Linguagens Formais e Autômatos

Aula 29 - NP completude

Prof. Dr. Daniel Lucrédio
Departamento de Computação / UFSCar
Última revisão: ago/2015

Referências bibliográficas

- **Introdução à teoria dos autômatos, linguagens e computação / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman** ; tradução da 2.ed. original de Vandenberg D. de Souza. - Rio de Janeiro : Elsevier, 2002 (Tradução de: Introduction to automata theory, languages, and computation - ISBN 85-352-1072-5)
  - Capítulo 10 - Seção 10.2 a 10.4
- **Introdução à teoria da computação / Michael Sipser** ; tradução técnica Ruy José Guerra Barretto de Queiroz ; revisão técnica Newton José Vieira. -- São Paulo : Thomson Learning, 2007 (Título original : Introduction to the theory of computation. "Tradução da segunda edição norte-americana" - ISBN 978-85-221-0499-4)
  - Capítulo 7 - Seções 7.4 e 7.5

NP-completude

- Enquanto não se tem a prova de que P≠NP
- Foi observado um fenômeno interessante
  - Certos problemas em NP tem sua complexidade relacionada àquela da classe inteira
  - São problemas cuja complexidade representa a “essência” de NP-P
  - Todos problemas em NP compartilham dessa “essência”
  - Esses problemas são chamados de NP-completos

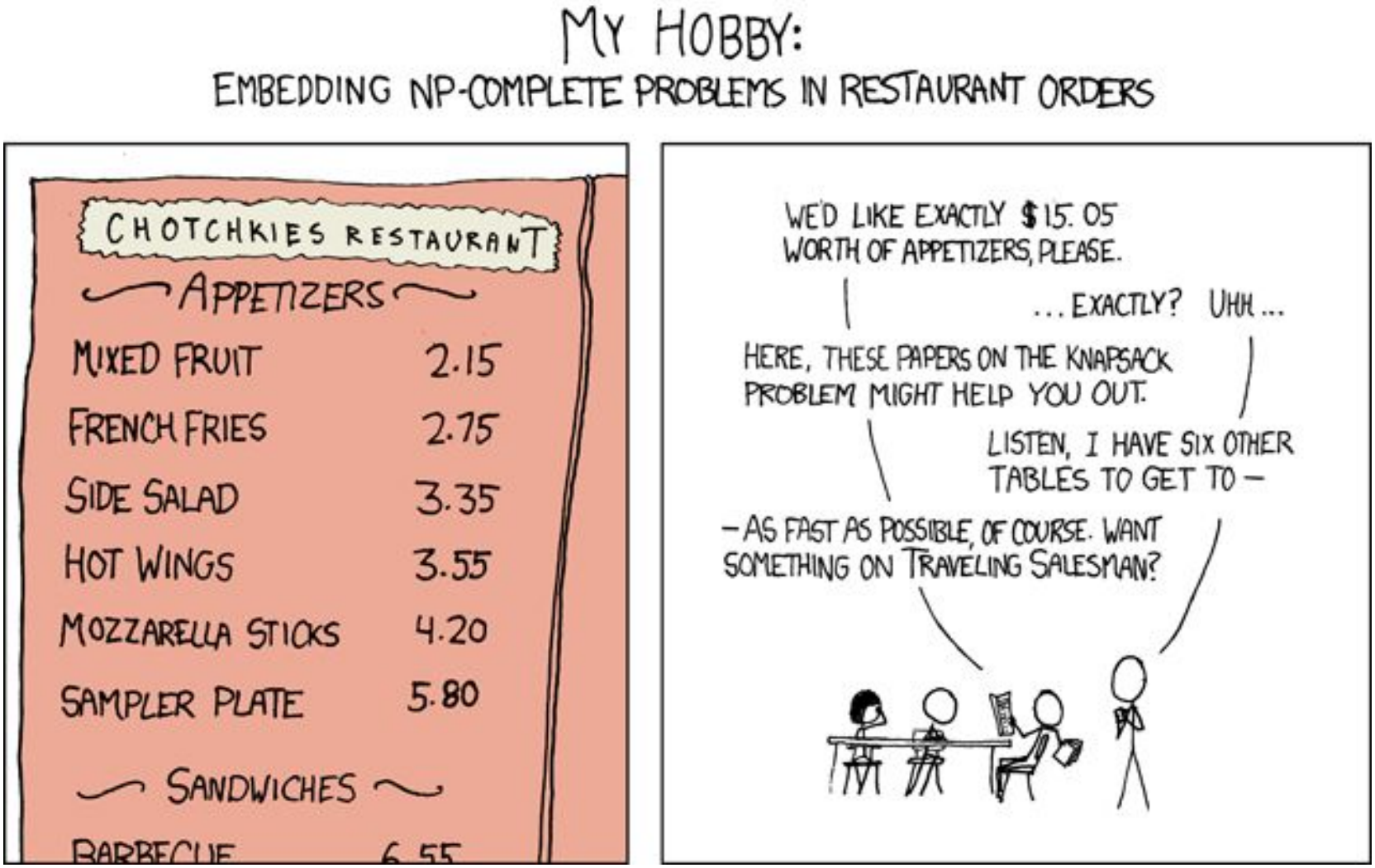
NP-completude

- A meta da observação desse fenômeno é o seguinte teorema:
  - Se algum problema NP-completo está em P, então P=NP
  - Ou seja, originalmente, era um caminho para se tentar provar que P=NP
- Mas existem também consequências práticas:
  - Se um problema é NP-completo, isso é uma forte evidência de que não existe solução polinomial!
    - Não é uma prova, devido à questão em aberto P vs NP
    - Mas para todos os efeitos práticos, assume-se que não vale a pena tentar encontrar um algoritmo de tempo polinomial
    - O melhor a fazer é buscar soluções alternativas

NP-completude na prática

- Segundo a wikipedia, existem mais de 3000 problemas reconhecidamente NP-completos
- Existe uma GRANDE chance de você se deparar com alguns deles durante sua vida profissional
- Portanto, o estudo da NP-completude não tem impacto somente na teoria
- Saber reconhecer um problema NP-completo, ou identificar um novo problema NP-completo é importante

NP-completude na prática



SAT = Problema da satisfazibilidade booleana

- Operações booleanas
  - Obs: VERDADEIRO = 1, FALSO = 0, E = &, OU = |, NÃO = !

0 & 0 = 0	0 | 0 = 0	!0 = 1
0 & 1 = 0	0 | 1 = 1	!1 = 0
1 & 0 = 0	1 | 0 = 1	
1 & 1 = 1	1 | 1 = 1	

- Fórmula booleana
  - Expressão envolvendo variáveis e operações booleanas. Exemplo:

$$\Phi = (!x \ \& \ y) \ | \ (x \ \& \ !z)$$

SAT

- Uma fórmula booleana é satisfazível se alguma atribuição de 0s e 1s às variáveis faz a fórmula ter valor 1
  - Ex: Φ = (!x & y) | (x & !z)
  - É satisfazível para x=0, y=1 e z=0

SAT={(Φ) | Φ é uma fórmula booleana satisfazível}

SAT

- SAT é NP-completo
- A prova virá a seguir
- Mas antes, veremos o conceito de
  - Redutibilidade em tempo polinomial

Redutibilidade em tempo polinomial

- Já vimos o conceito de redução
  - Se existe uma redução de A para B, então B é no mínimo tão difícil quanto A
- Mas anteriormente (no estudo da indecidibilidade), estávamos apenas interessados se um problema é decidível ou não
  - Agora, estamos interessados se um problema tem solução eficiente (polinomial) ou não
  - Portanto, o conceito de redução deve ser revisto

Redução em tempo polinomial

- Redução = algoritmo que converte instâncias de um problema A em instâncias de um problema B
- Exs:
  - Conversão de uma instância do PCPM para uma instância do PCP
  - Conversão de uma instância do Lu para uma instância do PCPM
- Uma redução em tempo polinomial é um algoritmo DE TEMPO POLINOMIAL que converte instâncias de um problema A em instâncias de um problema B
  - Ou, de forma equivalente, uma MT que executa em tempo polinomial e faz tal conversão

Redução em tempo polinomial

- Para provar complexidade de um problema, não basta usar qualquer redução
  - Pois uma redução em tempo exponencial de A para B não garante que B é tão difícil quanto A
  - Exemplo: A está em P, e queremos provar que B também está em P
    - Se A se reduz a B exponencialmente, significa que mesmo que eu encontrar uma solução polinomial para B, a mesma não implica em uma solução polinomial para A, já que a conversão é exponencial

Redução em tempo polinomial

- Portanto, agora além de serem válidas, as reduções devem executar em tempo polinomial
- Posso usar isso de duas formas:
  - A está em P, e quero provar que B está em P
    - Reduzo A para B em tempo polinomial
  - A está em NP, e quero provar que B está em NP
    - Reduzo A para B em tempo polinomial
  - A é NP-completo, e quero provar que B é NP-completo
    - Reduzo A para B em tempo polinomial

Definição formal de NP-completude

Uma linguagem B é NP-completa se satisfaz duas condições:

1. B está em NP, e
2. Toda A em NP é redutível em tempo polinomial a B

SAT

- Faremos agora a prova de que SAT é NP-completo
  - Usaremos a definição anterior
- SAT é a primeiro problema que provaremos ser NP-completo
  - Depois usaremos apenas reduções a partir de SAT para provar que outros problemas são também NP-completos
- Na sua vida profissional, é assim que você irá provar que um problema X é NP-completo
  - Comece a partir de um problema NP-completo conhecido (como SAT, CLIQUE, entre outros)
  - E encontre uma redução em tempo polinomial de um problema conhecido para X

SAT

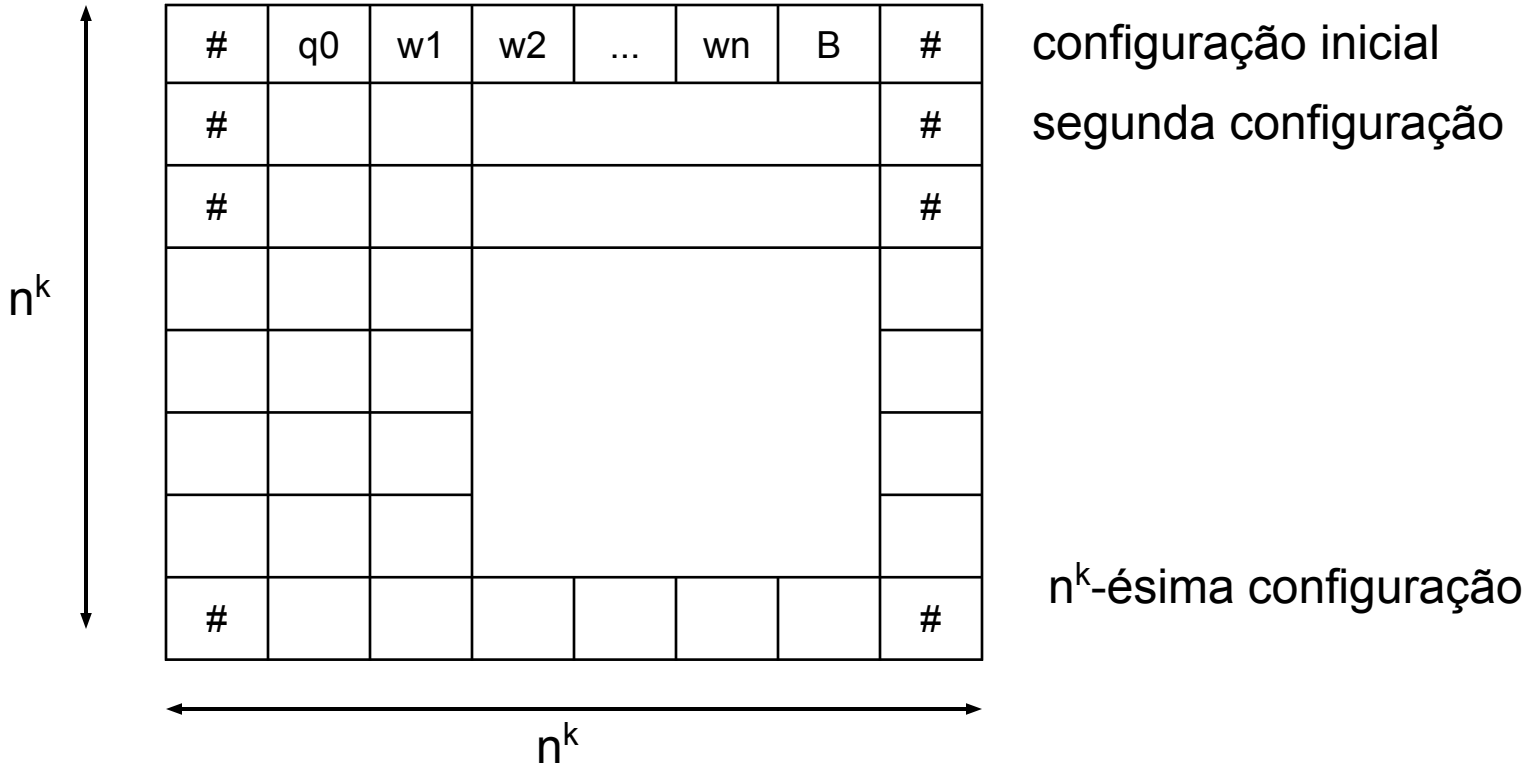
- Prova de que SAT é NP-completo
- 1: provar que SAT é NP
  - Basta encontrar um verificador
    - Certificado c é a atribuição correta de valores às variáveis, tal que o resultado é 1
    - Algoritmo verificador calcula o resultado das operações. Se for 1, aceita. Caso contrário, rejeita. Esse algoritmo é certamente polinomial.
  - Similarmente, bastaria projetar uma NTM de tempo polinomial

SAT

- Prova de que SAT é NP-completo
- 2: provar que qualquer linguagem em NP é redutível em tempo polinomial a SAT
  - Seja A uma linguagem qualquer em NP
  - Com certeza existe uma NTM N que decide A em tempo n^k para alguma constante k (não entendeu? Volte e reveja os conceitos dessa aula)
  - Sendo não-determinística, N pode levar a vários ramos de execução para uma entrada w
    - Alguns ramos são de aceitação
    - Determinar se N aceita w se resume a encontrar um ramo de aceitação

SAT

- Um ramo de execução pode ser visto como uma tabela – chamada tableau



SAT

- Um tableau é de aceitação se qualquer linha dele for uma configuração de aceitação
- Determinar se N aceita w é equivalente ao problema de se determinar se existe um tableau de aceitação para N sobre w
- Vamos reduzir o problema A para SAT
- Ou seja, decidir o SAT gerado implica que A também é decidido
  - Ou seja, decidir o SAT gerado implica que encontramos um tableau de aceitação para N sobre w

SAT

- O objetivo é montar uma fórmula booleana, certo?
- Então montaremos uma fórmula booleana que, se satisfeita, representa exatamente um tableau de aceitação
- $\Phi = \Phi_{início} \,\&\, \Phi_{movimento} \,\&\, \Phi_{aceita}$
- Onde:
  - $\Phi_{início}$ é uma fórmula booleana que diz quando N inicia corretamente
  - $\Phi_{movimento}$ é uma fórmula booleana que diz quando N faz movimentos corretos de uma CI para outra
  - Φ_{aceita} é uma fórmula booleana que diz quando N termina corretamente

SAT

- Antes, vamos definir as variáveis que serão geradas
  - Afinal, as fórmulas contém variáveis
- Uma variável consiste em $X_{ij"a"}$
  - Ela “pergunta” se, na linha i, coluna j, o símbolo é “a”.
  - $X_{ij"a"} = 1$, se, na linha i, coluna j, existe o símbolo “a”
  - $X_{ij"a"} = 0$, se, na linha i, coluna j, não existe o símbolo “a”
- Exemplo, para uma MT que opera sobre o alfabeto {a,b}, com um tableau mostrado abaixo:

#	q0	a	b	b	B	B	#
#							#
#							#

$X_{11"\#"} = 1$
$X_{11"q0"} = 0$
$X_{12"q0"} = 1$
$X_{12"q2"} = 0$
$X_{14"b"} = 1$
$X_{14"a"} = 0$

SAT

- $\Phi_{início}$
  - Essa fórmula booleana, se satisfeita, diz se N inicia corretamente, ou seja:
    - [q0]w1w2w3...wnBB...BB
    - (primeira configuração instantânea de N)
    - Ou, na notação de tableau:
    - # q0 w1 w2 ... wn B B ... B #
- Portanto montamos $\Phi_{início}$ da seguinte forma:

$$\Phi_{início} = x_{11"\#"} \,\&\, x_{12"qo"} \,\&\, x_{13"w1"} \,\&\, x_{14"w2"} \,\&\, \ldots \,\&\, x_{1n"wn"} \,\&\, x_{1n+1"B"} \,\&\, \ldots \,\&\, x_{1n^k"\#"}$$

- Uma solução para $\Phi_{início}$ corresponde ao início correto de N

SAT

- Φ_{aceita}
  - A fórmula booleana que “pergunta” se N aceita é simplesmente uma varredura por todas as células em busca de um estado de aceitação

$$\Phi_{aceita} = |_{1 \leq i,j \leq n} k\ X_{ij”qf”}$$

  - Ou seja, é um OU (|) entre variáveis que representam uma busca, para todo i, j da tabela, por todo estado de aceitação qf
- A solução para Φ_{aceita}, ou seja, valores para as variáveis de tal forma que a fórmula resulte em 1, implica que o tableau correspondente possui um estado de aceitação (um símbolo “qf”) em alguma célula

SAT

- $\Phi_{movimento}$
- Essa última fórmula garante que cada linha da tabela corresponde a uma configuração que representa um movimento correto a partir da linha anterior
  - Conforme a função de transição de N
  - Ex: suponha que uma das regras de transição seja:
  - $\delta(q3,a) = (q2,b,E)$

#	...						#	
#	a	b	q3	a	b	B	#	i-1
#	a	q2	b	b	b	B	#	i
#	a	b	b	b	q1	B	#	i+1
#	...						#	

i é uma configuração válida, pois o movimento de i-1 para i está previsto em δ

i+1 é uma configuração inválida, pois não existe movimento possível que leva de i para i+1

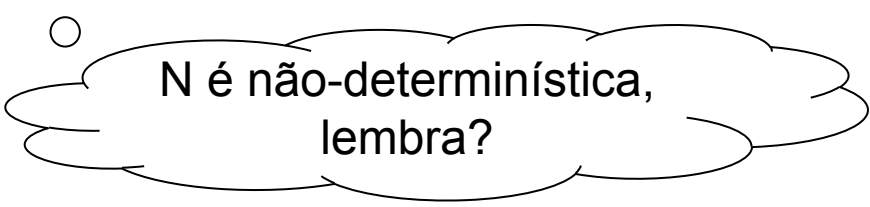
SAT

- A redução sendo construída faz essa checagem em janelas de 2 x 3 células
  - Uma janela 2 x 3 é legal se essa janela não viola as ações especificadas pela função de transição de N

#	...					#
#	...	X_{ij-1}	X_{ij}	X_{ij+1}	...	#
#		X_{i+1j-1}	X_{i+1j}	X_{i+1j+1}		#
#	...					#

SAT

- Exemplo. Suponha que N aceita entradas sobre o alfabeto {a,b,c}, e tenha estados {q1,q2}, e δ seja:
  - δ(q1,a) = {(q1,b,D)}
  - δ(q1,b) = {(q2,c,E),(q2,a,D)}
- Exemplos de janelas legais:



a	q1	B
q2	a	c

a	a	q1
a	a	b

a	b	a
a	b	q2

a	q1	B
a	a	q2

#	b	a
#	b	a

- Exemplos de janelas ilegais:

a	b	a
a	a	a

a	q1	a
q1	a	a

b	q1	b
q2	b	q2

SAT

- Se a linha superior for a configuração inicial e
- Toda janela na tabela for legal
  - Cada linha da tabela é uma configuração que representa um movimento correto de N
- Voltando à definição de $\Phi_{movimento}$

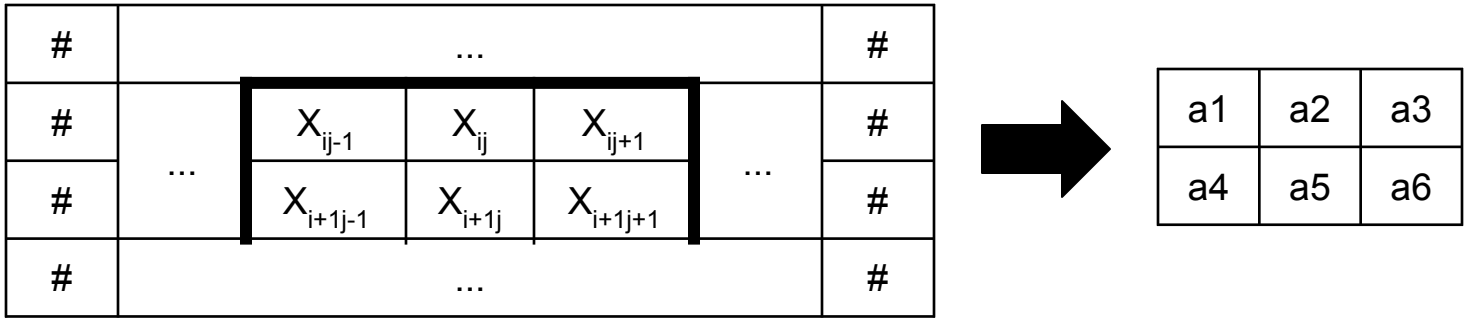
$$\Phi_{movimento} = \&_{1<i\leq n^k, 1<j\leq n^k} (\text{a janela } (i,j) \text{ é legal})$$

SAT

- A fórmula para (a janela (i,j) é legal) é dada a seguir:

$$\bigvee_{a1,a2,...,a6} (X_{ij-1"a1"} \,\&\, X_{ij"a2"} \,\&\, X_{ij+1"a3"} \,\&\, X_{i+1j-1"a4"} \,\&\, X_{i+1j"a5"} \,\&\, X_{i+1j+1"a6"})$$

- Onde as variáveis representam uma janela 3x2, e a1,a2,...,a6 são valores que a tornam legal



SAT

- Resumindo, dada uma Máquina de Turing não-determinística que executa em tempo polinomial N = (Q,Σ,Γ,δ,q0,B,F) e uma entrada w
  - Mostramos uma redução de N para uma instância do SAT $\Phi = \Phi_{início} \,\&\, \Phi_{movimento} \,\&\, \Phi_{aceita}$
  - De tal forma que se encontrarmos uma solução para:
  - $\Phi_{início} \rightarrow$ significa que encontramos uma configuração inicial para N sobre w
  - $\Phi_{movimento} \rightarrow$ significa que encontramos movimentos válidos de N sobre w
  - $\Phi_{aceita} \rightarrow$ significa que descobrimos que N aceita w

SAT

- Ou seja, com essa redução, encontrar uma solução para Φ implica em encontrar uma execução que leva à aceitação de w por N
  - Importante: para qualquer N e qualquer w!
- Importante 2: N representa qualquer linguagem (problema) NP
  - Ou seja, acabamos de reduzir todo problema NP para SAT, que é (parte da) segunda condição para NP-completude:

Uma linguagem B é NP-completa se satisfaz duas condições:

1. B está em NP, e
2. Toda A em NP é redutível em tempo polinomial a B

SAT

- Ainda falta uma análise para completarmos a prova de que SAT é NP-completo
- O algoritmo de redução que acabamos de apresentar precisa executar EM TEMPO POLINOMIAL
- Passemos à análise
  - Considerando que o tableau tem tamanho n^k x n^k
- Porque?
  - Resposta: porque o tableau representa um ramo de execução de uma NTM de tempo polinomial
  - n^k é um polinômio
  - Nunca vão ser necessários mais do que n^k passos para uma execução
    - Nem mais do que n^k células da fita de N

SAT

- Intuitivamente, basta observar que a fórmula obtida tem uma natureza altamente repetitiva
  - $\Phi_{início} \rightarrow$ um fragmento para cada célula da primeira linha
    - Tamanho = $O(n^k)$
  - $\Phi_{movimento} \rightarrow$ cada fragmento possui um tamanho fixo para cada célula do tableau
    - Tamanho = $O(n^{2k})$
  - $\Phi_{aceita} \rightarrow$ cada fragmento possui um tamanho fixo para cada célula do tableau
    - Tamanho = $O(n^{2k})$
- Como consequência, o tamanho total de Φ é $O(n^{2k})$
  - Que é um polinômio

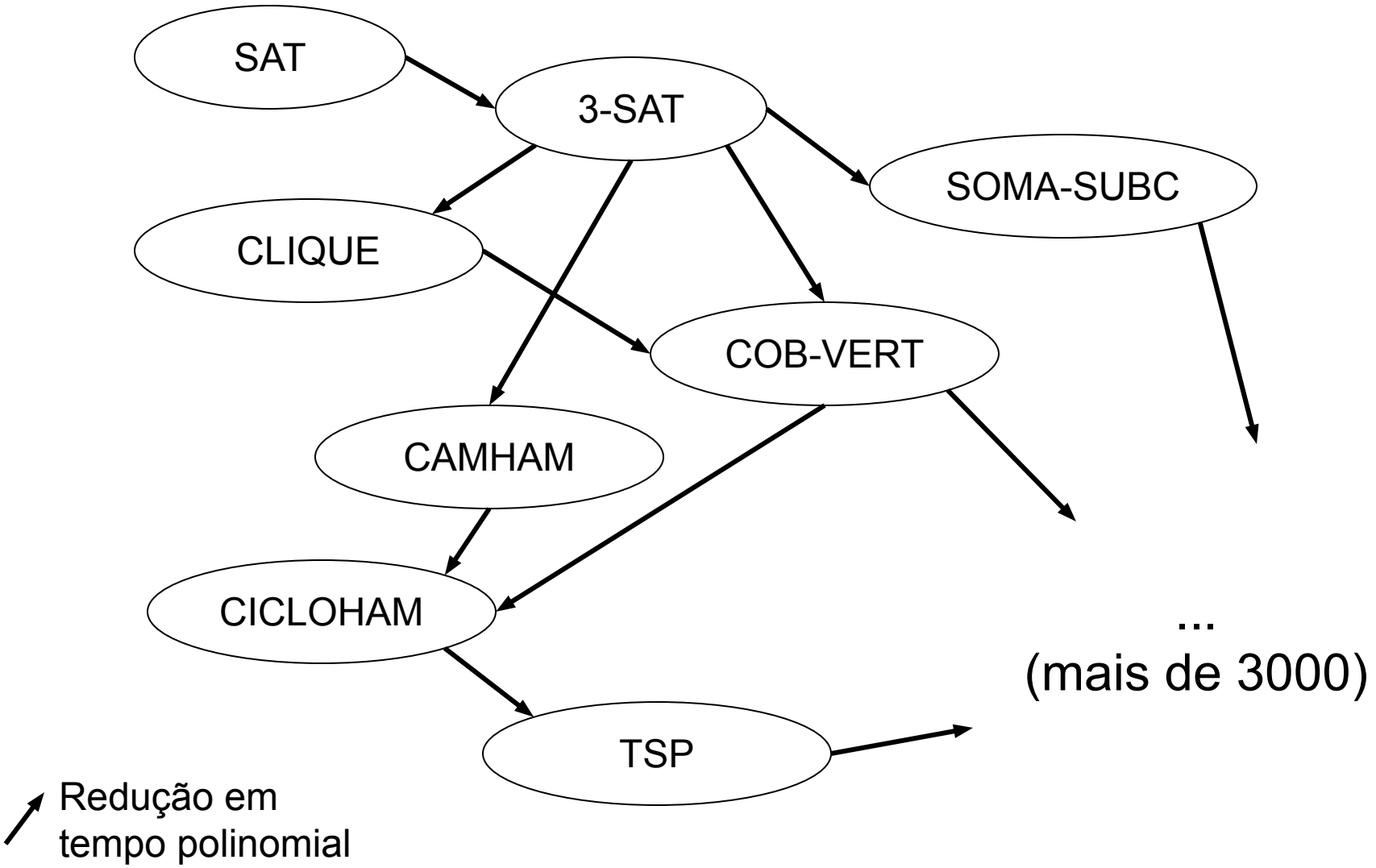
Teorema de Cook-Levin

- A demonstração anterior mostrou que SAT é NP-completo
  - É chamada de teorema de Cook-Levin
- SAT é a “semente” para a prova de NP-completude de vários outros problemas
  - Ou seja, não é encessário fazer uma prova tão longa
  - Basta escolher um problema NP-completo “conhecido” e encontrar uma redução em tempo polinomial

Outros problemas NP-Completos

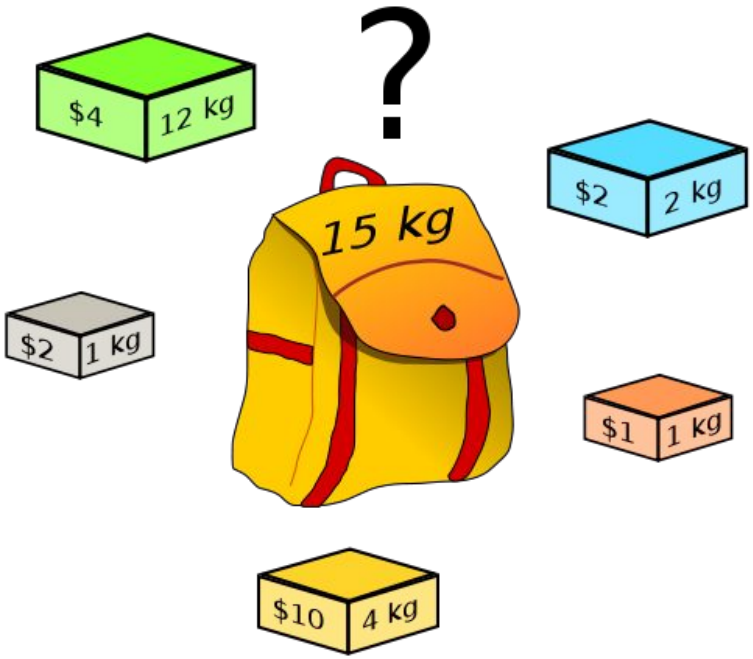
- 3SAT: uma versão “modificada”, onde as fórmulas booleanas estão na forma normal conjuntiva de três literais:
  - (x1 | x2 | !x3) & (x3 | !x5 | x4) & (x4 | x5 | x6)
- COB-VERT: uma cobertura de vértices de um grafo direcionado é um subconjunto dos nós onde toda aresta de G toca um dos nós.
  - COB-VERT = {(G,k) | G é um grafo não-direcionado que tem uma cobertura de vértices de k nós}
- CAMHAM
- CICLOHAM
- TSP
- CLIQUE
- SOMA-SUBC

Problemas NP-completos



Problemas NP-completos

- Nem toda redução é complicada



Problema da mochila
Tenho $$ e Kg limitados...
Quais itens escolher?

Problema do cardápio. Você só tem 10 R$. Qual o máximo de comida você consegue pedir?

Redução é trivial

Cardápio	
Batata frita	R$ 5,50
Mandioca frita	R$ 6,00
Iscas de peixe	R$ 9,50
Mesa de frios	R$ 7,00
Queijos diversos	R$ 12,00
Torradas e patês	R$ 4,20
Amendoins	R$ 1,50

Problemas NP-difíceis

- São problemas tão difíceis como qualquer problema NP
- Problemas NP-difíceis são aqueles para os quais:
  - É possível obter uma redução em tempo polinomial a partir de um problema NP-completo (condição 2 da NP-completude)
  - Mas não é possível provar que existe uma solução não-determinística polinomial, ou que existe um verificador (condição 1 da NP-completude)
- Ou seja, em teoria, um problema NP-difícil pode ser ainda mais difícil que todos os problemas NP-completos
  - Na prática, problemas NP-difíceis são igualmente intratáveis, e portanto concluir que um problema é NP-difícil resulta na impossibilidade de resolvê-lo

Complexidade de espaço

- Classes PSPACE e NPSPACE
  - Linguagens decididas por DTMs e NTMS, respectivamente
  - Utilizando quantidade de espaço (fita) polinomial
- Outra classificação de problemas
  - Diferente de P e NP, pois espaço é reutilizável
  - Por exemplo, SAT pode ser resolvido em espaço linear
- Existe o mesmo conceito de problemas PSPACE-completos e PSPACE-difíceis

Como resolver problemas intratáveis?

- RP – Polinomial aleatório
  - Máquinas de Turing baseadas em aleatoriedade
  - Para alcançar tempos de execução médios polinomiais (ainda que existam casos exponenciais, mas esses são pouco frequentes)
  - Algoritmo Monte Carlo
  - Garante algumas respostas, mas pode encerrar sem chegar a uma conclusão
- ZPP – Polinomial probabilística de erro zero
  - MT aleatória que sempre dá a resposta certa, mas pode levar um tempo exponencial
  - Algoritmo Las Vegas
  - Quicksort

Como resolver problemas intratáveis?

- Aproximações
- Heurísticas
- Estatísticas
- Soluções localizadas

Fim

Aula 29 - NP completude