

## Undecidability

Recall that  $\Sigma^*$  represents  
a countable number of strings over a finite alphabet.

All possible languages would be  
every possible arrangement of string,  
which is the power set of  $\Sigma^*$  and is uncountably infinite.

What does this mean?

While there are a countable number of regular context-free,  
context-sensitive and Turing acceptable languages,  
there are languages outside this domain.

There are problems for which no solution exists.

Slide Lecture 15 –445

The Halting Problem:

Does some TM program contain an infinite loop?

This is a problem for which no solution exists.

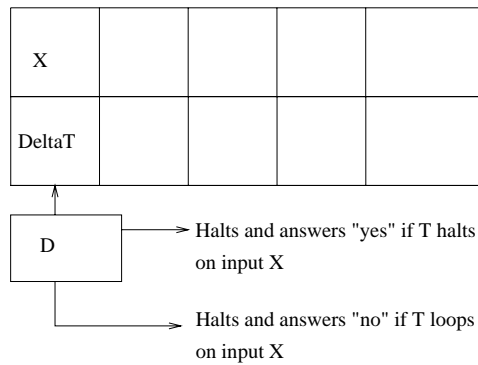
### Proof

We attempt to construct a “Universal Testing Program”  
which takes an arbitrary program  $P$  of some language  
and answers this question:

Does there exist an input  $W$   
such that  $P$  loops indefinitely on  $W$ ?

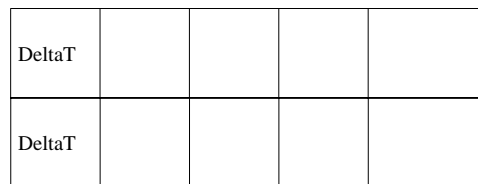
Slide Lecture 15 –446

Assume we have a TM, D, which decides the halting problem.  
 It takes as input the description of the Turing machine,  
 Delta-T, and the input, X, to the Turing machine.



Slide Lecture 15 -447

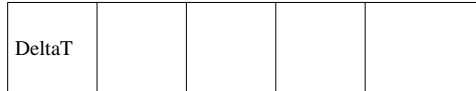
We are interested in a special case where  
 the input is also the program description, Delta-T.



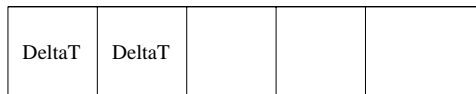
Slide Lecture 15 -448

We can construct a TM “E” which is composed of “D” plus states to copy Delta-T.

E TAKES

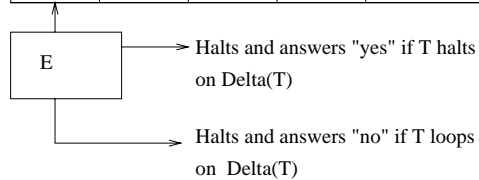
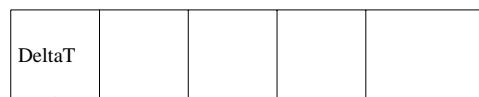


AND GENERATES



AND TURNS CONTROL OVER TO "D"

Slide Lecture 15 –449



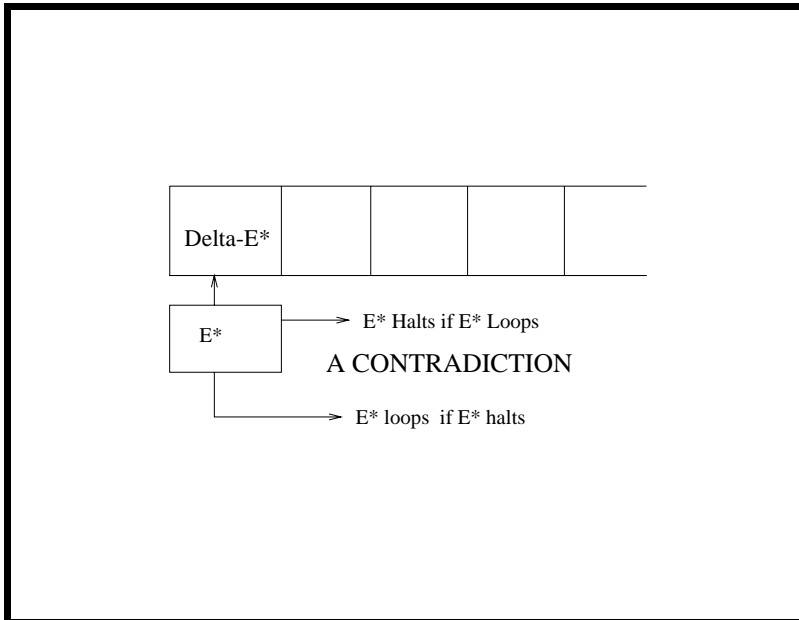
Finally, we modify “E” slightly to obtain  $E^*$ . We add two new states:

If T loops on Delta-T, enter state  $q_h$  and halt.

If T halts on Delta-T, enter state  $q_L$  and loop.

Now let  $T = E^*$ . (run  $E^*$ ) on itself.

Slide Lecture 15 –450



Slide Lecture 15 –451

We could also do this in Pascal,C,Lisp etc.,

Let  $P^*$  be our test program.

$P^*(P)$  :

If  $P$  halts .... $P^*$  loops.

If  $P$  loops .... $P^*$  halts.

$P^*(P^*)$  :

If  $P^*$  halts .... $P^*$  loops.

If  $P^*$  loops .... $P^*$  halts.

Slide Lecture 15 –452

Of course, this proof depends on the fact that Turing Machines have infinite memory.

This isn't a problem for finite state machines, since we know that if we remove  $\lambda$  - loops, all finite state machines halt.

But of course there are things we can't compute with a finite state machine.

**Slide Lecture 15 –453**

$\mathcal{P}$  and  $\mathcal{NP}$

**NP-Complete and NP-Hard**

$\mathcal{P}$  = P-Time = Polynomial Time Decidable  
on a Deterministic TM

$\mathcal{NP}$  = NP-Time = Polynomial time Decidable  
on a Nondeterministic TM

**Slide Lecture 15 –454**

Martin:

The class  $\mathcal{P}$  is the set of all languages  $L$  that can be accepted in polynomial time. There exists a TM  $T$  and a natural number  $k$  so that the time complex of  $T$  on an input of length  $n$  is  $O(n^k)$ .

The class  $\mathcal{NP}$  is the set of all languages  $L$  that can be accepted in nondeterministic polynomial time by a Nondeterministic Turing machine.

**Slide Lecture 15 –455**

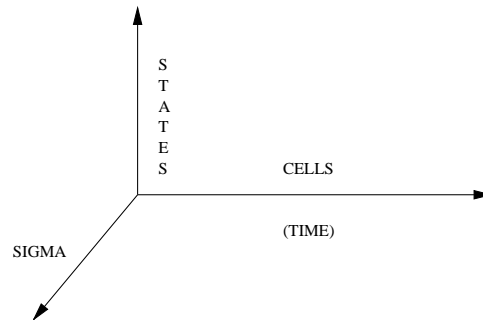
All languages accepted in P-time are decided in P-time.

All languages accepted in NP-time are decided in NP-time.

Let  $F(n)$  represent the maximum (polynomial) time required to accept any string of length  $n$ . Run the TM for time  $2^{*}F(n)$ . If the string is not accepted, it is rejected.

**Slide Lecture 15 –456**

All NP-time computations carried out by a  
 Nondeterministic Turing Machine can be expressed by a  
 Boolean Expression  
 with a polynomial number of variables.



Slide Lecture 15 –457

Let  $H_{t,l}$  be true if the tape head is at cell  $l$  at time  $t$ .

Let  $Q_{t,j}$  be true if the TM is in state  $q_j$  at time  $t$ .

Let  $S_{t,k}$  be true if the TM reads symbol  $\sigma_k$  at time  $t$ .

Since time is polynomial,  
 and the number of states and input symbols is constant,  
 the space of  $HXS$  is polynomial.

Transitions can be written

$$(Q_{t,j} \wedge H_{t,l} \wedge S_{t,k}) \implies (Q_{t+1,j'} \wedge H_{t+1,l'} \wedge S_{t+1,k'})$$

Slide Lecture 15 –458

There are other constraints,  
 E.G. The machine must be in 1 of  $r$  states at each time step.

$$(Q_{t,0} \wedge \overline{Q_{t,1}} \wedge \overline{Q_{t,2}} \dots \overline{Q_{t,r-1}}) \vee$$

$$(\overline{Q_{t,0}} \wedge Q_{t,1} \wedge \overline{Q_{t,2}} \dots \overline{Q_{t,r-1}}) \vee$$

$$(\overline{Q_{t,0}} \wedge \overline{Q_{t,1}} \wedge Q_{t,2} \dots \overline{Q_{t,r-1}}) \vee$$

...

$$(\overline{Q_{t,0}} \wedge \overline{Q_{t,1}} \wedge \overline{Q_{t,2}} \dots Q_{t,r-1})$$

But these constraints are all polynomial.  
 See Martin, pages 436 to 442 for the DETAILS.

Slide Lecture 15 –459

Given a specific input string,  
 there is a way to assign values  $\{0, 1\}$   
 to the resulting Boolean Expression  
 so that it evaluates to true  
 IFF the Turing Machine accepts the input string.  
 Thus *every problem in  $\mathcal{NP}$  can be express as a Boolean Satisfiability problem.*

In other words:

**Cook's Theorem**

*Boolean Satisfiability problem (SAT) is NP-Complete.*

Slide Lecture 15 –460

*Boolean Satisfiability* is also “solved” in NP-time by a Nondeterministic Turing Machine.

The Nondeterministic TM magically picks the correct value for each Boolean variable. It thus “solves” this problem in linear time.

Since every problem in  $\mathcal{NP}$  can be expressed as a Boolean Satisfiability problem in P-time, we say that every problem in  $\mathcal{NP}$  *reduces* to Boolean Satisfiability.

**Slide Lecture 15 –461**

More Formally,

Let  $\Sigma_1$  and  $\Sigma_2$  be finite alphabets, and  $L_1$  and  $L_2$  be languages in  $\Sigma_1^*$  and  $\Sigma_2^*$  respectively.  $L_1$  is said to be *polynomial-time reducible* to  $L_2$  if there is a total function

$$f : \Sigma_1^* \longrightarrow \Sigma_2^* \text{ such that}$$

$$x \in L_1 \text{ IFF } f(x) \in L_2$$

where  $f$  can be computed in polynomial time. I.E., there is a Turing Machine computing  $f$  with polynomial time complexity.

**Slide Lecture 15 –462**

We use the notation

$$L_1 <_p L_2$$

to indicate  $L_1$  is polynomial time reducible to  $L_2$ .

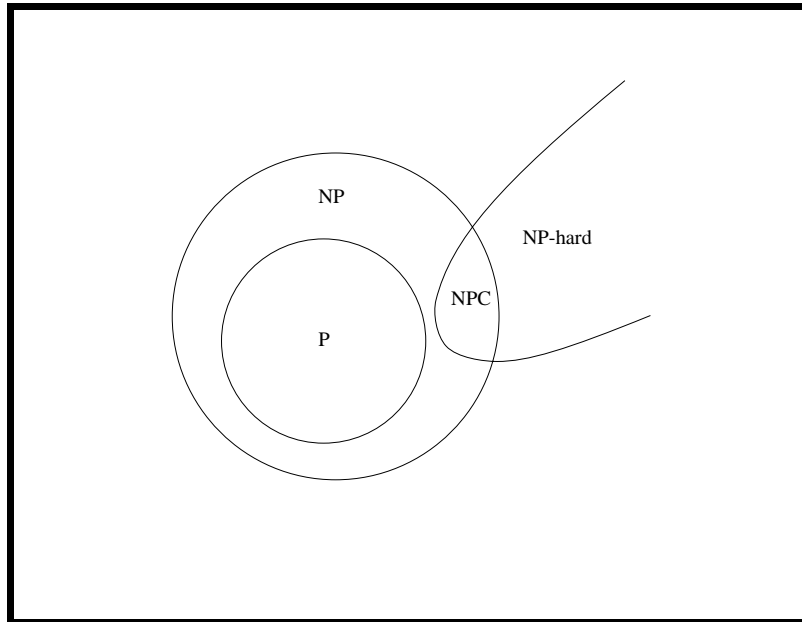
Slide Lecture 15 –463

A language  $L \in \Sigma^*$  is **NP-hard** if for every language  $L' \in \mathcal{NP}$ ,

$$L' <_p L$$

If language  $L$  is NP-hard and  $L \in \mathcal{NP}$ ,  $L$  is **NP-Complete**.

Slide Lecture 15 –464



Slide Lecture 15 –465

Since every Nondeterministic Turing Machine  
can be expressed as a Boolean Satisfiability Problem,  
SAT is NP-hard.

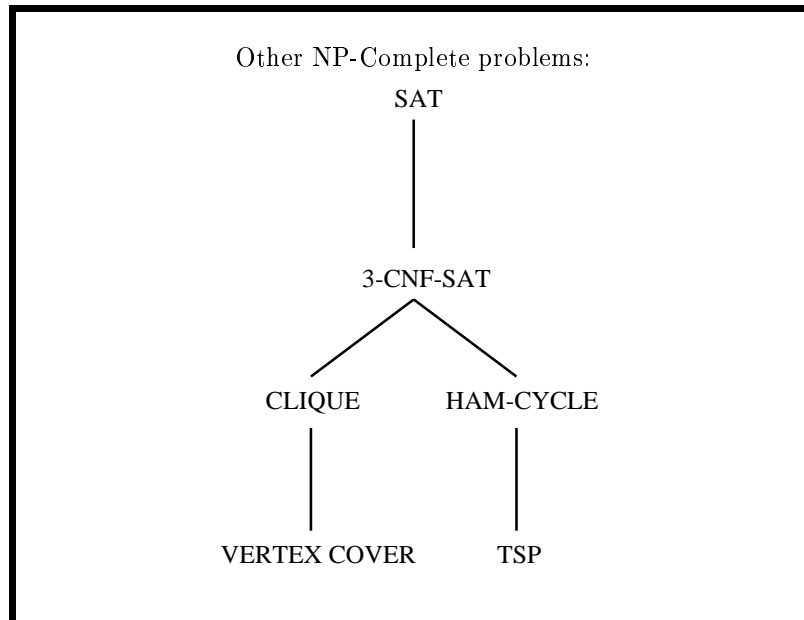
Since every Boolean Expression can be evaluated  
in linear time (wrt number of variables)  
by a Nondeterministic TM,

$$\text{SAT} \in \mathcal{NP}$$

and thus

SAT is NP-Complete

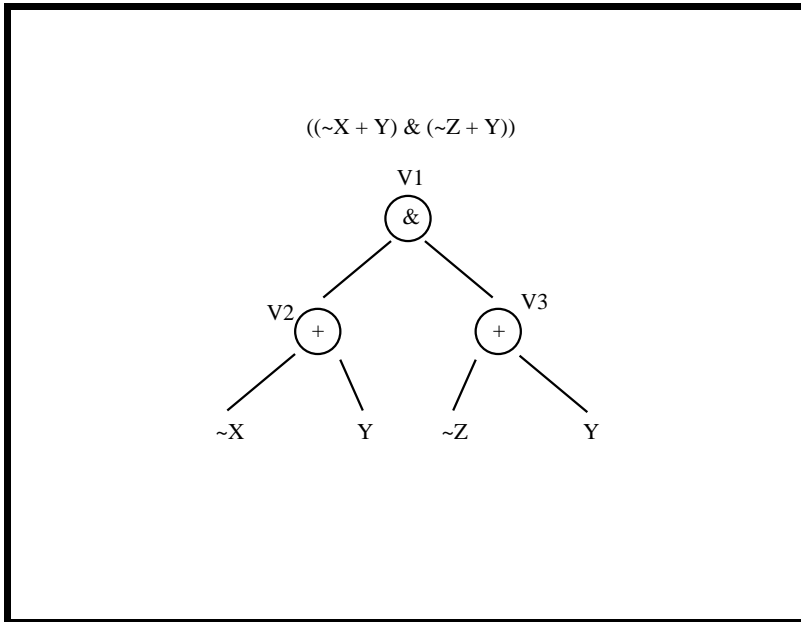
Slide Lecture 15 –466



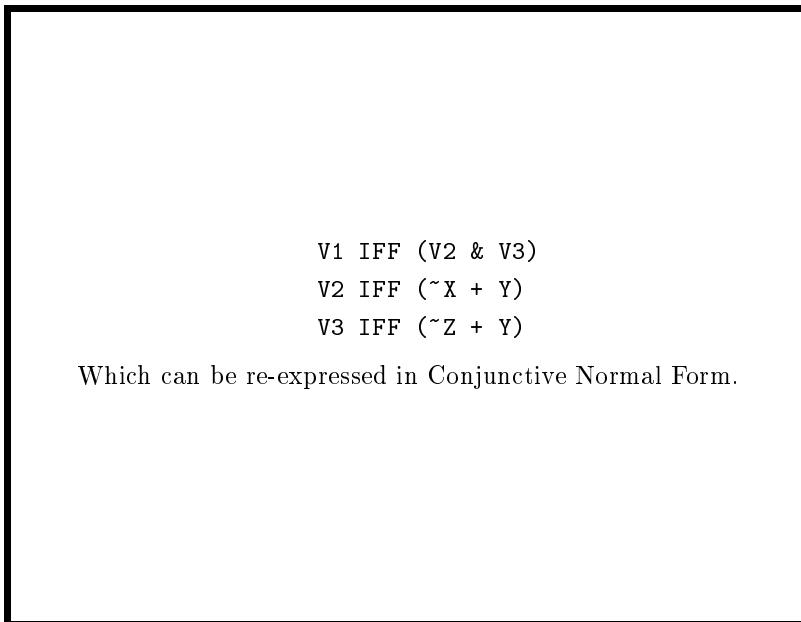
Slide Lecture 15 –467

CLIQUE = Complete Subgraph Problem  
(Often expressed as Maximum subgraph problem)  
HAM-CYCLE = The Hamiltonian Circuit Problem.  
TSP = The Traveling Salesman Problem.

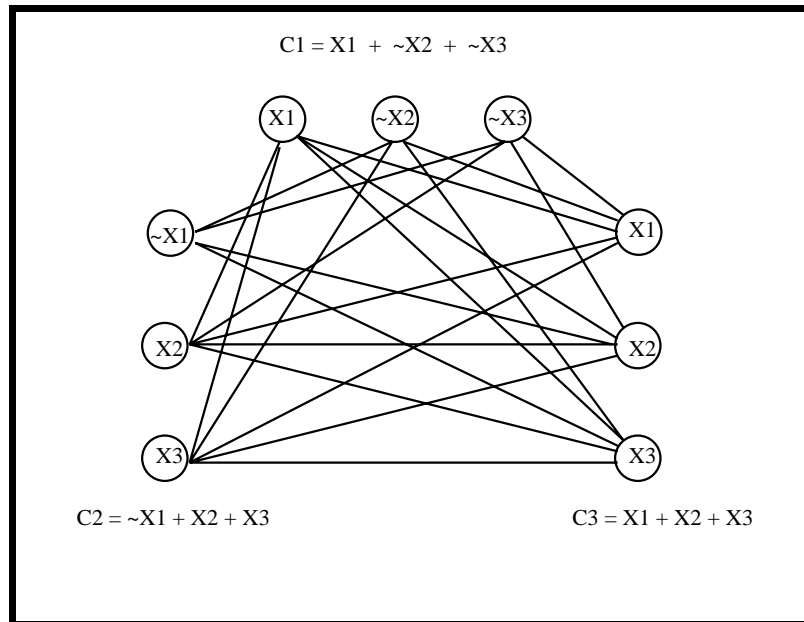
Slide Lecture 15 –468



Slide Lecture 15 –469



Slide Lecture 15 –470



Slide Lecture 15 –471

Since every SAT problem can be expressed  
as a 3-CNF-SAT problem,  
and every 3-CNF-SAT can be expressed  
as a Clique (complete subgraph) problem

$SAT <_p 3\text{-CNF} <_p \text{CLIQUE}$

So CLIQUE is NP-Hard.

CLIQUE is also in NP, since  
we can find the CLIQUE in linear time  
given a Nondeterministic TM.

Thus, CLIQUE is NP-Complete.

Slide Lecture 15 –472

THE OPEN QUESTION

$$\mathcal{P} = \mathcal{NP}$$

It is believed that

$$\mathcal{P} \neq \mathcal{NP}$$

There are more than 1000 NP-Complete problems. None are known to have a P-time solution.