Efficient Algorithms and Problem Complexity
– Reductions, NP-completeness, and Cook’s Theorem –

Frank Drewes
Department of Computing Science
Umeå University
Today's Menu

1. Reductions
2. NP-Completeness
3. Cook's Theorem
What are We Aiming at?

- We suspect some problems in NP to be significantly harder than each of the problems in P.
- How should we define significance (or insignificance) in this context?
- The definition should consider
  1. the difference between problems in P to be insignificant, but
  2. the difference between problems in and outside P to be significant.
- The Composition Theorem gives us a strong hint: We can apply any polynomial-time preprocessing to a problem in P without leaving P!
- The preprocessing may in particular translate one problem into another.
- Such translations are called polynomial-time reductions.
Polynomial-Time Reductions

Definition (polynomial-time reduction)
Let \( A \) and \( B \) be decision problems. A polynomial-time reduction from \( A \) to \( B \)

is a function \( f \) such that

1. for every input \( x, x \in A \iff f(x) \in B \), and
2. \( f \) can be computed in (deterministic!) polynomial time.

We say that \( A \) is polynomial-time reducible to \( B \), and write \( A \leq_p B \).

Note: \( \leq_p \) is a preorder (also called a quasi-order), that is,

- it is transitive (\( A \leq_p B \leq_p C \) implies \( A \leq_p C \)) [WHY?]
- it is reflexive (\( A \leq_p A \)) [WHY?].
So, What Does this Intuitively Mean?

Importantly, \( \leq_p \) is not acyclic:

For \( A \neq B \) we may have both \( A \leq_p B \) and \( B \leq_p A \). We write this as \( A \equiv_p B \) and say that \( A \) and \( B \) are polynomial-time equivalent.

- \( A \leq_p B \) formalizes that \( A \) is not significantly harder than \( B \) – the difference is only a polynomial preprocessing.
- In particular, \( A \equiv_p B \) means that the difference is only polynomial in either direction – they are of the same complexity if we disregard polynomial preprocessing.
- On the other hand, if \( A \leq_p B \) but \( B \not\leq_p A \), then \( B \) is indeed significantly harder than \( A \), i.e., the difference is more than “just” a polynomial.
Insignificance of the Difference Between Problems in P

Now, is “the difference between problems in P insignificant”?

We want that a polynomial-time reduction from $A$ to $B$ exists if $A, B \in P$.

Here is such a reduction $f$: Choose any $y^+ \in B$ and $y^- \notin B$, and define

$$f(x) = \begin{cases} 
  y^+ & \text{if } x \in A \\
  y^- & \text{otherwise.}
\end{cases}$$

- **Checking requirement 1**: by the definition of $f$, $x \in A \iff f(x) \in B$.
- **Checking requirement 2**: computing $f$ in polynomial time works by deciding whether $x \in A$ and outputting either $y^+$ or $y^-$. 
- Intuitively, the reduction does all the work itself.
- It can do so, because it has all the resources $A$ requires.
- There are exactly two problems $B$ for which this does not work. Which ones?
Significance of the Difference Between Problems In and Outside P

Theorem (P is (backwards) closed under reductions)

If $A \leq_P B$ and $B \in P$ then $A \in P$.

In other words, if $B \in P$ but $A \notin P$ then $A \not\leq_P B$.

Proof: Let $M$ decide $B$ and let $M_0$ compute a reduction $f$ from $A$ to $B$, both in polynomial time.

- $M' = M \circ M_0$ runs in polynomial time (Composition Theorem).
- We have $M'(x) = M(M_0(x)) = M(f(x))$. Therefore,

  $$x \in A \iff f(x) \in B \iff M(f(x)) = 1 \iff M'(x) = 1.$$ 

- Thus, $M'$ decides $A$ in polynomial time.
NP-Completeness

We ultimately want to understand whether there are problems in \( \text{NP} \) that are not in \( \text{P} \). So, it makes sense to look at the hardest problems in \( \text{NP} \).

**Definition (NP-complete)**

A decision problem \( B \) is **NP-complete** if

- \( B \in \text{NP} \) and
- \( A \leq_p B \) for every problem \( A \in \text{NP} \).

- If the first condition is dropped, we say that \( B \) is **NP-hard**.
- For classes above \( \text{NP} \) (such as \( \text{EXP} \)), completeness is defined similarly.
- For \( \text{P} \) and classes inside \( \text{P} \), we would need another type of reduction (because of the reasoning on Slide 6).
To prove $P = NP$, it suffices to solve one NP-complete problem efficiently.

**Theorem**

If there is an NP-complete problem $B$ such that $B \in P$, then $P = NP$.

**Proof:** Since $B$ is NP-complete, we have $A \leq_p B$ for all $A \in NP$. By the closedness of $P$ under reductions (Slide 7), this means that $A \in P$. 
Proving NP-Completeness

Lemma

If \( A \in \text{NP} \) and \( B \leq_p A \) for an NP-complete problem \( B \), then \( A \) is NP-complete.

Proof: This is because, for every \( C \in \text{NP} \), we have \( C \leq_p B \leq_p A \).

This yields the most common way to prove that \( A \) is NP-complete:

1. show that \( A \in \text{NP} \),
2. choose a suitable NP-complete problem \( B \), so that you manage to find a reduction \( f \) from \( B \) to \( A \) (note the direction!!!),
3. argue that \( f \) is polynomial-time computable,
4. show that \( f(x) \in A \) iff \( x \in B \) (both directions!!!).
Cook’s Theorem

For the method on the previous slide to be useful, someone has to find a first NP-complete problem...

Theorem (Cook 1971)
SAT is NP-complete.

The proof...
- shows how to construct a polynomial-time reduction from $A$ to SAT, for arbitrary $A \in \text{NP}$,
- cannot be based on anything else than the existence of a nondeterministic polynomial-time decision algorithm for $A$.

Rather than nRAMs, we use Turing machines (TMs) for the proof sketch, because they are simpler.
Proof Sketch of Cook’s Theorem

We use the characterization of \( \text{NP} \) by polynomially bounded binary relations. Let \( M \) be a deterministic TM and \( p, q \) polynomials such that

- \( M \) accepts only strings of the form \( x \# y \), where \( y \in \{0, 1\}^{p(n)} \) \( (n = |x|) \),
- \( x \in A \) if and only if \( \exists y \in \{0, 1\}^{p(n)} \) with \( M(x \# y) = \text{yes} \), and
- \( M \) runs in at most \( q(n) \) steps.
- When \( M \) halts (after \( q(n) \) steps), its head scans the first tape cell.
- \( M \) accepts the input by writing a special symbol \( \alpha \) in the first tape cell and entering a special accepting state \( z_{yes} \).

**Major proof step:** From \( x \# y \) we construct, in polynomial time, a Boolean formula (in CNF) that encodes the entire computation of \( M \) with this input.
A computation of $M$ as a $q(n) \times q(n)$-array of symbols:

$$
\begin{array}{cccccccc}
\alpha 
& b & b & \cdots & \# & 0 & 1 & \cdots \\
\hline
a & b & b & \cdots & \# & 0 & 1 & \cdots \\
\hline
\cdots & b & a & \cdots & \# & 0 & 1 & \cdots \\
\hline
\cdots & b & b & \cdots & \# & 0 & 1 & \cdots \\
\hline
\alpha 
& \text{yes} 
& \cdots
\end{array}
$$

- Encode contents of cell $C_{ij}$ by $k$ boolean variables $\overline{x}_{ij} = x_{ij}^1, \ldots, x_{ij}^k$.
- Row 1 should contain the input
  $\Rightarrow$ we use formulas $\varphi^j_b \equiv \text{“} \overline{x}_{1j} \text{ represents } b \text{”}$.
- Contents of $C_{i+1j}$ is given by contents of $C_{ij-1}, C_{ij}, C_{ij+1}$
  $\Rightarrow$ we use $\varphi^i_j \equiv \text{“} \overline{x}_{i+1j} \text{ represents the successor of } \overline{x}_{ij-1}, \overline{x}_{ij}, \overline{x}_{ij+1} \text{”}$.
- Cell $C_{q(n)1}$ must indicate acceptance
  $\Rightarrow$ we use $\varphi_{\text{yes}} \equiv \text{“} \overline{x}_{q(n)1} \text{ represents } \alpha^{\text{yes}} \text{”}$.
Proof Sketch of Cook’s Theorem

Computing the reduction for input $x \# y = b_1 \cdots b_m$:

1. Compute $q(n)$.
2. Print “$\varphi_1^{b_1} \land \cdots \land \varphi_m^{b_m} \land \varphi_{m+1}^{b_m} \cdots \varphi_{q(n)}^{b_m} \land$ blanks”.
3. For $i = 2, \ldots, q(n)$ and $j = 1, \ldots, q(n)$ print “$\land \varphi_i^j$”.
4. Print “$\land \varphi_{\text{yes}}$”.

$\Rightarrow$ the resulting formula $\varphi$ is satisfiable if and only if $M$ accepts $x \# y$.

But we have only $x$ and want $\varphi$ to be satisfiable if there exists such a $y$!

Easy solution: In Step 2 above, print $\varphi_i^j$ only for the initial part $x \#$. Leave the remaining values of variables in row 1 unspecified.