Algorithms and Complexity: The Complexity of Theorem-Proving Procedures

Thomas Pipilikas

INTER-INSTITUTIONAL GRADUATE PROGRAM “ALGORITHMS, LOGIC AND DISCRETE MATHEMATICS”

- Presented at the 3rd Annual ACM SIGACT Symposium on the Theory of Computing (STOC, May 3-5, 1971)

- Presented at the 3rd Annual ACM SIGACT Symposium on the Theory of Computing (STOC, May 3-5, 1971)
  - STOC 2020 Online, June 22-26, 2020
  - STOC 2021 Rome, Italy, June 21-25, 2021

- Presented at the 3rd Annual ACM SIGACT Symposium on the Theory of Computing (STOC, May 3-5, 1971)
    - STOC 2020 Online, June 22-26, 2020
    - STOC 2021 Rome, Italy, June 21-25, 2021
- In 1982, Cook received the Turing award. His citation reads:

  For his advancement of our understanding of the complexity of computation in a significant and profound way. His seminal paper, The Complexity of Theorem Proving Procedures, presented at the 1971 ACM SIGACT Symposium on the Theory of Computing, laid the foundations for the theory of NP-Completeness. The ensuing exploration of the boundaries and nature of NP-complete class of problems has been one of the most active and important research activities in computer science for the last decade.

Overview

Cook's Theorem

- Any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a formula in CNF is satisfiable (SAT).

Overview

Cook's Theorem

- Any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a formula in CNF is satisfiable (SAT).
- If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

Overview

Cook's Theorem

- Any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a formula in CNF is satisfiable (SAT).
- If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

- P-reducibility stands for polynomial-time Turing reduction (aka Cook reduction, 1971).

Overview

Cook's Theorem

- Any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a formula in CNF is satisfiable (SAT).
- If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

- P-reducibility stands for polynomial-time Turing reduction (aka Cook reduction, 1971).
- vs polynomial-time many-one reduction (aka Karp reduction, 1972).

Overview

Cook's Theorem

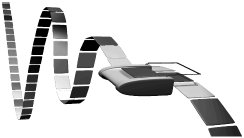
- Any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a formula in CNF is satisfiable (SAT).
- If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

- P-reducibility stands for polynomial-time Turing reduction (aka Cook reduction, 1971).
- vs polynomial-time many-one reduction (aka Karp reduction, 1972).
- NP-completeness.

Turing Machine

In his 1948 essay, "*Intelligent Machinery*", Turing wrote that his machine consisted of:

> *...an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol, and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings.*

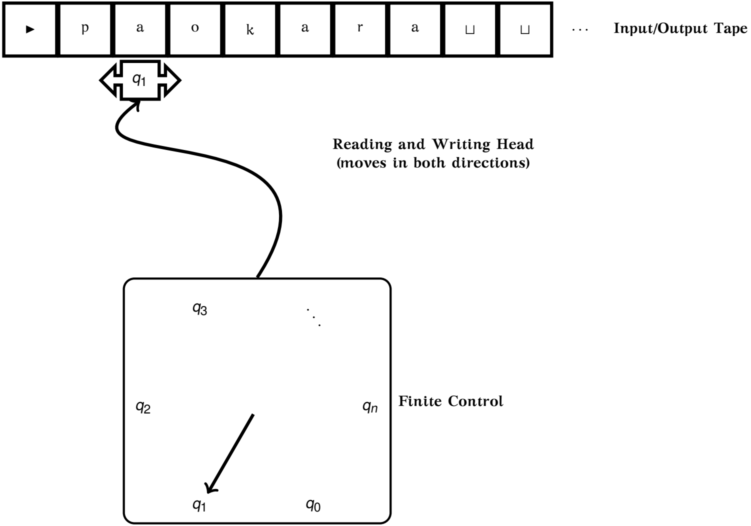


Formal Definition of Turing Machine

A *(one-tape) Turing machine* can be formally defined as a 6-tuple $M = \langle Q, \Gamma, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite, non-empty set of states;
- Γ is a finite, non-empty set of tape alphabet symbols;
- $\sqcup \in \Gamma$ is the *blank symbol* (the only symbol allowed to occur on the tape infinitely often at any step during the computation);
- $\blacktriangleright \in \Gamma$ is a special symbol, that defines the*begining* of the tape.
- $\Sigma \subseteq \Gamma \setminus \{b, \blacktriangleright\}$ is the set of input alphabet symbols, that is, the set of symbols allowed to appear in the initial tape contents;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *final states* or *accepting states*. The initial tape contents is said to be accepted by M if it eventually halts in a state from F.
- $\delta : (Q \setminus F) \times \Gamma \cup \{\blacktriangleright\} \not\to Q \times \Gamma \cup \{\blacktriangleright\} \times \{L, R\}$ is a partial function called the *transition function*, where L is left shift, R is right shift and for any $q \in Q \setminus F$, $\delta(q, \blacktriangleright) = (\ldots, \blacktriangleright, R)$, if δ is defined on $(q, \blacktriangleright)$. If δ is not defined on the current state and the current tape symbol, then the machine halts (rejects).

Turing Machine



Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,
- a TM M rejects the string w iff M with input w ends in a non-accepting state,

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,
- a TM M rejects the string w iff M with input w ends in a non-accepting state,
- a TM M recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,
- a TM M rejects the string w iff M with input w ends in a non-accepting state,
- a TM M recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a TM M decides a set (language) L iff for any $w \in L$, M accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, M rejects $\tilde{w}$,

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,
- a TM M rejects the string w iff M with input w ends in a non-accepting state,
- a TM M recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a TM M decides a set (language) L iff for any $w \in L$, M accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, M rejects $\tilde{w}$,
- L is recursive enumerable , iff there is a TM M that recognises L,

Turing Machine

The computation of a TM M starts with the state register initialized with q_0 and the head reading the first square of the tape.
We say that:

- a TM M accepts the string w iff M with input w ends in an accepting state,
- a TM M rejects the string w iff M with input w ends in a non-accepting state,
- a TM M recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a TM M decides a set (language) L iff for any $w \in L$, M accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, M rejects $\tilde{w}$,
- L is recursive enumerable , iff there is a TM M that recognises L,
- L is recursive, iff there is a TM M that decides L.

Church-Turing Thesis

Every realistic model of computation, yet discovered, has been shown to be equivalent.

Church-Turing Thesis

Every realistic model of computation, yet discovered, has been shown to be equivalent.

Church-Turing thesis

- Every known and "unknown" models of notion of computation (calculability) are effectively equivalent.

Church-Turing Thesis

Every realistic model of computation, yet discovered, has been shown to be equivalent.

Church-Turing thesis

- Every known and "unknown" models of notion of computation (calculability) are effectively equivalent.
- Every effective computation can be carried out by a Turing machine.

Church

Turing

Nondeterministic Turing Machine

In contrast to a deterministic Turing machine, in a nondeterministic Turing machine (NTM) the set of rules may prescribe more than one action to be performed for any given situation,

Nondeterministic Turing Machine

In contrast to a deterministic Turing machine, in a nondeterministic Turing machine (NTM) the set of rules may prescribe more than one action to be performed for any given situation,

i.e. $\delta \subseteq ((Q \setminus F) \times \Gamma \cup \{\blacktriangleright\}) \times Q \times \Gamma \cup \{\blacktriangleright\} \times \{L, R\}$ where again for any $q, q' \in Q \setminus F$ and any $y \in \Gamma$, $(q, \blacktriangleright, q', y, L) \notin \delta$.

Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,

Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,
- a NTM N rejects the string w iff every computation of N with input w ends in a non-accepting state,

Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,
- a NTM N rejects the string w iff every computation of N with input w ends in a non-accepting state,
- a NTM N recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,

Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,
- a NTM N rejects the string w iff every computation of N with input w ends in a non-accepting state,
- a NTM N recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a NTM N decides a set (language) L iff for any $w \in L$, N accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, N rejects $\tilde{w}$,

Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,
- a NTM N rejects the string w iff every computation of N with input w ends in a non-accepting state,
- a NTM N recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a NTM N decides a set (language) L iff for any $w \in L$, N accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, N rejects $\tilde{w}$,
- NTMs and TMs decide (recognise) the same languages.

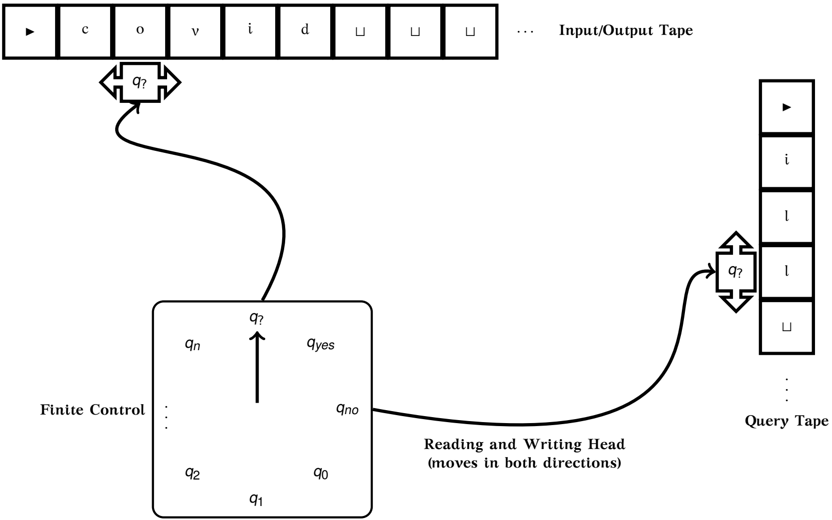
Nondeterministic Turing Machine

- a NTM N accepts the string w iff there is a computation of N with input w ends in an accepting state,
- a NTM N rejects the string w iff every computation of N with input w ends in a non-accepting state,
- a NTM N recognises a set (language) L, when for any $w \in \Sigma$, M accepts w iff $w \in L$,
- a NTM N decides a set (language) L iff for any $w \in L$, N accepts w and for any $\tilde{w} \in \Sigma^* \setminus L$, N rejects $\tilde{w}$,
- NTMs and TMs decide (recognise) the same languages.(complexity?)

Query Machine

A *query machine* is a multitape Turing machine with a distinguished tape called the *query tape*, and three distinguished states called the *query state* ($q_?$), *yes state* (q_{yes}), and *no state* (q_{no}), respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if $u \in T$ and the no state if $u \notin T$. We think of an "*oracle*", which knows T, placing M in the yes state or no state.

Query Machine



Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

- P-reducibility is a transitive relation.

Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

- P-reducibility is a transitive relation.
- The relation E on sets of strings, given by $(S, T) \in E$ iff each of S and T is P-reducible to the other, is an equivalence relation.

Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

- P-reducibility is a transitive relation.
- The relation E on sets of strings, given by $(S, T) \in E$ iff each of S and T is P-reducible to the other, is an equivalence relation.
- The equivalence class containing a set S will be denoted by $\deg(S)$ (the polynomial degree of difficulty of S).

Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

- P-reducibility is a transitive relation.
- The relation E on sets of strings, given by $(S, T) \in E$ iff each of S and T is P-reducible to the other, is an equivalence relation.
- The equivalence class containing a set S will be denoted by $\deg(S)$ (the polynomial degree of difficulty of S).
- We will denote $\deg(\{0\})$ by $\mathscr{L}_*$, where 0 denotes the zero function.

Polynomial-time Turing reduction

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial $Q(n)$ such that for each input string w, the T-computation of M with input w halts within $Q(|w|)$ steps ($|w|$ is the length of w) and ends in an accepting state iff $w \in S$.

- P-reducibility is a transitive relation.
- The relation E on sets of strings, given by $(S, T) \in E$ iff each of S and T is P-reducible to the other, is an equivalence relation.
- The equivalence class containing a set S will be denoted by $\deg(S)$ (the polynomial degree of difficulty of S).
- We will denote $\deg(\{0\})$ by $\mathscr{L}_*$, where 0 denotes the zero function.($\mathscr{L}_* = \mathrm{P}$!!!)

Propositional Calculus

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ.

Propositional Calculus

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ.

Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol.

Propositional Calculus

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ.

Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol.

Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols.

Propositional Calculus

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ.

Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol.

Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols.

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet.

Propositional Calculus

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ.

Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol.

Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols.

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet.

Cook's Theorem will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood.

Special Set of Strings

1. The *subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, *\}$ by listing the successive rows of its adjacency matrix, separated by $*s$. We let {subgraph pairs} denote the set of strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to a subgraph of G_2.

Special Set of Strings

1. The *subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, *\}$ by listing the successive rows of its adjacency matrix, separated by $*s$. We let {subgraph pairs} denote the set of strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to a subgraph of G_2.
2. The *graph isomorphism problem* will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to G_2.

Special Set of Strings

1. The *subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, *\}$ by listing the successive rows of its adjacency matrix, separated by $*$s. We let {subgraph pairs} denote the set of strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to a subgraph of G_2.
2. The *graph isomorphism problem* will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to G_2.
3. The set {primes} is the set of all binary notations for prime numbers.

Special Set of Strings

1. The *subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, *\}$ by listing the successive rows of its adjacency matrix, separated by $*s$. We let {subgraph pairs} denote the set of strings $\overline{G_1} ** \overline{G_2}$ such that G_1 is isomorphic to a subgraph of G_2.
2. The *graph isomorphism problem* will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\overline{G_1} ** \overline{G_2}$ such that G_1 is isomorphic to G_2.
3. The set {primes} is the set of all binary notations for prime numbers.
4. The set {DNF tautologies} is the set of strings representing tautologies in disjunctive normal form.

Special Set of Strings

1. The *subgraph problem* is the problem given two finite undirected graphs, determine whether the first is isomorphic to a subgraph of the second. A graph G can be represented by a string $\overline{G}$ on the alphabet $\{0, 1, *\}$ by listing the successive rows of its adjacency matrix, separated by $*s$. We let {subgraph pairs} denote the set of strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to a subgraph of G_2.
2. The *graph isomorphism problem* will be represented by the set, denoted by {isomorphic graphpairs}, of all strings $\overline{G_1} * * \overline{G_2}$ such that G_1 is isomorphic to G_2.
3. The set {primes} is the set of all binary notations for prime numbers.
4. The set {DNF tautologies} is the set of strings representing tautologies in disjunctive normal form.
5. The set D_3 consists of those tautologies in disjunctive normal form in which each disjunct has at most three conjuncts (each of which is an atom or negation of an atom).

Cook's Theorem

Theorem (Cook's Theorem)

If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to $\{$DNF tautologies$\}$.

Cook's Theorem

Theorem (Cook's Theorem)

If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to $\{$DNF tautologies$\}$.

Corollary

Each of the sets in definitions 1)–5) is P-reducible to $\{$DNF tautologies$\}$.

Cook's Theorem (Proof)

Sketching the proof

Cook's Theorem (Proof)

Sketching the proof

- Suppose a NTM M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial.

Cook's Theorem (Proof)

Sketching the proof

- Suppose a NTM M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial.
- Given an input w for M, we will construct a proposition formula $A(w)$ in conjunctive normal form (CNF) such that $A(w)$ is satisfiable iff M accepts w.

Cook's Theorem (Proof)

Sketching the proof

- Suppose a NTM M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial.
- Given an input w for M, we will construct a proposition formula $A\,(w)$ in conjunctive normal form (CNF) such that $A\,(w)$ is satisfiable iff M accepts w.
- $\neg A\,(w)$ is easily put in DNF (using De Morgan's laws), and $\neg A\,(w)$ is a tautology iff $w \notin S$.

Cook's Theorem (Proof)

Sketching the proof

- Suppose a NTM M accepts a set S of strings within time $Q(n)$, where $Q(n)$ is a polynomial.
- Given an input w for M, we will construct a proposition formula $A(w)$ in conjunctive normal form (CNF) such that $A(w)$ is satisfiable iff M accepts w.
- $\neg A(w)$ is easily put in DNF (using De Morgan's laws), and $\neg A(w)$ is a tautology iff $w \notin S$.
- Since the whole construction can be carried out in time bounded by a polynomial in $|w|$, the theorem will be proved.

Cook's Theorem (Proof)

Let us number the squares of M from left to right $1, 2, \ldots$.

Cook's Theorem (Proof)

Let us number the squares of M from left to right $1, 2, \ldots$.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_l\}$ and the set of states is $\{q_1, \ldots, q_r\}$.

Cook's Theorem (Proof)

Let us number the squares of M from left to right $1, 2, \ldots$.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_l\}$ and the set of states is $\{q_1, \ldots, q_r\}$.

Let us fix an input w to M of length n, and suppose $w \in S$.

Cook's Theorem (Proof)

Let us number the squares of M from left to right $1, 2, \ldots$.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_l\}$ and the set of states is $\{q_1, \ldots, q_r\}$.

Let us fix an input w to M of length n, and suppose $w \in S$.

Then there is a computation of M with input w that ends in an accepting state within $T = Q(n)$ steps.

Cook's Theorem (Proof)

Let us number the squares of M from left to right $1, 2, \ldots$.

Suppose the tape alphabet for M is $\{\sigma_1, \ldots, \sigma_l\}$ and the set of states is $\{q_1, \ldots, q_r\}$.

Let us fix an input w to M of length n, and suppose $w \in S$.

Then there is a computation of M with input w that ends in an accepting state within $T = Q(n)$ steps.

Notice that since the computation has at most $T = Q(n)$ steps, no tape square beyond T is scanned.

Cook's Theorem (Proof)

The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings (listed in the next frame), refer to such a computation.

Cook's Theorem (Proof)

The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings (listed in the next frame), refer to such a computation.
Proposition symbols:

Cook's Theorem (Proof)

The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings (listed in the next frame), refer to such a computation.

Proposition symbols:

- $P^i_{s,t}$ is true iff tape square number s at step t contains the symbol σ_i, where $i \in [l]$, $s, t \in [T]$.

Cook's Theorem (Proof)

The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings (listed in the next frame), refer to such a computation.

Proposition symbols:

- $P^i_{s,t}$ is true iff tape square number s at step t contains the symbol σ_i, where $i \in [l]$, $s, t \in [T]$.
- Q^i_t is true iff at step t the machine is in state q_i, where $i \in [r]$, $t \in [T]$.

Cook's Theorem (Proof)

The formula $A(w)$ will be built from many different proposition symbols, whose intended meanings (listed in the next frame), refer to such a computation.

Proposition symbols:

- $P^i_{s,t}$ is true iff tape square number s at step t contains the symbol σ_i, where $i \in [l]$, $s, t \in [T]$.
- Q^i_t is true iff at step t the machine is in state q_i, where $i \in [r]$, $t \in [T]$.
- $S_{s,t}$ is true iff at time t square number s is scanned by the tape head, where $s, t \in [T]$.

Cook's Theorem (Proof)

The formula $A(w)$ is a conjunction $B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge I$ formed as follows. Notice $A(w)$ is in CNF.

Cook's Theorem (Proof)

The formula $A(w)$ is a conjunction $B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge I$ formed as follows. Notice $A(w)$ is in CNF.

B will assert that at each step t, one and only one square is scanned.

Cook's Theorem (Proof)

The formula $A(w)$ is a conjunction $B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge I$ formed as follows. Notice $A(w)$ is in CNF.

B will assert that at each step t, one and only one square is scanned.

B is a conjunction $B_1 \wedge B_2 \wedge \cdots \wedge B_T$, where B_t asserts that at time t one and only one square is scanned:

$$B_t = (S_{1,t} \vee S_{2,t} \vee \cdots \vee S_{T,t}) \wedge \left[\bigwedge_{1 \leq i < j \leq T} (\neg S_{i,t} \vee \neg S_{j,t}) \right]$$

Cook's Theorem (Proof)

For $s \in [T]$ and $t \in [T_j]$ $C_{s,t}$ asserts that at square s and time t there is one and only one symbol. C is the conjunction of all the $C_{s,t}$.

Cook's Theorem (Proof)

For $s \in [T]$ and $t \in [T_j]$ $C_{s,t}$ asserts that at square s and time t there is one and only one symbol. C is the conjunction of all the $C_{s,t}$.

D asserts that for each t there is one and only one state.

Cook's Theorem (Proof)

For $s \in [T]$ and $t \in [T_j]$ $C_{s,t}$ asserts that at square s and time t there is one and only one symbol. C is the conjunction of all the $C_{s,t}$.

D asserts that for each t there is one and only one state.

E asserts the initial conditions are satisfied:

$$E = Q_1^0 \wedge S_{1,1} \wedge P_{1,1}^{i_1} \wedge P_{2,1}^{i_2} \wedge \cdots \wedge P_{n,1}^{i_n} \wedge P_{n+1,1}^{1} \wedge \cdots \wedge P_{T,1}^{1}$$

where $w = \sigma_{i_1} \cdots \sigma_{i_n}$, q_0 is the initial state and $\sigma_1 = \sqcup$ is the blank symbol.

Cook's Theorem (Proof)

F, G and H assert that for each time t the values of the P's, Q's and S's are updated properly.

Cook's Theorem (Proof)

F, G and H assert that for each time t the values of the P's, Q's and S's are updated properly.

For example, G is the conjunction over all t, i, j of $G^t_{i,j}$, where $G^t_{i,j}$ asserts that if at time t the machine is in state q_i scanning symbol s_j, then at time $t+1$ the machine is in state q_k, where q_k is the state given by the transition function (relation) for M.

Cook's Theorem (Proof)

F, G and H assert that for each time t the values of the P's, Q's and S's are updated properly.

For example, G is the conjunction over all t, i, j of $G^t_{i,j}$, where $G^t_{i,j}$ asserts that if at time t the machine is in state q_i scanning symbol s_j, then at time $t+1$ the machine is in state q_k, where q_k is the state given by the transition function (relation) for M. (!)

(or states:

(nondeterminism)

$$G^t_{i,j} = \bigwedge_{s=1}^{T} \left(\neg Q^i_t \vee \neg S_{s,t} \vee \neg P^j_{s,t} \vee Q^k_{t+1} \right)$$

Cook's Theorem (Proof)

Finally, the formula I asserts that the machine reaches an accepting state at some time. The machine M should be modified so that it continues to compute in some trivial fashion after reaching an accepting state, so that $A(w)$ will be satisfied.

Cook's Theorem (Proof)

Finally, the formula I asserts that the machine reaches an accepting state at some time. The machine M should be modified so that it continues to compute in some trivial fashion after reaching an accepting state, so that $A(w)$ will be satisfied.

It is now straightforward to verify that $A(w)$ has all the properties asserted in the first paragraph of the proof.

$\square$

Cook's Theorem (Proof)

Finally, the formula I asserts that the machine reaches an accepting state at some time. The machine M should be modified so that it continues to compute in some trivial fashion after reaching an accepting state, so that $A(w)$ will be satisfied.

It is now straightforward to verify that $A(w)$ has all the properties asserted in the first paragraph of the proof.

□

sort of NP-Completeness

Theorem

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {*tautologies*}, {*DNF tautologies*}, D_3, {*subgraph pairs*}.

sort of NP-Completeness

Theorem

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark

We have not been able to add either {primes} or {isomorphic graphpairs} to the above list. To show {tautologies} is P-reducible to {primes} would seem to require some deep results in number theory

sort of NP-Completeness

Theorem

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark

We have not been able to add either {primes} or {isomorphic graphpairs} to the above list. To show {tautologies} is P-reducible to {primes} would seem to require some deep results in number theory (actually in P)

sort of NP-Completeness

Theorem

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark

We have not been able to add either {primes} or {isomorphic graphpairs} to the above list. To show {tautologies} is P-reducible to {primes} would seem to require some deep results in number theory (actually in P), while showing {tautologies} is P-reducible to {isomorphic graphpairs} would probably upset a conjecture of Corneil's from which he deduces that the graph isomorphism problem can be solved in polynomial time

sort of NP-Completeness

Theorem

The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D_3, {subgraph pairs}.

Remark

We have not been able to add either {primes} or {isomorphic graphpairs} to the above list. To show {tautologies} is P-reducible to {primes} would seem to require some deep results in number theory (actually in P), while showing {tautologies} is P-reducible to {isomorphic graphpairs} would probably upset a conjecture of Corneil's from which he deduces that the graph isomorphism problem can be solved in polynomial time (believed to be NP-intermediate).

sort of NP-Completeness

- D_2 consisting of all DNF tautologies with at most two conjuncts per disjunct, is in $\mathscr{L}_*$ (Davis-Putnam procedure). Hence D_2 cannot be added to the list in theorem 2 (unless all sets in the list are in $\mathscr{L}_*$).

sort of NP-Completeness

- D_2 consisting of all DNF tautologies with at most two conjuncts per disjunct, is in $\mathscr{L}_*$ (Davis-Putnam procedure). Hence D_2 cannot be added to the list in theorem 2 (unless all sets in the list are in $\mathscr{L}_*$). (P vs NP !!!)

sort of NP-Completeness

- D_2 consisting of all DNF tautologies with at most two conjuncts per disjunct, is in $\mathscr{L}_*$ (Davis-Putnam procedure). Hence D_2 cannot be added to the list in theorem 2 (unless all sets in the list are in $\mathscr{L}_*$). (P vs NP !!!)
- {DNF tautologies} is actually coNP-complete.

Proof of the 2nd Theorem

Sketching the proof

Proof of the 2nd Theorem

Sketching the proof

- By the corollary, each of the sets is P-reducible to {DNF tautologies}.

Proof of the 2nd Theorem

Sketching the proof

- By the corollary, each of the sets is P-reducible to {DNF tautologies}.
- Obviously {DNF tautologies} is P-reducible to {tautologies}.

Proof of the 2nd Theorem

Sketching the proof

- By the corollary, each of the sets is P-reducible to {DNF tautologies}.
- Obviously {DNF tautologies} is P-reducible to {tautologies}.
- It remains to show {DNF tautologies} is P-reducible to D_3 and

Proof of the 2nd Theorem

Sketching the proof

- By the corollary, each of the sets is P-reducible to {DNF tautologies}.
- Obviously {DNF tautologies} is P-reducible to {tautologies}.
- It remains to show {DNF tautologies} is P-reducible to D_3 and
- D_3 is P-reducible to {subgraph pairs}.

Proof of the 2nd Theorem

To show {DNF tautologies} is P-reducible to D_3:

Proof of the 2nd Theorem

To show {DNF tautologies} is P-reducible to D_3:

Let $A = B_1 \vee \cdots \vee B_k$ be a proposition formula in DNF, where $B_1 = R_1 \wedge \cdots \wedge R_s$, and each R_i is an atom or negation of an atom, and $s > 3$.

Proof of the 2nd Theorem

To show {DNF tautologies} is P-reducible to D_3:

Let $A = B_1 \vee \cdots \vee B_k$ be a proposition formula in DNF, where $B_1 = R_1 \wedge \cdots \wedge R_s$, and each R_i is an atom or negation of an atom, and $s > 3$.

Then A is a tautology if and only if A' is a tautology where

$$A' = (P \wedge R_3 \wedge \cdots \wedge R_s) \vee (\neg P \wedge R_1 \wedge R_2) \vee B_2 \vee B_3 \vee \cdots \vee B_k,$$

where P is a new atom.

Proof of the 2nd Theorem

To show {DNF tautologies} is P-reducible to D_3:

Let $A = B_1 \vee \cdots \vee B_k$ be a proposition formula in DNF, where $B_1 = R_1 \wedge \cdots \wedge R_s$, and each R_i is an atom or negation of an atom, and $s > 3$.

Then A is a tautology if and only if A' is a tautology where

$$A' = (P \wedge R_3 \wedge \cdots \wedge R_s) \vee (\neg P \wedge R_1 \wedge R_2) \vee B_2 \vee B_3 \vee \cdots \vee B_k,$$

where P is a new atom.

Since we have reduced the number of conjuncts in B_1, this process may be repeated until eventually a formula is found with at most three conjuncts per disjunct.

Proof of the 2nd Theorem

To show {DNF tautologies} is P-reducible to D_3:

Let $A = B_1 \vee \cdots \vee B_k$ be a proposition formula in DNF, where $B_1 = R_1 \wedge \cdots \wedge R_s$, and each R_i is an atom or negation of an atom, and $s > 3$.

Then A is a tautology if and only if A' is a tautology where

$$A' = (P \wedge R_3 \wedge \cdots \wedge R_s) \vee (\neg P \wedge R_1 \wedge R_2) \vee B_2 \vee B_3 \vee \cdots \vee B_k,$$

where P is a new atom.

Since we have reduced the number of conjuncts in B_1, this process may be repeated until eventually a formula is found with at most three conjuncts per disjunct.

Clearly the entire process is bounded in time by a polynomial in the length of A.

Proof of the 2nd Theorem

To show D_3 is P-reducible to {subgraph pairs}:

Proof of the 2nd Theorem

To show D_3 is P-reducible to {subgraph pairs}:

Suppose $A = C_1 \vee \cdots \vee C_k$ in D_3, where $C_i = R_{i1} \wedge R_{i2} \wedge R_{i3}$, and each R_{ij} is an atom or negation of an atom.

Proof of the 2nd Theorem

To show D_3 is P-reducible to {subgraph pairs}:

Suppose $A = C_1 \vee \cdots \vee C_k$ in D_3, where $C_i = R_{i1} \wedge R_{i2} \wedge R_{i3}$, and each R_{ij} is an atom or negation of an atom.

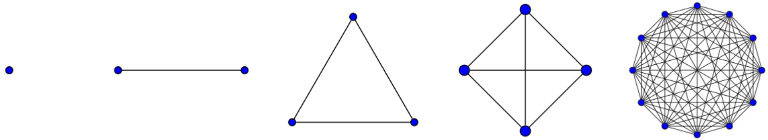
Let $G_1 = K_k$ be the complete graph with k vertices.

Proof of the 2nd Theorem

To show D_3 is P-reducible to {subgraph pairs}:

Suppose $A = C_1 \vee \cdots \vee C_k$ in D_3, where $C_i = R_{i1} \wedge R_{i2} \wedge R_{i3}$, and each R_{ij} is an atom or negation of an atom.

Let $G_1 = K_k$ be the complete graph with k vertices.



K_1 K_2 K_3 K_4 K_{12}

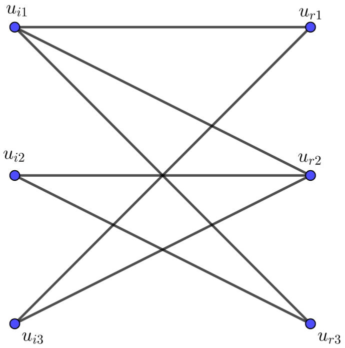
Proof of the 2nd Theorem

Let G_2 be the graph with vertices $\left\{u_{ij}\right\}$, $i \in [k]$, $j \in [3]$, such that u_{ij} is connected by an edge to u_{rs} iff $i \neq r$ and the two literals $\left(R_{ij}, R_{rs}\right)$ do not form an opposite pair (that is they are neither of the form $(P, \neg P)$ nor of the form $(\neg P, P)$).

Proof of the 2nd Theorem

Let G_2 be the graph with vertices $\{u_{ij}\}$, $i \in [k]$, $j \in [3]$, such that u_{ij} is connected by an edge to u_{rs} iff $i \neq r$ and the two literals (R_{ij}, R_{rs}) do not form an opposite pair (that is they are neither of the form $(P, \neg P)$ nor of the form $(\neg P, P)$).

$C_i = R_{i1} \wedge R_{i2} \wedge R_{i3} = R \wedge \neg P \wedge X \qquad C_r = R_{r1} \wedge R_{r2} \wedge R_{r3} = P \wedge R \wedge \neg X$



Proof of the 2nd Theorem

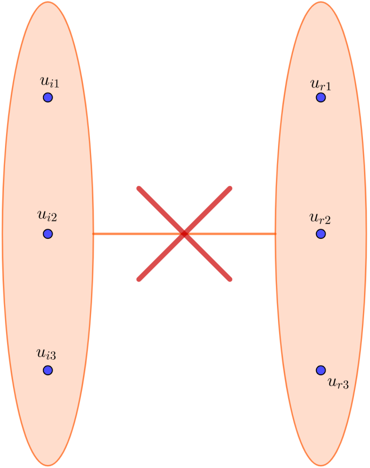
In which cases are there no edges between vertices corresponding to C_i, C_r?

Proof of the 2nd Theorem

In which cases are there no edges between vertices corresponding to C_i, C_r?

$C_i = \neg P \wedge \neg P \wedge \neg P$

$C_r = P \wedge P \wedge P$

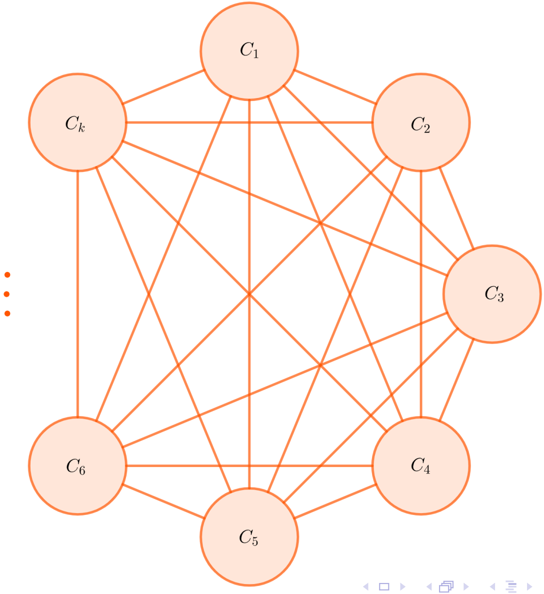


Proof of the 2nd Theorem

When is K_k isomorphic to a subgraph of G_2?

Proof of the 2nd Theorem

When is K_k isomorphic to a subgraph of G_2?



Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi\colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.

Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi \colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.
- The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified.

Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi\colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.
- The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified.
- Then G_1 is isomorphic to a subgraph of G_2 iff $A \notin D_3$.

Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi\colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.
- The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified.
- Then G_1 is isomorphic to a subgraph of G_2 iff $A \notin D_3$. (coNP)

Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi \colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.
- The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified.
- Then G_1 is isomorphic to a subgraph of G_2 iff $A \notin D_3$. (coNP)
- Such a construction can be carried out in polynomial time.

Proof of the 2nd Theorem

- Thus there is a falsifying truth assignment to the formula A iff there is a graph homomorphism $\phi \colon G_1 \to G_2$ such that for each i, $\phi(i) = u_{ij}$ for some j.
- The homomorphism tells for each i which of R_{i1}, R_{i2}, R_{i3} should be falsified, and the selective lack of edges in G_2 guarantees that the resulting truth assignment is consistently specified.
- Then G_1 is isomorphic to a subgraph of G_2 iff $A \notin D_3$. (coNP)
- Such a construction can be carried out in polynomial time.

This completes the proof of the 2nd theorem. $\square$

Discussion ☕

Theorem 1 and its corollary give strong evidence that it is not easy to determine whether a given proposition formula is a tautology, even if the formula is in normal disjunctive form. Theorems 1 and 2 together suggest that it is fruitless to search for a polynomial decision procedure for the subgraph problem, since success would bring polynomial decision procedures to many other apparently intractible problems. Of course the same remark applies to any combinatorial problem to which {tautologies} is P-reducible.

Furthermore, the theorems suggest that {tautologies} is a good candidate for an interesting set not in $\mathscr{L}_*$, and I feel it is worth spending considerable effort trying to prove this conjecture. Such a proof would be a major breakthrough in complexity theory.

In view of the apparent complexity of {DNF tautologies}, it is interesting to examine the Davis-Putnam procedure. This procedure was designed to determine whether a given formula in conjunctive normal form is satisfiable, but of course the "dual" procedure determines whether a given formula in disjunctive normal form is a tautology. I have not yet been able to find a series of examples showing the procedure (treated sympathetically to avoid certain pitfalls) must require more than polynomial time. Nor have I found an interesting upper bound for the time required.

The Legend Himself

An interview conducted with Cook by Bruce Kapron for the ACM on February 25, 2016.

Bibliography

- Cook, Stephen A. "*The complexity of theorem-proving procedures.*" Proceedings of the third annual ACM symposium on Theory of computing. 1971.
- Zhou, JianMing, and Yu Li. "*What is Cook's theorem?.*" arXiv preprint arXiv:1501.01910 (2015).
- Zhou, JianMing, and Yu Li. "*Inquiry of P-reduction in Cook's 1971 Paper–from Oracle machine to Turing machine.*" arXiv preprint arXiv:1905.06311 (2019).