Teoria da Computação

Introdução à Teoria da Complexidade

Mirtha Lina Fernández Venero
mirtha.lina@ufabc.edu.br

novembro 2017

Universidade Federal do ABC

Sumário

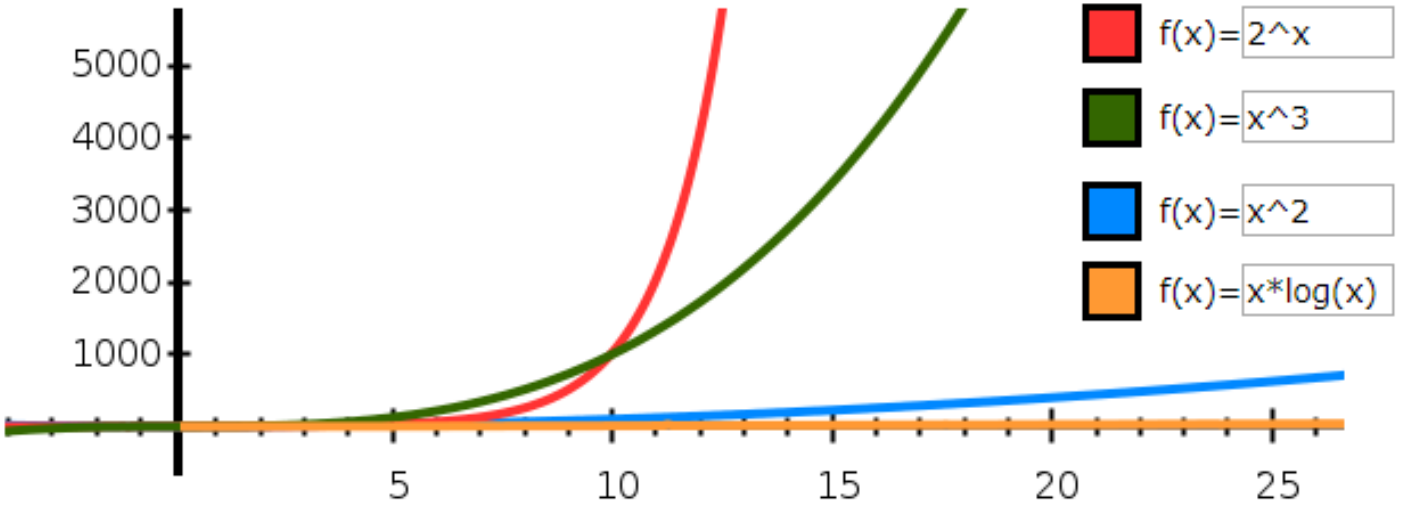
Complexidade Computacional

Na prática, nem todos os problemas decidíveis podem ser resolvidos para todas as instâncias. **Por quê?**

Complexidade Computacional

Complexidade Assintótica de Algoritmos: Usa a notações assintóticas O, Ω, Θ, o e ω para o custo dum algoritmo

- Qualquer função exponencial cresce assintoticamente mais rápido que qualquer função polinomial: $n^b = o(a^n), a > 1$
- Algoritmos polinomiais são melhores do que os exponenciais



https://graphsketch.com/

Complexidade Computacional

Complexidade Assintótica de Algoritmos: Usa a notações assintóticas O, Ω, Θ, o e ω para o custo dum algoritmo

- Qualquer função exponencial cresce assintoticamente mais rápido que qualquer função polinomial: $n^b = o(a^n), a > 1$
- Algoritmos polinomiais são melhores do que os exponenciais

Complexidade de Problemas:

- Por que há problemas computacionais mais difíceis do que outros?
- Como analisar a complexidade dum problema?
- Quantas classes de problemas existem?
- Quais problemas são os mais difíceis?
- O que fazer com esses problemas?

Complexidade dos Problemas e Intratabilidade

Problemas Computacionais: **decisão**, busca/otimização e contagem/enumeração

- Alguns problemas tem algoritmos polinomiais, e.g. caminho mínimo, árvore geradora mínima, componentes conexas, programação linear, etc

- Para outros problemas não existe algoritmo polinomial (**intratáveis**). Por exemplo, qualquer problema com saída de tamanho exponencial (Torres de Hanoi, todos os subconjuntos/permutações/caminhos hamiltonianos, etc)

Complexidade dos Problemas e Intratabilidade

Problemas Computacionais: **decisão**, busca/otimização e contagem/enumeração

- Alguns problemas tem algoritmos polinomiais, e.g. caminho mínimo, árvore geradora mínima, componentes conexas, programação linear, etc
- Existem problemas no meio, i.e. para eles existe um algoritmo exponencial porém (até hoje) nenhum algoritmo polinomial nem uma prova de que ele não existe
- Para outros problemas não existe algoritmo polinomial (**intratáveis**). Por exemplo, qualquer problema com saída de tamanho exponencial (Torres de Hanoi, todos os subconjuntos/permutações/caminhos hamiltonianos, etc)

Complexidade dos Problemas e Intratabilidade

Problemas Computacionais: **decisão**, busca/otimização e contagem/enumeração

- Alguns problemas tem algoritmos polinomiais, e.g. caminho mínimo, árvore geradora mínima, componentes conexas, programação linear, etc
- Existem problemas no meio, i.e. para eles existe um algoritmo exponencial porém (até hoje) nenhum algoritmo polinomial nem uma prova de que ele não existe

Tratáveis	Intratáveis?
Primo?	Fatoração inteira
Caminho simples mínimo	Caminho simples máximo
Caminho Euleriano	Caminho hamiltoniano
Programação linear	SAT
2-CNF SAT	3-CNF SAT

Sumário

Complexidade de Tempo - MT determinística

- O **tempo de execução ou complexidade de tempo** duma MT **determinística** M (que sempre pára) é uma função $f : \mathbf{N} \to \mathbf{N}$ onde $f(n)$ é o número máximo de passos que M usa sobre qualquer entrada de tamanho n.
- As notações assintóticas O e o são usadas para omitir detalhes
- Dada uma função $f : \mathbf{N} \to \mathbf{R}^+$, a classe de complexidade de tempo $\mathbf{TIME}(t(n))$ é o conjunto de todas as linguagens decidíveis por alguma MT com tempo de execução $O(t(n))$

Exemplo: Qual a complexidade de tempo da linguagem $\{ 0^n 1^n \mid n \geq 0 \}$?

Teorema: Para toda MT multifita de tempo $t(n) > n$ existe uma MT equivalente de uma fita e de tempo $O(t^2(n))$

Complexidade de Tempo - MT não determinística

- O **tempo de execução ou complexidade de tempo** duma MT **não determinística** M (que sempre pára) é uma função $f : \mathbf{N} \rightarrow \mathbf{N}$ onde $f(n)$ é o número de passos que M usa sobre qualquer ramo de sua computação para qualquer entrada de tamanho n.
- Dada uma função $f : \mathbf{N} \rightarrow \mathbf{R}^+$, a classe de complexidade de tempo $\mathbf{NTIME}(t(n))$ é o conjunto de todas as linguagens decidíveis por alguma MT não determinística com tempo de execução $O(t(n))$

Exemplo: Qual a complexidade de tempo da linguagem $\{x | s \in (0+1)^*, s = wxyxz, |x| = 4\}$, i.e. todas as cadeias de zeros e uns tais que tem uma subcadeia de tamanho 4 que se repete (não necessariamente de forma consecutiva)?

Complexidade de Tempo - MT não determinística

- O **tempo de execução ou complexidade de tempo** duma MT **não determinística** M (que sempre pára) é uma função $f : \mathbf{N} \to \mathbf{N}$ onde $f(n)$ é o número de passos que M usa sobre qualquer ramo de sua computação para qualquer entrada de tamanho n.
- Dada uma função $f : \mathbf{N} \to \mathbf{R}^+$, a classe de complexidade de tempo $\mathbf{NTIME}(t(n))$ é o conjunto de todas as linguagens decidíveis por alguma MT não determinística com tempo de execução $O(t(n))$

Teorema: Para toda MT não determinística de uma fita de tempo $t(n) > n$ existe uma MT determinística de uma fita equivalente e de tempo $2^{O(t(n))}$.

Classes de Linguagens: $\mathcal{P}$, $\mathcal{NP}$ e co-$\mathcal{NP}$

A classe $\mathcal{P}$ é o conjunto de linguagens que são decidíveis em tempo polinomial por uma MT determinística.

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

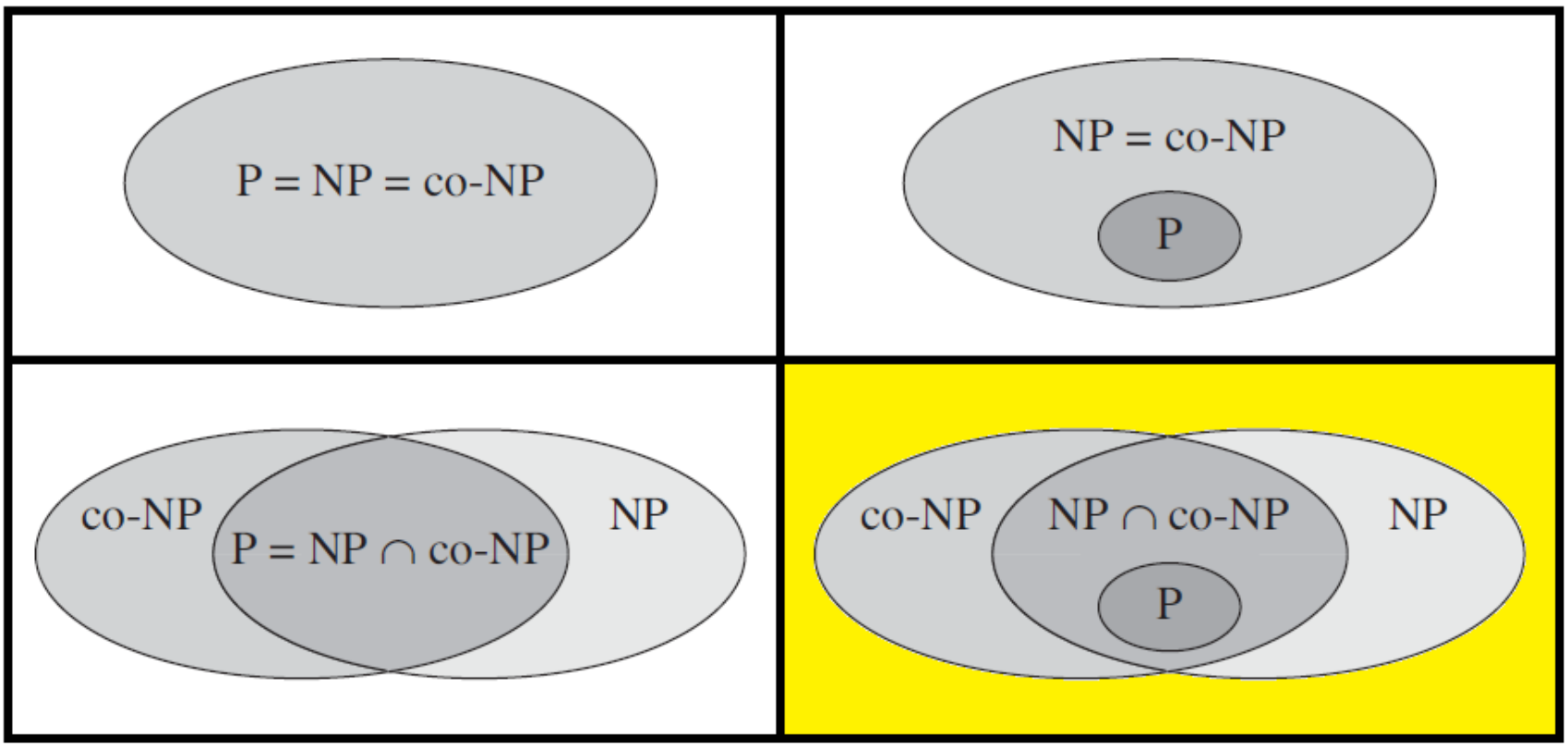
A classe $\mathcal{NP}$ é o conjunto de linguagens que são decidíveis em tempo polinomial por uma MT não-determinística.

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

A classe co-$\mathcal{NP}$ é o conjunto das linguagens L^c tal que $L \in NP$

$$\mathcal{P} \subseteq \mathcal{NP},\ \mathcal{P} \subseteq \text{co-}\mathcal{NP}$$

Relação entre $\mathcal{P}$, $\mathcal{NP}$ e co-$\mathcal{NP}$

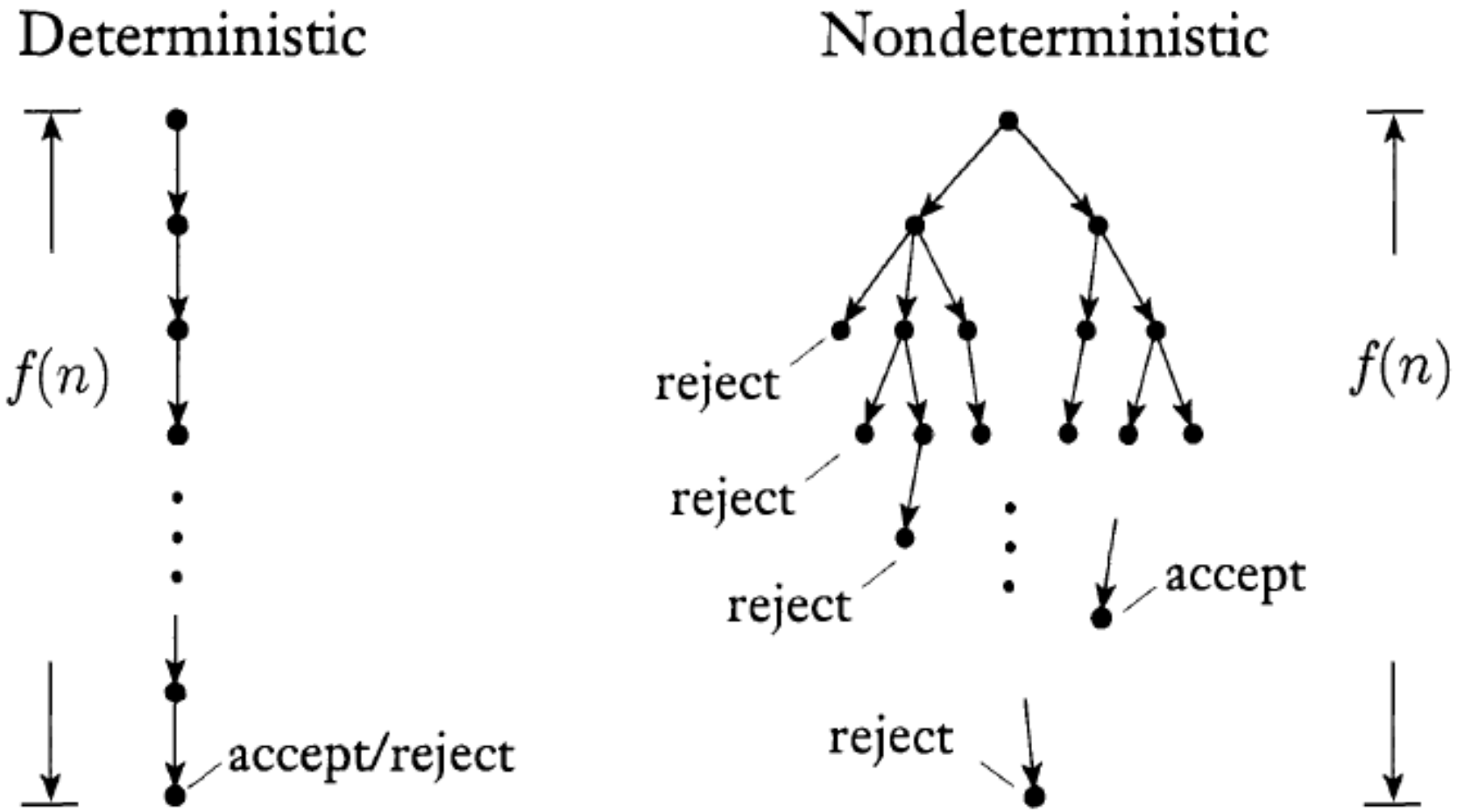


Por que a última relação é a mais provável?

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms

Universidade Federal do ABC

$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$: o não determinismo tem (ou não) mais poder?

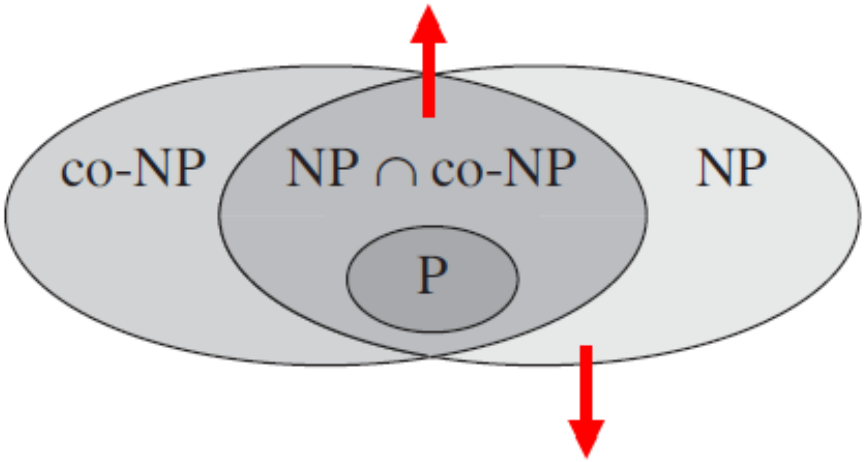


Adaptado de M. Sipser. Introduction to the Theory of Computation

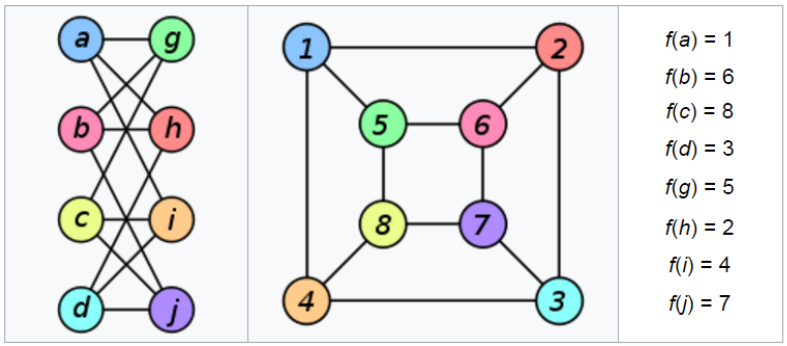
Universidade Federal do ABC

Classes de Linguagens: $\mathcal{P}$, $\mathcal{NP}$ e co-$\mathcal{NP}$

Fatoração de inteiros: $i \stackrel{?}{=} p * q$ com $p, q > 1$



Isomorfismo de grafos: $G_1 \equiv f(G_2)$?



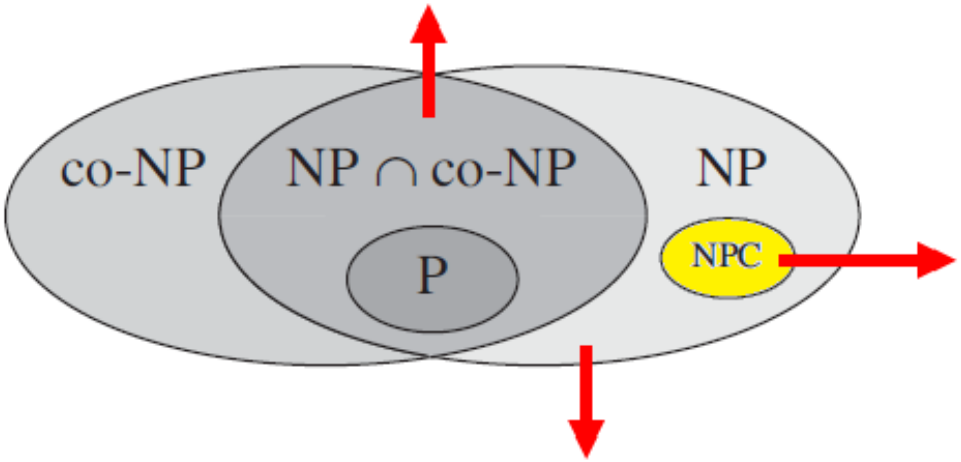
https://en.wikipedia.org/wiki/Graph_isomorphism

A classe NP também pode ser definida como o conjunto das linguagens que podem ser verificadas por um algoritmo de tempo polinomial.

Um algoritmo A verifica uma linguagem L se para qualquer cadeia x ∈ L, existe um certificado e que A pode usar para provar x ∈ L. Além disso, para qualquer cadeia x∉ L, não deve existir certificado comprovando que x ∈ L.

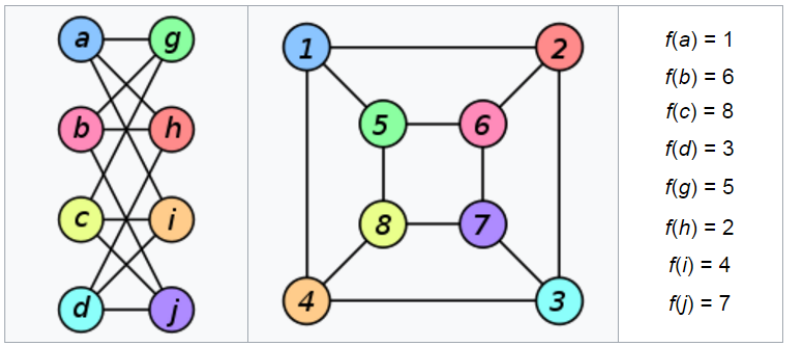
Classes de Linguagens: $\mathcal{P}$, $\mathcal{NP}$ e co-$\mathcal{NP}$

Fatoração de inteiros: $i \stackrel{?}{=} p * q$ com $p, q > 1$



Os problemas mais difíceis da classe NP são chamados de **problemas NP-completos**.

Isomorfismo de grafos: $G_1 \equiv f(G_2)$?



https://en.wikipedia.org/wiki/Graph_isomorphism

Universidade Federal do ABC

Sumário

$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ e $\mathcal{NP}$-completude

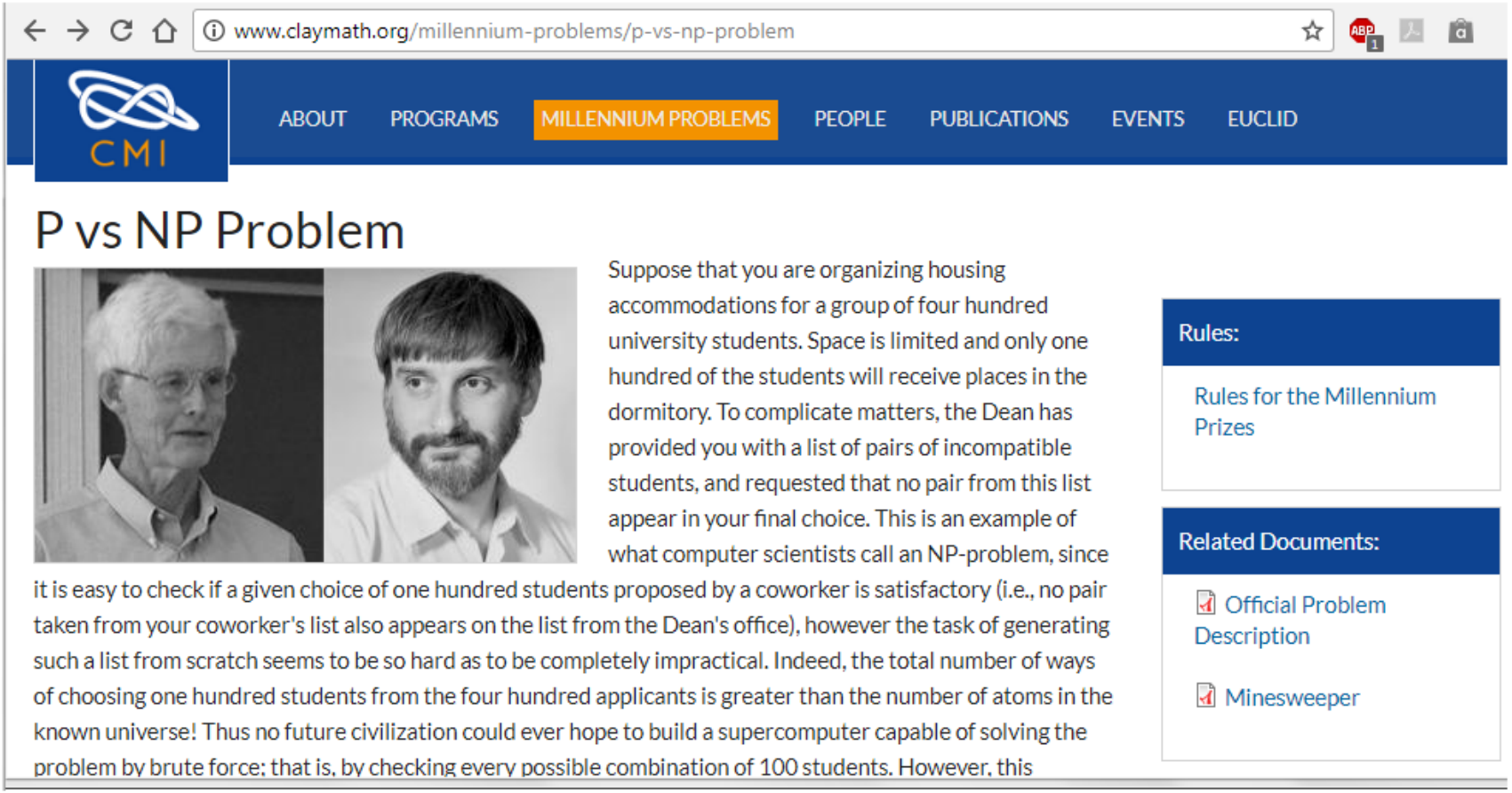
A teoria da NP-completude e a questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ surgiu no início dos anos 70 de resultados obtidos em paralelo por Stephen Cook e Leonid Levin. Posteriormente, enriquecida por Richard Karp.



Cook, Stephen. (1971). "The complexity of theorem proving procedures", Proc. 3rd Annual ACM STOC, 151-158
Levin, Leonid (1973). "Universal search problems". Problems of Information Transmission. 9 (3): 265-266
Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". Complexity of Computer Computations, 85-103
Wikipedia, Turing Award Winners, The Theory of Computing Hall of Fame, University of Waterloo

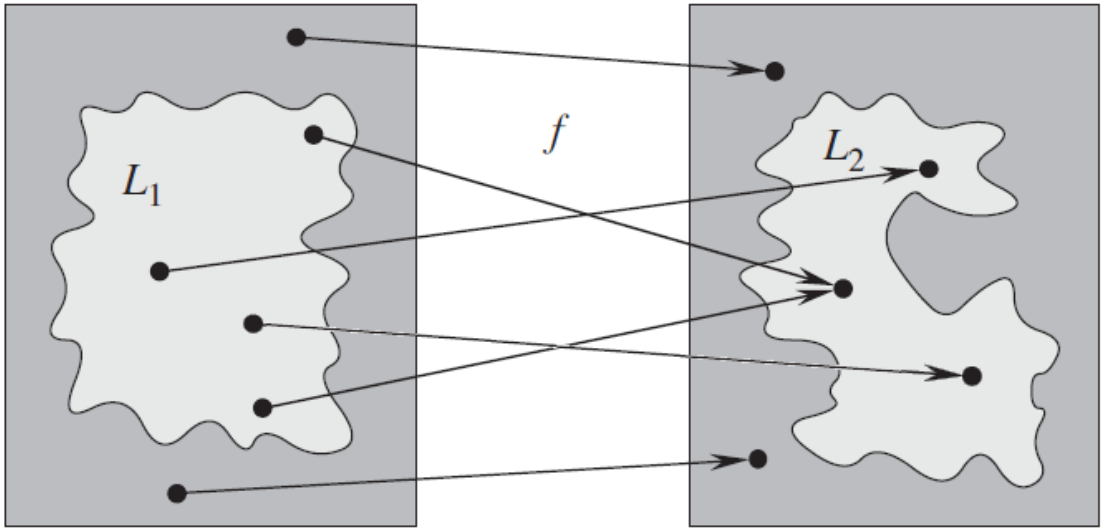
$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$, um dos sete "Millennium Prize Problems"

Teorema (de Cook-Levin): O problema da satisfazibilidade booleana (**SAT**) pertence a $\mathcal{P}$ se e somente se $\mathcal{P} = \mathcal{NP}$.



Redutibilidade em tempo polinomial

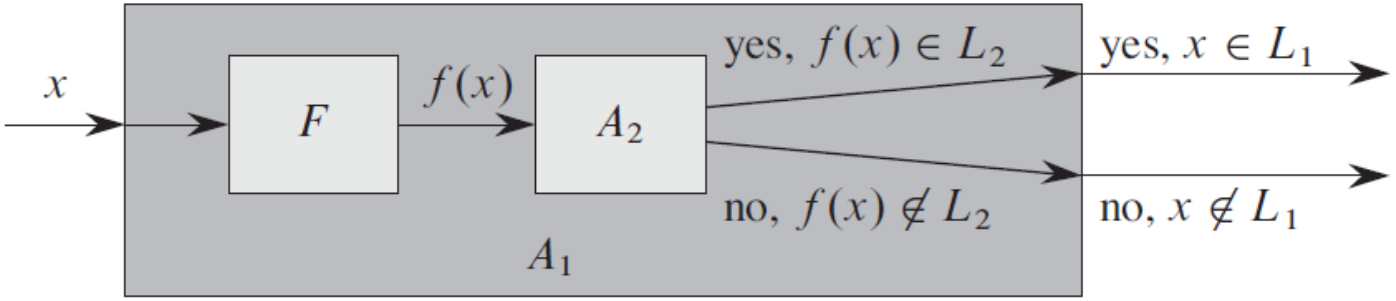
- Uma função $f : \Sigma^* \to \Sigma^*$ é **computável em tempo polinomial** se existe uma máquina de Turing de complexidade polinomial que pára com $f(w)$ na fita para qualquer $w \in \Sigma^*$.
- Uma linguagem L_1 é **redutível em tempo polinomial** a uma linguagem L_2 se existe uma função f computável em tempo polinomial tal que $w \in L_1 \Leftrightarrow f(w) \in L_2$. A redução de tempo polinomial é denotada por $L_1 \leq_P L_2$.



Redutibilidade em tempo polinomial

- Uma função $f : \Sigma^* \to \Sigma^*$ é **computável em tempo polinomial** se existe uma máquina de Turing de complexidade polinomial que pára com $f(w)$ na fita para qualquer $w \in \Sigma^*$.
- Uma linguagem L_1 é **redutível em tempo polinomial** a uma linguagem L_2 se existe uma função f computável em tempo polinomial tal que $w \in L_1 \Leftrightarrow f(w) \in L_2$. A redução de tempo polinomial é denotada por $L_1 \leq_P L_2$.

Teorema: Se $L_1 \leq_P L_2$ e $L_2 \in \mathcal{P}$ então $L_1 \in \mathcal{P}$.



$\mathcal{NP}$-completude e Teorema de Cook-Levin

- Um problema L_c é $\mathcal{NP}$-**difícil** se $\forall L \in \mathcal{NP}$, $L \leq_P L_c$ (L redutível em tempo polinomial a L_c).
- L_c é dito $\mathcal{NP}$-**completo** se é $\mathcal{NP}$-**difícil** e $L_c \in \mathcal{NP}$.

Teorema: Se L_c é $\mathcal{NP}$-completo e $L_c \in P$ então $\mathcal{P} = \mathcal{NP}$.

Teorema: **SAT**=$\{\phi \mid \phi \textit{ é uma fórmula booleana satisfazível}\}$ é $\mathcal{NP}$-completo

Ideia da Prova: **SAT**$\in \mathcal{NP}$ porque dada uma fórmula booleana e uma atribuição de valores a suas variáveis, pode-se verificar se a fórmula é satisfazível usando uma MT determinista de tempo polinomial. Para mostrar que SAT é $\mathcal{NP}$-difícil toda MT não determinística de tempo polinomial que resolve um problema em NP é reduzida a uma fórmula

Ver e.g. Garey&Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness.

Outros problemas $\mathcal{NP}$-completos

- **Forma normal conjuntiva** (CNF): Fórmula booleana resultante da conjunção de disjunções de literais (variáveis booleanas ou sua negação).

  Exemplo: $(x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4)$

- **3CNF**: Toda disjunção todas as disjunções têm três literais

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

3CNF-SAT$=\{ \ \phi \mid \phi \ é \ uma$ **3CNF-**$fórmula \ satisfazível \ \}$

Teorema: 3CNF-SAT é $\mathcal{NP}$-completo.

Ideia da Prova: Modificação da prova de **SAT** que já gera uma CNF (ver Sipser). A transformação de CNF a 3CNF é $O(n)$. Por exemplo, $(x_1 \vee x_1 \vee \neg x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee z) \ \wedge \ (\neg z \vee x_3 \vee \neg x_4)$.

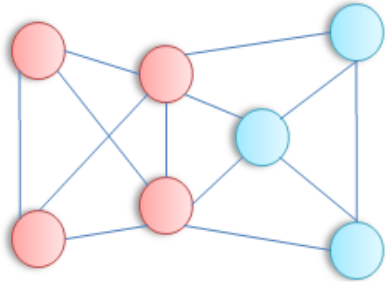
Outros problemas $\mathcal{NP}$-completos

É sempre assim? Não, Karp achou um método mais "simples"

Teorema: Se L_c é $\mathcal{NP}$-completo e $L_c \leq_P L$, para $L \in NP$, então L é $\mathcal{NP}$-completo.

Teorema: **SAT** $\leq_P$ **3CNF-SAT** (ver Cormen)

Exemplo: Dado um grafo não direcionado G, um k-clique é um subgrafo completo de G com k nós.

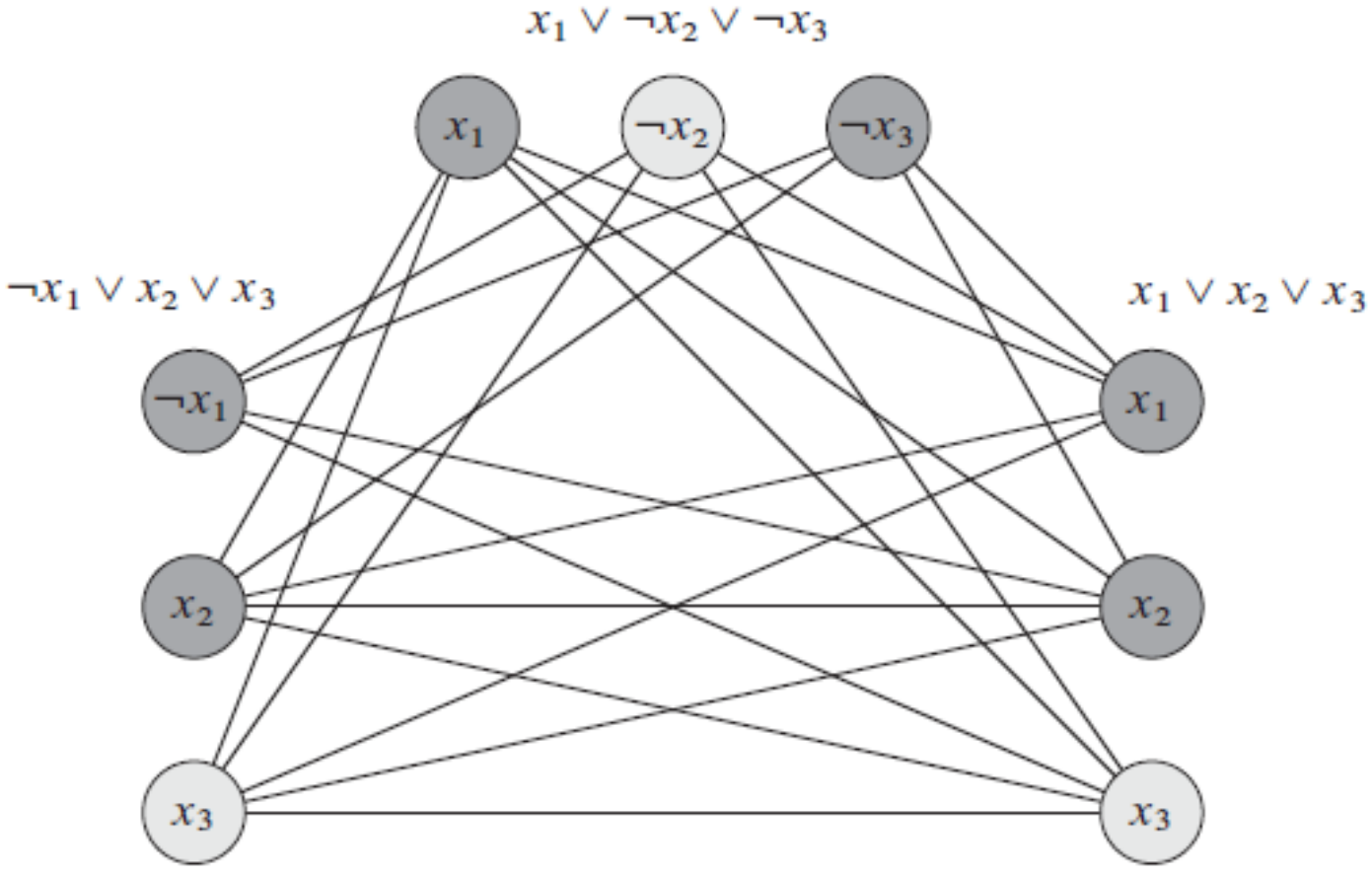


CLIQUE $= \{ (G, k) \mid G \; é \; um \; grafo \; com \; um \; k - clique \}$

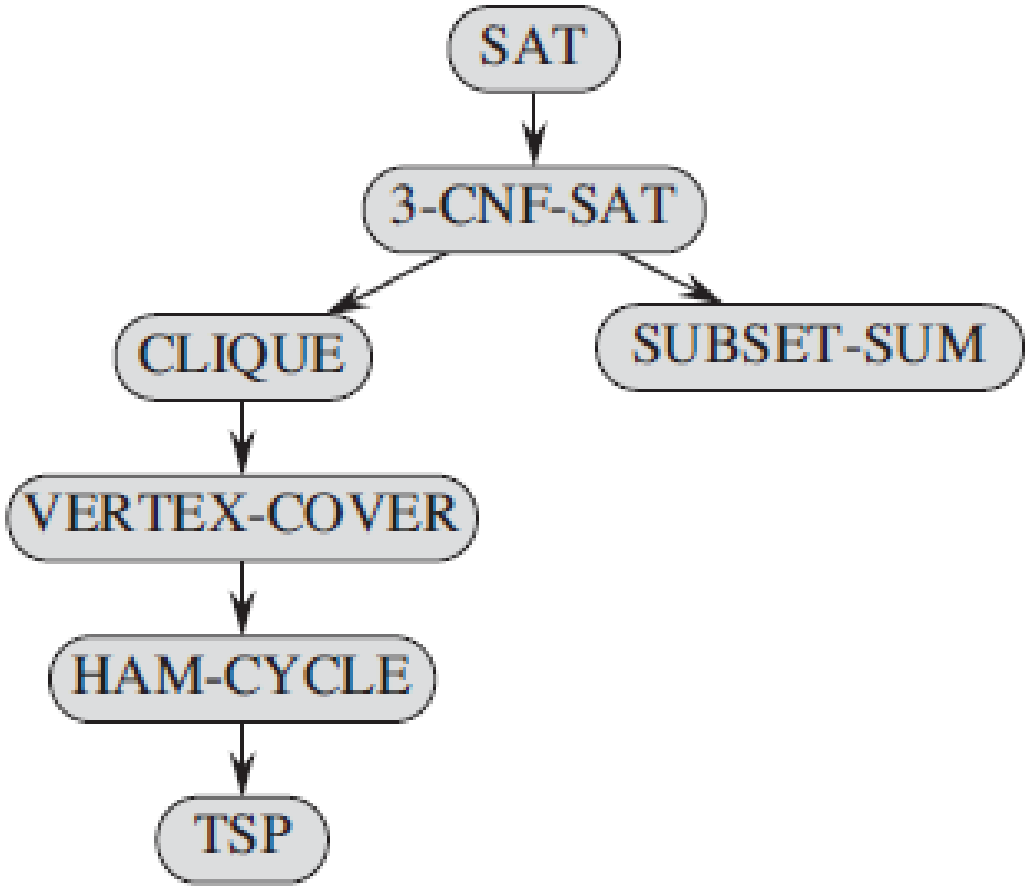
Teorema: **CLIQUE** é $\mathcal{NP}$-completo.

Outros problemas $\mathcal{NP}$-completos

Teorema: 3CNF-SAT $\leq_P$ CLIQUE.



Outros problemas $\mathcal{NP}$-completos



Ver mais e.g. http://adriann.github.io/npc/npc.html e

http://cgi.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated_np.html

Sumário

co-$\mathcal{NP}$ e $\mathcal{NP}$-completude

- Uma linguagem L_c é **completa** para uma classe de linguagens $\mathcal{C}$ se $L_c \in \mathcal{C}$ e $L \leq_P L_c$, $\forall L \in \mathcal{C}$.

Exemplo: **TAUT** é co-$\mathcal{NP}$-completo.

Teorema: $L \in \mathcal{NP}$ é $\mathcal{NP}$-completo se e somente se L^c é co-$\mathcal{NP}$-completo.

Teorema: $\mathcal{NP} =$ co-$\mathcal{NP}$ se e somente se existe um problema $\mathcal{NP}$-completo tal que seu complemento está em $\mathcal{NP}$.

Atenção: **SAT**^{c} inclui todas as fórmulas que não são satisfazíveis mas também todas as cadeias que não são fórmulas booleanas válidas.

Ver Hopcroft, Motwani & Ullman. Introduction to Automata Theory, Languages and Computation.

Complexidade de Espaço

- A **complexidade de espaço** duma MT **determinística** M (que sempre pára) é uma função $f : \mathbf{N} \to \mathbf{N}$ onde $f(n)$ é o número máximo de células que M lê para qualquer entrada de tamanho n.
- O **tempo de execução ou complexidade de tempo** duma MT **não determinística** M (que sempre pára) é uma função $f : \mathbf{N} \to \mathbf{N}$ onde $f(n)$ é o número de passos que M usa sobre qualquer ramo de sua computação para qualquer entrada de tamanho n.

Complexidade de Espaço

Dada uma função $f : \mathbf{N} \to \mathbf{R}^+$

- a classe $\mathbf{SPACE}(t(n))$ é o conjunto das linguagens decidíveis por alguma MT determinística com complexidade de espaço $O(t(n))$
- a classe $\mathbf{NSPACE}(t(n))$ é o conjunto das linguagens decidíveis por alguma MT não determinística com complexidade de espaço $O(t(n))$

Teorema: Para qualquer função $f : \mathbf{N} \to \mathbf{R}^+$, com $f(n) > n$, $PSPACE(f(n)) \subseteq NPSPACE(f^2(n))$.

$$\mathcal{PSPACE} = \bigcup_k SPACE(n^k) = \bigcup_k NSPACE(n^k) = \mathcal{NPSPACE}$$

$\mathcal{EXPTIME}$ e $\mathcal{EXPSPACE}$

A classe $\mathcal{EXPTIME}$ é o conjunto de linguagens que são decidíveis em tempo exponencial por uma MT determinística.

$$\mathcal{EXPTIME} = \bigcup_k TIME(2^{n^k})$$

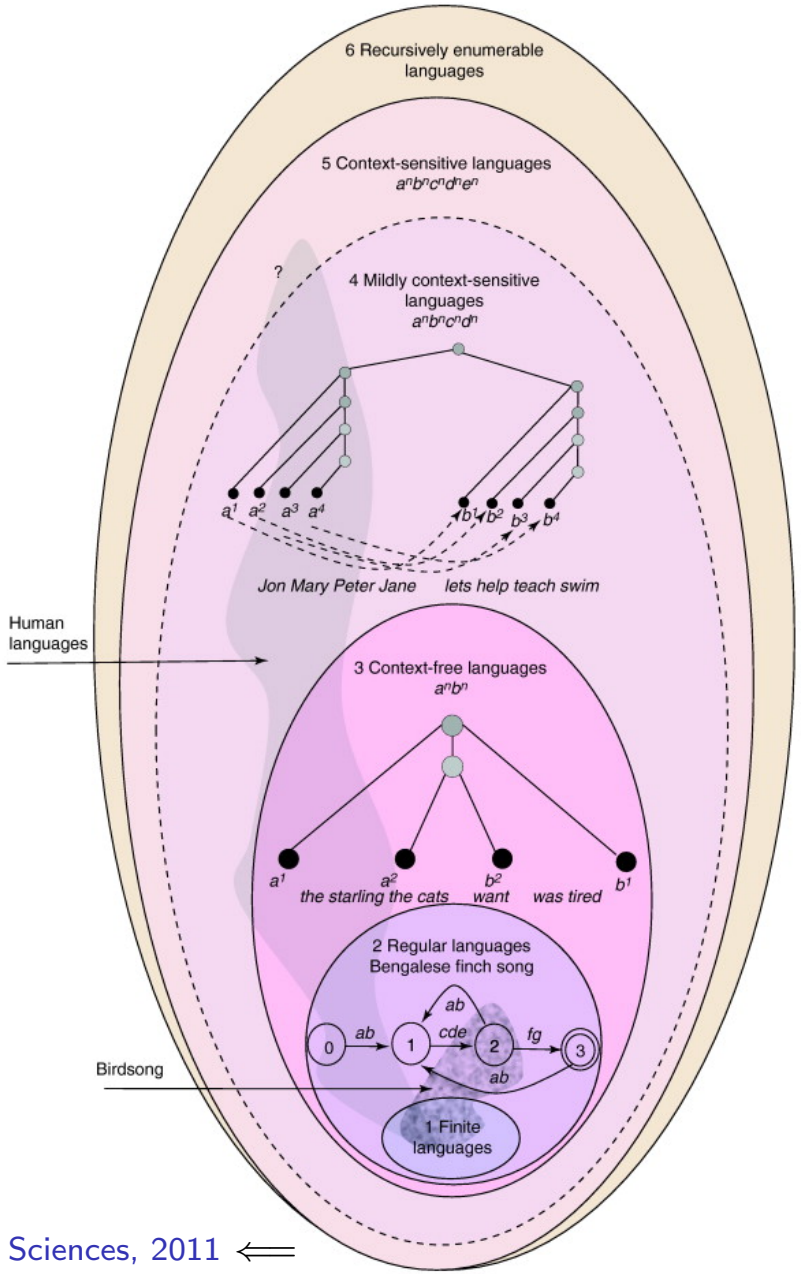
A classe $\mathcal{EXPSPACE}$ é o conjunto das linguagens decidíveis por alguma MT determinística com complexidade de espaço exponencial.

$$\mathcal{EXPSPACE} = \bigcup_k SPACE(2^{n^k})$$

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXPTIME} \subseteq \mathcal{EXPSPACE}$$

Hierarquia de Chomsky

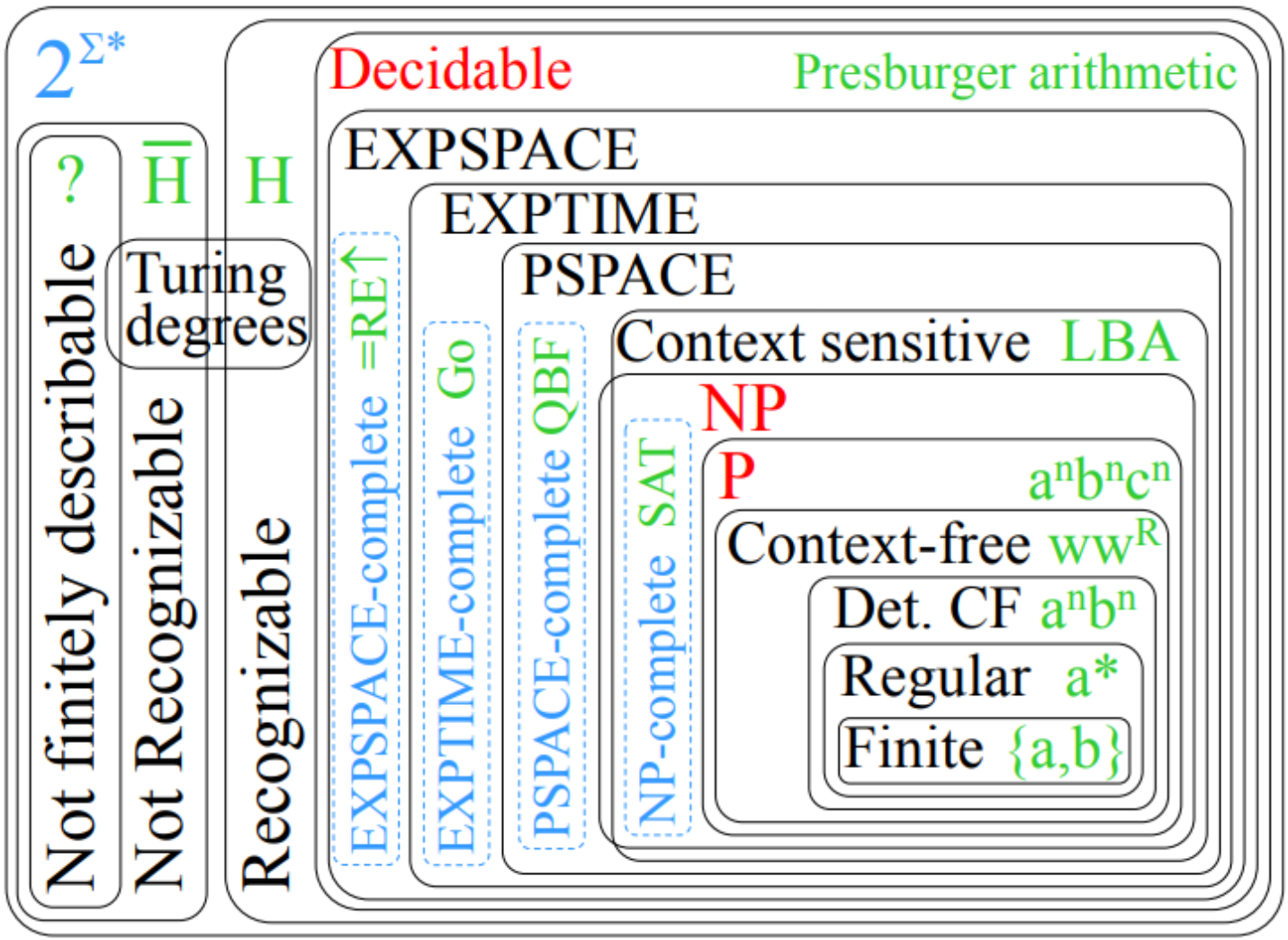
Type	Allowable Productions	Recognizing Automaton	Parsing Complexity
0	Unrestricted	Turing Machine	Undecidable
1	$\alpha \rightarrow \beta$ where $\lvert\alpha\rvert \leq \lvert\beta\rvert$ $\alpha \in V^*NV^*$ $\beta \in V^+$	Linear Bounded Automaton	NP Complete
2	$A \rightarrow \alpha$ $A \in N$ $\alpha \in V^*$	Pushdown Automaton	$O(n^3)$
3	$A \rightarrow xB$ $A \rightarrow x$ $A, B \in N$ $x \in T^*$	Finite Automaton	$O(n)$



http://www.cs.utexas.edu/users/novak/cs343283.html

Songs to syntax: the linguistics of birdsong, Trends in Cognitive Sciences, 2011 ⟸

Hierarquia de Chomsky estendida



http://www.cs.virginia.edu/~robins/cs3102/slides/Theory_Slides_Computational_Universality.pdf

Sumário

Como lidar com problemas computacionalmente difíceis?

"If a problem has no solution, it may not be a problem, but a fact — not to be solved, but to be coped with over time" - Shimon Peres

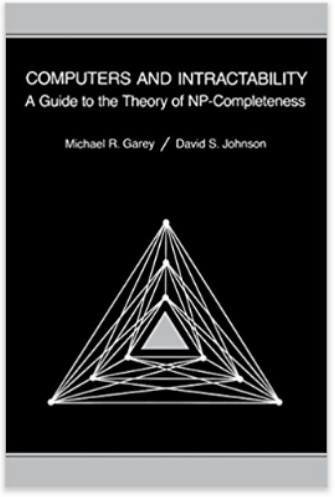
- Instâncias pequenas não são problema
- Casos especiais importantes podem ter algoritmos eficientes: **Horn-SAT**$\in \mathcal{P}$

  Buning, H.K.; Karpinski, Marek; Flogel, A. (1995). "Resolution for Quantified Boolean Formulas". Information and Computation. Elsevier. 117 (1), 12-18.
- Na prática, grandes instâncias podem resolvidas de forma eficiente: The International SAT Competitions web page
- Para problemas de otimização, achar uma solução que se aproxime à ótima (algoritmos de aproximação): Pex. Christofides algorithm for the TSP with the triangle inequality.

  Goodrich, Michael T.; Tamassia, Roberto (2015), "The Christofides Approximation Algorithm", Algorithm Design and Applications, Wiley, 513-514.

Sumário

Bibliografia Básica



1. **M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979**
2. M. Sipser. Introduction to the Theory of Computation, Cengage Learning 3rd ed, 2012
3. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein. Introduction to Algorithms, 3rd Edition, McGraw-Hill. 2009
4. J. E. Hopcroft, R. Motwani and J. D. Ullman. Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 2nd edition, 2001.
5. H. R. Lewis, C. H. Papadimitriou. Elementos de Teoria da Computação. 2a edição, Bookman Companhia Ed., 2004.