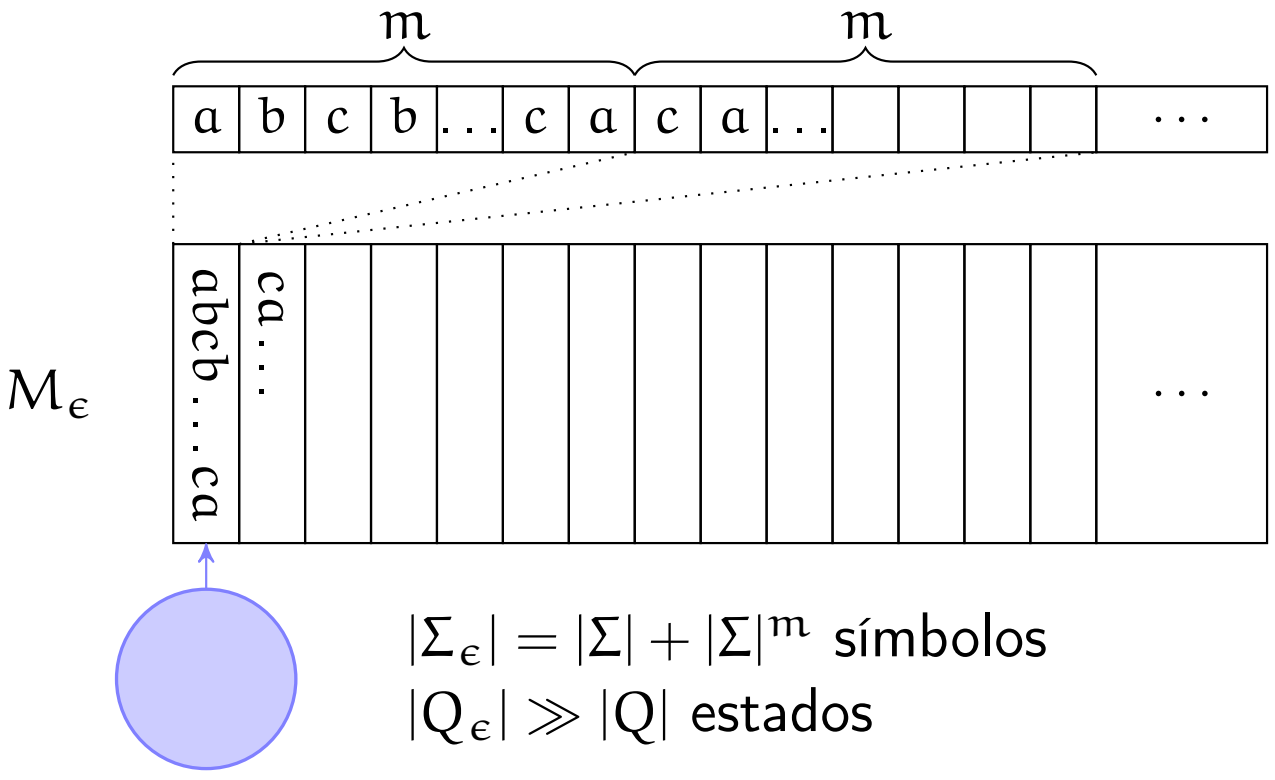
Prova: *Speedup* linear



Prova: *Speedup* linear

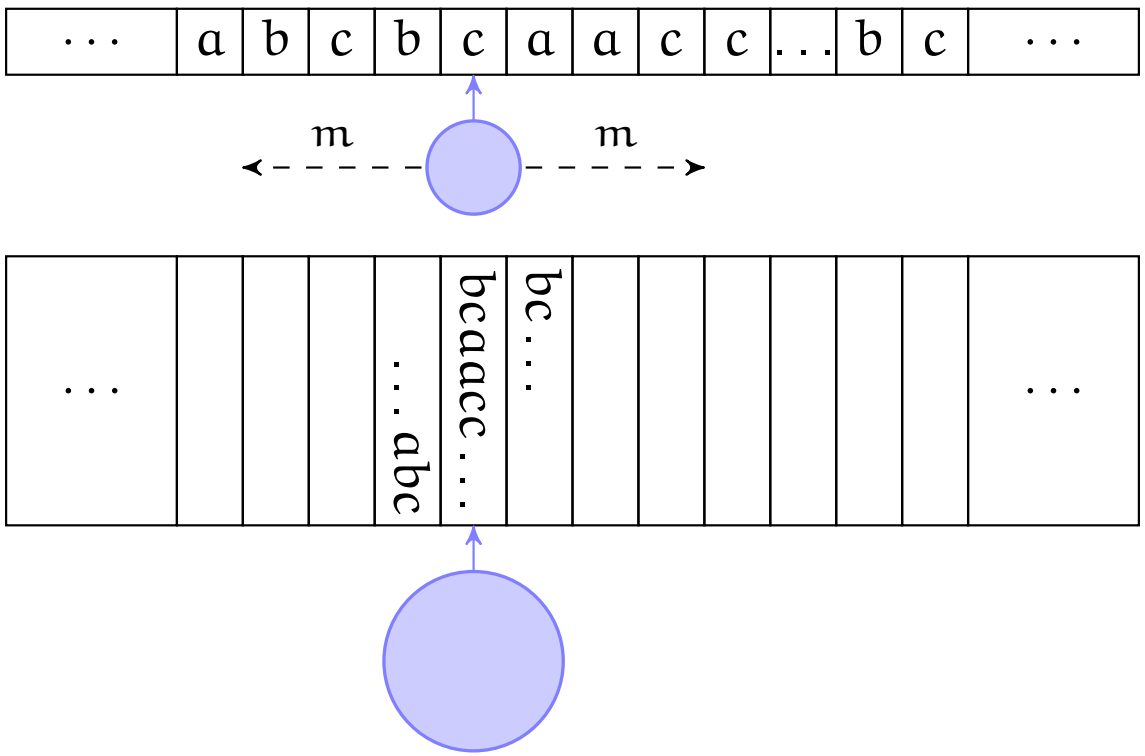


Seja m um inteiro (dependendo de ϵ e M),



Prova: *Speedup* linear—simulação de M por M_ϵ

1. Copiar os n símbolos de entrada para as n/m células de M_ϵ [$n+2$ passos]
2. Simular m passos de M em seis passos de M_ϵ [$6f(n)/m$ passos]
   - E,D,D,E: "ler" símbolos relevantes e armazená-los nos estados
   - E,D ou D,E: fazer as m substituições
3. Total de $6f(n)/m+n+2$ passos, fazendo-se $m = 6/\epsilon$ obtém-se o resultado



Voltar

Prova: $\exists f(\mathbf{TIME}(f(n)) = \mathbf{TIME}(2^{f(n)}))$

$$\text{Pred}(i,k) \quad \text{sse} \quad \forall M_{0 \leqslant j \leqslant i},\ \forall x \in \Sigma_j^*,\ \text{tal que } |x| \leqslant i,$$
$$\text{steps}(M_j, x) \leqslant k \text{ e } \text{steps}(M_j, x) > 2^k$$

$$N(i) = \sum_{j=0}^{i} |\text{alfabeto}_j|^i$$

$$f(i) = 2^{\underbrace{2^{\cdot^{\cdot^{\cdot^{2}}}}}_{s \text{ vezes}}}$$

em que

$$s \leqslant N(i) \quad \text{e} \quad \text{Pred}(i, 2^{\underbrace{2^{\cdot^{\cdot^{\cdot^{2}}}}}_{s \text{ vezes}}})$$

◂ Voltar

Prova: $\text{Para}_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$

Suponha que $\text{Para}_f \in \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$ e seja

$$\text{Diag}_f(T) = \begin{cases} \text{“não”} & \text{se } T_{\text{Para}_f}(T, T) = \text{“sim”} \\ \text{“sim”} & \text{senão} \end{cases}$$

Diag_f roda em tempo $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$

$\text{Diag}_f(\text{Diag}_f) = \text{“sim”}$ $\qquad$ $T_{\text{Para}_f}(\text{Diag}_f, \text{Diag}_f) = \text{“não”}$

$\perp$

Diag_f não aceita sua descrição em tempo $f(n)$

$\text{Diag}_f(\text{Diag}_f) = \text{“não”}$ $\qquad$ $T_{\text{Para}_f}(\text{Diag}_f, \text{Diag}_f) = \text{“sim”}$

$\perp$

Diag_f aceita sua descrição em tempo $f(n)$

◂ Voltar

Prova: Existe um oráculo B tal que $\mathbf{P}^{\mathrm{B}} = \mathbf{NP}^{\mathrm{B}}$

Se $\mathrm{B} = \{\langle \mathsf{T}, w, 1^{\mathrm{k}} \rangle : w \text{ é aceita por T usando no máximo k células}\}$, então

$$\mathbf{P}^{\mathrm{B}} \subseteq \mathbf{NP}^{\mathrm{B}} \qquad \text{(trivial)}$$

$$\mathbf{NP}^{\mathrm{B}} \subseteq \mathbf{P}^{A}$$

Prova: Existe um oráculo B tal que $\mathbf{P}^{\mathrm{B}} = \mathbf{NP}^{\mathrm{B}}$

Se $\mathrm{B} = \{\langle \mathsf{T}, w, 1^{\mathrm{k}} \rangle : w$ é aceita por T usando no máximo k células$\}$, então

$$\mathbf{P}^{\mathrm{B}} \subseteq \mathbf{NP}^{\mathrm{B}} \qquad \text{(trivial)}$$

$$\mathrm{L} \in \mathbf{NP}^{\mathrm{B}} \subseteq \mathbf{P}^{A}$$

Prova: Existe um oráculo B tal que $\mathbf{P}^{\mathrm{B}} = \mathbf{NP}^{\mathrm{B}}$

Se $\mathrm{B} = \{\langle \mathsf{T}, w, 1^{\mathrm{k}} \rangle : w$ é aceita por T usando no máximo k células$\}$, então

$$\mathbf{P}^{\mathrm{B}} \subseteq \mathbf{NP}^{\mathrm{B}} \qquad \text{(trivial)}$$

$$\mathrm{L} \in \mathbf{NP}^{\mathrm{B}} \subseteq \mathbf{P}^{A}$$

$w \in \mathrm{L}$

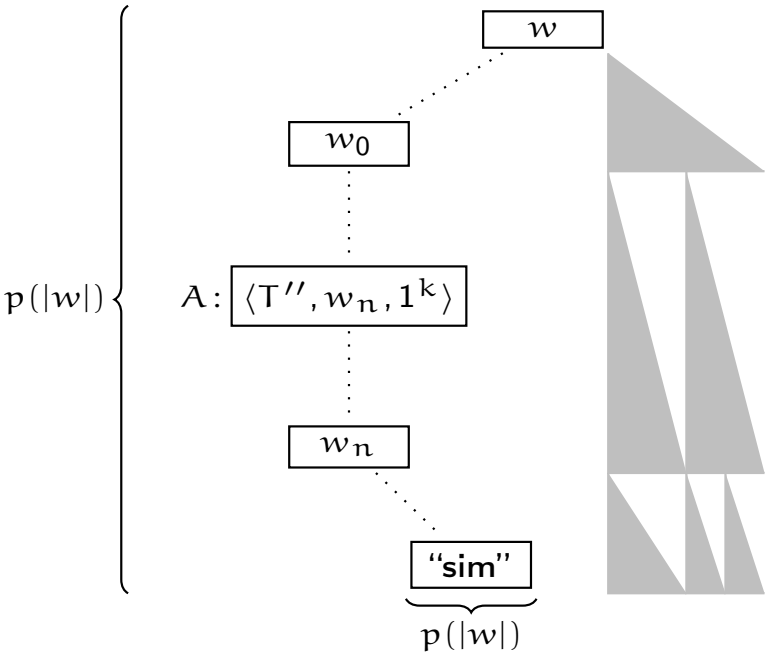
Prova: Existe um oráculo B tal que $\mathbf{P}^B = \mathbf{NP}^B$

Se $B = \{\langle T, w, 1^k\rangle : w$ é aceita por T usando no máximo k células$\}$, então

$$\mathbf{P}^B \subseteq \mathbf{NP}^B \qquad \text{(trivial)}$$

$$L \in \mathbf{NP}^B \subseteq \mathbf{P}^A$$

$w \in L$ T

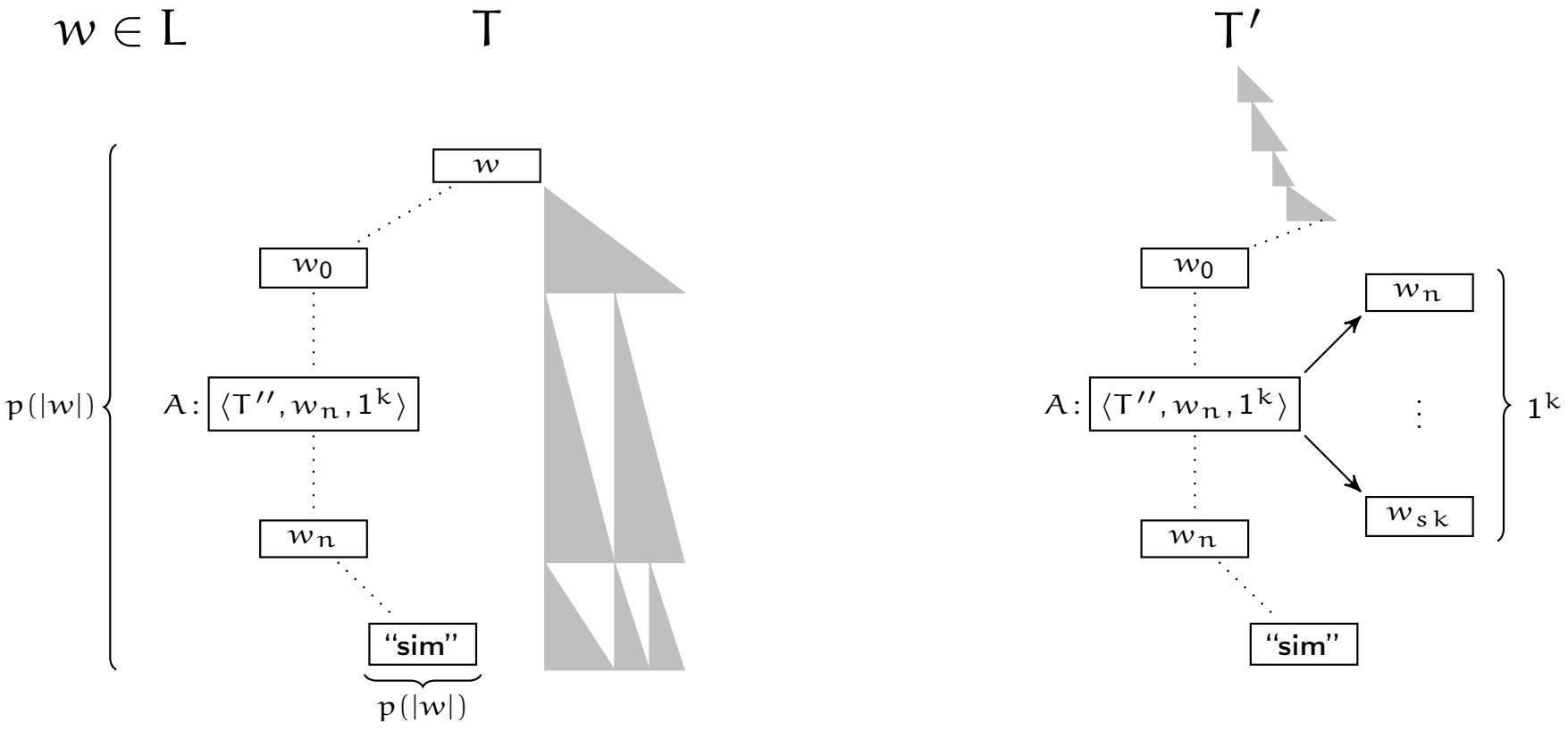


Prova: Existe um oráculo B tal que $\mathbf{P}^B = \mathbf{NP}^B$

Se $B = \{\langle T, w, 1^k\rangle : w \text{ é aceita por } T \text{ usando no máximo } k \text{ células}\}$, então

$$\mathbf{P}^B \subseteq \mathbf{NP}^B \qquad \text{(trivial)}$$

$$L \in \mathbf{NP}^B \subseteq \mathbf{P}^A$$

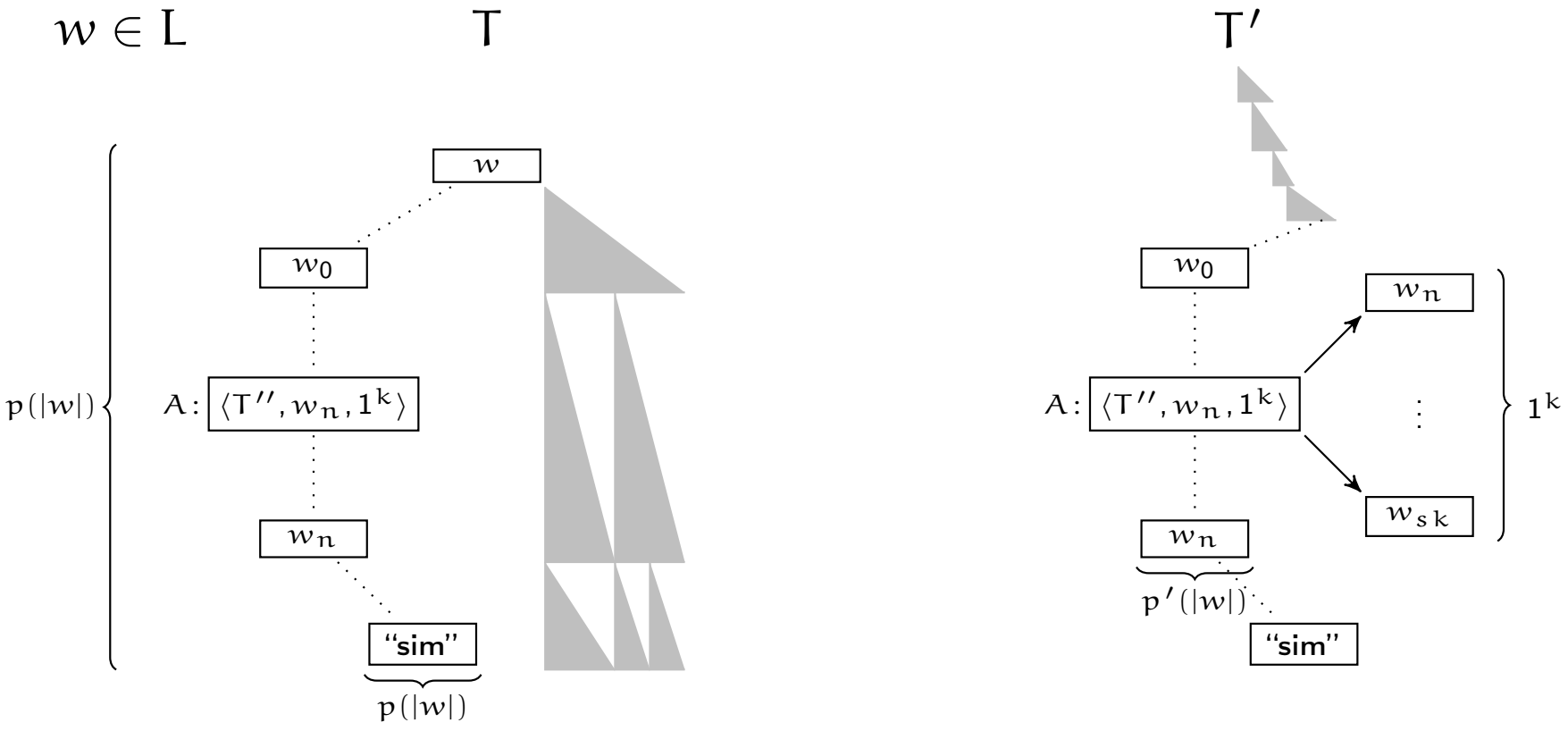


Prova: Existe um oráculo B tal que $\mathbf{P}^B = \mathbf{NP}^B$

Se $B = \{\langle T, w, 1^k\rangle : w \text{ é aceita por } T \text{ usando no máximo } k \text{ células}\}$, então

$$\mathbf{P}^B \subseteq \mathbf{NP}^B \qquad \text{(trivial)}$$

$$L \in \mathbf{NP}^B \subseteq \mathbf{P}^A$$



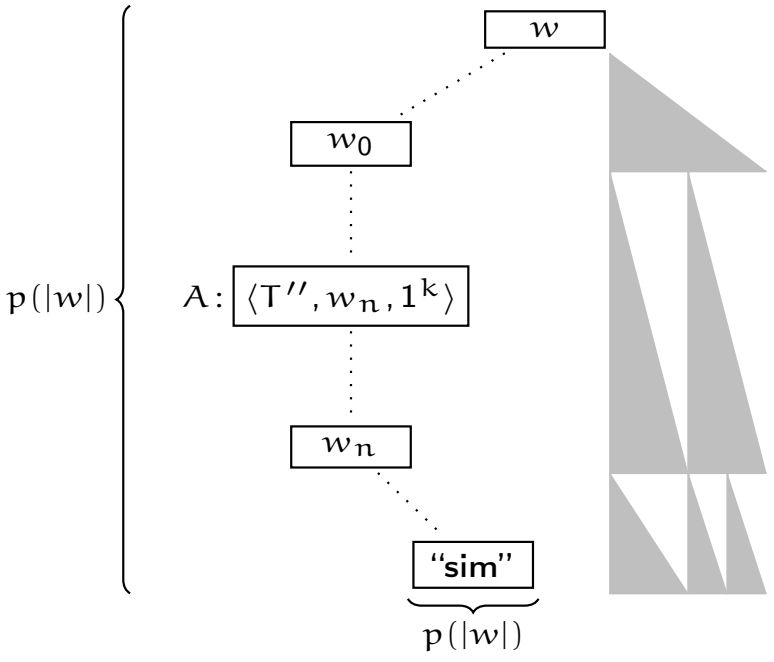
Prova: Existe um oráculo B tal que $\mathbf{P}^B = \mathbf{NP}^B$

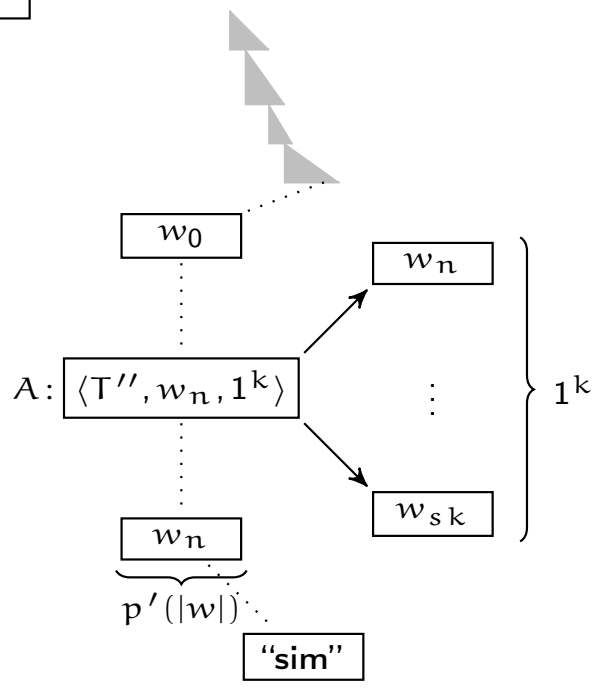
Se $B = \{\langle T, w, 1^k\rangle : w \text{ é aceita por } T \text{ usando no máximo } k \text{ células}\}$, então

$$\mathbf{P}^B \subseteq \mathbf{NP}^B \qquad \text{(trivial)}$$

$$L \in \mathbf{NP}^B \subseteq \mathbf{P}^A \ni L$$

$w \in L$ $\quad T$ $\quad A: \langle T', w, 1^{p'(|w|)}\rangle$ $\quad T'$





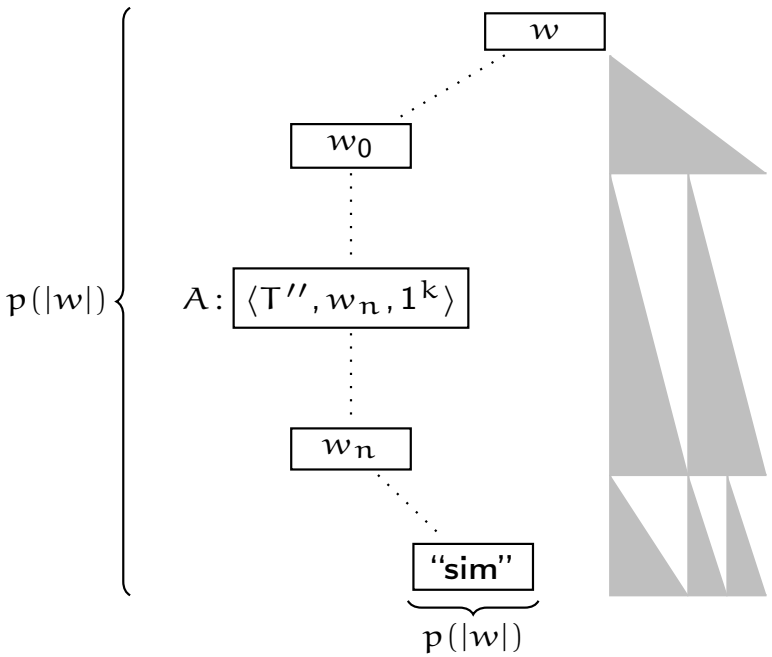
Prova: Existe um oráculo B tal que $\mathbf{P}^B = \mathbf{NP}^B$

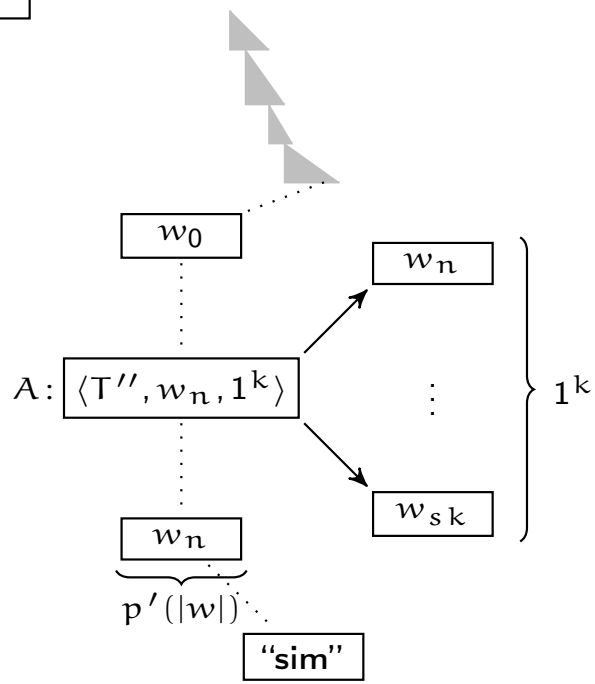
Se $B = \{\langle T, w, 1^k\rangle : w \text{ é aceita por } T \text{ usando no máximo } k \text{ células}\}$, então

$$\mathbf{P}^B \subseteq \mathbf{NP}^B \qquad \text{(trivial)}$$

$$L \in \mathbf{NP}^B \subseteq \mathbf{P}^A \ni L$$

$w \in L$ $\quad T \quad$ $A: \langle T', w, 1^{p'(|w|)}\rangle$ $\quad T'$





◂ Voltar

Prova: Existe um oráculo C tal que $\mathbf{P}^C \neq \mathbf{NP}^C$

Sejam

- X um oráculo qualquer
- $\{T_0, T_1, \ldots, T_n, \ldots\}$ uma enumeração construtiva de $\mathbf{P}^X$ tal que, para todo x e k, $T_k(x)$ para em no máximo $|x|^k + k$ passos
- $L_x = \{0^n : \exists x(x \in X \text{ e } |x| = n)\}$ $\qquad (L_x \in \mathbf{NP}^X)$

Vamos definir um oráculo C (do tipo X) tal que $L_C \notin \mathbf{P}^C$

> **Obs.** $|x|^k + k < 2^{|x|}$, i.e., a quantidade de strings de tamanho $|x|$ é maior do que qualquer polinômio em $|x|$.

Qualquer $T_k \in \mathbf{P}^C$ calculando $T_k(w)$ pode submeter no máximo $|x|^k + k$ strings de tamanho $|x|$ ao oráculo C,

$$\mathbf{NP}^C \not\subset \mathbf{P}^C$$

Prova: Existe um oráculo C tal que $\mathbf{P}^C \neq \mathbf{NP}^C$ (cont.)

Diagonalização em partes para definir C:

$$C(0) = \varnothing \qquad C(i) = \{w : w \in C \text{ e } |w| = i\} \qquad \|W\| = \max\{|w| : w \in W\}$$

Objetivo

- Se $T_i^{C(i-1)}(0^i) =$ "sim", então $C(i) = \varnothing$
- Se $T_i^{C(i-1)}(0^i) =$ "não", então

$$C(i) = \{\min\{w : |w| > \|C(i-1)\|^{i-1} + i - 1 \quad \text{e} \quad 2^{|w|} > |w|^i + i\}\}$$

Suponha que $L_C \in \mathbf{P}^C$. Então existe T_i que aceita L_C. Logo,

$$T_i^C(0^i) = \text{"sim"} \Rightarrow 0^i \in C(i) = \varnothing$$
$$T_i^C(0^i) = \text{"não"} \Rightarrow 0^i \notin C(i) \ni 0^i$$

O que é absurdo. Portanto,

$$L_C \notin \mathbf{P}^C$$

◂ Voltar

Prova: Lema

Sejam C_1 e C_2 construtivamente enumeráveis. Então

$$\exists T_j^i (i = 1, 2)(j \in \mathbb{N}),\ T_j^i \text{ decide } L_j^i \in C_i$$

Define-se G tal que $G \in \mathbf{P}$ e

$$\forall j \exists z \in \Sigma^*,\ z \in L_i \Delta L_j^i \text{ e } z \in G \qquad (\dagger)$$

Fazendo-se $L = (L_1 \cap G) \cup (L_2 \cap \bar{G})$,

$$(\dagger) \Rightarrow \forall j \exists z \in \Sigma^*,\ z \in L \Delta L_j^i \text{ e } z \in G$$

Prova: Lema (cont.)

Sejam

$$Z_1^{j,n}p = \text{menor palavra } z \text{ com } |z| \geqslant n \text{ e } z \in L_1 \Delta L_j^1$$
$$Z_2^{j,n}p = \text{menor palavra } z \text{ com } |z| \geqslant n \text{ e } z \in L_2 \Delta L_j^i$$
$$R_k(n) = \max_{i \leqslant n}\{|Z_k^{i,n}|\} + 1$$

Então

$$L_1 \notin C_1 \Rightarrow L_1 \Delta L_1^i \neq \varnothing, \text{ para todo } i$$
$$C_1 \text{ fechada por var. finita} \Rightarrow \forall j, n \geqslant j, \exists z \in L_1 \Delta L_1^j$$

Logo, R_1 é total e computável. Sejam

$$R(n) \geqslant \max\{R_1(n), R_2(n)\} \qquad \text{(tempo construtível)}$$
$$G = \{x : R^{2n}(0) \leqslant |x| \leqslant R^{2n+1}(0) \text{ e } n \geqslant 0\}$$

Então $G \in \mathbf{P}$

Prova: Lema (cont.)

Seja T uma máquina de Turing que conta o tempo de R. Então

$$T(x) \text{ para em exatamente } R(|x|) \text{ passos}$$

Algoritmo

1. Executar $|x|$ passos de $T(1^0)$ verificando se
   $$R(0) > |x| \Rightarrow R(0) > |x| > 0$$
2. Executar $|x|$ passos de $T(1^{R(0)})$ verificando se
   $$R^2(0) > |x| \Rightarrow R^2(0) > |x| > R(0)$$

$\vdots$

n. Executar $|x|$ passos de $T(1^{R^n(0)})$ verificando se
   $$R^n(0) > |x| \Rightarrow R^n(0) > |x| > R^{n-1}(0)$$

Até achar o intervalo e, então, verificar se n é par ou impar

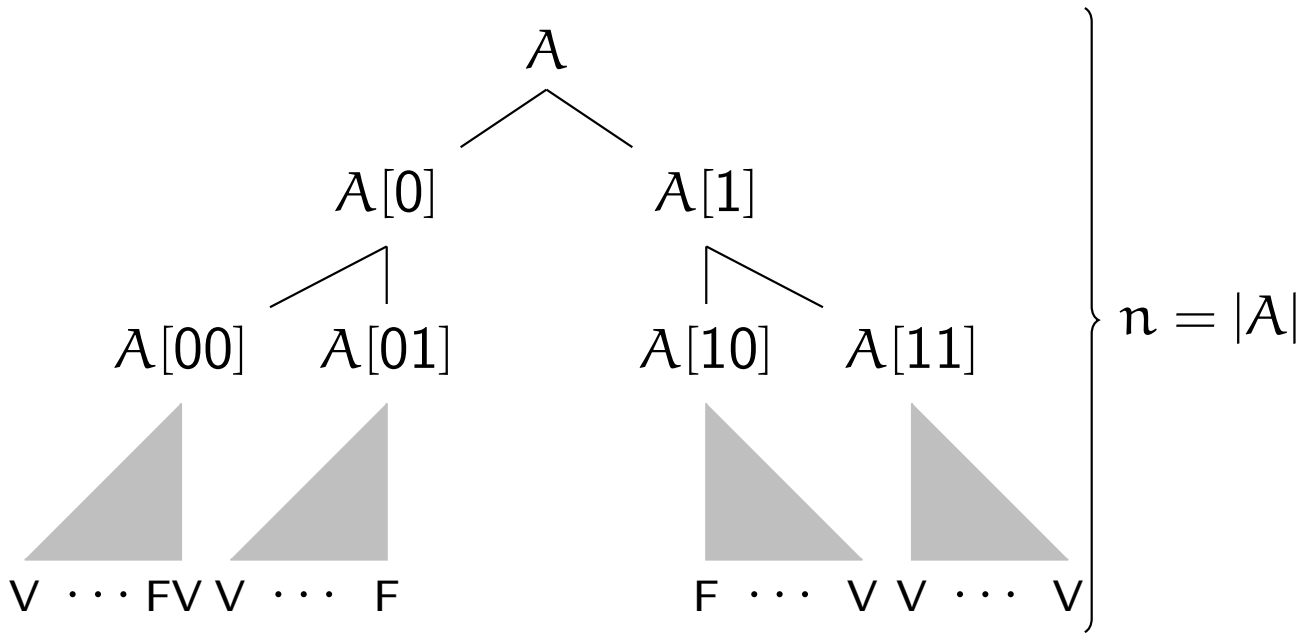
◂ Voltar

Prova: $\exists L$ unária $\mathbf{NP}$-completa $\Rightarrow \mathbf{P} = \mathbf{NP}$

Se $\exists L$ unária e $\mathbf{NP}$-completa, então existe uma redução R polinomial tal que

$$x \in \textsc{Sat} \Leftrightarrow R(x) \in L$$

Seja $t \in \{0,1\}^*$ e seja $A[t]$ a fórmula A avaliada parcialmente por t (ordem nas letras),



Prova: $\exists$L unária **NP**-completa $\Rightarrow$ **P** = **NP** (cont.)

Construção de algoritmo poli para SAT

```
function verif_1(A[t])
    if |t| = n then
        return A[t] ≠ {}
    else
        return verif_1(A[t0]) or verif_1(A[t1])
    end
end
```

```
function verif_2(A[t])
    if |t| = n then
        return A[t] ≠ {}
    end
    if Tab(hash(t)) then
        return Tab(hash(t))
    else
        insere o resultado em Tab
        return verif_2(A[t0]) or verif_2(A[t1])
    end
end
```

Propriedades desejáveis do *hashing*

1. Ter domínio pequeno (polinomial)
2. Se $\text{hash}(t) = \text{hash}(t')$ então $A[t] \in \text{SAT}$ sse $A[t'] \in \text{SAT}$

$$\boxed{(1),\ (2) \Longrightarrow \text{hash}(t) = R(A[t]).}$$

Prova: $\exists L$ unária $\mathbf{NP}$-completa $\Rightarrow \mathbf{P} = \mathbf{NP}$ (cont.)

Estimativa de complexidade de verif_2

1. Se C é o número de chamadas recursivas então verif_2 é $\mathcal{O}(C \cdot p(n))$
2. Existe uma sequência de avaliações parciais $\{t_1, \ldots, t_k\}$ tal que
   - $k \geqslant C/2n$
   - todas as chamadas associadas são recursivas
   - nenhum t_i é prefixo de nenhum t_j $(i \neq j)$

Logo, se o valor máximo de k é $p(n)$, então

$$p(n) \geqslant C/2n \Rightarrow \mathcal{O}(np^2(n)) \text{ é a cota para } \text{verif}_2$$