“Deolalikar claimed that he had tamed the wildness of algorithms and shown that P indeed doesn’t equal NP. Within a few hours of his e-mail, the paper got an impressive endorsement: ‘This appears to be a relatively serious claim to have solved P versus NP,’ emailed Stephen Cook of the University of Toronto, the scientist who had initially formulated the question. That evening, a blogger posted Deolalikar’s paper. And the next day, long before researchers had had time to examine the 103-page paper in detail, the recommendation site Slashdot picked it up, sending a fire hose of tens of thousands of readers and dozens of journalists to the paper.”

\[
P = \bigcup_{k=1}^{\infty} \text{DTIME}[n^k]
\]

\(P\) is a good mathematical wrapper for "truly feasible".
NTIME[t(n)]: a mathematical fiction

input $w$

$|w| = n$
Many optimization problems we want to solve are NP complete.
Many optimization problems we want to solve are NP complete.
Many optimization problems we want to solve are NP complete.
Descriptive Complexity

Query
\[ q_1 \ q_2 \ \cdots \ q_n \]
\[ \mapsto \]
Computation
\[ a_1 \ a_2 \ \cdots \ a_i \ \cdots \ a_m \]

Neil Immerman

P versus NP: Approaches, Rebuttals, and Does It Matter?
Restrict attention to the complexity of computing individual bits of the output, i.e., **decision problems**.
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?

How rich a language do we need to express property $S$?
Restrict attention to the complexity of computing individual bits of the output, i.e., decision problems.

How hard is it to check if input has property $S$?

How rich a language do we need to express property $S$?

There is a constructive isomorphism between these two approaches.
Graph \[ G = (\{v_1, \ldots, v_n\}, E, s, t) \]

Binary \[ A_w = (\{p_1, \ldots, p_8\}, S) \]

String \[ S = \{p_2, p_5, p_7, p_8\} \]

\[ w = 01001011 \]

Vocabularies: \[ \tau_g = (E^2, s, t), \quad \tau_s = (S^1) \]
First-Order Logic

input symbols: from $\tau$
variables: $x, y, z, \ldots$
boolean connectives: $\land, \lor, \neg$
quantifiers: $\forall, \exists$
numeric symbols: $=, \leq, +, \times, \min, \max$

$\alpha \equiv \forall x \exists y (E(x, y)) \in \mathcal{L}(\tau_g)$

$\beta \equiv \exists x \forall y (x \leq y \land S(x)) \in \mathcal{L}(\tau_s)$

$\beta \equiv S(\min) \in \mathcal{L}(\tau_s)$
$$\Phi_{3\text{--color}} \equiv \exists R^1 \ G^1 \ B^1 \ \forall x \ y \ ((R(x) \lor G(x) \lor B(x)) \ \land \ \ (E(x, y) \rightarrow (\neg(R(x) \land R(y)) \land \neg(G(x) \land G(y)) \ \land \ \neg(B(x) \land B(y))))$$

\begin{tikzpicture}
    \node (s) at (0,0) {s};
    \node (b) at (1,1) {b};
    \node (d) at (2,2) {d};
    \node (t) at (3,1) {t};
    \node (a) at (0,-1) {a};
    \node (c) at (1,-1) {c};
    \node (f) at (3,-2) {f};
    \node (g) at (1,-2) {g};
    \node (e) at (2,-2) {e};

    \draw (s) -- (b);
    \draw (s) -- (a);
    \draw (s) -- (g);
    \draw (s) -- (d);
    \draw (b) -- (c);
    \draw (b) -- (d);
    \draw (b) -- (f);
    \draw (c) -- (e);
    \draw (d) -- (t);
    \draw (t) -- (f);
    \draw (f) -- (g);
    \draw (g) -- (c);

\end{tikzpicture}
Fagin’s Theorem: \( \text{NP} = \text{SO} \exists \)

\[ \Phi_{3\text{-color}} \equiv \exists R^1 G^1 B^1 \forall x y ((R(x) \lor G(x) \lor B(x)) \land (E(x, y) \rightarrow (\neg(R(x) \land R(y)) \land \neg(G(x) \land G(y)) \land \neg(B(x) \land B(y)))) \]
Recursive

EXPTIME

PSPACE

Polynomial-Time Hierarchy

Arithmetic Hierarchy

co–r.e. complete

co–r.e.

FON

r.e. complete

FO ∈ (N)

Recursive

EXPTIME

PSPACE

NP complete

co–NP complete

SO

NP ∩ co–NP

P
complete

"truly feasible"

P

NP

SO

NP

co–NP

SO

FO

FO(CFL)

FO(REGULAR)

FO
Addition is First-Order

\[ Q_+ : \text{STRUC}[\tau_{AB}] \rightarrow \text{STRUC}[\tau_s] \]

\[
\begin{array}{cccccccc}
A & & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
B & + & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
S & & s_1 & s_2 & \ldots & s_{n-1} & s_n \\
\end{array}
\]
Addition is First-Order

\[ Q_+ : \text{STRUC}[\tau_{AB}] \rightarrow \text{STRUC}[\tau_s] \]

\[
\begin{array}{c}
A \\
B \\
S \\
\hline
a_1 & a_2 & \cdots & a_{n-1} & a_n \\
b_1 & b_2 & \cdots & b_{n-1} & b_n \\
s_1 & s_2 & \cdots & s_{n-1} & s_n \\
\end{array}
\]

\[ C(i) \equiv (\exists j > i) \left( (A(j) \land B(j)) \land \left( (\forall k. j > k > i) (A(k) \lor B(k)) \right) \right) \]
Addition is First-Order

$Q_+ : \text{STRUC}[\tau_{AB}] \rightarrow \text{STRUC}[\tau_s]$

\[
\begin{array}{c}
A \quad a_1 \quad a_2 \quad \ldots \quad a_{n-1} \quad a_n \\
B \quad + \quad b_1 \quad b_2 \quad \ldots \quad b_{n-1} \quad b_n \\
S \quad \quad \quad s_1 \quad s_2 \quad \ldots \quad s_{n-1} \quad s_n \\
\end{array}
\]

\[
C(i) \equiv (\exists j > i) \left( (A(j) \land B(j)) \land (\forall k. j > k > i)(A(k) \lor B(k)) \right)
\]

\[
Q_+(i) \equiv A(i) \oplus B(i) \oplus C(i)
\]
Parallel Machines:

\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]
Parallel Machines: Quantifiers are Parallel

$$\text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}]$$

Assume array $$A[x]: x = 1, \ldots, r$$ in memory.
Parallel Machines: Quantifiers are Parallel

\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]

Assume array \( A[x] : x = 1, \ldots, r \) in memory.

\[ \forall x(A(x)) \equiv \text{write}(1); \text{proc } p_i : \text{if } (A[i] = 0) \text{ then } \text{write}(0) \]
\[ \text{FO} = \text{CRAM}[1] = \text{AC}^0 = \text{Logarithmic-Time Hierarchy} \]
Inductive Definitions

\[ E^*(x, y) \equiv x = y \lor E(x, y) \lor \exists z (E^*(x, z) \land E^*(z, y)) \]
Inductive Definitions

\[ E^*(x, y) \equiv x = y \lor E(x, y) \lor \exists z (E^*(x, z) \land E^*(z, y)) \]

\[ \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z (R(x, z) \land R(z, y)) \]
Inductive Definitions

\[ E^*(x, y) \equiv x = y \lor E(x, y) \lor \exists z(E^*(x, z) \land E^*(z, y)) \]

\[ \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z(R(x, z) \land R(z, y)) \]

\[ G \in \text{REACH} \iff G \models (\text{LFP}\varphi_{tc})(s, t) \]
Inductive Definitions

\[ E^*(x, y) \equiv x = y \lor E(x, y) \lor \exists z (E^*(x, z) \land E^*(z, y)) \]

\[ \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z (R(x, z) \land R(z, y)) \]

\[ G \in \text{REACH} \iff G \models (\text{LFP} \varphi_{tc})(s, t) \]

Thus, \( \text{REACH} \in \text{IND}[\log n] \).
Inductive Definitions

\[ E^*(x, y) \equiv x = y \lor E(x, y) \lor \exists z (E^*(x, z) \land E^*(z, y)) \]

\[ \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z (R(x, z) \land R(z, y)) \]

\[ G \in \text{REACH} \iff G \models (\text{LFP} \varphi_{tc})(s, t) \]

Thus, \( \text{REACH} \in \text{IND}[\log n] \).

Next, we’ll show that \( \text{REACH} \in \text{FO}[\log n] \).
1. Dummy universal quantification for base case:

\[ \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z (R(x, z) \land R(z, y)) \]

\[ \varphi_{tc}(R, x, y) \equiv (\forall z. M_1)(\exists z)(R(x, z) \land R(z, y)) \]

\[ M_1 \equiv \neg(x = y \lor E(x, y)) \]
\( \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z \ (R(x, z) \land R(z, y)) \)

1. Dummy universal quantification for base case:

\[
\varphi_{tc}(R, x, y) \equiv (\forall z. M_1)(\exists z)(R(x, z) \land R(z, y))
\]

\[
M_1 \equiv \neg(x = y \lor E(x, y))
\]

2. Using \( \forall \), replace two occurrences of \( R \) with one:

\[
\varphi_{tc}(R, x, y) \equiv (\forall z. M_1)(\exists z)(\forall uv. M_2)R(u, v)
\]

\[
M_2 \equiv (u = x \land v = z) \lor (u = z \land v = y)
\]
\( \varphi_{tc}(R, x, y) \equiv x = y \lor E(x, y) \lor \exists z (R(x, z) \land R(z, y)) \)

1. Dummy universal quantification for base case:

\[
\varphi_{tc}(R, x, y) \equiv (\forall z. M_1)(\exists z)(R(x, z) \land R(z, y)) \\
M_1 \equiv \neg(x = y \lor E(x, y))
\]

2. Using \( \forall \), replace two occurrences of \( R \) with one:

\[
\varphi_{tc}(R, x, y) \equiv (\forall z. M_1)(\exists z)(\forall uv. M_2)R(u, v) \\
M_2 \equiv (u = x \land v = z) \lor (u = z \land v = y)
\]

3. Requantify \( x \) and \( y \).

\[
M_3 \equiv (x = u \land y = v)
\]

\[
\varphi_{tc}(R, x, y) \equiv [ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\exists xy. M_3) ] R(x, y)
\]

Every FO inductive definition is equivalent to a quantifier block.
\[QB_{tc} \equiv [ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\forall xy. M_3) ] \]

\[\varphi_{tc}(R, x, y) \equiv [ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\exists xy. M_3) ] R(x, y) \]
\[ \mathcal{Q}B_{tc} \equiv [ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\forall xy. M_3) ] \]

\[ \varphi_{tc}(R, x, y) \equiv [ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\exists xy. M_3) ]R(x, y) \]

\[ \varphi_{tc}(R, x, y) \equiv [ \mathcal{Q}B_{tc} ]R(x, y) \]
\[ \mathcal{QB}_{tc} \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)] \]

\[ \varphi_{tc}(R, x, y) \equiv [(\forall z.M_1)(\exists z)(\forall uv.M_2)(\exists xy.M_3)]R(x, y) \]

\[ \varphi_{tc}(R, x, y) \equiv [\mathcal{QB}_{tc}]R(x, y) \]

\[ \varphi_{tc}(\emptyset) \equiv [\mathcal{QB}_{tc}]^{\prime}(\text{false}) \]
\[
\begin{align*}
QB_{tc} \equiv \left[ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\exists xy. M_3) \right] \\
\varphi_{tc}(R, x, y) &\equiv \left[ (\forall z. M_1)(\exists z)(\forall uv. M_2)(\exists xy. M_3) \right] R(x, y) \\
\varphi_{tc}(R, x, y) &\equiv [QB_{tc}] R(x, y) \\
\varphi^r_{tc}(\emptyset) &\equiv [QB_{tc}]^r (\text{false}) \\
\text{Thus, for any structure } \mathcal{A} \in \text{STRUC}[\tau_g],
\end{align*}
\]

\[
\begin{align*}
\mathcal{A} \in \text{REACH} &\iff \mathcal{A} \models (\text{LFP}\varphi_{tc})(s, t) \\
&\iff \mathcal{A} \models ([QB_{tc}]^{1+\log \|\mathcal{A}\|} \text{false})(s, t)
\end{align*}
\]
CRAM\[t(n)\] = concurrent parallel random access machine; polynomial hardware, parallel time \(O(t(n))\)

IND\[t(n)\] = first-order, depth \(t(n)\) inductive definitions

FO\[t(n)\] = \(t(n)\) repetitions of a block of restricted quantifiers:

\[QB = [(Q_1x_1.M_1) \cdots (Q_kx_k.M_k)]; \quad M_i \text{ quantifier-free}\]

\[\varphi_n = \underbrace{[QB][QB] \cdots [QB]}_{t(n)} M_0\]
Thm: For all constructible, polynomially bounded $t(n)$,

$$\text{CRAM}[t(n)] = \text{IND}[t(n)] = \text{FO}[t(n)]$$

Thm: For all $t(n)$, even beyond polynomial,

$$\text{CRAM}[t(n)] = \text{FO}[t(n)]$$
For $t(n)$ poly bdd,

$$\text{CRAM}[t(n)] = \text{IND}[t(n)] = \text{FO}[t(n)]$$
For all $t(n)$,  
\[
\text{CRAM}[t(n)] = \text{FO}[t(n)]
\]
**Thm:** For \( v = 1, 2, \ldots \), \( \text{DSPACE}[n^v] = \text{VAR}[v + 1] \)

Number of variables corresponds to amount of hardware.

Since variables range over a universe of size \( n \), a constant number of variables can specify a polynomial number of gates:

A bounded number of variables corresponds to polynomially much hardware.
Key Issue: Parallel Time versus Amount of Hardware
Key Issue: Parallel Time versus Amount of Hardware

- We would love to understand this tradeoff.
Key Issue: Parallel Time versus Amount of Hardware

- We would love to understand this tradeoff.
- Is there such a thing as an inherently sequential problem? No one knows.
Key Issue: Parallel Time versus Amount of Hardware

- We would love to understand this tradeoff.
- Is there such a thing as an inherently sequential problem? No one knows.
- Same tradeoff as number of variables vs. number of iterations of a quantifier block.
We would love to understand this tradeoff.

Is there such a thing as an inherently sequential problem? No one knows.

Same tradeoff as number of variables vs. number of iterations of a quantifier block.

One second-order variable can name $2^n$ gates.
Key Issue: Parallel Time versus Amount of Hardware

We would love to understand this tradeoff.

Is there such a thing as an inherently sequential problem? No one knows.

Same tradeoff as number of variables vs. number of iterations of a quantifier block.

One second-order variable can name $2^n$ gates.

Thus, $\text{SO}[t(n)] = \text{CRAM-HARD}[t(n), 2^{n^{O(1)}}]$. 
\[ \text{SO}[t(n)] = \text{CRAM-HARD} \left[ t(n), 2^n^{O(1)} \right] \]
Recent Breakthroughs in Descriptive Complexity

**Theorem** [Ben Rossman] Any first-order formula with any numeric relations ($\leq, +, \times, \ldots$) that means “I have a clique of size $k$” must have at least $k/4$ variables.

Creative new proof idea using Håstad’s Switching Lemma gives the essentially optimal bound.

This lower bound is for a fixed formula, if it were for a sequence of polynomially-sized formulas, i.e., a fixed-point formula, it would follow that $\text{CLIQUE} \not\in \mathbf{P}$ and thus $\mathbf{P} \neq \mathbf{NP}$.

Best previous bounds:

- $k$ variables necessary and sufficient without ordering or other numeric relations [I 1980].
- Nothing was known with ordering except for the trivial fact that 2 variables are not enough.
**Theorem** [Martin Grohe] Fixed-Point Logic with Counting captures Polynomial Time on all classes of graphs with excluded minors.

Grohe proves that for every class of graphs with excluded minors, there is a constant \( k \) such that two graphs of the class are isomorphic iff they agree on all \( k \)-variable formulas in fixed-point logic with counting.

Using Ehrenfeucht-Fraïssé games, this can be checked in polynomial time, \( (O(n^k \log n)) \). In the same time we can give a canonical description of the isomorphism type of any graph in the class. Thus every class of graphs with excluded minors admits the same general polynomial time canonization algorithm: we’re isomorphic iff we agree on all formulas in \( C_k \) and in particular, you are isomorphic to me iff your \( C_k \) canonical description is equal to mine.
What We Know

- **Diagonalization**: more of the same resource gives us more:

  \[ \text{DTIME}[n] \subsetneq \text{DTIME}[n^2], \]

  same for DSPACE, NTIME, NSPACE, \ldots
What We Know

- **Diagonalization**: more of the same resource gives us more:
  \[ \text{DTIME}[n] \subsetneq \text{DTIME}[n^2], \]
  same for DSPACE, NTIME, NSPACE, . . .

- **Natural Complexity Classes have Natural Complete Problems**
  SAT for NP, CVAL for P, QSAT for PSPACE, . . .
What We Know

- **Diagonalization**: more of the same resource gives us more:
  \[ \text{DTIME}[n] \subset \neq \text{DTIME}[n^2], \]
  same for DSPACE, NTIME, NSPACE, ... 

- **Natural Complexity Classes have Natural Complete Problems**
  SAT for NP, CVAL for P, QSAT for PSPACE, ... 

- **Major Missing Idea**: concept of work or conservation of energy in computation, i.e.,
  in order to solve SAT or other hard problem we must do a certain amount of computational work.
Strong Lower Bounds on $\text{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: $\text{FO}$. 
Strong Lower Bounds on $\text{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: FO.
- [Beame-Håstad]: hierarchy remains strict up to $\text{FO}[\log n / \log \log n]$. 
[Sipser]: strict first-order alternation hierarchy: $\text{FO}$.

[Beame-Håstad]: hierarchy remains strict up to $\text{FO}[\log n/\log \log n]$.

$\text{NC}^1 \subseteq \text{FO}[\log n/\log \log n]$ and this is tight.
Strong Lower Bounds on $\text{FO}[t(n)]$ for small $t(n)$

- [Sipser]: strict first-order alternation hierarchy: $\text{FO}$.

- [Beame-Håstad]: hierarchy remains strict up to $\text{FO}[\log n / \log \log n]$.

- $\text{NC}^1 \subseteq \text{FO}[\log n / \log \log n]$ and this is tight.

- Does $\text{REACH}$ require $\text{FO}[\log n]$? This would imply $\text{NC}^1 \neq \text{NL}$. 
Does It Matter? How important is $P \neq NP$?

- Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can’t be.
Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can’t be.

We conjecture that SAT requires $\text{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\text{DTIME}[n]$. 
Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can’t be.

We conjecture that SAT requires $\text{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\text{DTIME}[n]$.

Basic trade-offs are not understood, e.g., trade-off between time and number of processors. Are any problems inherently sequential? How can we best use multi-cores?
Much is known about approximation, e.g., some NP complete problems, e.g., Knapsack, Euclidean TSP, can be approximated as closely as we want, others, e.g., Clique, can’t be.

We conjecture that SAT requires $\text{DTIME}[\Omega(2^{\epsilon n})]$ for some $\epsilon > 0$, but no one has yet proved that it requires more than $\text{DTIME}[n]$.

Basic trade-offs are not understood, e.g., trade-off between time and number of processors. Are any problems inherently sequential? How can we best use multi-cores?

**SAT solvers** are impressive new general purpose problem solvers, e.g., used in model checking, AI planning, code synthesis. How good are current SAT solvers? How much can they be improved?
Descriptive Complexity

**Fact:** For constructible \( t(n) \), \( \text{FO}[t(n)] = \text{CRAM}[t(n)] \)

**Fact:** For \( k = 1, 2, \ldots \), \( \text{VAR}[k + 1] = \text{DSPACE}[n^k] \)

The complexity of computing a query is closely tied to the complexity of describing the query.

\[
\begin{align*}
P = \text{NP} & \iff \text{FO(LFP)} = \text{SO} \\
\text{ThC}^0 = \text{NP} & \iff \text{FO(MAJ)} = \text{SO} \\
P = \text{PSPACE} & \iff \text{FO(LFP)} = \text{SO(TC)}
\end{align*}
\]