Computational Complexity
计算复杂性

张方国

中山大学计算机学院

isszhfg@mail.sysu.edu.cn

Lecture 6. NP-Completeness

- Definitions and existence of NP-complete problems
- Karp的21个NPC
- CSAT is NPC
- SAT is NPC
- 3SAT is NPC

Definitions of NP-completeness

NP-completeness of a problem Π combines two conditions:

- Π is in the class (i.e., Π being in $\mathcal{NP}$ or $\mathcal{PC}$, depending on whether Π is a decision or a search problem).
- Each problem in the class is reducible to Π. This condition is called NP-hardness.

NP-completeness of decision problem, restricted notion A set S is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every set in $\mathcal{NP}$ is Karp-reducible to S.

NP-completeness of search problem, restricted notion A binary relation R is $\mathcal{PC}$-complete if it is in $\mathcal{PC}$ and every relation in $\mathcal{PC}$ is Levin-reducible to R.

S. A. Cook. The complexity of theorem proving procedures, Proceedings, Third Annual ACM Symposium on the Theory of Computing. New York: ACM. 1971: 151-158.

NPC问题，又称NP完全问题或NP完备问题，是NP中最难的问题。

因此NP完备问题应该是最不可能被多项式归约为P问题的集合。

如果任何一个NPC问题多项式时间可解，则P=NP。

The Existence of NP-complete Problems

Theorem

There exist NP-complete relations and sets.

Proof: The relation R_u consists of pairs $(< M, x, 1^t >, y)$ such that M accepts the input pair(x, y) within t steps, where $|y| \leq t$. The corresponding set $S_u = \{\bar{x} : \exists y \ s.t.(\bar{x}, y) \in R_u\}$ consists of triples $< M, x, 1^t >$ such that machine M accepts some input of the form $(x, \cdot)$ within t steps.

$$f : S \to S_u$$

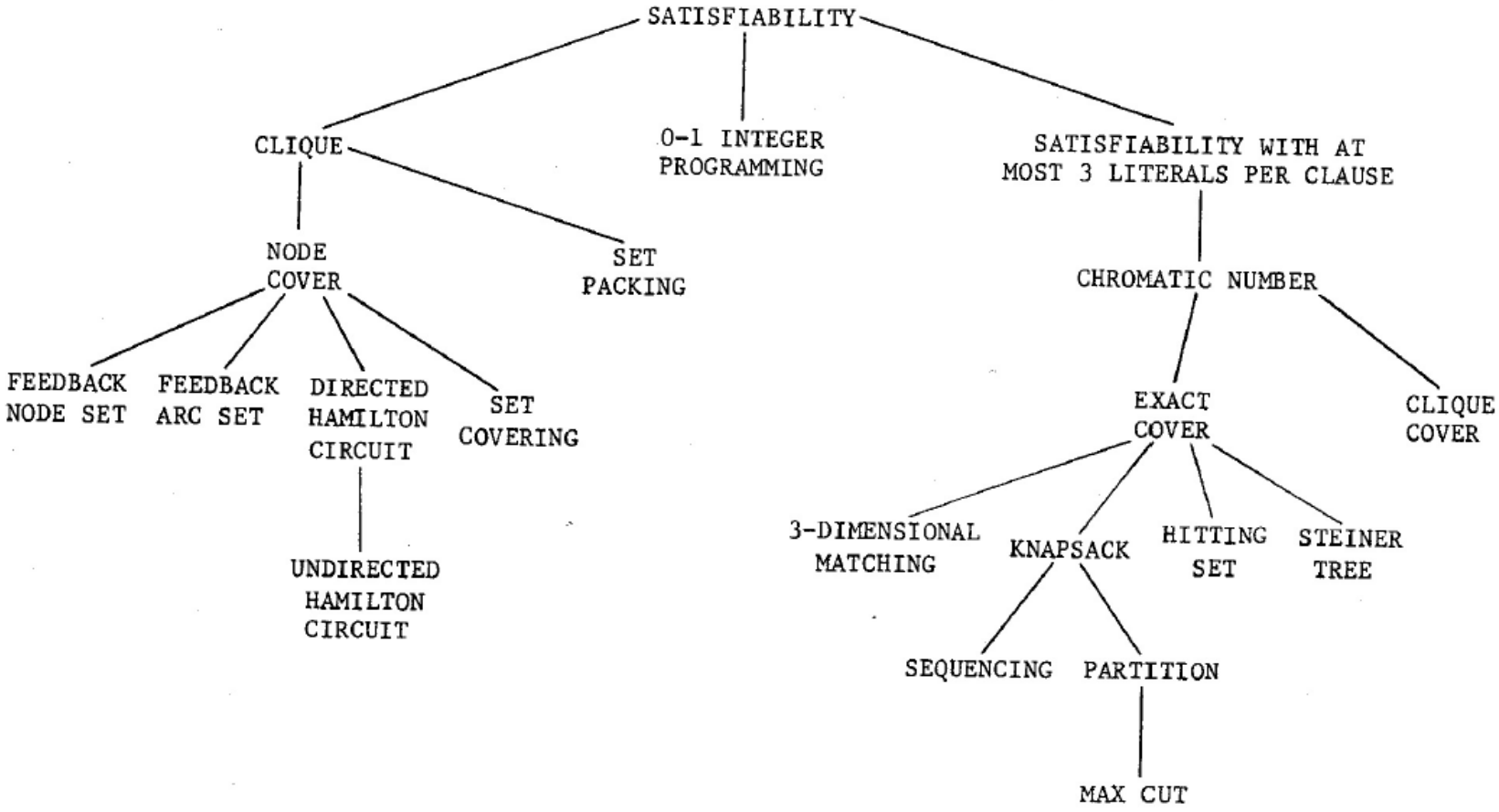
$$x \to f(x) =< M_R, x, 1^{t_R(|x|+p_R(|x|))} >$$

Some natural NP-complete problems

Proposition

If an NP-complete problem Π is reducible to some problem Π' in NP then Π' is NP-complete. Furthermore, reducibility via Karp-reductions(resp., Levin-reductions) is preserved.

Karp's 21 NPC Problems



NPC问题的证明思路

CSAT(按NPC定义证明出第一个NPC)$\hookrightarrow$ SAT $\hookrightarrow$ CNF-SAT $\hookrightarrow$ 3-SAT.

CNF-SAT $\hookrightarrow$ Set Cover.

Circuit satisfiability(CSAT) and formula satisfiability(SAT)

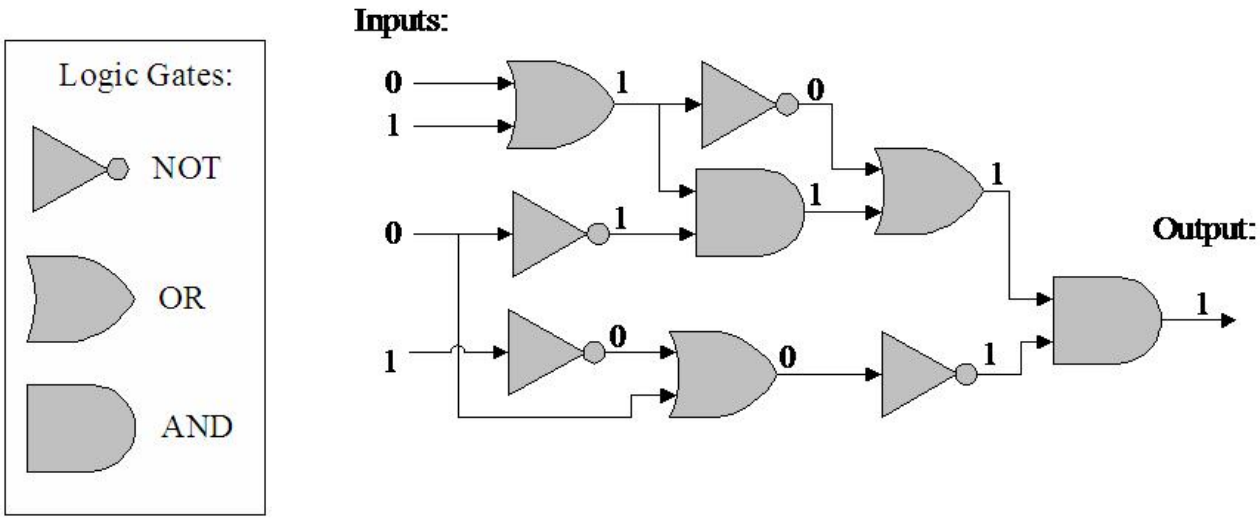
The evaluation of circuit C on input z is denoted $C(z)$. We focus on circuits with a single output, and let CSAT denote the set of satisfiable Boolean circuits (i.e., a circuit C is in CSAT if there exists an input z such that $C(z) = 1$). We also consider the related relation $R_{CSAT} = \{(C, z) : C(z) = 1\}$.

Theorem

(NP-completeness of CSAT:) The set (resp., relation) CSAT (resp., R_{CSAT}) is $\mathcal{NP}$-complete (resp.,$\mathcal{PC}$-complete).

Proof:

The computation of polynomial-time algorithms can be emulated by polynomial-size circuits.



The NP-hardness of CSAT is an immediate consequence of the fact that Boolean circuits can emulate algorithms.

证明思路: We will reduce an arbitrary set $S \in \mathcal{NP}$ to CSAT. i.e., we will construct a Karp-reduction that maps an instance x (for S) to a circuit, denoted $f(x) = C_x$, such that $C_x(y) = 1$ if and only if M_R accepts the input (x, y) within $t_R(|x| + p_R(|x|))$ steps. Thus, $x \in S$ if and only if there exists $y \in \{0,1\}^{p_R(|x|)}$ such that $C_x(y) = 1$(i.e., if and only if $C_x \in CSAT$)

Any function that has time complexity t has circuit complexity $poly(t)$, i.e., "The computation of polynomial-time algorithms can be emulated by polynomial-size circuits".

The circuit C_x is from the computation of M_R on input (x, y), where x is fixed and y represents a generic string of length at most $p_R(|x|)$.

Such a computation proceeds for $t = t_R(|x| + p_R(|x|))$ steps, and corresponds to a sequence of $t + 1$ instantaneous configurations, each of length t. Each such configuration can be encoded by t pairs of symbols, where the first symbol in each pair indicates the contents of a cell and the second symbol indicates either a state of the machine or the fact that the machine is not located in this cell.

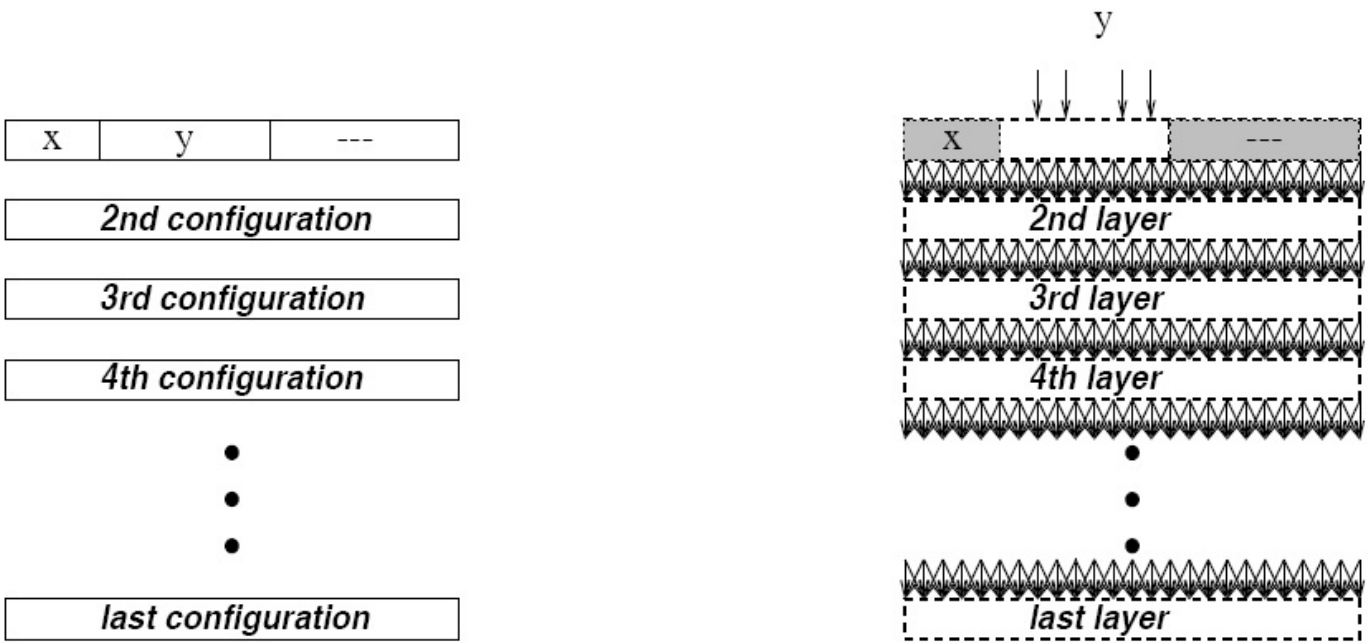


Figure 4.1: The schematic correspondence between the configurations in the computation of $M(x, y)$ (on the left) and the evaluation of the circuit C_x on input y (on the right), where x is fixed and y varies. The value of x (as well as a sequence of blanks) is hard-wired (marked grey) in the first layer of C_x, and directed edges connect consecutive layers.

证明一个问题A属于NP完全的证明方法:

首先证明A属于NP ;

最后证明是NP难的，其证明过程如下:

a) 首先选择一个已知属于NP完全的问题A';

b) 设计一个计算函数f的算法，能将A'的每个实例$x \in \{0,1\}*$映射到A的实例$f(x)$;

c) 证明上述转换函数f满足，对任意的$x \in \{0,1\}*$，$x \in A'$当且仅当有$f(x) \in A$;

d) 证明计算函数f的算法是多项式时间算法。

Formula satisfiability(SAT)

布尔公式可满足性指的是一个给定的布尔公式是否是可满足的。用形式语言可以描述如下:

$$SAT = \{\phi : \phi \ is \ a \ satisfiable \ boolean \ formula\}.$$

可满足性问题可以描述为，对于给定的ϕ，判定ϕ是否属于SAT。

Theorem

(NP-completeness of SAT:) The set (resp., relation) SAT (resp., R_{SAT}) is $\mathcal{NP}$-complete (resp.,$\mathcal{PC}$-complete).

We further restrict our attention to formulae given in conjunctive normal form (CNF).

We denote by SAT the set of satisfiable CNF formulae (i.e., a CNF formula ϕ is in SAT if there exists an truth assignment τ such that $\phi(\tau) = 1$).

We also consider the related relation $R_{SAT} = \{(\phi, \tau) : \phi(\tau) = 1\}$.

Proof: 首先我们证明SAT属于NP：对于一个布尔公式ϕ以及它的一个可满足赋值构成的证据y，验证算法只要将公式中的变元用其赋值代替，然后计算公式ϕ的值，这个过程显然能够在多项式时间内完成，如果$\phi(y) = 1$，则它是可满足的。

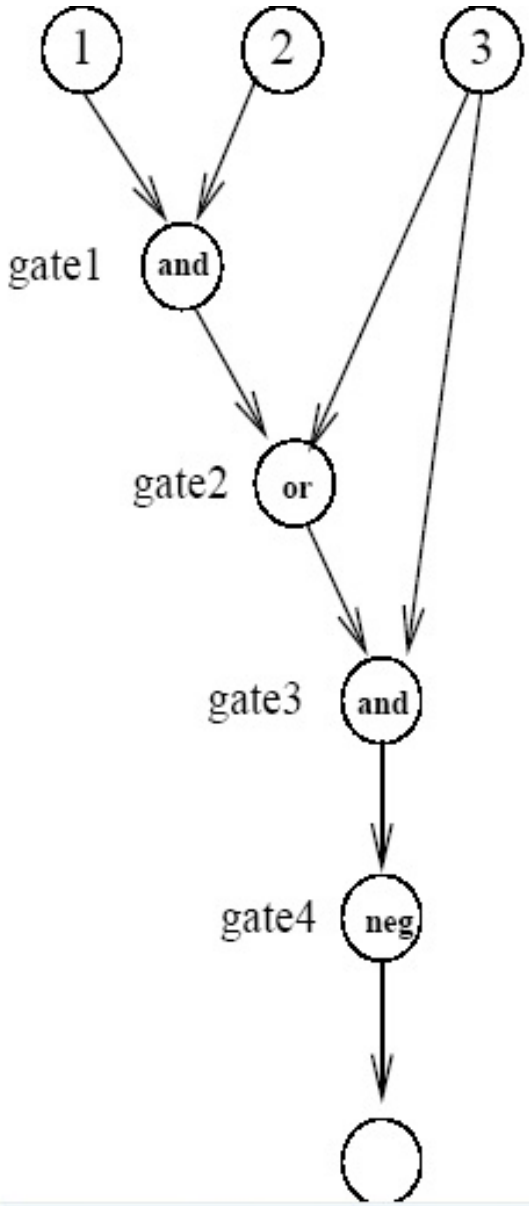
NP困难的证明：Reduce CSAT to SAT.

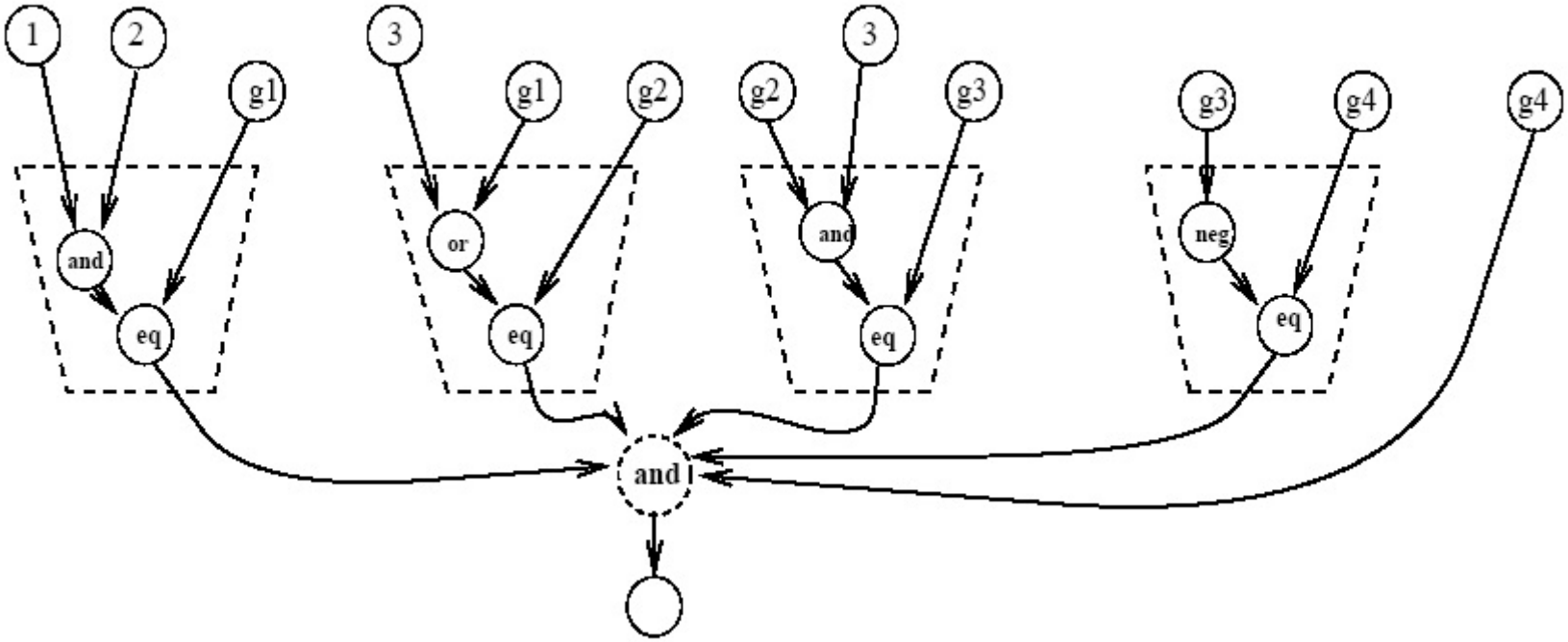
首先我们要证明CSAT的任意一个实例能够在多项式时间内约简到可满足性问题的一个实例。我们可以按照下列方式进行约简，对于组合电路的每个输入，布尔公式有对应的变元。电路中一个逻辑门能够表达为一个公式，公式由逻辑门的输入和输出决定

Given a Boolean circuit C, with n input terminals and m gates, we first construct m constant-size formulae on $n+m$ variables, where the first n variables correspond to the input terminals of the circuit, and the other m variables correspond to its gates. The i^{th} formula will depend on the variable that correspond to the i^{th} gate and the 1-2 variables that correspond to the vertices that feed into this gate.

This (constant-size) formula will be satisfied by a truth assignment if and only if this assignment matches the gate's functionality. Note that these constant-size formulae can be written as constant size CNF formulae.

Taking the conjunction of these m formulae as well as the variable associated with the gate that feeds into the output terminal, we obtain a formula ϕ in CNF


1
2
3
gate1
and
gate2
or
gate3
and
gate4
neg


1
2
g1
and
eq
3
g1
g2
or
eq
g2
3
g3
and
eq
g3
g4
neg
eq
g4
and

"The set SAT is $\mathcal{NP}$-complete" 就是著名的Cook-Levin定理，简称Cook定理。虽然，本书是从证明CSAT属于NP完全开始的，是因为CSAT的证明比较容易理解，而直接证明SAT属于NP完全则要复杂些，但是证明它们的思路是一样的。事实上，SAT才是第一个被证明的NP完全问题。Cook定理揭开了计算复杂性中NP完全理论的研究。在此基础上关于NP完全问题的研究，成为过去几十年来计算机科学最活跃和重要的研究领域之一。Stephen A. Cook因为其在计算复杂性领域的开创性贡献而获得1982年的图灵奖。

kSAT

If a boolean formula is in CNF and every clause(子句) consists of exactly k literals(文字), we say the boolean formula is an instance of k-SAT. Say the formula is in k-CNF.

Example: 3-SAT formula:

$$(x+y+z)(x+-y+z)(x+y+-z)(x+-y+-z)$$

or

$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee \overline{z})$$

Some facts about kSAT:

1. Every boolean formula has an equivalent CNF formula. But the size of the CNF formula may be exponential in the size of the original.
2. Not every boolean formula has a k-SAT equivalent
3. 2SAT is in P;
4. 3SAT is NP-complete.

3SAT is NP-complete

Proof. We know that since 3-SAT is merely a special case of SAT, it must be in NP.

To show that it 3-SAT hard for NP, we will reduce SAT to it by transforming any instance of the satisfiability problem to an instance of 3-SAT. This means we must demonstrate how to convert clauses which do not contain exactly three literals into ones which do. It is easy if a clause contains two literals. Let us take (x_1, x_2) as an example. This is equivalent to the pair (x_1, x_2, u) $(x_1, x_2, \overline{u})$, where u is a new variable. Note that each clause of the pair contains exactly three literals.

Now we will transform clauses such as (x) which contain one literal. This will require two steps. We begin by converting it to the pair of two literal clauses: (x, u_1) $(x, \overline{u_1})$, much as before. Then we change each of these just as before and get: (x, u_1, u_2) $(x, \overline{u_1}, \overline{u_2})$.

One case remains. We might have a clause such as $(x_1, ..., x_k)$ which contains more than three literals. We shall arrange these literals as a cascade of three literal clauses. Consider the sequence of clauses:

$$(x_1, x_2, u_1)(x_3, \overline{u_1}, u_2), ..., (x_{k-2}, \overline{u_{k-4}}, u_{k-3}), (x_{k-1}, x_k, \overline{u_{k-3}})$$

If the original clause were satisfiable then one of the x_i's had to be true. Let us set all of the u_i's to true up to the point in the sequence where x_i was encountered and false thereafter. So, if the original clause was satisfiable, this collection is satisfiable too.

Now for the other part of the proof. Suppose the original clause is not satisfiable. This means that all of the x_i's are false. We claim that in this case the collection of clauses we constructed is unsatisfiable also. Assume that there is some way to satisfy the sequence of clauses. For it to be satisfiable, the last clause must be satisfiable. For the last clause to be satified, u_{k-3} must be false since x_{k-1} and x_k are false. This in turn forces u_{k-4} to be false. Thus all of the ui's all the way down the line have got to be false. And when we reach the first clause we are in big trouble since u1 is false. So, if the xi's are all false there is nothing we can do with the truth values for the ui's that satisfies all of the clauses.

Note that the above transformation is indeed a polynomial time mapping. We reduce SAT to 3-SAT.

Thank You Very Much!