Complexity Classes and Reductions

JASS 2006 Course One: Proofs and Computers

Bernhard Häupler

Technische Universität München

April 2006

What this talk is about:

- Complexity classes and Problems

What this talk is about:

- Complexity classes and Problems
- Function Problems

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles
- Reductions and Completeness

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles
- Reductions and Completeness
- Boolean Circuits

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles
- Reductions and Completeness
- Boolean Circuits
- P, NP and $PSPACE$ completeness results

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles
- Reductions and Completeness
- Boolean Circuits
- P, NP and $PSPACE$ completeness results
- Problems in FP^{NP}

What this talk is about:

- Complexity classes and Problems
- Function Problems
- Oracles
- Reductions and Completeness
- Boolean Circuits
- P, NP and $PSPACE$ completeness results
- Problems in FP^{NP}
- Polynomial Hierarchy

Definitions:

$TIME(f(n)) :=$ Languages decidable in time $O(f(n))$ by a DTM

$NTIME(f(n)) :=$ Languages decidable in time $O(f(n))$ by a NTM

$SPACE(f(n)) :=$ Languages decidable in space $O(f(n))$ by a DTM (besides the (read only) input and the (write only) output)

$NSPACE(f(n)) :=$ Languages decidable in space $O(f(n))$ by a NTM

Important Complexity Classes

$$L := SPACE(\log n) \qquad NL := NSPACE(\log n)$$

$$P := \bigcup_k TIME(n^k) \qquad NP := \bigcup_k NTIME(n^k)$$

$$coNP := \mathrm{P}(\Sigma^*) \setminus NP$$

$$PSPACE := \bigcup_k SPACE(n^k)$$

$$EXP := \bigcup_k TIME(2^{n^k}) \qquad NEXP := \bigcup_k NTIME(2^{n^k})$$

$$2\text{-}EXP := \bigcup_k TIME(2^{2^{n^k}})$$

$$ELEMENTARY := \bigcup_k k\text{-}EXP$$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
    - $L \subseteq P$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
    - $L \subseteq P$
    - $NL \subseteq P$
    - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes
- Nondeterministic Space

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes
- Nondeterministic Space
  - $coNSPACE = NSPACE$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes
- Nondeterministic Space
  - $coNSPACE = NSPACE$
  - $NSPACE(f(n)) \subseteq SPACE(f^2(n))$

Relationships between Complexity Classes

Important relationships:

- Hierarchy in *PSPACE*
  - $L \subseteq P$
  - $NL \subseteq P$
  - $\Rightarrow L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
  - $L \subset PSPACE$
- Linear Speedup
  - $TIME(f(n)) = TIME(\epsilon f(n) + n + 2)$
  - $SPACE(f(n)) = SPACE(\epsilon f(n) + 2)$
  - same for nondeterministic classes
- Nondeterministic Space
  - $coNSPACE = NSPACE$
  - $NSPACE(f(n)) \subseteq SPACE(f^2(n))$
  - $\Rightarrow PSPACE = NPSPACE = coNPSPACE$

Function problems

Definition (Function problems)

A function problem is abstracted by a binary relation $R \subseteq \Sigma^* \times \Sigma^*$.
The task is: Given an input x, find an output y with $(x, y) \in R$.

Function problems

Definition (Function problems)

A function problem is abstracted by a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. The task is: Given an input x, find an output y with $(x, y) \in R$.

FC is the class of all function problems computable by a TM in C

Function problems

Definition (Function problems)

A function problem is abstracted by a binary relation $R \subseteq \Sigma^* \times \Sigma^*$.
The task is: Given an input x, find an output y with $(x, y) \in R$.

FC is the class of all function problems computable by a TM in C

Definition (Decision problems)

A decision problem is abstracted by a language $L \subseteq \Sigma^*$.
The task is: Given an input x, decide whether $x \in L$.

Function problems

Definition (Function problems)

A function problem is abstracted by a binary relation $R \subseteq \Sigma^* \times \Sigma^*$.
The task is: Given an input x, find an output y with $(x, y) \in R$.

FC is the class of all function problems computable by a TM in C

Definition (Decision problems)

A decision problem is abstracted by a language $L \subseteq \Sigma^*$.
The task is: Given an input x, decide whether $x \in L$.

$L(R) := \{x \mid \exists y : (x, y) \in R\}$
is the decision problem related to the function problem R

Oracles

Definition: (Oracle TM)

An oracle TM $M^?$ has 3 additional states (q_{query}, q_{yes} and q_{no}) and one additional query-string qs.
After being in state q_{query} $M^?$ continues in state q_{yes} / q_{no} depending on the answer of the oracle on input qs.

Oracles

Definition: (Oracle TM)

An oracle TM $M^?$ has 3 additional states (q_{query}, q_{yes} and q_{no}) and one additional query-string qs.
After being in state q_{query} $M^?$ continues in state q_{yes} / q_{no} depending on the answer of the oracle on input qs.

Definition: (Oracle Complexity Class)

$C^O =$ Languages decidable by an oracle TM $M^? \in C$ with oracle O

Reductions: Idea

Idea: If problem A reduces to B then B is at least as hard as A
We write therefore $A \leq B$

Reductions: Definitions

Definition: (Reductions)

Let f, g, h be functions:

- **Cook:** $A \in P^B$

Reductions: Definitions

Definition: (Reductions)

Let f, g, h be functions:

- **Cook:** $A \in P^B$
- **Karp:** $\exists f \in FP: \; x \in A \iff f(x) \in B$

Reductions: Definitions

Definition: (Reductions)

Let f, g, h be functions:

- **Cook:** $A \in P^B$
- **Karp:** $\exists f \in FP : \; x \in A \iff f(x) \in B$
- **Logspace:** $\exists f \in FL : \; x \in A \iff f(x) \in B$

Reductions: Definitions

Definition: (Reductions)

Let f, g, h be functions:

- **Cook:** $A \in P^B$
- **Karp:** $\exists f \in FP: \; x \in A \iff f(x) \in B$
- **Logspace:** $\exists f \in FL: \; x \in A \iff f(x) \in B$
- **Levin:** $\exists f, g, h \in FP:$
  $x \in L(R_1) \iff f(x) \in L(R_2)$
  $\forall x, z: \; (f(x), z) \in R_2 \implies (x, g(x, z)) \in L(R_1)$
  $\forall (x, y) \in R_1: \; (f(x), h(x, y)) \in L(R_2)$

Reductions: Definitions

Definition: (Reductions)

Let f, g, h be functions:

- **Cook:** $A \in P^B$
- **Karp:** $\exists f \in FP: \; x \in A \iff f(x) \in B$
- **Logspace:** $\exists f \in FL: \; x \in A \iff f(x) \in B$
- **Levin:** $\exists f, g, h \in FP:$
  $x \in L(R_1) \iff f(x) \in L(R_2)$
  $\forall x, z: \; (f(x), z) \in R_2 \implies (x, g(x, z)) \in L(R_1)$
  $\forall (x, y) \in R_1: \; (f(x), h(x, y)) \in L(R_2)$
- **L-Reduction:** like Karp but preserves approximability

Reductions: Hierarchy and Closure

Lemma:

$$A \leq_{\log} B \implies A \leq_K B \implies A \leq_C B$$

Reductions: Hierarchy and Closure

Lemma:

$$A \leq_{\log} B \implies A \leq_K B \implies A \leq_C B$$

Proof:

1. $L \subseteq P$
2. compute $f(x)$ and ask oracle

Reductions: Hierarchy and Closure

Lemma:

$A \leq_{\log} B \implies A \leq_K B \implies A \leq_C B$

Proof:

1. $L \subseteq P$
2. compute $f(x)$ and ask oracle

Definition: (Closure under Reduction)

C is closed under reduction $:\iff \quad A \leq B \wedge B \in C \implies A \in C$

Reductions: Hierarchy and Closure

Lemma:

$A \leq_{\log} B \implies A \leq_K B \implies A \leq_C B$

Proof:

1. $L \subseteq P$
2. compute $f(x)$ and ask oracle

Definition: (Closure under Reduction)

C is closed under reduction $:\iff \quad A \leq B \wedge B \in C \implies A \in C$

L, NL, P, NP, $coNP$, $PSPACE$, EXP are closed under $\leq_{\log}$

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
   - run the P^B TM

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \land B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \land B \in P^C \implies A \in P^C$
    - run the P^B TM
    - instead of asking the oracle compute answer with P^C TM

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
   - run the P^B TM
   - instead of asking the oracle compute answer with P^C TM
   - polynomial queries which take polynomial time can be computed in P

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
   - run the P^B TM
   - instead of asking the oracle compute answer with P^C TM
   - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
   - run the P^B TM
   - instead of asking the oracle compute answer with P^C TM
   - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$
3. **Logspace:**

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
    - run the P^B TM
    - instead of asking the oracle compute answer with P^C TM
    - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$
3. **Logspace:**
    - like Karp

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
    - run the P^B TM
    - instead of asking the oracle compute answer with P^C TM
    - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$
3. **Logspace:**
    - like Karp
    - but $f_{AB}(x)$ could be polynomial long

Reductions: Transitivity

Lemma: (Transitivity)

$\leq_C$, $\leq_K$, $\leq_{\log}$, and $\leq_{Levin}$ are transitive.

Proof: $(A \leq B \wedge B \leq C \implies A \leq C)$

1. **Cook:** $A \in P^B \wedge B \in P^C \implies A \in P^C$
    - run the P^B TM
    - instead of asking the oracle compute answer with P^C TM
    - polynomial queries which take polynomial time can be computed in P
2. **Karp:** $f_{AC} = f_{BC} \circ f_{AB}$
3. **Logspace:**
    - like Karp
    - but $f_{AB}(x)$ could be polynomial long
    - $\Rightarrow$ each time f_{BC} needs input compute only this char with f_{AB}

Definition

Definition: (Completeness)

A is complete for $C :\iff A \in C \wedge \forall L \in C : L \leq Ar$
(maximal elements of the preorder given by $\leq$)

Boolean Circuits

Boolean Circuits

Lemma:

For $n > 2$ there is a n-ary boolean function which needs more than $m = \frac{2^n}{2n}$ gates.

Boolean Circuits

Lemma:

For $n > 2$ there is a n-ary boolean function which needs more than $m = \frac{2^n}{2n}$ gates.

Proof: $\left((n+5)m^2\right)^m = \left((n+5)\frac{2^{2n}}{4n^2}\right)^{\frac{2^n}{2n}} < \left(2^{2n}\right)^{\frac{2^n}{2n}} = 2^{2^n}$

Boolean Circuits

Lemma:

For $n > 2$ there is a n-ary boolean function which needs more than $m = \frac{2^n}{2n}$ gates.

Proof: $$\left((n+5)m^2\right)^m = \left((n+5)\frac{2^{2n}}{4n^2}\right)^{\frac{2^n}{2n}} < \left(2^{2n}\right)^{\frac{2^n}{2n}} = 2^{2^n}$$

There is no natural family of boolean functions known, which needs more than linear number of gates.

Boolean Circuits

Lemma:

For $n > 2$ there is a n-ary boolean function which needs more than $m = \frac{2^n}{2n}$ gates.

Proof: $\left((n+5)m^2\right)^m = \left((n+5)\frac{2^{2n}}{4n^2}\right)^{\frac{2^n}{2n}} < \left(2^{2n}\right)^{\frac{2^n}{2n}} = 2^{2^n}$

There is no natural family of boolean functions known, which needs more than linear number of gates.

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Proof:

Give each gate a variable and „translate"

- **variable gate:** $g \iff x$

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Proof:

Give each gate a variable and „translate“

- **variable gate:** $g \iff x$
- **True gate:** g
- **False gate:** $\neg g$

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Proof:

Give each gate a variable and „translate“

- **variable gate:** $g \iff x$
- **True gate:** g
- **False gate:** $\neg g$
- **not gate:** $g \iff \neg h$
- **and gate:** $g \iff a \wedge b$
- **or gate:** $g \iff a \vee b$

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Proof:

Give each gate a variable and „translate“

- **variable gate:** $g \iff x$
- **True gate:** g
- **False gate:** $\neg g$
- **not gate:** $g \iff \neg h$
- **and gate:** $g \iff a \wedge b$
- **or gate:** $g \iff a \vee b$
- **output gate:** g

Boolean Circuits

Lemma:

$CIRCUIT\_SAT \leq_{\log} SAT$

Proof:

Give each gate a variable and „translate“

- **variable gate:** $g \iff x$
- **True gate:** g
- **False gate:** $\neg g$
- **not gate:** $g \iff \neg h$
- **and gate:** $g \iff a \wedge b$
- **or gate:** $g \iff a \vee b$
- **output gate:** g

The conjunction of these clauses is equivalent to the circuit.

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

CIRCUIT_VALUE is *P* *complete*

Lemma:

CIRCUIT_VALUE is *P* *complete*

Proof:

- Having an arbitrary language $L \in P$ decided by a TM M in time n^k and an input x

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- Having an arbitrary language $L \in P$ decided by a TM M in time n^k and an input x
- want to build a boolean circuit that is satisfiable $\iff x \in L \iff M$ accepts x

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- Having an arbitrary language $L \in P$ decided by a TM M in time n^k and an input x
- want to build a boolean circuit that is satisfiable $\iff x \in L \iff M$ accepts x
- W.L.O.G. M has only one string

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- Having an arbitrary language $L \in P$ decided by a TM M in time n^k and an input x
- want to build a boolean circuit that is satisfiable $\iff x \in L \iff M$ accepts x
- W.L.O.G. M has only one string
- interpret the computation on x as a $|x|^{k+1} \times |x|^{k+1}$ computation table with alphabet $\Sigma \cup \Sigma \times K$
- ...

Computation Table

$\sqcup$	$\triangleright$	O_{q_0}	T	t	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_O	t	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t_O	O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	O_O	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	O	$\sqcup_O$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t	$O_{O'}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T	t_{q_r}	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_{q_r}	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	$@_{q_r}$	T	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	T_{q_0}	t	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	t_T	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	t	$@_T$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	$t_{T'}$	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	no	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$	$\triangleright$	@	@	no	@	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...
- a char depends only on the 3 chars above it

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char

$CIRCUIT\_VALUE$ is P $complete$

Lemma:

$CIRCUIT\_VALUE$ is P $complete$

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char
- with polynomial copies of C build a circuit G which computes the table

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char
- with polynomial copies of C build a circuit G which computes the table
- add a circuit which tests for accepting states

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char
- with polynomial copies of C build a circuit G which computes the table
- add a circuit which tests for accepting states
- the leftest and rightest columns are (set to) $\sqcup$ and the input x is known

CIRCUIT_VALUE is *P complete*

Lemma:

CIRCUIT_VALUE is *P complete*

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char
- with polynomial copies of C build a circuit G which computes the table
- add a circuit which tests for accepting states
- the leftest and rightest columns are (set to) $\sqcup$ and the input x is known
- $\Rightarrow$ no free variables occur

CIRCUIT_VALUE is *P* complete

Lemma:

CIRCUIT_VALUE is *P* complete

Proof:

- ...
- a char depends only on the 3 chars above it
- translate it into binary and write a circuit C which computes one char
- with polynomial copies of C build a circuit G which computes the table
- add a circuit which tests for accepting states
- the leftest and rightest columns are (set to) $\sqcup$ and the input x is known
- $\Rightarrow$ no free variables occur
- the value of G is True $\iff$ M accepts x

CIRCUIT_SAT is *NP complete*

Lemma:

CIRCUIT_SAT is *NP complete*

CIRCUIT_SAT is *NP complete*

Lemma:

CIRCUIT_SAT is *NP complete*

Proof:

- W.L.O.G. NTM M has single string and 2 nondeterministic choices (0,1) at each step

CIRCUIT_SAT is *NP complete*

Lemma:

CIRCUIT_SAT is *NP complete*

Proof:

- W.L.O.G. NTM M has single string and 2 nondeterministic choices (0,1) at each step
- a char depends only on the 3 chars above it and the choice

CIRCUIT_SAT is *NP complete*

Lemma:

CIRCUIT_SAT is *NP complete*

Proof:

- W.L.O.G. NTM M has single string and 2 nondeterministic choices (0,1) at each step
- a char depends only on the 3 chars above it and the choice
- build a circuit for the whole computation table

CIRCUIT_SAT is *NP complete*

Lemma:

CIRCUIT_SAT is *NP complete*

Proof:

- W.L.O.G. NTM M has single string and 2 nondeterministic choices (0,1) at each step
- a char depends only on the 3 chars above it and the choice
- build a circuit for the whole computation table
- this circuit is satisfiable $\iff$ an choice assignment exists that leads to an accepting state $\iff$ M accepts x

$HAMILTON\_PATH$ is NP $complete$

Lemma:

$HAMILTON\_PATH$ is NP $complete$

Proof:

$HAMILTON\_PATH \in NP \ \wedge \ SAT \leq_{\log} 3SAT \leq_{\log} HAMILTON\_PATH$

HAMILTON_PATH is *NP complete*

Lemma:

HAMILTON_PATH is *NP complete*

Proof:

$HAMILTON\_PATH \in NP \;\wedge\; SAT \leq_{\log} 3SAT \leq_{\log} HAMILTON\_PATH$

1. guess and verify

$HAMILTON\_PATH$ is NP complete

Lemma:

$HAMILTON\_PATH$ is NP *complete*

Proof:

$HAMILTON\_PATH \in NP \ \wedge \ SAT \leq_{\log} 3SAT \leq_{\log} HAMILTON\_PATH$

1. guess and verify
2. without proof (simple logic)

HAMILTON_PATH is *NP complete*

Lemma:

HAMILTON_PATH is *NP complete*

Proof:

$HAMILTON\_PATH \in NP \ \wedge \ SAT \leq_{\log} 3SAT \leq_{\log} HAMILTON\_PATH$

1. guess and verify
2. without proof (simple logic)
3. given a boolean expression Φ, construct a graph G:
   G has a Hamilton path $\iff$ Φ is satisfiable:
   - each variable $\mapsto$ choice gadget
     (allowing the true or false path to traverse)
   - each clause $\mapsto$ constraint gadget
     (forming a circle iff all variables are false)
   - consistency guaranteed through xor-gadgets
     (substitutes two edges so that only one can be traversed)

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

$TSP(D)$ is NP complete

Lemma:

$TSP(D)$ is NP *complete*

Proof:

$TSP(D) \in NP \;\wedge\; HAMILTON\_PATH \leq_{\log} TSP(D)$

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

Proof:

$TSP(D) \in NP \;\wedge\; HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

Proof:

$TSP(D) \in NP \ \wedge \ HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify
2. given graph G with n nodes, construct a complete weighted graph G' with n nodes and a budget B:

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

Proof:

$TSP(D) \in NP \ \wedge \ HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify
2. given graph G with n nodes, construct a complete weighted graph G' with n nodes and a budget B:
   - edges in G' have weight 1 if they exist in G else 2

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

Proof:

$TSP(D) \in NP \;\wedge\; HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify
2. given graph G with n nodes, construct a complete weighted graph G' with n nodes and a budget B:
   - edges in G' have weight 1 if they exist in G else 2
   - Budget $B = n + 1$

$TSP(D)$ is NP $complete$

Lemma:

$TSP(D)$ is NP $complete$

Proof:

$TSP(D) \in NP \ \wedge \ HAMILTON\_PATH \leq_{\log} TSP(D)$

1. guess and verify
2. given graph G with n nodes, construct a complete weighted graph G' with n nodes and a budget B:
   - edges in G' have weight 1 if they exist in G else 2
   - Budget $B = n + 1$
   - G' has a TSP-Tour with budged $B \iff$ G has a Hamilton path

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x| + 1$ first symbols of its string?

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x| + 1$ first symbols of its string?

Lemma:

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x|+1$ first symbols of its string?

Lemma:

IN_PLACE_ACCEPTANCE is *PSPACE complete*

Proof: $IN\_PLACE\_ACCEPTANCE \in PSPACE \quad \wedge$
$L \in PSPACE \implies L \leq_{\log} IN\_PLACE\_ACCEPTANCE$

1. simulate M on x, count steps and reject $\iff$
   M rejects, leaves the place, or operates more than $|K||x||\Sigma|^{|x|}$ steps

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x| + 1$ first symbols of its string?

Lemma:

IN_PLACE_ACCEPTANCE is *PSPACE complete*

Proof: $IN\_PLACE\_ACCEPTANCE \in PSPACE \quad \wedge$
$L \in PSPACE \implies L \ \leq_{\log} \ IN\_PLACE\_ACCEPTANCE$

1. simulate M on x, count steps and reject $\iff$
M rejects, leaves the place, or operates more than $|K||x||\Sigma|^{|x|}$ steps
2. DTM M decides L in n^k space:
$x \in L \iff M$ accepts x in $|x|^k$ space

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x|+1$ first symbols of its string?

Lemma:

IN_PLACE_ACCEPTANCE is *PSPACE complete*

Proof: $IN\_PLACE\_ACCEPTANCE \in PSPACE \;\; \wedge$
$L \in PSPACE \implies L \;\leq_{\log}\; IN\_PLACE\_ACCEPTANCE$

1. simulate M on x, count steps and reject $\iff$
   M rejects, leaves the place, or operates more than $|K||x||\Sigma|^{|x|}$ steps
2. DTM M decides L in n^k space:
   $x \in L \iff M$ accepts x in $|x|^k$ space
   $\iff M$ accepts $x\sqcup^{|x|^k}$ in place

IN_PLACE_ACCEPTANCE is *PSPACE complete*

IN_PLACE_ACCEPTANCE : Given a DTM M and an input x, does M accept x without ever leaving the $|x|+1$ first symbols of its string?

Lemma:

IN_PLACE_ACCEPTANCE is *PSPACE complete*

Proof: $IN\_PLACE\_ACCEPTANCE \in PSPACE \ \wedge$
$L \in PSPACE \implies L \ \leq_{\log} \ IN\_PLACE\_ACCEPTANCE$

1. simulate M on x, count steps and reject $\iff$
   M rejects, leaves the place, or operates more than $|K||x||\Sigma|^{|x|}$ steps
2. DTM M decides L in n^k space:
   $$\begin{aligned} x \in L &\iff M \text{ accepts x in } |x|^k \text{ space} \\ &\iff M \text{ accepts } x\sqcup^{|x|^k} \text{ in place} \\ &\iff (M, x\sqcup^{|x|^k}) \in IN\_PLACE\_ACCEPTANCE \end{aligned}$$

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^{?} \in FP$ which decides TSP with $TSP(D)$ oracle

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} complete

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^{?} \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} complete

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$
   - $\Rightarrow$ exact cost C can be computed by binary search asking $|x|$ queries
   - test every edge:

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$
   - $\Rightarrow$ exact cost C can be computed by binary search asking $|x|$ queries
   - test every edge:
     - set its cost to $C + 1$

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$
   - $\Rightarrow$ exact cost C can be computed by binary search asking $|x|$ queries
   - test every edge:
     - set its cost to $C + 1$
     - ask $TSP(D)$ oracle whether now an tour with budged C exists
     - reset the cost only if the answer is „no“

Optimization Problems in FP^{NP}

Lemma:

TSP is FP^{NP} *complete*

Proof: TSP is FP^{NP} hard $\wedge$ $TSP \in FP^{NP}$

1. without proof
2. Construct TM $M^? \in FP$ which decides TSP with $TSP(D)$ oracle
   - optimum cost C is an integer between 0 and $2^{|x|}$
   - $\Rightarrow$ exact cost C can be computed by binary search asking $|x|$ queries
   - test every edge:
     - set its cost to $C + 1$
     - ask $TSP(D)$ oracle whether now an tour with budged C exists
     - reset the cost only if the answer is „no“
   - all edges with cost $< C + 1$ form an optimal tour

Optimization Problems in FP^{NP}

Corollary:

$MAXIMUM\_WEIGHTED\_SAT \in FP^{NP}$

Proof: Construct TM $M^{?} \in FP$

Optimization Problems in FP^{NP}

Corollary:

$MAXIMUM\_WEIGHTED\_SAT \in FP^{NP}$

Proof: Construct TM $M^{?} \in FP$

- compute the largest possible weight of satisfied clauses by binary search

Optimization Problems in FP^{NP}

Corollary:

$MAXIMUM\_WEIGHTED\_SAT \in FP^{NP}$

Proof: Construct TM $M^? \in FP$

- compute the largest possible weight of satisfied clauses by binary search
- test each variable one-by-one

Optimization Problems in FP^{NP}

Corollary:

$MAXIMUM\_WEIGHTED\_SAT \in FP^{NP}$

Proof: Construct TM $M^{?} \in FP$

- compute the largest possible weight of satisfied clauses by binary search
- test each variable one-by-one

Corollary:

$WEIGHTED\_MAX\_CUT \in FP^{NP}$

$KNAPSACK \in FP^{NP}$

$WEIGHTED\_BISECTION\_WIDTH \in FP^{NP}$

...

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$$

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P = coNP^{\Sigma_i^P}$

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P = coNP^{\Sigma_i^P}$

$PH = \bigcup_i \ \Sigma_i^P$ is called polynomial hierarchy

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P = coNP^{\Sigma_i^P}$

$PH = \bigcup_i \Sigma_i^P$ is called polynomial hierarchy

$$\Delta_1^P = P^P = P \qquad \Sigma_1^P = NP^P = NP \qquad \Pi_1^P = coNP^P = coNP$$

Polynomial Hierarchy

Definition: (Polynomial Hierarchy)

$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$

and for all $i \geq 0$:

- $\Delta_{i+1}^P = P^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$
- $\Pi_{i+1}^P = coNP^{\Sigma_i^P}$

$PH = \bigcup_i \ \Sigma_i^P$ is called polynomial hierarchy

$\Delta_1^P = P^P = P$	$\Sigma_1^P = NP^P = NP$	$\Pi_1^P = coNP^P = coNP$
$\Delta_2^P = P^{NP}$	$\Sigma_2^P = NP^{NP}$	$\Pi_2^P = coNP^{NP}$
...	...	...

Definitions

Definition (polynomial bounded relation)

A polynomial bounded relation is a relation $R \subseteq (\Sigma^*)^{l+1}$ with $\exists k \in \mathbf{N} : \ \forall (x, y_1, y_2, ..., y_l) \in R : \ |y_i| \leq |x|^k$.

Definitions

Definition (polynomial bounded relation)

A polynomial bounded relation is a relation $R \subseteq (\Sigma^*)^{l+1}$ with $\exists k \in \mathbf{N} : \ \forall (x, y_1, y_2, ..., y_l) \in R : \ |y_i| \leq |x|^k$.

Definition (C-verifiable relation)

A C-verifiable relation R is a polynomial bounded relation, which is decidable in C: $\{x; y_1; y_2; ...; y_l \mid (x, y_1, y_2, ..., y_l) \in R\} \in C$

Characterization of NP

Lemma: (Characterization of NP)

$$NP = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } P\text{-verifiable}\}$$

Characterization of NP

Lemma: (Characterization of NP)

$$NP = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } P\text{-verifiable}\}$$

Proof:

„$\Leftarrow$“: R is P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in NP$

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \text{ is } P\text{-verifiable}\}$

Proof:

„$\Leftarrow$“: R is P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in NP$

- construct NTM M' which on input x
  - guesses polynomial bounded y
  - verify whether $(x,y) \in R$
  - accept $x \iff (x,y) \in R$

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } P\text{-verifiable}\}$

Proof:

„$\Leftarrow$“: R is P-verifiable $\Longrightarrow \{x \mid \exists y : (x, y) \in R\} \in NP$

- construct NTM M' which on input x
  - guesses polynomial bounded y
  - verify whether $(x, y) \in R$
  - accept $x \iff (x, y) \in R$
- $M' \in NP$

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{\ \{x \mid \exists y : (x, y) \in R\} \ \mid\ R \text{ is } P\text{-verifiable}\}$

Proof:

„$\Leftarrow$“: R is P-verifiable $\Longrightarrow \{x \mid \exists y : (x, y) \in R\} \in NP$

- construct NTM M' which on input x
  - guesses polynomial bounded y
  - verify whether $(x, y) \in R$
  - accept $x \iff (x, y) \in R$
- $M' \in NP$
- M' accepts $x \iff \exists y : (x, y) \in R$

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \Longrightarrow \exists R$ P-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{\ \{x \mid \exists y : (x, y) \in R\} \ \mid\ R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \implies \exists R\ P\text{-verifiable} :\ L = \{x \mid \exists y : (x, y) \in R\}$

- have TM $M \in NP$ deciding L

Characterization of NP

Lemma: (Characterization of NP)

$$NP = \{\ \{x \mid \exists y : (x,y) \in R\} \ \mid\ R \text{ is } P\text{-verifiable}\}$$

Proof:

„⇒“: $L \in NP \implies \exists R\ P\text{-verifiable}: \ L = \{x \mid \exists y : (x,y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \implies \exists R\ P\text{-verifiable} :\ L = \{x \mid \exists y : (x, y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y
- $R = \{(x, y) \mid y \text{ is witness for x}\}$ is the searched relation

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \Longrightarrow \exists R$ P-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y
- $R = \{(x, y) |\ y$ is witness for x$\}$ is the searched relation
  - polynomial bounded y (because of the polynomial running time of M)

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \Longrightarrow \exists R$ P-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y
- $R = \{(x, y) \mid y$ is witness for x$\}$ is the searched relation
  - polynomial bounded y (because of the polynomial running time of M)
  - polynomial decidable (by DTM M' using y to determine the computation path of M)

Characterization of NP

Lemma: (Characterization of NP)

$NP = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } P\text{-verifiable}\}$

Proof:

„⇒“: $L \in NP \Longrightarrow \exists R$ P-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have TM $M \in NP$ deciding L
- for input $x \in L$ encode the choices of an accepting path of M into a witness y
- $R = \{(x,y) \mid y$ is witness for x$\}$ is the searched relation
  - polynomial bounded y (because of the polynomial running time of M)
  - polynomial decidable (by DTM M' using y to determine the computation path of M)
  - $\exists y : (x,y) \in R \iff M$ accepts $x \iff x \in L$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\} \ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$$\Sigma_i^P = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$$

Proof: (by induction on i)

$i = 1$: exactly the characterization of NP

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$$\Sigma_i^P = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$$

Proof: (by induction on i)

$i = 1$: exactly the characterization of NP

$(i-1) \to i$:

„$\Leftarrow$“: R is Π_{i-1}^P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in \Sigma_i^P$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof: (by induction on i)

$i = 1$**:** exactly the characterization of NP

$(i-1) \to i$**:**

„$\Leftarrow$“: R is Π_{i-1}^P-verifiable $\Longrightarrow \{x \mid \exists y : (x, y) \in R\} \in \Sigma_i^P$

- construct NTM $M^? \in NP$ which on input x
  - guesses polynomial bounded y
  - aks an oracle $K \in \Sigma_{i-1}^P$ whether $(x, y) \in R$
  - accepts $x \iff (x, y) \in R$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof: (by induction on i)

$i = 1$**:** exactly the characterization of NP

$(i-1) \to i$**:**

„$\Leftarrow$“: R is Π_{i-1}^P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in \Sigma_i^P$

- construct NTM $M^? \in NP$ which on input x
    - guesses polynomial bounded y
    - aks an oracle $K \in \Sigma_{i-1}^P$ whether $(x,y) \in R$
    - accepts $x \iff (x,y) \in R$
- $M^K \in \Sigma_i^P$ (since $\Sigma_{i-1}^P \subseteq \Sigma_i^P$)

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof: (by induction on i)

$i = 1$: exactly the characterization of NP

$(i-1) \to i$:

„$\Leftarrow$“: R is Π_{i-1}^P-verifiable $\Longrightarrow \{x \mid \exists y : (x,y) \in R\} \in \Sigma_i^P$

- construct NTM $M^? \in NP$ which on input x
  - guesses polynomial bounded y
  - aks an oracle $K \in \Sigma_{i-1}^P$ whether $(x,y) \in R$
  - accepts $x \iff (x,y) \in R$
- $M^K \in \Sigma_i^P$ (since $\Sigma_{i-1}^P \subseteq \Sigma_i^P$)
- M^K accepts $x \iff \exists y : (x,y) \in R$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P\text{-verifiable} : \ L = \{x \mid \exists y : (x, y) \in R\}$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„$\Rightarrow$“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x,y) \mid\ y$ is certificate for $x\}$

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x,y) \mid y$ is certificate for $x\}$
  - R is polynomial bounded

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x, y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x, y) \mid y \text{ is certificate for } x\}$
  - R is polynomial bounded
  - $x \notin K$ is Π_{i-1}^P-decidable

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„$\Rightarrow$“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x,y) \mid\ y$ is certificate for $x\}$
  - R is polynomial bounded
  - $x \notin K$ is Π_{i-1}^P-decidable
  - $x \in K$ is Π_{i-2}^P-verifiable (by induction)

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x, y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x, y) |\ y$ is certificate for $x\}$
  - R is polynomial bounded
  - $x \notin K$ is Π_{i-1}^P-decidable
  - $x \in K$ is Π_{i-2}^P-verifiable (by induction)
  - $\Rightarrow R$ is Π_{i-1}^P-verifiable

Characterization of Σ_i^P

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x,y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Proof:

„⇒“: $L \in \Sigma_i^P \implies \exists R\ \Pi_{i-1}^P$-verifiable : $L = \{x \mid \exists y : (x,y) \in R\}$

- have NTM $M^? \in NP^?$ deciding L with oracle $K \in \Sigma_{i-1}^P$
- for input $x \in L$ encode all choices and queries of $M^?$ into a certificate y of x (example: $y = (c_0,\ c_4,\ qs_1 \notin K,\ c_1,\ qs_2 \in K +$ cert, ...))
- define $R = \{(x,y) \mid\ y$ is certificate for $x\}$
  - R is polynomial bounded
  - $x \notin K$ is Π_{i-1}^P-decidable
  - $x \in K$ is Π_{i-2}^P-verifiable (by induction)
  - $\Rightarrow R$ is Π_{i-1}^P-verifiable

Characterization of PH

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{\ \{x \mid \exists y : (x, y) \in R\}\ \mid\ R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Corollary: (Characterization of Π_i^P)

$\Pi_i^P = \{\ \{x \mid \forall y :\ |y| < |x|^k \Rightarrow (x, y) \in R\ \}\ \mid\ R \text{ is } \Sigma_{i-1}^P\text{-verifiable}\}$

Characterization of PH

Lemma: (Characterization of Σ_i^P)

$\Sigma_i^P = \{ \{x \mid \exists y : (x,y) \in R\} \mid R \text{ is } \Pi_{i-1}^P\text{-verifiable}\}$

Corollary: (Characterization of Π_i^P)

$\Pi_i^P = \{ \{x \mid \forall y : |y| < |x|^k \Rightarrow (x,y) \in R \} \mid R \text{ is } \Sigma_{i-1}^P\text{-verifiable}\}$

Corollary:

$L \in \Sigma_i^P \iff$
$\exists R : R \text{ is } P\text{-verifiable} \wedge L = \{x \mid \exists y_1 \forall y_2 \exists y_3 \ldots : (x, y_1, y_2, \ldots, y_i) \in R\}$

$L \in \Pi_i^P \iff$
$\exists R : R \text{ is } P\text{-verifiable} \wedge L = \{x \mid \forall y_1 \exists y_2 \forall y_3 \ldots : (x, y_1, y_2, \ldots, y_i) \in R\}$

Problems in PH

MINIMUM_CIRCUIT : Given a Boolean circuit C, is it true that there is no circuit with fewer gates computing the same Boolean function?

Problems in PH

MINIMUM_CIRCUIT : Given a Boolean circuit C, is it true that there is no circuit with fewer gates computing the same Boolean function?

Lemma:

$MINIMUM\_CIRCUIT \in \Pi_2^P$

Problems in PH

MINIMUM_CIRCUIT : Given a Boolean circuit C, is it true that there is no circuit with fewer gates computing the same Boolean function?

Lemma:

$MINIMUM\_CIRCUIT \in \Pi_2^P$

Proof:

- C is accepted $\iff \forall C' : \ |C'| < |C| : \ \exists$ input $x : \ C'(x) \neq C(x)$
- and $C'(x) \neq C(x)$ can be checked in polynomial time

Problems in PH

MINIMUM_CIRCUIT : Given a Boolean circuit C, is it true that there is no circuit with fewer gates computing the same Boolean function?

Lemma:

$MINIMUM\_CIRCUIT \in \Pi_2^P$

Proof:

- C is accepted $\iff \forall C' : \; |C'| < |C| : \; \exists$ input $x : \; C'(x) \neq C(x)$
- and $C'(x) \neq C(x)$ can be checked in polynomial time

$QSAT_i$: Decide whether a quantified boolean expression with i alternations of quantifiers (beginning with an existential quantifier) is satisfiable

Lemma:

$QSAT_i$ is Σ_i^P complete

PH collapses

Definition: (Collapse of PH)

PH collapses to the *i*th level means: $\forall j > i: \; \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

PH collapses

Definition: (Collapse of PH)

PH collapses to the *i*th level means: $\forall j > i : \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the *i*th level.

PH collapses

Definition: (Collapse of PH)

PH collapses to the ith level means: $\forall j > i: \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the ith level.

Proof: $\Sigma_i^P = \Pi_i^P \implies \Sigma_{i+1}^P = \Sigma_i^P$

PH collapses

Definition: (Collapse of PH)

PH collapses to the *i*th level means: $\forall j > i: \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the *i*th level.

Proof: $\Sigma_i^P = \Pi_i^P \implies \Sigma_{i+1}^P = \Sigma_i^P$

$L \in \Sigma_{i+1}^P \iff L = \{x \mid \exists y: \ (x, y) \in R\}$ with R is Π_i^P-verifiable

PH collapses

Definition: (Collapse of PH)

PH collapses to the ith level means: $\forall j > i : \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the ith level.

Proof: $\Sigma_i^P = \Pi_i^P \Longrightarrow \Sigma_{i+1}^P = \Sigma_i^P$

$$L \in \Sigma_{i+1}^P \iff L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Pi_i^P\text{-verifiable}$$
$$\iff L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Sigma_i^P\text{-verifiable}$$

PH collapses

Definition: (Collapse of PH)

PH collapses to the *i*th level means: $\forall j > i: \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the *i*th level.

Proof: $\Sigma_i^P = \Pi_i^P \Longrightarrow \Sigma_{i+1}^P = \Sigma_i^P$

$$\begin{aligned} L \in \Sigma_{i+1}^P \quad &\Longleftrightarrow \quad L = \{x \mid \exists y: \ (x,y) \in R\} \text{ with } R \text{ is } \Pi_i^P\text{-verifiable} \\ &\Longleftrightarrow \quad L = \{x \mid \exists y: \ (x,y) \in R\} \text{ with } R \text{ is } \Sigma_i^P\text{-verifiable} \\ &\Longleftrightarrow \quad L = \{x \mid \exists y: \ (x,y) \in R\} \text{ with} \\ &\qquad \left[(x,y) \in R \iff \exists z: (x,y,z) \in S \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable}\right] \end{aligned}$$

PH collapses

Definition: (Collapse of PH)

PH collapses to the ith level means: $\forall j > i: \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the ith level.

Proof: $\Sigma_i^P = \Pi_i^P \Longrightarrow \Sigma_{i+1}^P = \Sigma_i^P$

$$
\begin{aligned}
L \in \Sigma_{i+1}^P \iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Pi_i^P\text{-verifiable} \\
\iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Sigma_i^P\text{-verifiable} \\
\iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with} \\
& \left[(x,y) \in R \iff \exists z : (x,y,z) \in S \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable}\right] \\
\iff & L = \{x \mid \exists y, z : \ (x,y,z) \in S\} \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable}
\end{aligned}
$$

PH collapses

Definition: (Collapse of PH)

PH collapses to the ith level means: $\forall j > i : \ \Sigma_j^P = \Pi_j^P = \Delta_j^P = \Sigma_i^P$

Lemma: (Collapse of PH)

If for some $i \leq 1$ $\Sigma_i^P = \Pi_i^P$ then PH collapses to the ith level.

Proof: $\Sigma_i^P = \Pi_i^P \implies \Sigma_{i+1}^P = \Sigma_i^P$

$$
\begin{aligned}
L \in \Sigma_{i+1}^P \iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Pi_i^P\text{-verifiable} \\
\iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with } R \text{ is } \Sigma_i^P\text{-verifiable} \\
\iff & L = \{x \mid \exists y : \ (x,y) \in R\} \text{ with} \\
& \big[(x,y) \in R \iff \exists z : (x,y,z) \in S \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable}\big] \\
\iff & L = \{x \mid \exists y, z : \ (x,y,z) \in S\} \text{ with } S \text{ is } \Pi_{i-1}^P\text{-verifiable} \\
\iff & L \in \Sigma_i^P
\end{aligned}
$$

PH collapses

Corollary: (*PH* `complete Problems`)

If *PH* has complete problems, then it collapses to some finite level.

PH collapses

Corollary: (PH complete Problems)

If PH has complete problems, then it collapses to some finite level.

Corollary: (PH and $PSPACE$)

$PH \subseteq PSPACE$ and $PH = PSPACE \Longrightarrow PH$ collapses

PH collapses

Corollary: (*PH* complete Problems)

If *PH* has complete problems, then it collapses to some finite level.

Corollary: (*PH* and *PSPACE*)

$PH \subseteq PSPACE$ and $PH = PSPACE \implies PH$ collapses

Proof:

1. trivial
2. *PSPACE* has complete problems

References

O. Goldreich:
Introduction to Complexity Theory
Lecture Notes, 1999.

Markus Bläser:
Complexity Theory
Lecture Notes, 2005.

Clay Mathematics Institute, Cambridge, Massachusetts:
P vs NP Problem
http://www.claymath.org/millennium/P_vs_NP/.

Christos H. Papadimitriou:
Computational Complexity.
Addison Wesley, 1994.

THANK YOU