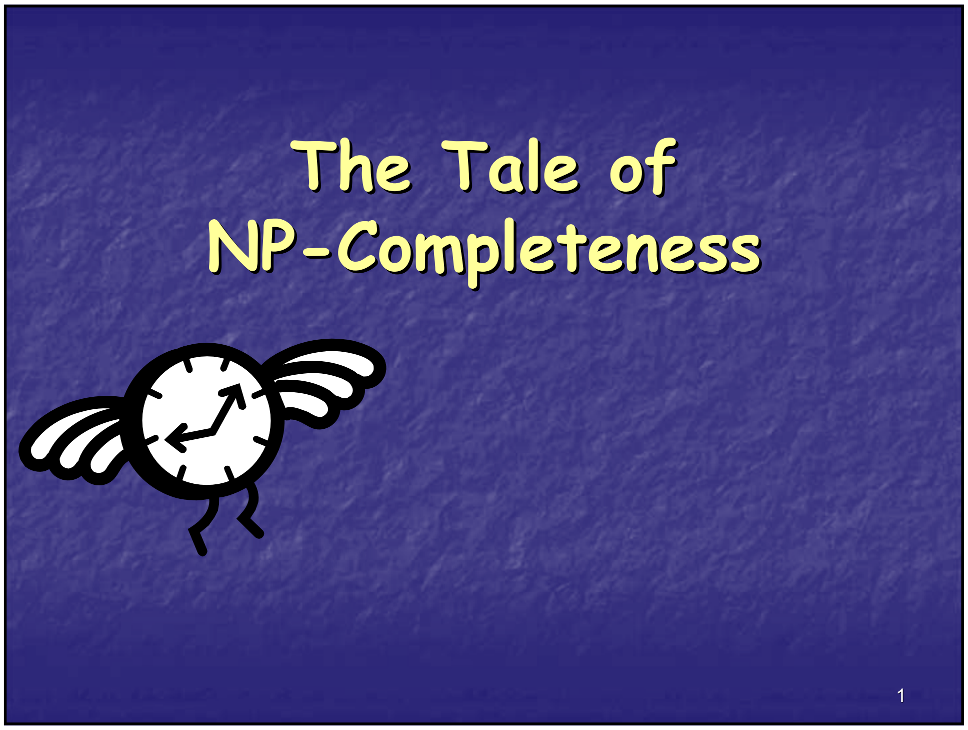
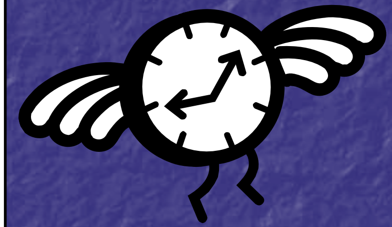
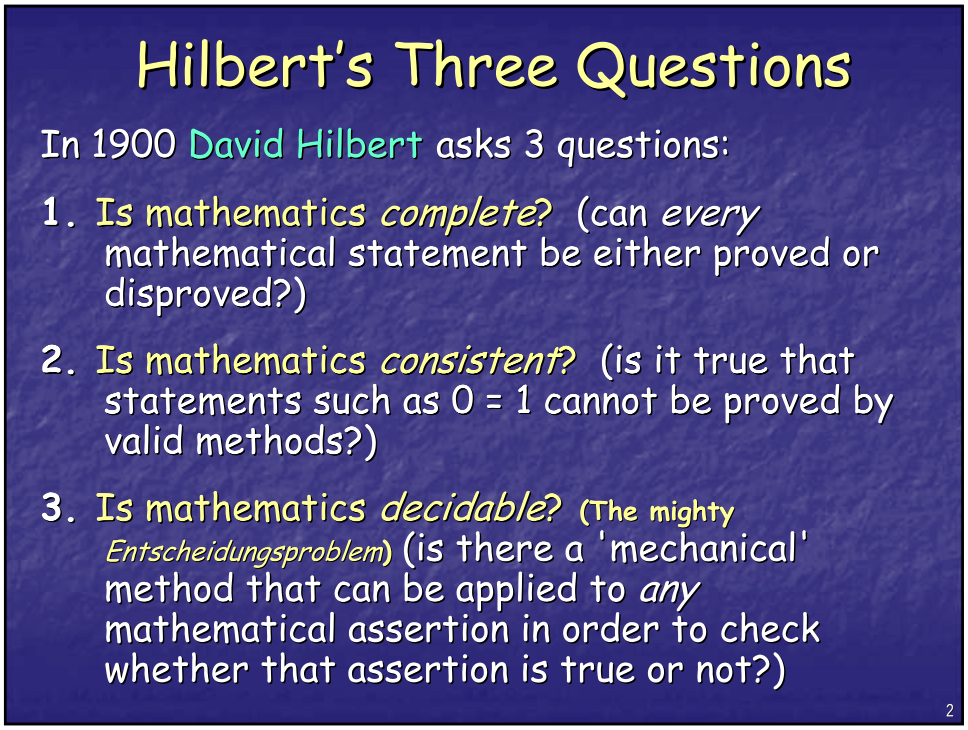


The Tale of NP-Completeness



1



Hilbert's Three Questions

In 1900 David Hilbert asks 3 questions:

1. Is mathematics *complete?* (can *every* mathematical statement be either proved or disproved?)
2. Is mathematics *consistent?* (is it true that statements such as 0 = 1 cannot be proved by valid methods?)
3. Is mathematics *decidable?* (The mighty *Entscheidungsproblem*) (is there a 'mechanical' method that can be applied to *any* mathematical assertion in order to check whether that assertion is true or not?)

2

Hilbert's Three NOs

Hilbert gets 3 NOs:

1931 – Kurt Gödel introduces the incompleteness theorems

1936 – Alonzo Church proves that there is no algorithm which decides for two given lambda calculus expressions whether they are equivalent.

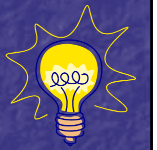
1936 – Alan Turing "builds" a-machines (a.k.a Turing Machines) and reduces the halting problem for TMs to the Entscheidungsproblem.

TMs became a platform for general computations.

3

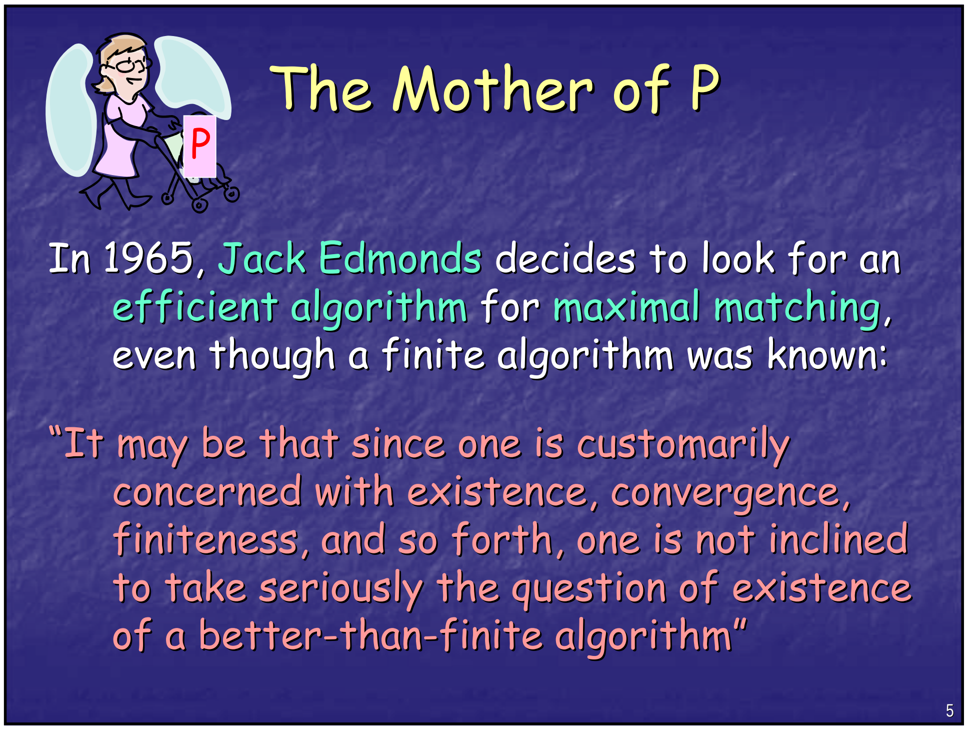
The Birth of Computational Complexity

The idea of "efficient algorithms" can be traced back to the ancient Greeks.



But our story starts in the 1960's when Juris Hartmanis and Richard E. Stearns, laid out the definition of quantified time and space complexity, and thus the computational complexity was born

4

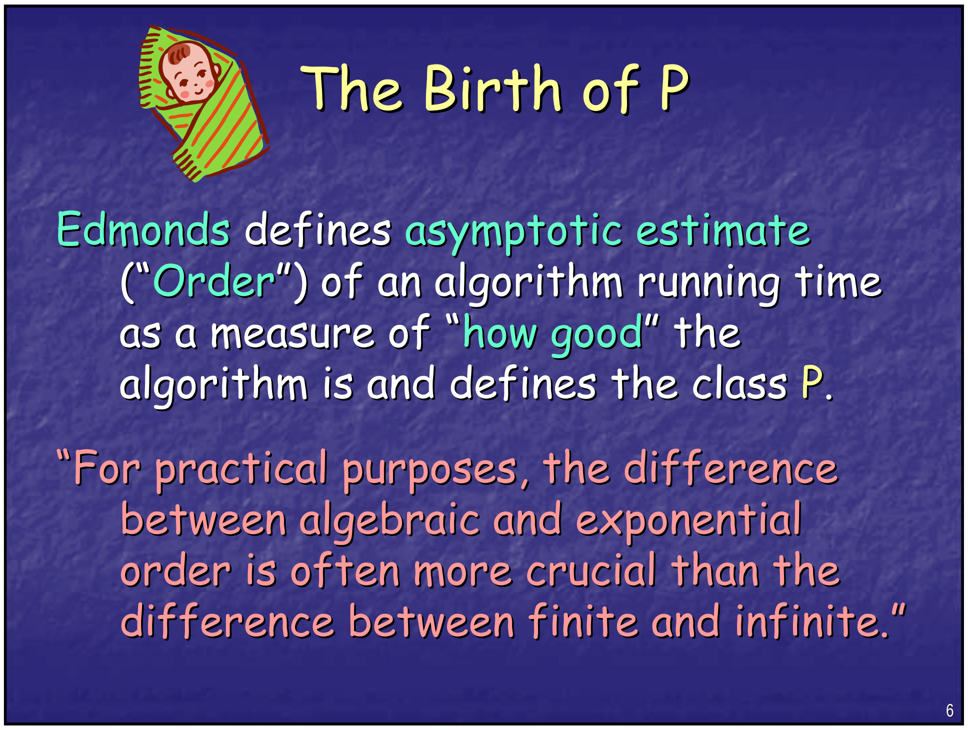


The Mother of P

In 1965, Jack Edmonds decides to look for an efficient algorithm for maximal matching, even though a finite algorithm was known:

"It may be that since one is customarily concerned with existence, convergence, finiteness, and so forth, one is not inclined to take seriously the question of existence of a better-than-finite algorithm"

5

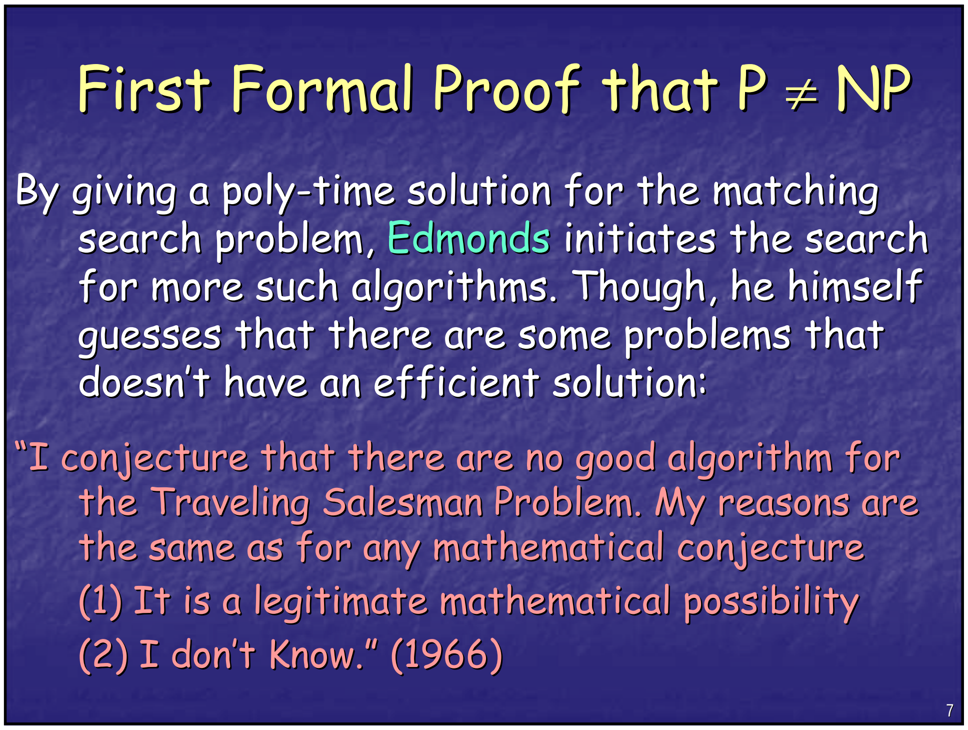


The Birth of P

Edmonds defines asymptotic estimate ("Order") of an algorithm running time as a measure of "how good" the algorithm is and defines the class P.

"For practical purposes, the difference between algebraic and exponential order is often more crucial than the difference between finite and infinite."

6



First Formal Proof that P $\neq$ NP

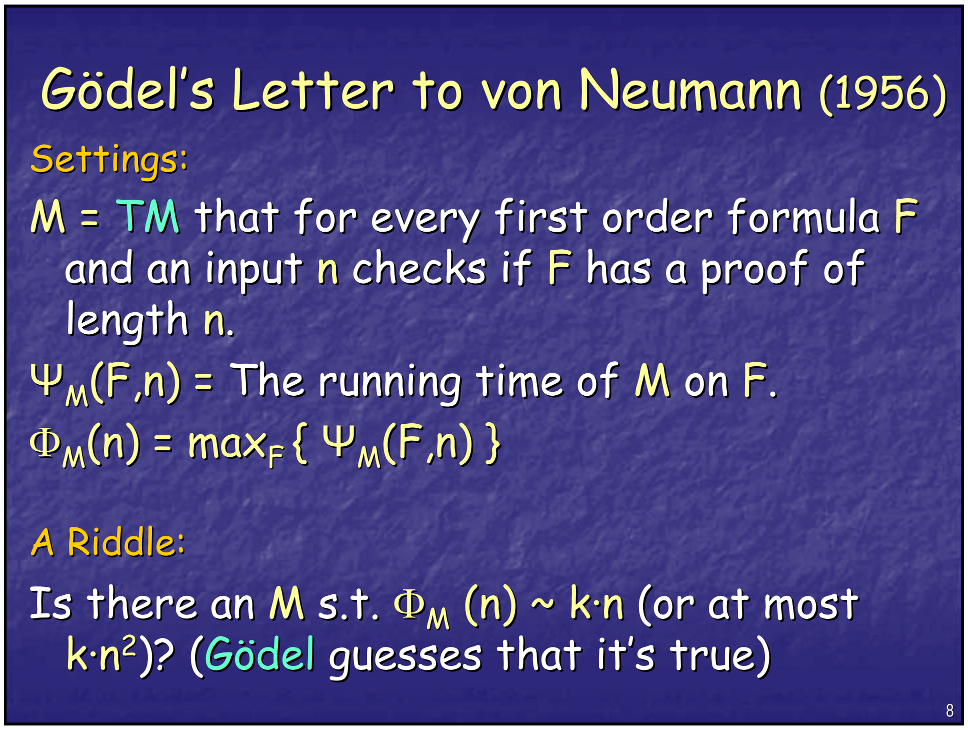
By giving a poly-time solution for the matching search problem, Edmonds initiates the search for more such algorithms. Though, he himself guesses that there are some problems that doesn't have an efficient solution:

"I conjecture that there are no good algorithm for the Traveling Salesman Problem. My reasons are the same as for any mathematical conjecture

(1) It is a legitimate mathematical possibility

(2) I don't Know." (1966)

7



Gödel's Letter to von Neumann (1956)

Settings:

M = TM that for every first order formula F and an input n checks if F has a proof of length n.

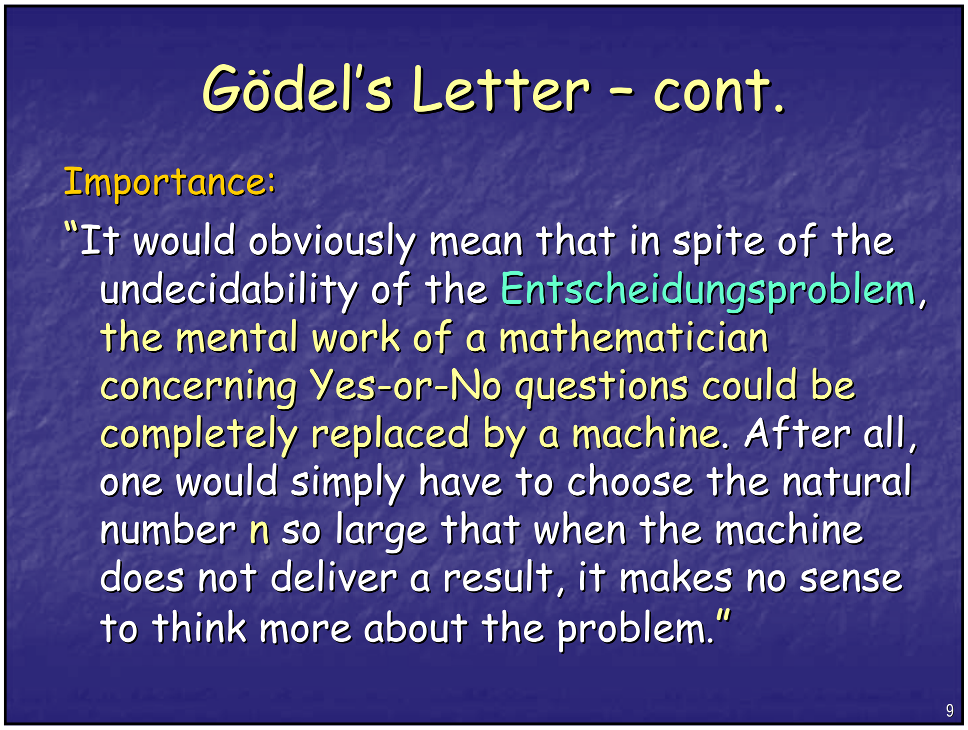
$\Psi_M(F,n)$ = The running time of M on F.

$\Phi_M(n) = \max_F \{ \Psi_M(F,n) \}$

A Riddle:

Is there an M s.t. $\Phi_M(n) \sim k \cdot n$ (or at most $k \cdot n^2$)? (Gödel guesses that it's true)

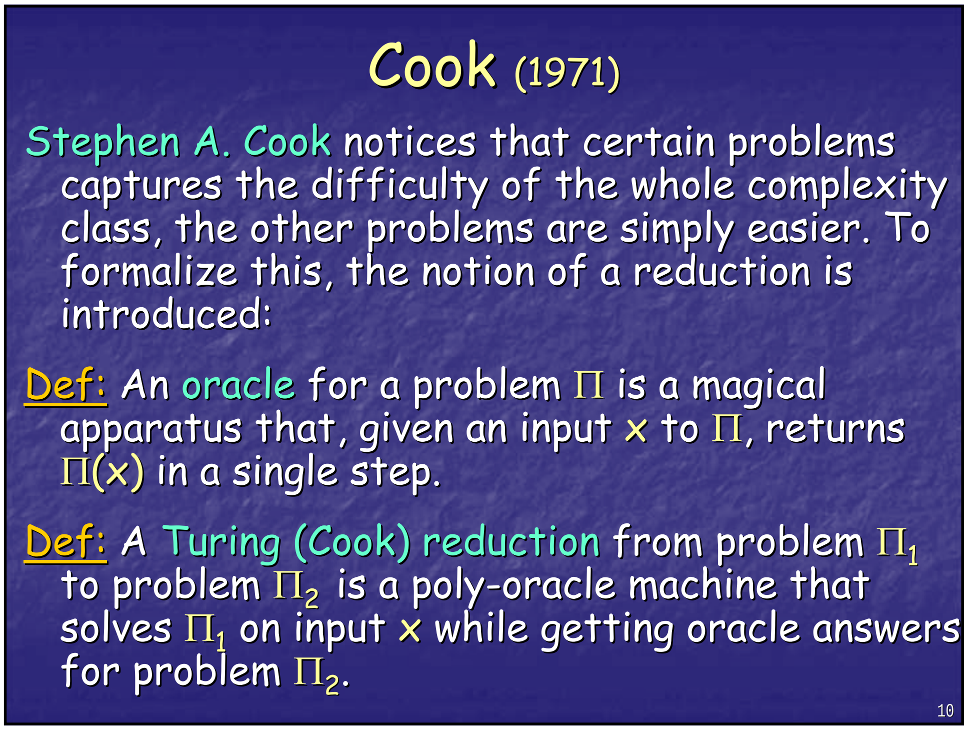
8



Gödel's Letter – cont.

Importance:

"It would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem."

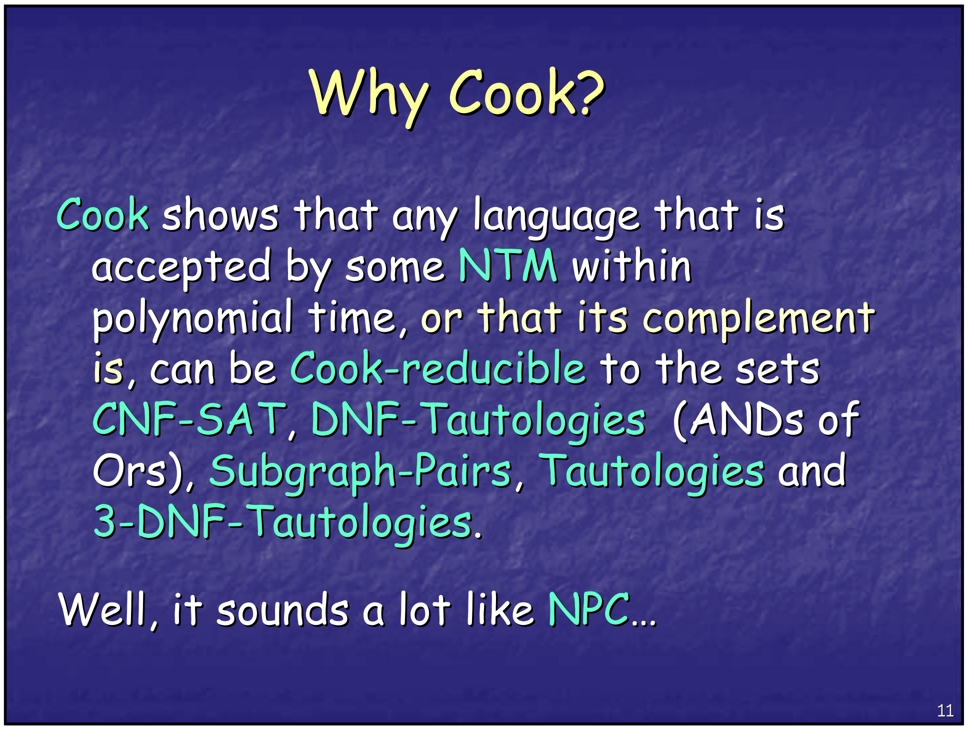


Cook (1971)

Stephen A. Cook notices that certain problems captures the difficulty of the whole complexity class, the other problems are simply easier. To formalize this, the notion of a reduction is introduced:

Def: An oracle for a problem Π is a magical apparatus that, given an input x to Π, returns $\Pi(x)$ in a single step.

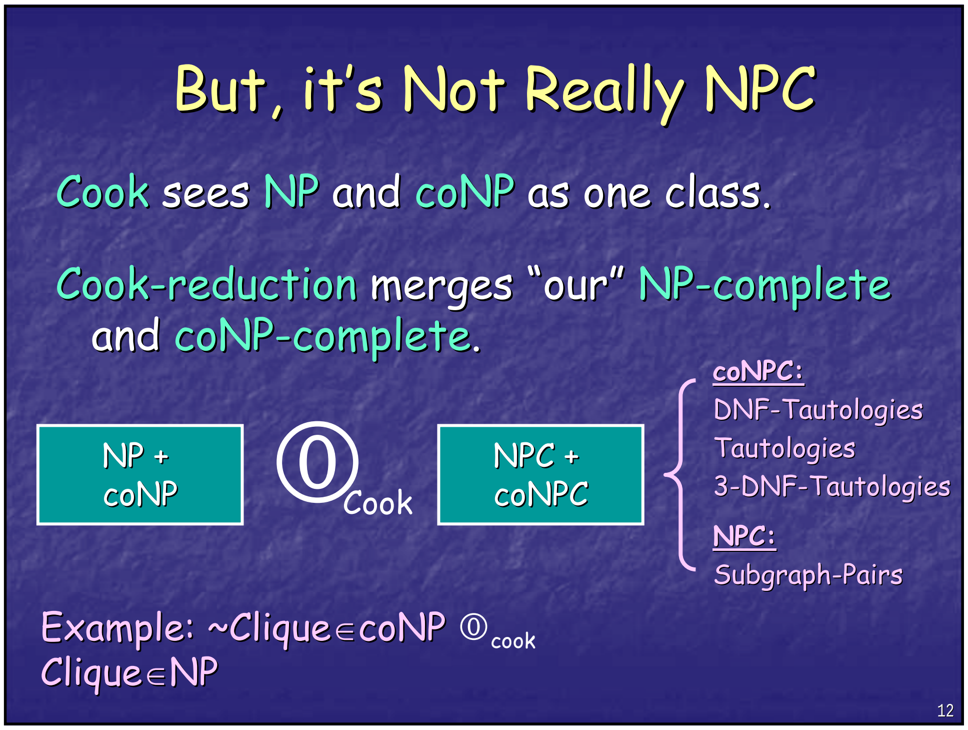
Def: A Turing (Cook) reduction from problem Π_1 to problem Π_2 is a poly-oracle machine that solves Π_1 on input x while getting oracle answers for problem Π_2.



Why Cook?

Cook shows that any language that is accepted by some NTM within polynomial time, or that its complement is, can be Cook-reducible to the sets CNF-SAT, DNF-Tautologies (ANDs of Ors), Subgraph-Pairs, Tautologies and 3-DNF-Tautologies.

Well, it sounds a lot like NPC…



But, it's Not Really NPC

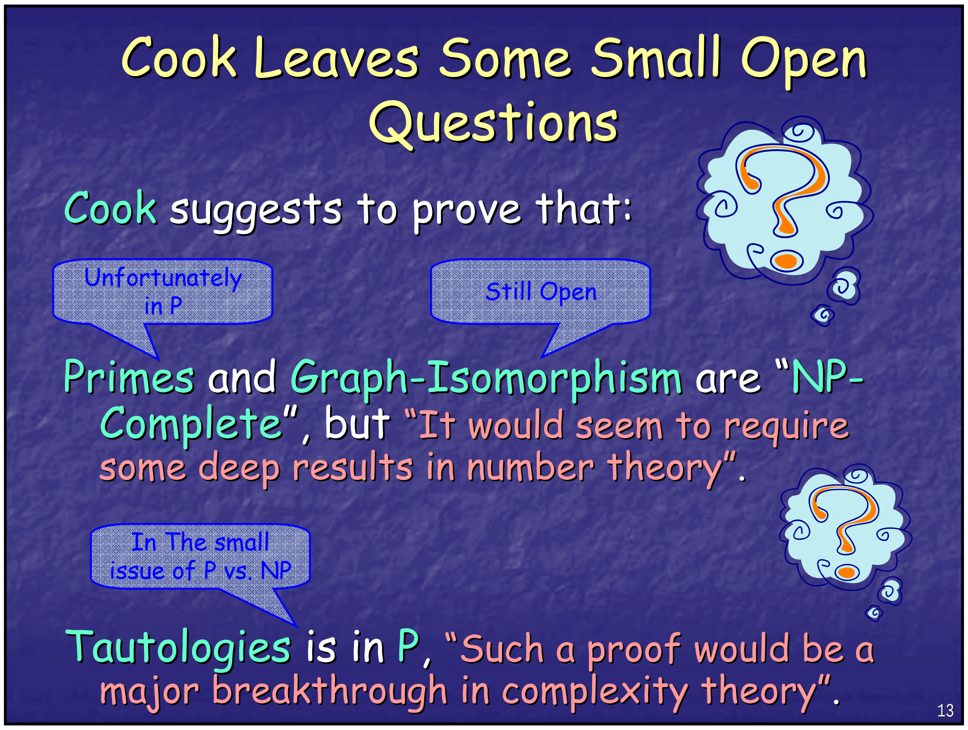
Cook sees NP and coNP as one class.

Cook-reduction merges "our" NP-complete and coNP-complete.

$\boxed{\text{NP + coNP}}$ ⓪$_{\text{Cook}}$ $\boxed{\text{NPC + coNPC}}$

coNPC:
DNF-Tautologies
Tautologies
3-DNF-Tautologies
NPC:
Subgraph-Pairs

Example: ~Clique∈coNP ⓪$_{\text{cook}}$ Clique∈NP



Cook Leaves Some Small Open Questions

Cook suggests to prove that:

Unfortunately in P

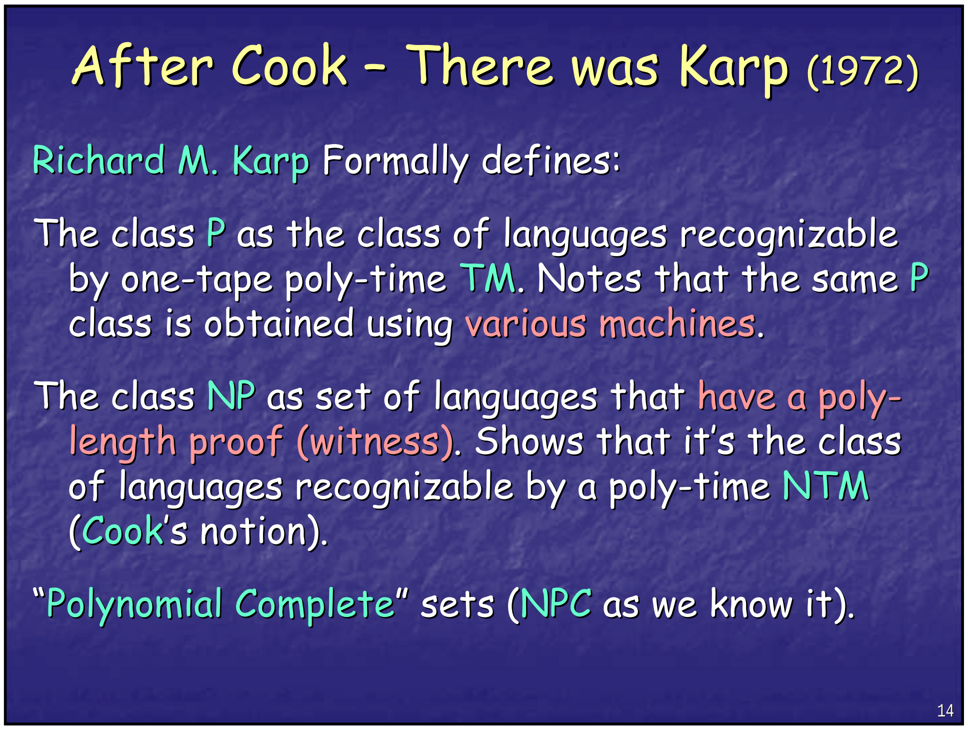
Still Open

Primes and Graph-Isomorphism are "NP-Complete", but "It would seem to require some deep results in number theory".

In The small issue of P vs. NP

Tautologies is in P, "Such a proof would be a major breakthrough in complexity theory".

13



After Cook – There was Karp (1972)

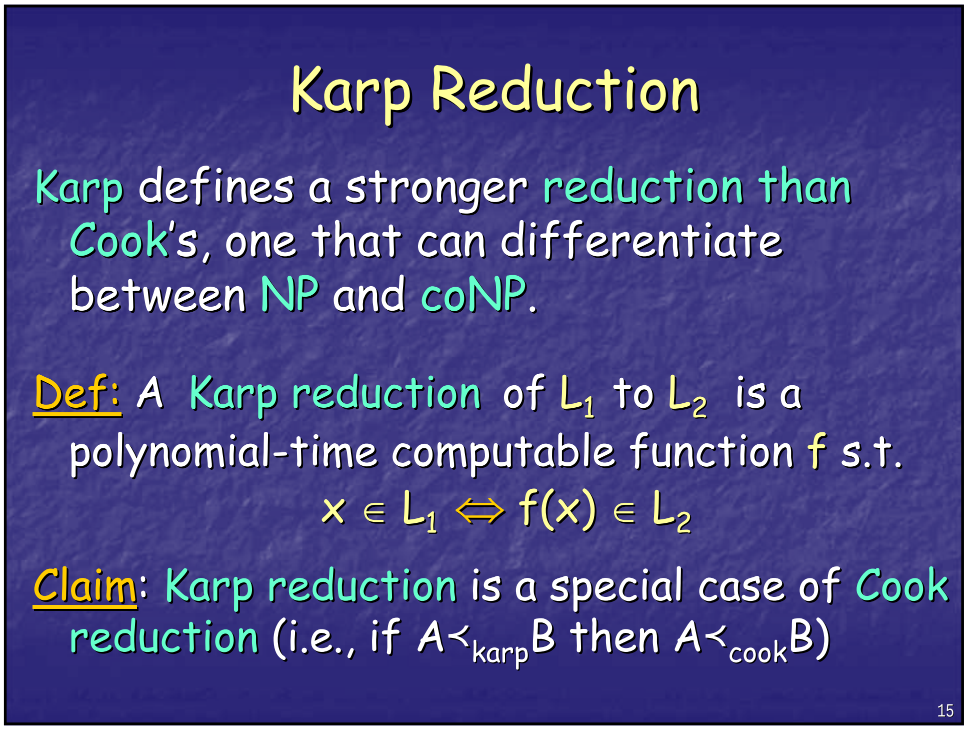
Richard M. Karp Formally defines:

The class P as the class of languages recognizable by one-tape poly-time TM. Notes that the same P class is obtained using various machines.

The class NP as set of languages that have a poly-length proof (witness). Shows that it's the class of languages recognizable by a poly-time NTM (Cook's notion).

"Polynomial Complete" sets (NPC as we know it).

14



Karp Reduction

Karp defines a stronger reduction than Cook's, one that can differentiate between NP and coNP.

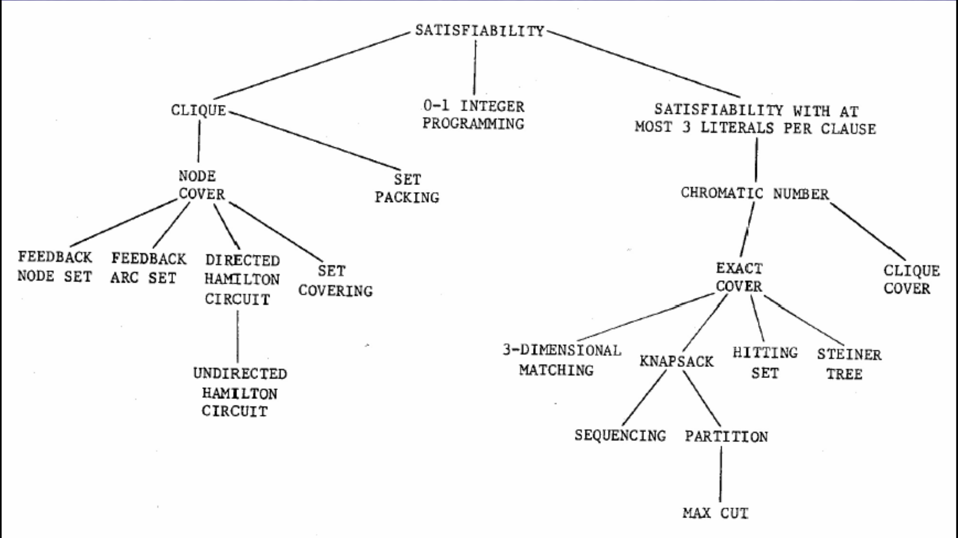
Def: A Karp reduction of L_1 to L_2 is a polynomial-time computable function f s.t.

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

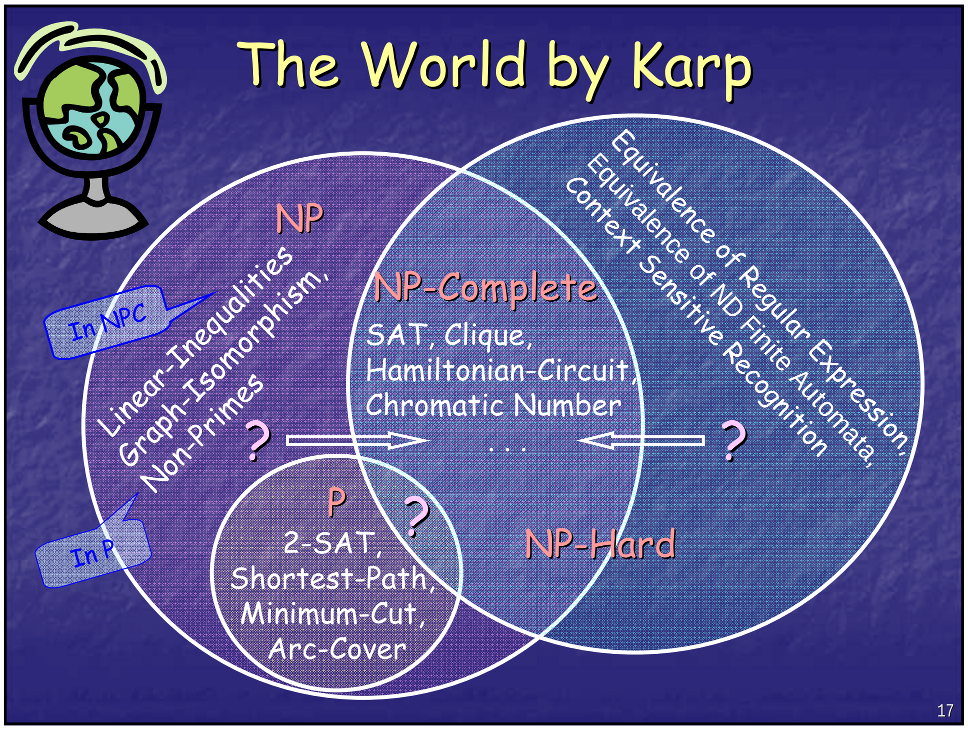
Claim: Karp reduction is a special case of Cook reduction (i.e., if $A <_{karp} B$ then $A <_{cook} B$)

15

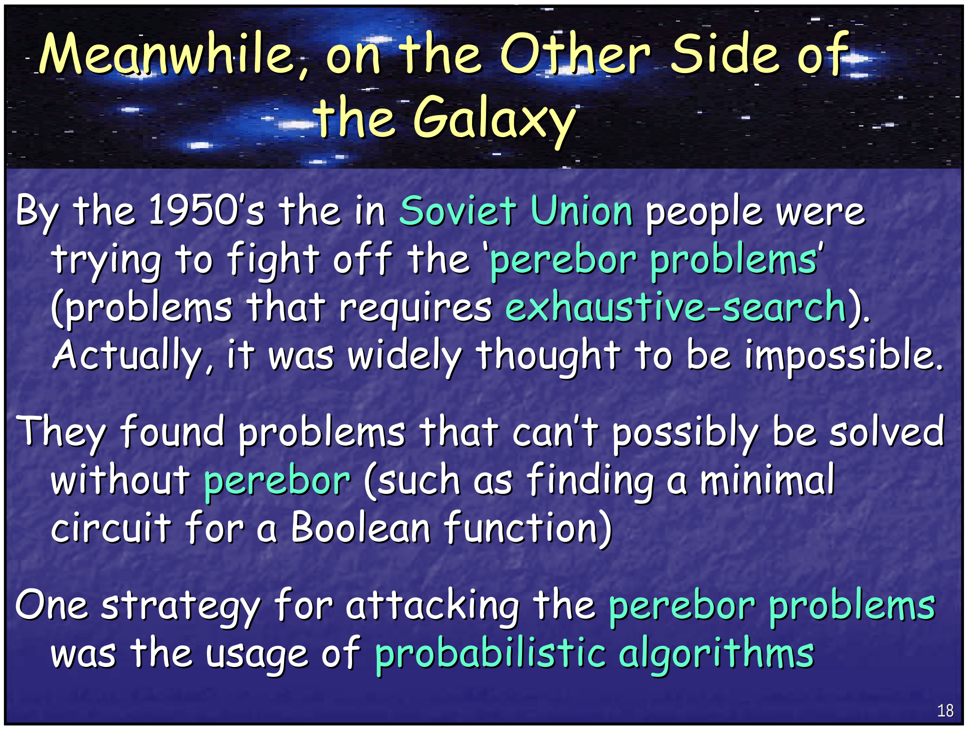
Karp's Complete Problems



16



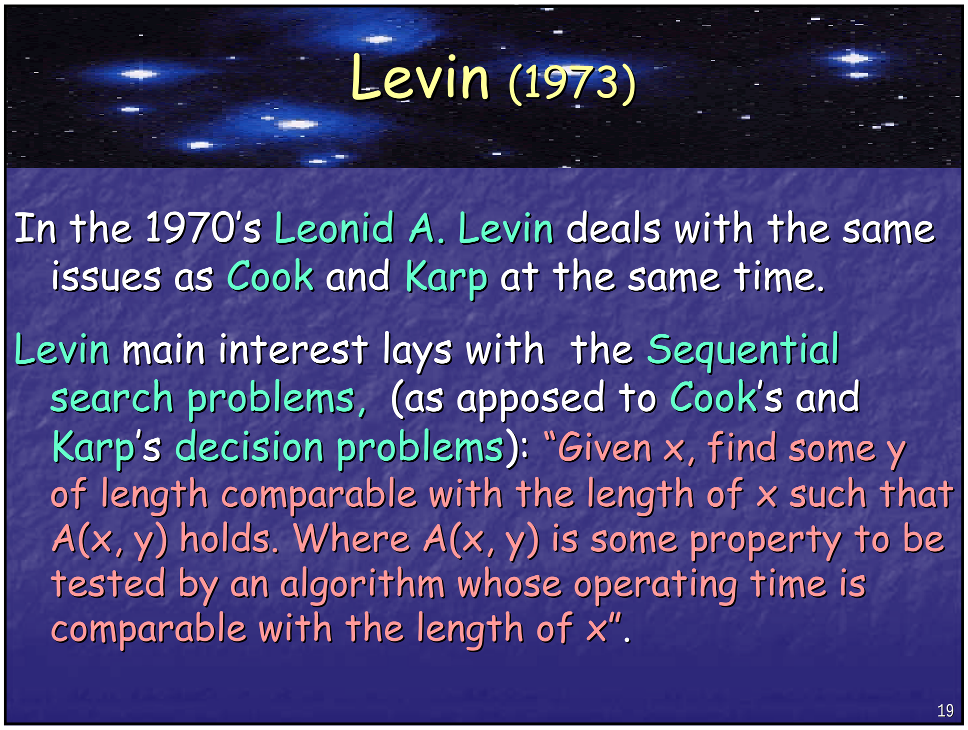
The World by Karp



Meanwhile, on the Other Side of the Galaxy

- By the 1950's the in Soviet Union people were trying to fight off the 'perebor problems' (problems that requires exhaustive-search). Actually, it was widely thought to be impossible.
- They found problems that can't possibly be solved without perebor (such as finding a minimal circuit for a Boolean function)
- One strategy for attacking the perebor problems was the usage of probabilistic algorithms

18

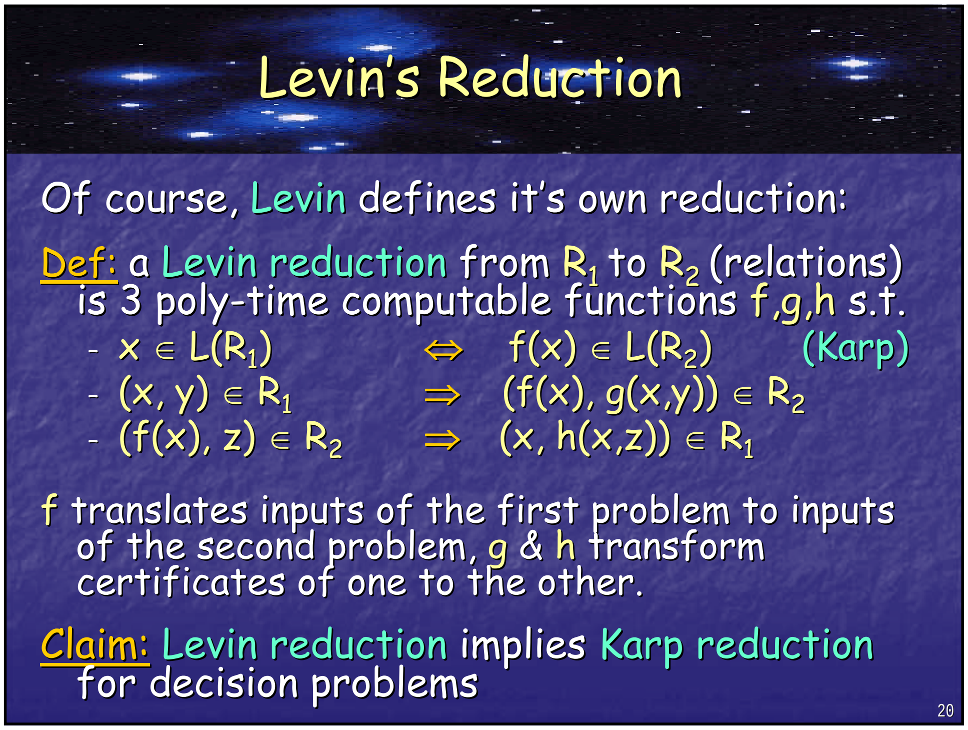


Levin (1973)

In the 1970's Leonid A. Levin deals with the same issues as Cook and Karp at the same time.

Levin main interest lays with the Sequential search problems, (as apposed to Cook's and Karp's decision problems): "Given x, find some y of length comparable with the length of x such that A(x, y) holds. Where A(x, y) is some property to be tested by an algorithm whose operating time is comparable with the length of x".

19



Levin's Reduction

Of course, Levin defines it's own reduction:

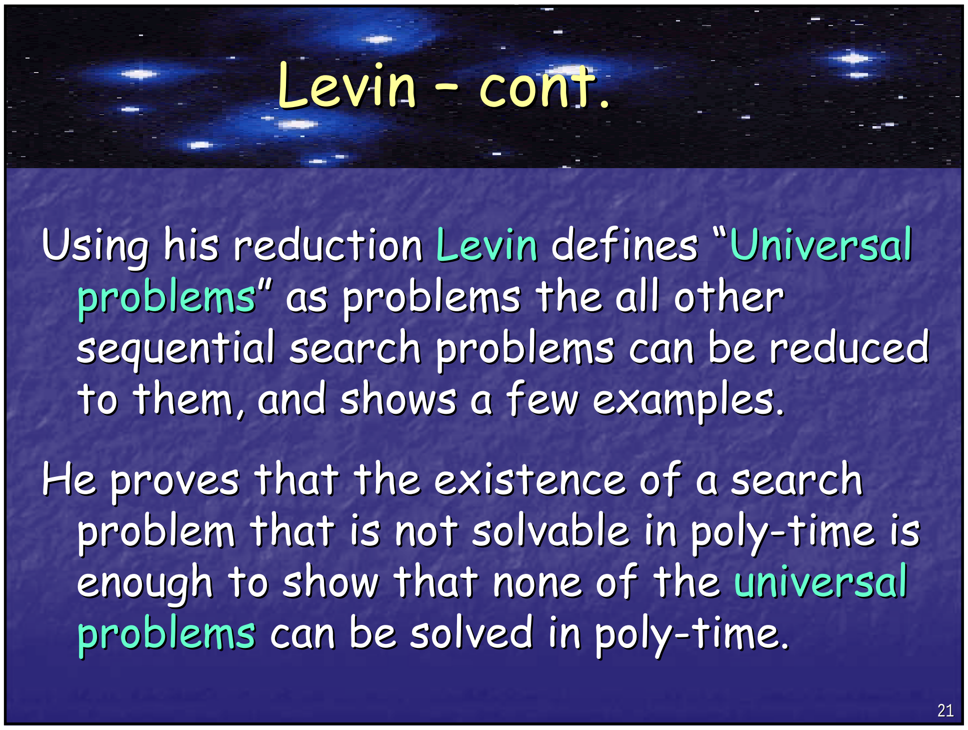
Def: a Levin reduction from R_1 to R_2 (relations) is 3 poly-time computable functions f,g,h s.t.

- $x \in L(R_1) \Leftrightarrow f(x) \in L(R_2)$ (Karp)
- $(x, y) \in R_1 \Rightarrow (f(x), g(x,y)) \in R_2$
- $(f(x), z) \in R_2 \Rightarrow (x, h(x,z)) \in R_1$

f translates inputs of the first problem to inputs of the second problem, g & h transform certificates of one to the other.

Claim: Levin reduction implies Karp reduction for decision problems

20

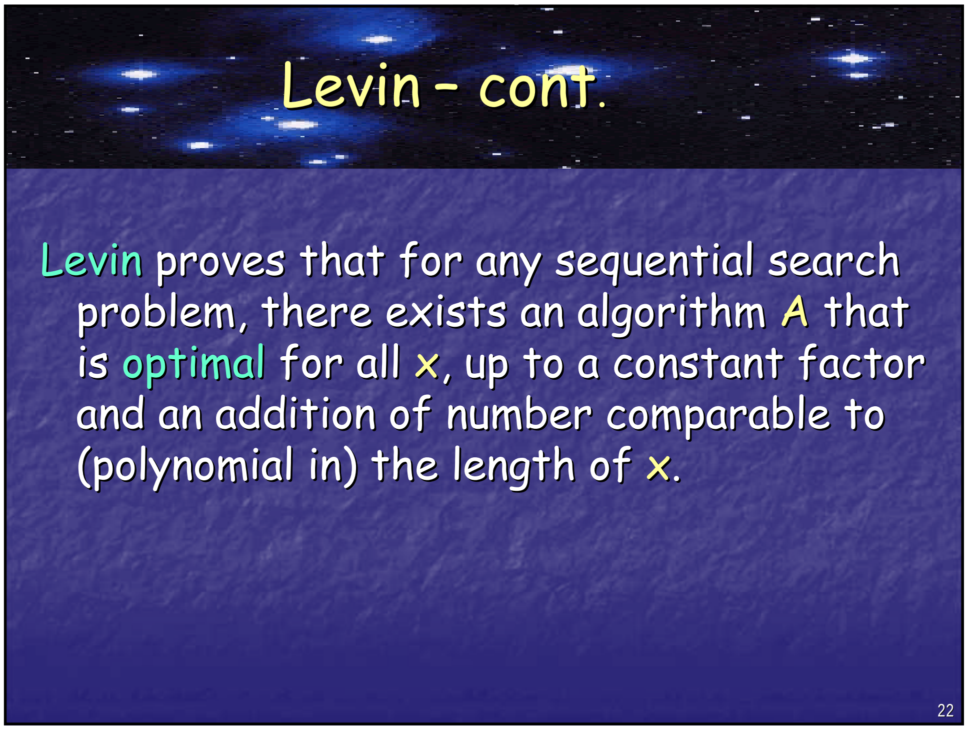


Levin – cont.

Using his reduction Levin defines "Universal problems" as problems the all other sequential search problems can be reduced to them, and shows a few examples.

He proves that the existence of a search problem that is not solvable in poly-time is enough to show that none of the universal problems can be solved in poly-time.

21



Levin – cont.

Levin proves that for any sequential search problem, there exists an algorithm A that is optimal for all x, up to a constant factor and an addition of number comparable to (polynomial in) the length of x.

22

Why is P vs. NP So Important?

Money - It's (literally) a million dollar question

Philophy –

- Is it really more difficult to find a proof than to check it?
- Is nature nondeterministic?

"Real Life" – A Classification tool for what's practical

Computational Hardness – Can be used by the force of good ($P \neq NP$ is crucial for the existence of one-way-functions)

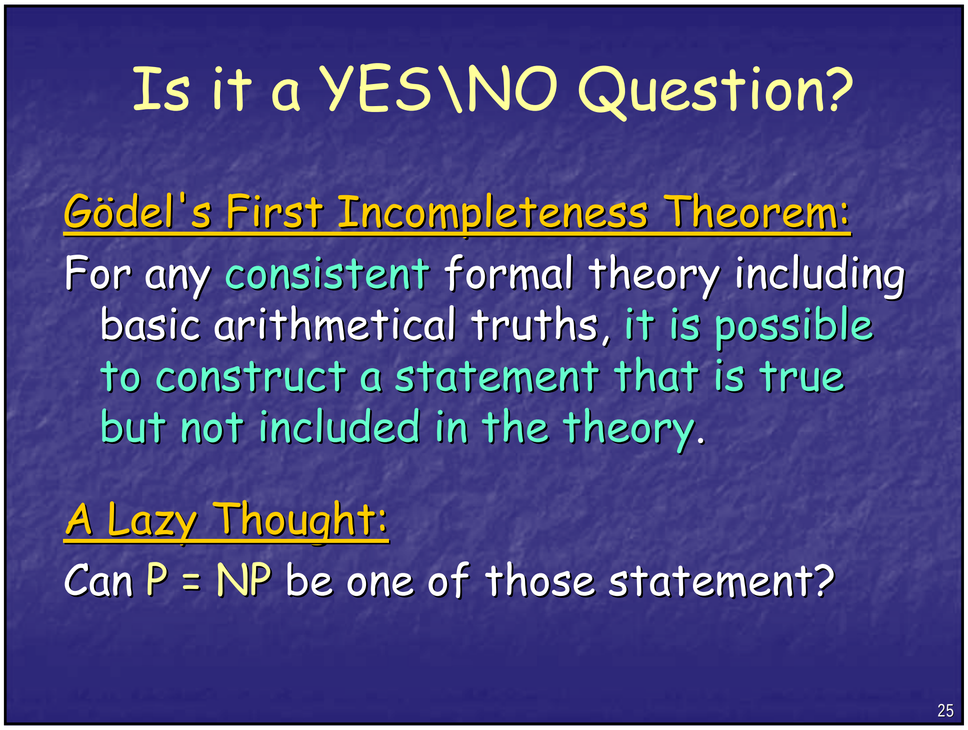
Is it Really THE Question?

Isn't it good enough to approximate \ get a probabilistic answer?

- Although some NPC problems are easy to approximate, other approximations are NPC.
- Depends who you ask…

Isn't it enough to take the average case (as apposed to worst case)?

- For example, existence of a one-way-function requires a problem that is hard on average.
- Depends who you ask…



Is it a YES\NO Question?

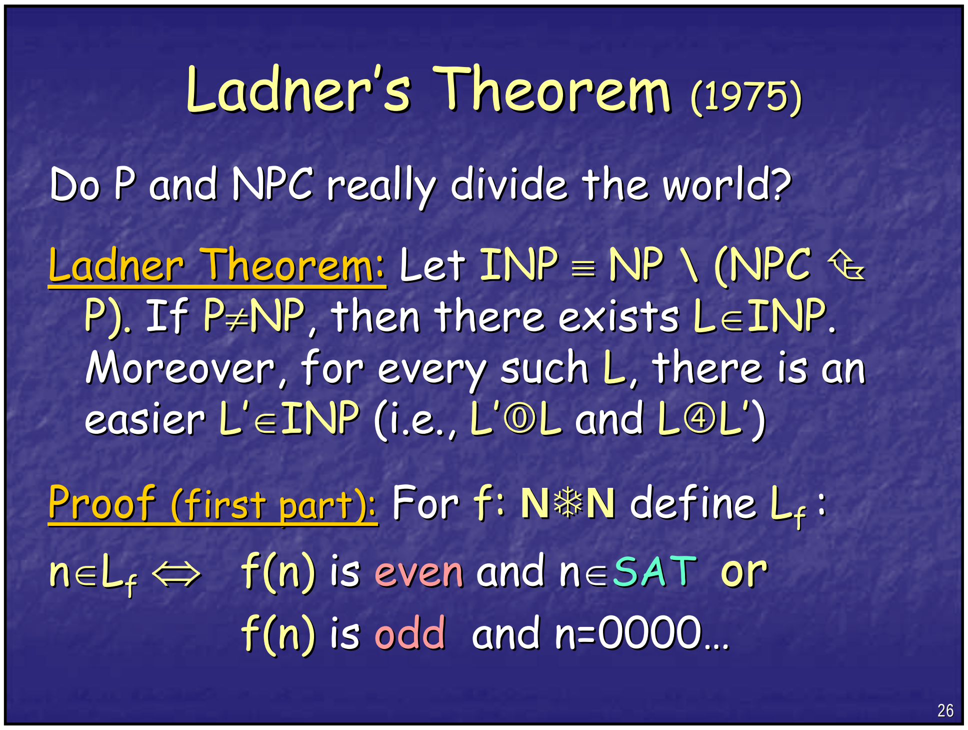
<u>Gödel's First Incompleteness Theorem:</u>

For any consistent formal theory including basic arithmetical truths, it is possible to construct a statement that is true but not included in the theory.

<u>A Lazy Thought:</u>

Can P = NP be one of those statement?

25



Ladner's Theorem (1975)

Do P and NPC really divide the world?

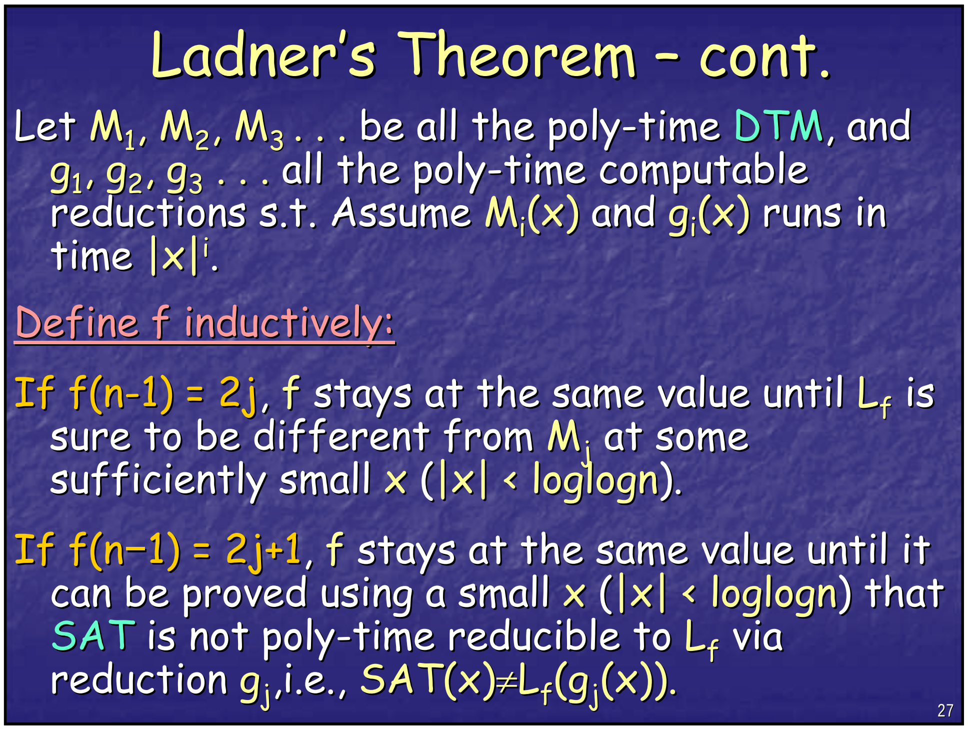
<u>Ladner Theorem:</u> Let INP ≡ NP \ (NPC ⮸ P). If P≠NP, then there exists L∈INP. Moreover, for every such L, there is an easier L'∈INP (i.e., L'⓪L and L④L')

<u>Proof (first part):</u> For f: **N❄N** define L_f :

$n \in L_f \Leftrightarrow$ f(n) is even and n∈SAT or

f(n) is odd and n=0000…

26



Ladner's Theorem – cont.

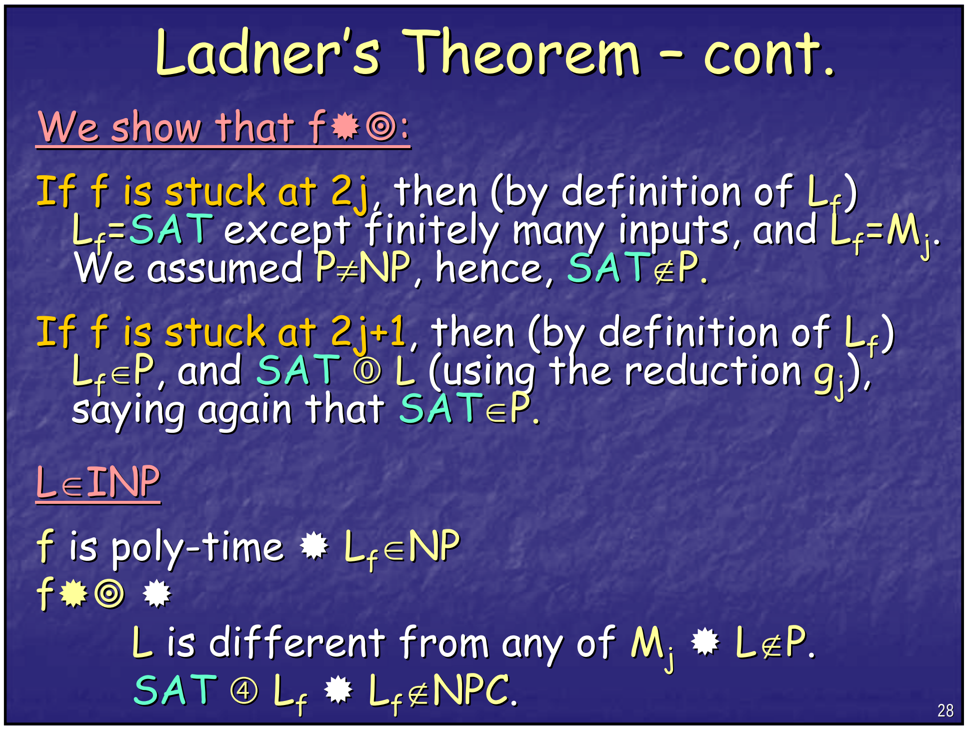
Let M_1, M_2, M_3 . . . be all the poly-time DTM, and g_1, g_2, g_3 . . . all the poly-time computable reductions s.t. Assume $M_i(x)$ and $g_i(x)$ runs in time $|x|^i$.

Define f inductively:

If $f(n-1) = 2j$, f stays at the same value until L_f is sure to be different from M_j at some sufficiently small x ($|x| < \log\log n$).

If $f(n-1) = 2j+1$, f stays at the same value until it can be proved using a small x ($|x| < \log\log n$) that SAT is not poly-time reducible to L_f via reduction g_j,i.e., $SAT(x) \neq L_f(g_j(x))$.

27



Ladner's Theorem – cont.

We show that f✸◎:

If f is stuck at 2j, then (by definition of L_f) L_f=SAT except finitely many inputs, and $L_f=M_j$. We assumed P≠NP, hence, SAT∉P.

If f is stuck at 2j+1, then (by definition of L_f) $L_f \in P$, and SAT ⓪ L (using the reduction g_j), saying again that SAT∈P.

L∈INP

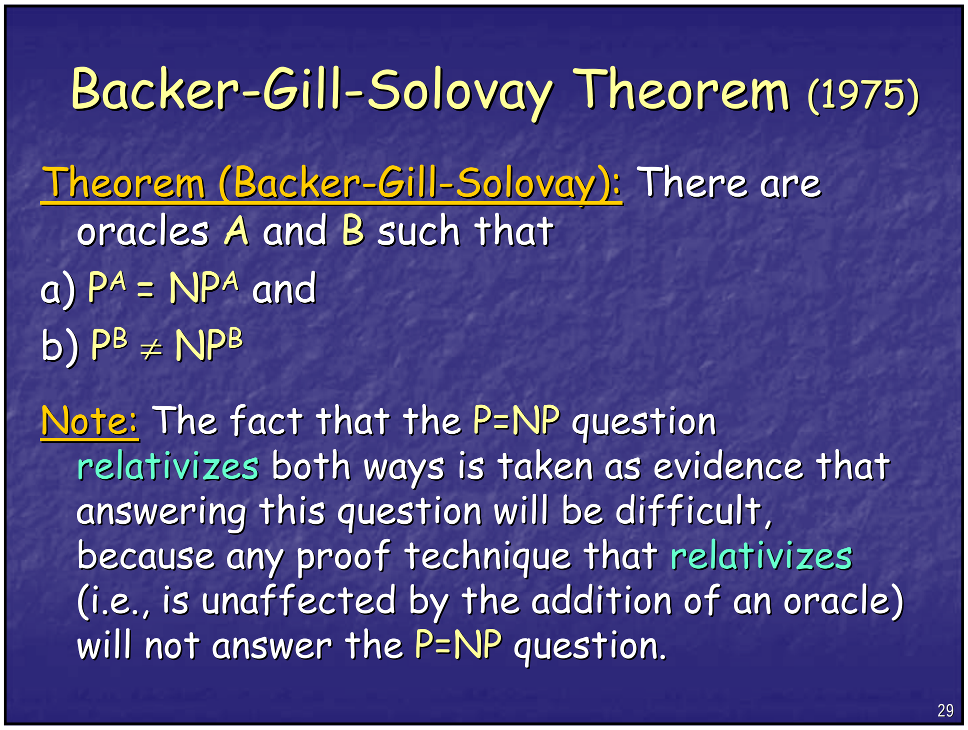
f is poly-time ✸ $L_f \in NP$

f✸◎ ✸

L is different from any of M_j ✸ L∉P.

SAT ④ L_f ✸ $L_f \notin NPC$.

28



Backer-Gill-Solovay Theorem (1975)

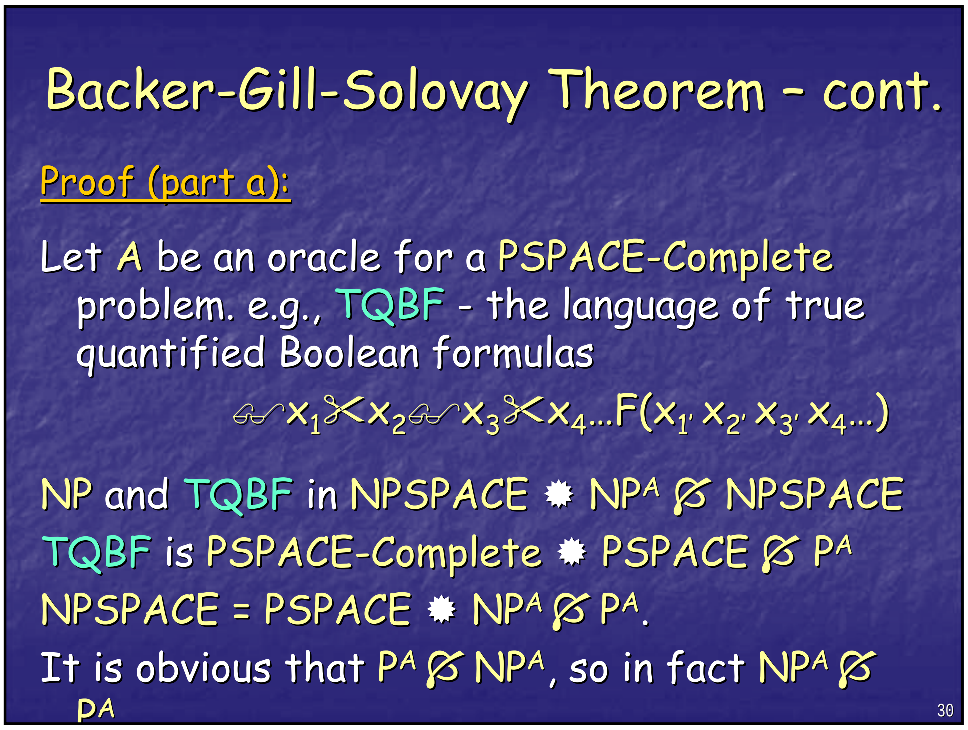
Theorem (Backer-Gill-Solovay): There are oracles A and B such that

a) $P^A = NP^A$ and

b) $P^B \neq NP^B$

Note: The fact that the P=NP question relativizes both ways is taken as evidence that answering this question will be difficult, because any proof technique that relativizes (i.e., is unaffected by the addition of an oracle) will not answer the P=NP question.

29



Backer-Gill-Solovay Theorem – cont.

Proof (part a):

Let A be an oracle for a PSPACE-Complete problem. e.g., TQBF - the language of true quantified Boolean formulas

$$\forall x_1 \exists x_2 \forall x_3 \exists x_4 \ldots F(x_1, x_2, x_3, x_4 \ldots)$$

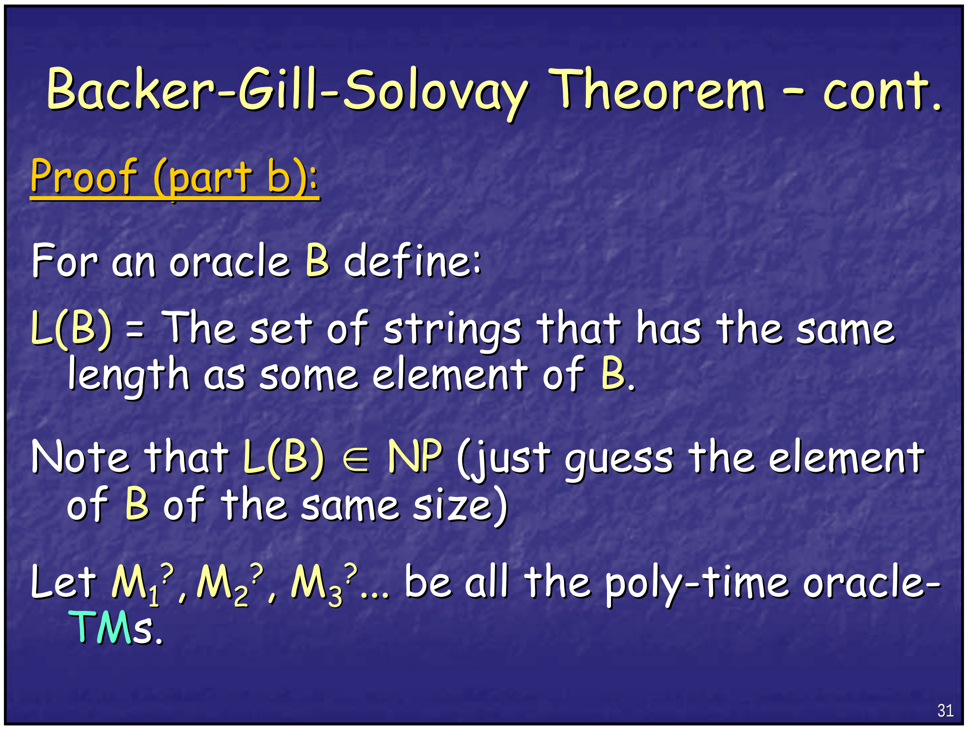
NP and TQBF in NPSPACE $\Rightarrow$ $NP^A \subseteq$ NPSPACE

TQBF is PSPACE-Complete $\Rightarrow$ PSPACE $\subseteq P^A$

NPSPACE = PSPACE $\Rightarrow$ $NP^A \subseteq P^A$.

It is obvious that $P^A \subseteq NP^A$, so in fact $NP^A \subseteq P^A$

30



Backer-Gill-Solovay Theorem – cont.

Proof (part b):

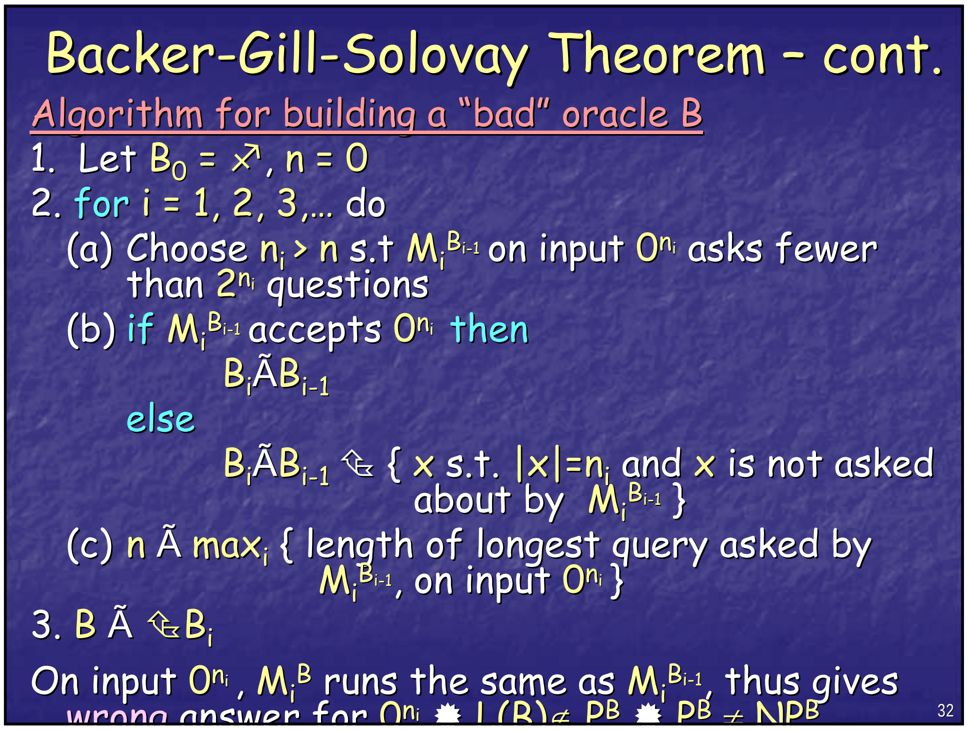
For an oracle B define:

L(B) = The set of strings that has the same length as some element of B.

Note that $L(B) \in NP$ (just guess the element of B of the same size)

Let $M_1^?, M_2^?, M_3^?$... be all the poly-time oracle-TMs.

31



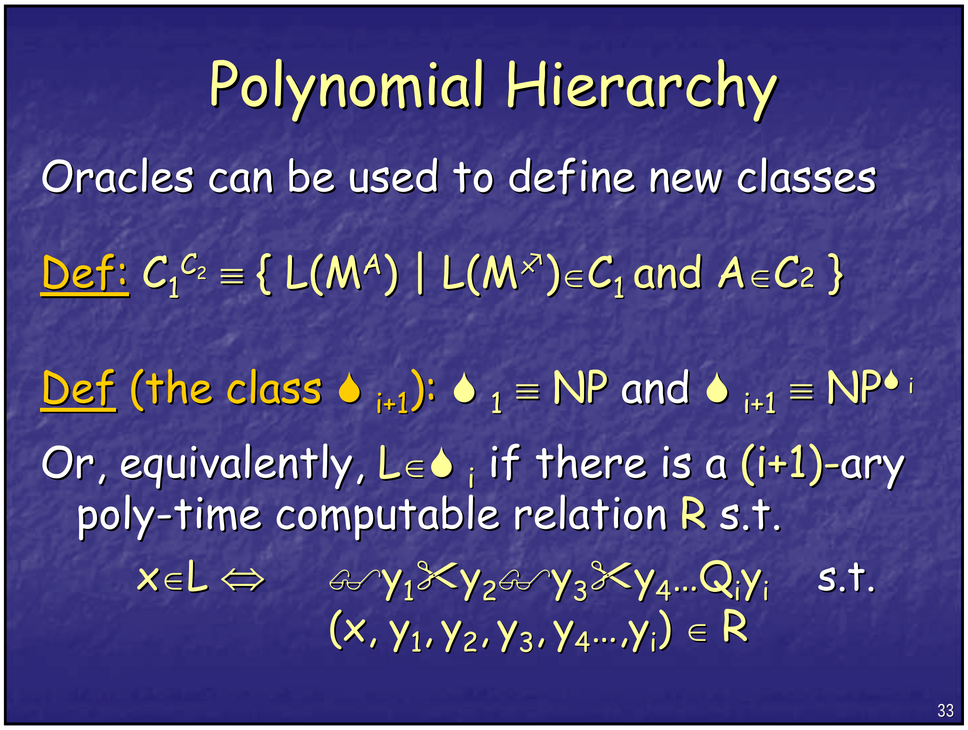
Backer-Gill-Solovay Theorem – cont.

Algorithm for building a “bad” oracle B

```
1. Let B_0 = ↗, n = 0
2. for i = 1, 2, 3,… do
   (a) Choose n_i > n s.t M_i^{B_{i-1}} on input 0^{n_i} asks fewer
       than 2^{n_i} questions
   (b) if M_i^{B_{i-1}} accepts 0^{n_i} then
              B_i Ã B_{i-1}
       else
              B_i Ã B_{i-1} ⇐ { x s.t. |x|=n_i and x is not asked
                                 about by M_i^{B_{i-1}} }
   (c) n Ã max_i { length of longest query asked by
                        M_i^{B_{i-1}}, on input 0^{n_i} }
3. B Ã ⇐B_i
```

On input 0^{n_i}, M_i^B runs the same as $M_i^{B_{i-1}}$, thus gives wrong answer for 0^{n_i} ✹ $L(B) \notin P^B$ ✹ $P^B \neq NP^B$

32



Polynomial Hierarchy

Oracles can be used to define new classes

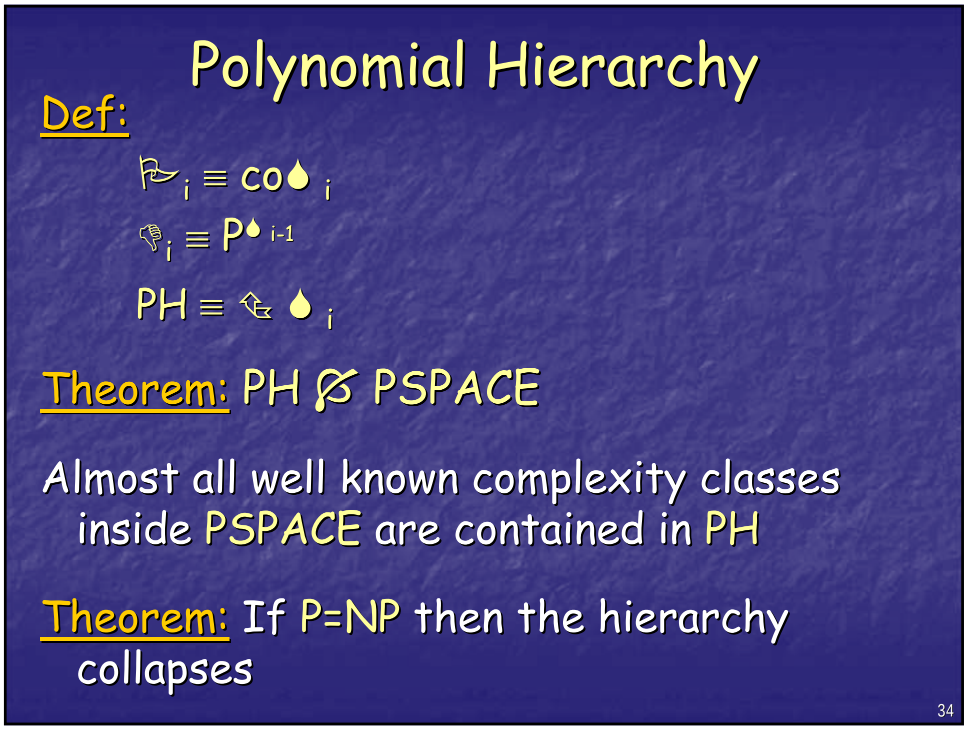
Def: $C_1^{C_2} \equiv \{ L(M^A) \mid L(M^{\text{⌖}}) \in C_1$ and $A \in C_2 \}$

Def (the class 💧 $_{i+1}$): 💧 $_1 \equiv$ NP and 💧 $_{i+1} \equiv$ NP$^{💧 i}$

Or, equivalently, L∈💧 $_i$ if there is a (i+1)-ary poly-time computable relation R s.t.

x∈L ⇔ 👓y_1✂y_2👓y_3✂y_4…$Q_i y_i$ s.t. $(x, y_1, y_2, y_3, y_4, \ldots, y_i) \in R$

33



Polynomial Hierarchy

Def:

🏳 $_i \equiv$ co💧 $_i$

👎 $_i \equiv$ P$^{💧 i-1}$

PH ≡ ⮴ 💧 $_i$

Theorem: PH ℘ PSPACE

Almost all well known complexity classes inside PSPACE are contained in PH

Theorem: If P=NP then the hierarchy collapses

34