3-10 NP完全理论初步

程龚

你能区分这两种“难”吗？

with hard problems only. We consider a problem to be hard if there is no known deterministic algorithm (computer program) that solves it efficiently. Efficiently means in a low-degree polynomial time. Our interpretation of hardness here is connected to the current state of our knowledge in algorithmics rather than to the unknown, real difficulty of the problems considered. Thus, a problem is hard if one would need years or thousands of years to solve it by deterministic programs for an input of a realistic size appearing in the current practice. This book provides a handbook of algorithmic methods that

constant k. Generally, we think of problems that are solvable by polynomial-time algorithms as being tractable, or easy, and problems that require superpolynomial time as being intractable, or hard.

你理解这段话了吗？

Although NP-complete problems are confined to the realm of decision problems, we can take advantage of a convenient relationship between optimization problems and decision problems. We usually can cast a given optimization problem as a related decision problem by imposing a bound on the value to be optimized. For

- 优化问题和对应的判定问题哪个更难？

优化问题有自己的“NP”和“P”

Definition 2.3.3.21. NPO *is the class of optimization problems, where* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \in \mathrm{NPO}$ *if the following conditions hold:*

(i) $L_I \in \mathrm{P}$,
(ii) there exists a polynomial p_U *such that*
  a) for every $x \in L_I$, *and every* $y \in \mathcal{M}(x)$, $|y| \leq p_U(|x|)$, *and*
  b) there exists a polynomial-time algorithm that, for every $y \in \Sigma_O^*$ *and every* $x \in L_I$ *such that* $|y| \leq p_U(|x|)$, *decides whether* $y \in \mathcal{M}(x)$, *and*
(iii) the function cost is computable in polynomial time.

Informally, we see that an optimization problem U is in NPO if

(i) one can efficiently verify whether a string is an instance of U,
(ii) the size of the solutions is polynomial in the size of the problem instances and one can verify in polynomial time whether a string y is a solution to any given input instance x, and
(iii) the cost of any solution can be efficiently determined.

Definition 2.3.3.23. PO *is the class of optimization problems* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *such that*

(i) $U \in \mathrm{NPO}$, *and*
(ii) there is a polynomial-time algorithm that, for every $x \in L_I$, *computes an optimal solution for* x.

我们为什么用多项式作为分界线？

- n^{100}也很大，不是吗？
- 多项式有什么优势？

我们为什么用多项式作为分界线？

First, although we may reasonably regard a problem that requires time $\Theta(n^{100})$ to be intractable, very few practical problems require time on the order of such a high-degree polynomial. The polynomial-time computable problems encountered in practice typically require much less time. Experience has shown that once the first polynomial-time algorithm for a problem has been discovered, more efficient algorithms often follow. Even if the current best algorithm for a problem has a running time of $\Theta(n^{100})$, an algorithm with a much better running time will likely soon be discovered.

Second, for many reasonable models of computation, a problem that can be solved in polynomial time in one model can be solved in polynomial time in another. For example, the class of problems solvable in polynomial time by the serial random-access machine used throughout most of this book is the same as the class of problems solvable in polynomial time on abstract Turing machines.[1] It is also the same as the class of problems solvable in polynomial time on a parallel computer when the number of processors grows polynomially with the input size.

Third, the class of polynomial-time solvable problems has nice closure properties, since polynomials are closed under addition, multiplication, and composition. For example, if the output of one polynomial-time algorithm is fed into the input of another, the composite algorithm is polynomial. Exercise 34.1-5 asks you to show that if an algorithm makes a constant number of calls to polynomial-time subroutines and performs an additional amount of work that also takes polynomial time, then the running time of the composite algorithm is polynomial.

这个定理有什么意义？

Lemma 34.1

Let Q be an abstract decision problem on an instance set I, and let e_1 and e_2 be polynomially related encodings on I. Then, $e_1(Q) \in \mathrm{P}$ if and only if $e_2(Q) \in \mathrm{P}$.

这个定理有什么意义？

Lemma 34.1

Let Q be an abstract decision problem on an instance set I, and let e_1 and e_2 be polynomially related encodings on I. Then, $e_1(Q) \in \mathrm{P}$ if and only if $e_2(Q) \in \mathrm{P}$.

- 有没有可能两种编码不是polynomially related，
  你能举个例子吗？
  在这种情况下，如何分析问题的难度呢？

你理解这些术语了吗？

- (an algorithm) **accepts** (a string/language)
  - (an algorithm) rejects (a string)
- (an algorithm) **decides** (language)

- accept和decide有什么区别？

你理解这些术语了吗？

- (an algorithm) **accepts** (a string/language)
  - (an algorithm) rejects (a string)
- (an algorithm) **decides** (language)

- accept和decide有什么区别？
- 为什么又有以下联系？

$$\mathrm{P} = \{L \subseteq \{0,1\}^* : \text{ there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}.$$

In fact, P is also the class of languages that can be accepted in polynomial time.

Theorem 34.2

$$\mathrm{P} = \{L : L \text{ is accepted by a polynomial-time algorithm}\}.$$

你理解这些术语了吗？

- Certificate
- (an algorithm) **verifies** (a string/algorithm)

你理解这些术语了吗？

- Certificate
- (an algorithm) **verifies** (a string/algorithm)

- 在HAM-CYCLE中，
  certificate是什么？如何verify？时间复杂度是多少？

你理解这些术语了吗？

- Certificate
- (an algorithm) **verifies** (a string/algorithm)

- 在HAM-CYCLE中，
  certificate是什么？如何verify？时间复杂度是多少？
- 在以下这些问题中，certificate分别是什么？如何verify？
  - GRAPH-ISOMORPHISM
  - CIRCUIT-SAT
  - (3-CNF-)SAT
  - CLIQUE
  - VERTEX-COVER
  - SUBSET-SUM
  - SOL-IP
  - EQ-POL
  - EQ-1BP
  - PRIM

你理解这些术语了吗？

- Certificate
- (an algorithm) **verifies** (a string/algorithm)

- 在HAM-CYCLE中，
  certificate是什么？如何verify？时间复杂度是多少？
- 在以下这些问题中，certificate分别是什么？如何verify？
  - GRAPH-ISOMORPHISM
  - CIRCUIT-SAT
  - (3-CNF-)SAT
  - CLIQUE
  - VERTEX-COVER
  - SUBSET-SUM
  - SOL-IP
  - EQ-POL
  - EQ-1BP
  - PRIM

- $L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}$.

这个问题属于NP，它真**呀！

- 当我们说一个问题属于NP时，
  我们是在夸它 容易 还是 困难？

$$L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

为什么P ⊆ NP?

$$\mathrm{P} = \{L \subseteq \{0,1\}^* : \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}.$$

$$L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

为什么P ⊆ NP?

$$\mathrm{P} = \{L \subseteq \{0,1\}^* : \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}.$$

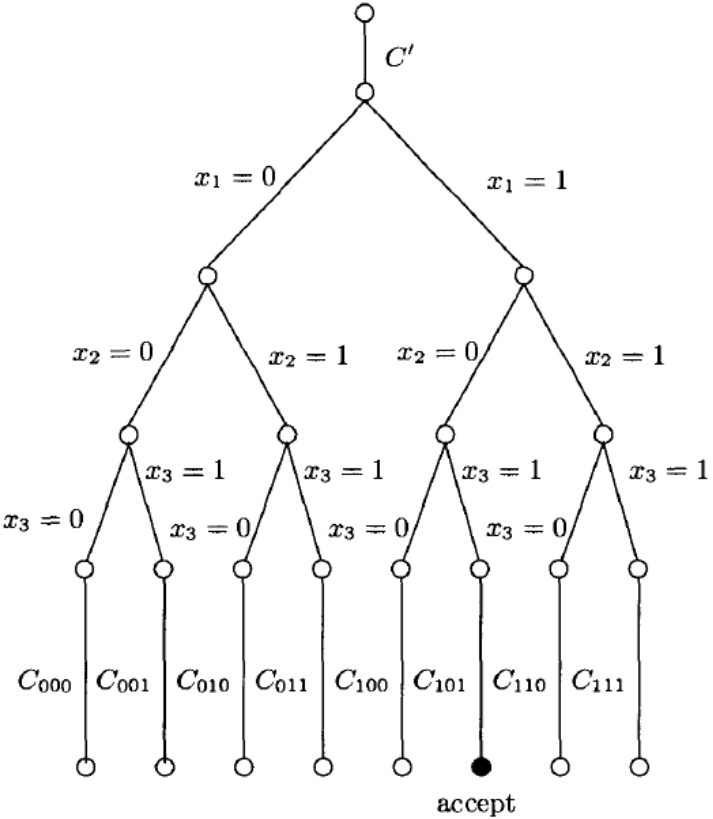
$$L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

Moreover, if $L \in \mathrm{P}$, then $L \in \mathrm{NP}$, since if there is a polynomial-time algorithm to decide L, the algorithm can be easily converted to a two-argument verification algorithm that simply ignores any certificate and accepts exactly those input strings it determines to be in L. Thus, $\mathrm{P} \subseteq \mathrm{NP}$.

为什么P ⊆ NP?

- 另一种视角
  - 确定性
  - 不确定性

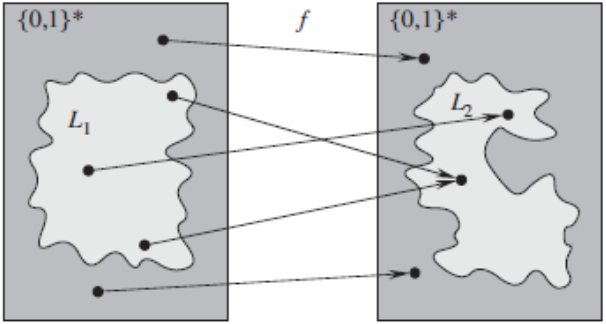
$$\Phi_x = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_3) \wedge \overline{x}_2$$



你认为NP ⊆ P吗？

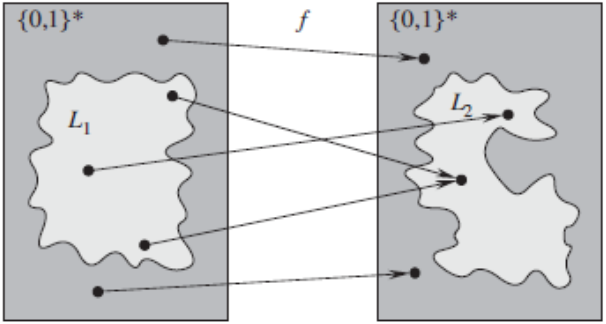
你理解这些术语了吗？

- Polynomial-time reducible



你理解这些术语了吗？

- Polynomial-time reducible



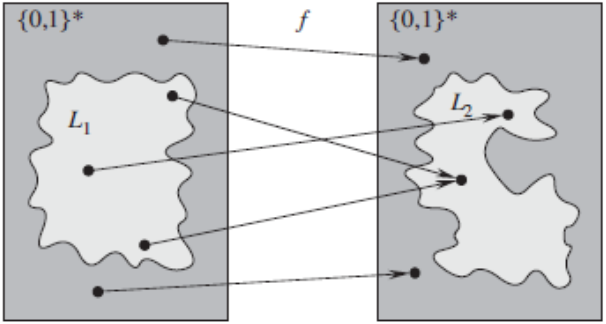
a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2 . \tag{34.1}$$

- 作为一种二元关系，它具有什么性质？

你理解这些术语了吗？

- Polynomial-time reducible



a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

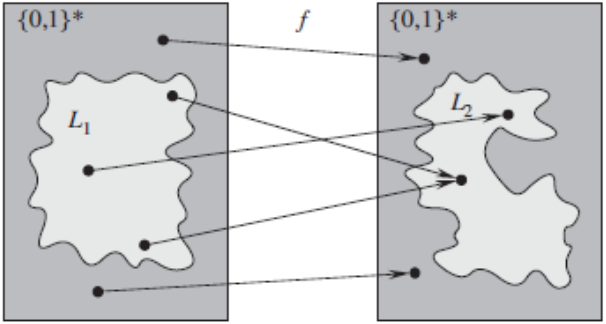
$$x \in L_1 \text{ if and only if } f(x) \in L_2 . \tag{34.1}$$

- 作为一种二元关系，它具有什么性质？

- 为什么判定L_1 至多 和判定L_2一样难？

你理解这些术语了吗？

- Polynomial-time reducible

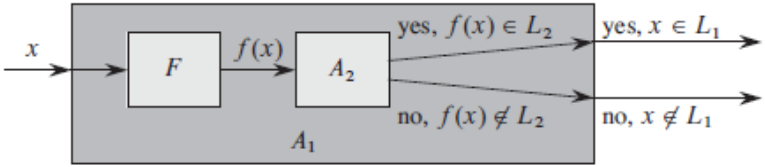


a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2 . \tag{34.1}$$

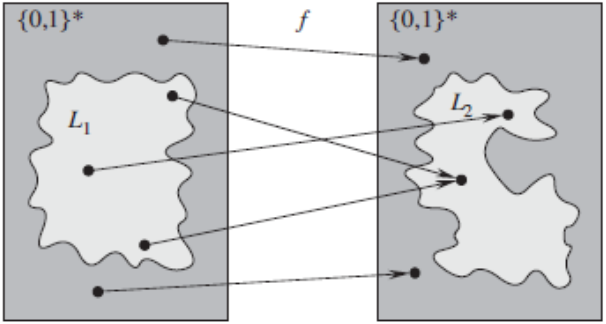
- 作为一种二元关系，它具有什么性质？

- 为什么判定L_1 至多 和判定L_2一样难？



你理解这些术语了吗？

- Polynomial-time reducible



a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

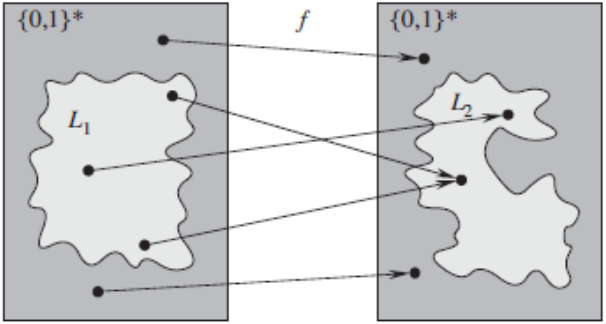
$$x \in L_1 \text{ if and only if } f(x) \in L_2 . \tag{34.1}$$

- 作为一种二元关系，它具有什么性质？

- 为什么判定L_1 至多 和判定L_2一样难？
- 如何用判定L_2的算法来判定L_1？

你理解这些术语了吗？

- Polynomial-time reducible

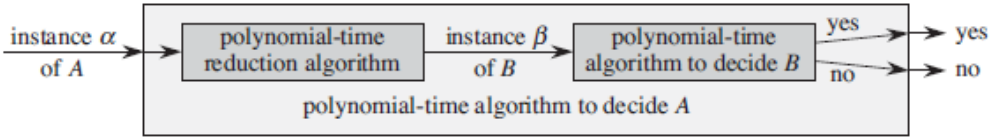


a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \le_P L_2$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2 . \tag{34.1}$$

- 作为一种二元关系，它具有什么性质？

- 为什么判定L_1 至多 和判定L_2一样难？
- 如何用判定L_2的算法来判定L_1？



你理解这些术语了吗？

- NP-complete
- NP-hard

你理解这些术语了吗？

- NP-complete
- NP-hard

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

1. $L \in \mathrm{NP}$, and
2. $L' \leq_{\mathrm{P}} L$ for every $L' \in \mathrm{NP}$.

If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***. We also define NPC to be the class of NP-complete languages.

你理解这些术语了吗？

- NP-complete
- NP-hard

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

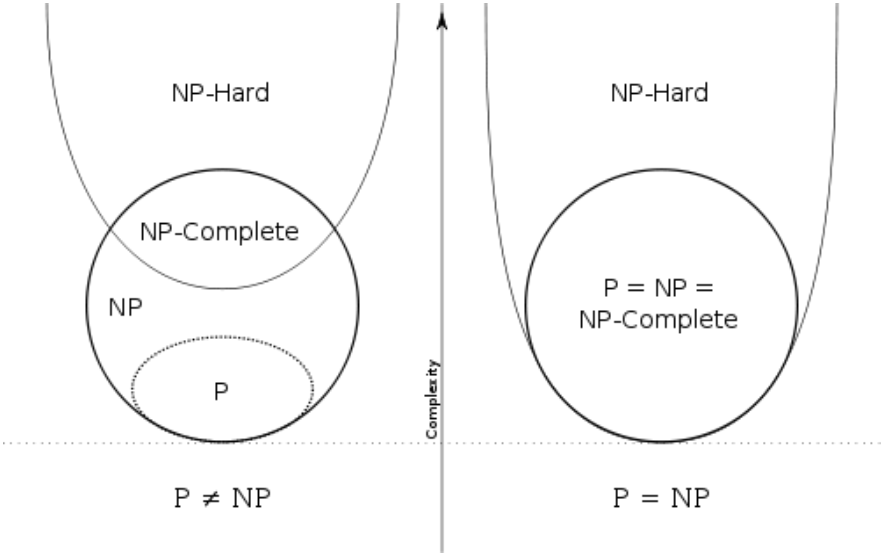
1. $L \in \mathrm{NP}$, and
2. $L' \leq_{\mathrm{P}} L$ for every $L' \in \mathrm{NP}$.

If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***. We also define NPC to be the class of NP-complete languages.

- 为什么这条定理成立？

Theorem 34.4
If any NP-complete problem is polynomial-time solvable, then P = NP. Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete problem is polynomial-time solvable.

P = NP?

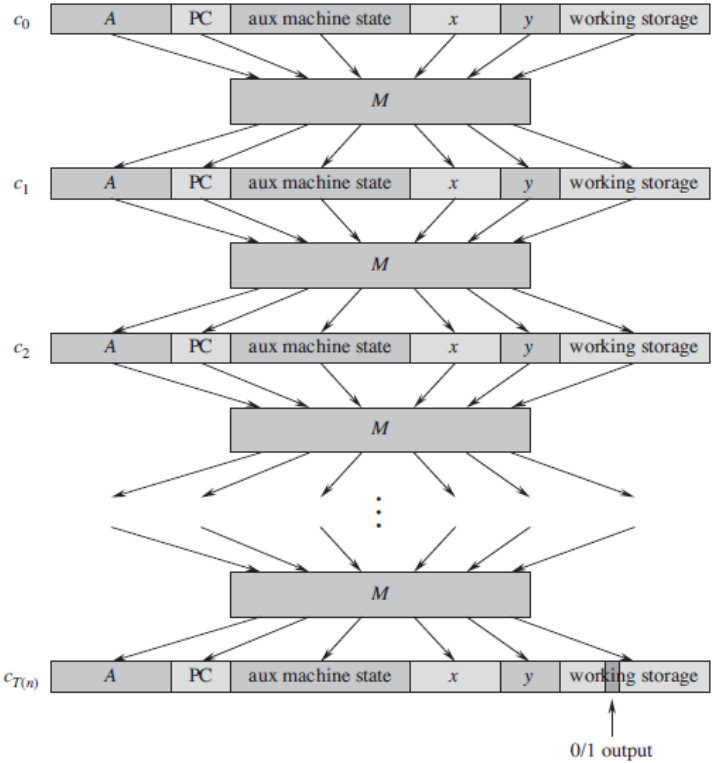


http://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/P_np_np-complete_np-hard.svg/500px-P_np_np-complete_np-hard.svg.png

你理解这个证明了吗？

Lemma 34.6
The circuit-satisfiability problem is NP-hard.

- 如何构造reduction？
  - 电路结构？
  - 输入输出？
- 如何证明多项式时间？

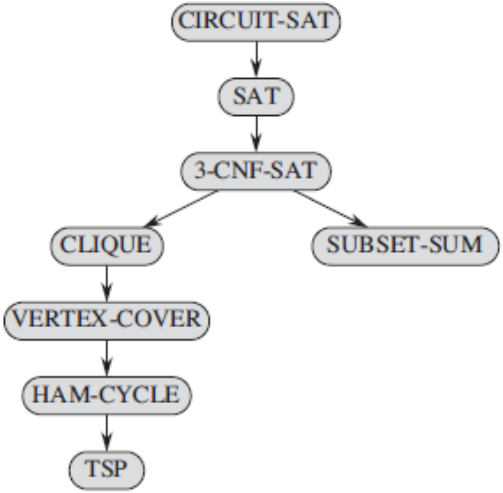


如何证明一个新问题是NP-complete/hard？

如何证明一个新问题是NP-complete/hard？

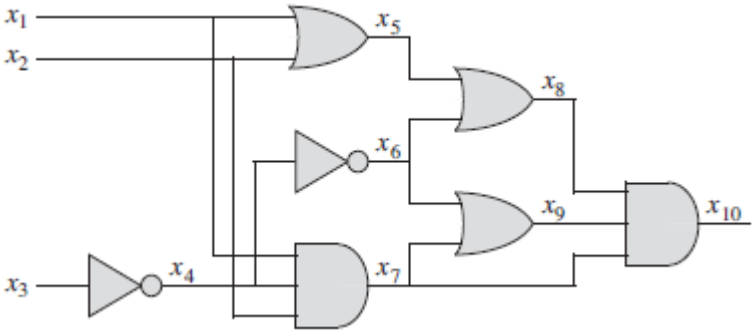
1. Prove $L \in \mathrm{NP}$.
2. Select a known NP-complete language L'.
3. Describe an algorithm that computes a function f mapping every instance $x \in \{0,1\}^*$ of L' to an instance $f(x)$ of L.
4. Prove that the function f satisfies $x \in L'$ if and only if $f(x) \in L$ for all $x \in \{0,1\}^*$.
5. Prove that the algorithm computing f runs in polynomial time.

你理解这张图了吗？


CIRCUIT-SAT
SAT
3-CNF-SAT
CLIQUE
SUBSET-SUM
VERTEX-COVER
HAM-CYCLE
TSP

你理解以下两个问题间的reduction了吗？

- CIRCUIT-SAT $\leq_P$ SAT

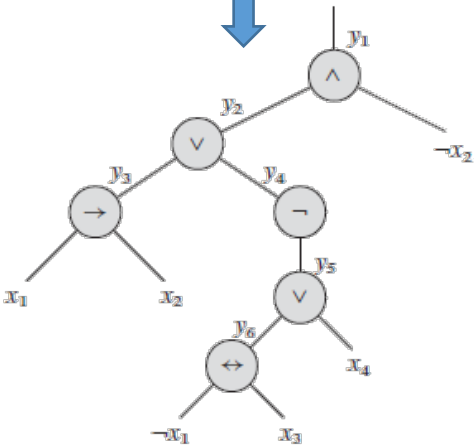


$$
\begin{aligned}
\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge (x_6 \leftrightarrow \neg x_4) \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)) .
\end{aligned}
$$

你理解以下两个问题间的reduction了吗？

- SAT $\leq_p$ 3-CNF-SAT

$$\phi = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2 \,.$$



$$\begin{aligned}\phi' \;=\; & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge\, (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge\, (y_3 \leftrightarrow (x_1 \to x_2)) \\ & \wedge\, (y_4 \leftrightarrow \neg y_5) \\ & \wedge\, (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge\, (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)) \,.\end{aligned}$$

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

$$\begin{aligned}\phi''_1 \;=\; & (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \\ & \wedge\, (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2) \,,\end{aligned}$$

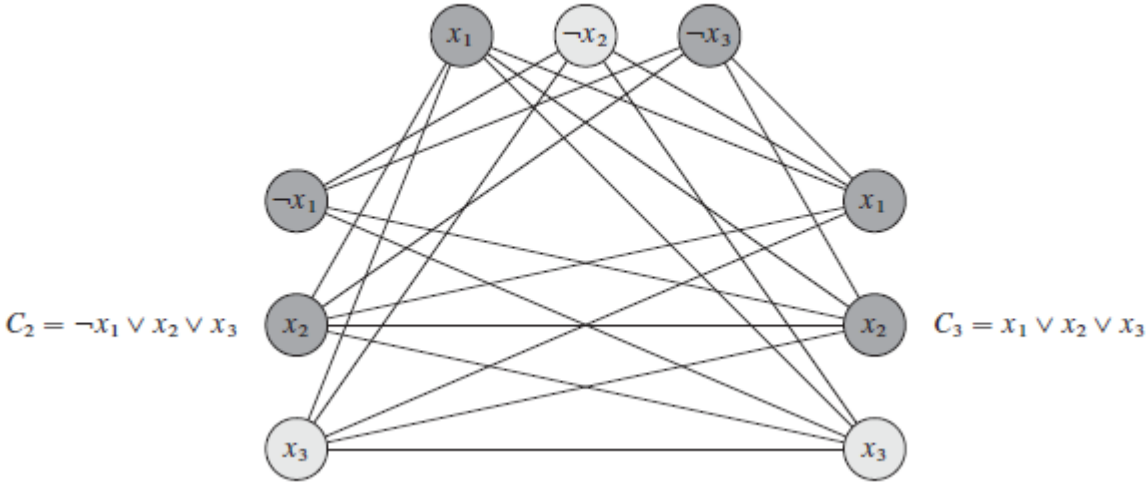
If C_i has 2 distinct literals, that is, if $C_i = (l_1 \vee l_2)$, where l_1 and l_2 are literals, then include $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ as clauses of ϕ'''.

你理解以下两个问题间的reduction了吗？

- 3-CNF-SAT $\leq_P$ CLIQUE

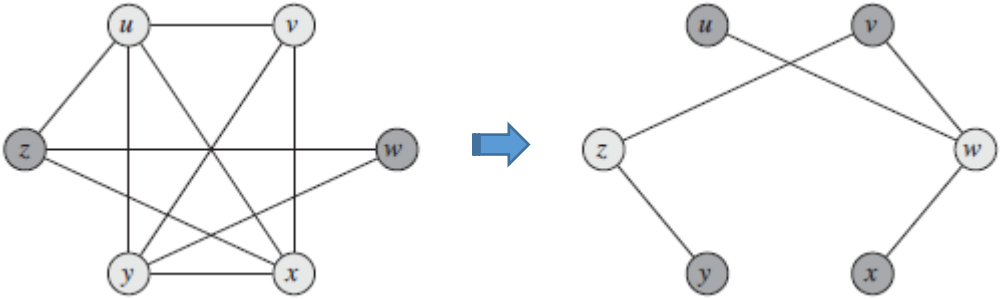
$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$



你理解以下两个问题间的reduction了吗？

- CLIQUE $\leq_P$ VERTEX-COVER



你理解以下两个问题间的reduction了吗？

- HAM-CYCLE $\leq_P$ TSP

你理解以下两个问题间的reduction了吗？

- 3-CNF-SAT $\leq_P$ SUBSET-SUM

$\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$, and $C_4 = (x_1 \vee x_2 \vee x_3)$.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v_1'	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v_3'	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s_3'	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

OT

- 请调研并介绍图灵机及其至少2种等价模型，讨论图灵机、P、NP之间的关系。