Reductions and hardness

Cook’s theorem

Reductions

COMP36111: Advanced Algorithms I
Lecture 8: Hardness and Reductions

Ian Pratt-Hartmann

Room KB2.38: email: ipratt@cs.man.ac.uk

2016–17
• Reading for this lecture:
  • Sipser: Chapter 7.
Outline

Reductions and hardness
- Reductions
  - Transitivity of reductions
  - Hardness and completeness

Cook’s theorem
- Cook’s theorem

Some easy reductions
- 3-SAT
- Integer linear programming
Reductions

• Recall the problems SAT and $k$-SAT

**SAT**
Given: A set of clauses $\Gamma$
Return: $Y$ if $\Gamma$ is satisfiable, and $N$ otherwise

**$k$-SAT**
Given: A set of clauses $\Gamma$ each of which has at most $k$ literals.
Return: $Y$ if $\Gamma$ is satisfiable, and $N$ otherwise.

• *Prima facie*, SAT looks harder than $k$-SAT. But is it?
• Let $P_1, P_2$ be problems over alphabets $\Sigma_1, \Sigma_2$, respectively.

• We say $P_1$ is (many-one logspace) reducible to $P_2$ if there is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that: (i) $f$ can be computed by a deterministic TM using at most $\log n$ space on any work tape; and (ii) for all $x \in \Sigma_1^*$, $x \in P_1$ if and only if $f(x) \in P_2$.

• In this case, we write

$$P_1 \leq_{\log}^m P_2$$

• We think of $P_1 \leq_{\log}^m P_2$ as stating any of the following:
  • $P_2$ is at least as hard as $P_1$;
  • $P_1$ is no harder than $P_2$;
  • if anyone shows me an easy way of solving $P_2$, I have an easy way of solving $P_1$. 
• Such reductions provide a way of showing that a problem is in a complexity class, because (sensible) complexity classes, such as

\textbf{LogSpace}, \textbf{NLogSpace}, \textbf{PTime}, \textbf{NPTime}, \ldots

are closed under many-one logspace reductions.

• \textbf{Warning}: Classes such as \textbf{TIME}(n), \textbf{TIME}(n^2) etc. are not closed under many-one logspace reductions.
Outline

Reductions and hardness
- Reductions
  - Transitivity of reductions
- Hardness and completeness

Cook’s theorem
- Cook’s theorem

Some easy reductions
- 3-SAT
- Integer linear programming
Furthermore, reducibility is a transitive relation, as the next theorem shows.

**Theorem**

If \( f_1 : \Sigma_1^* \rightarrow \Sigma_2^* \) and \( f_2 : \Sigma_2^* \rightarrow \Sigma_3^* \) are both computable in logarithmic space, then so is \( f_2 \circ f_1 : \Sigma_1^* \rightarrow \Sigma_3^* \).

The following picture is **not** a proof!
• Here is a Turing machine that will compute $f_2 \circ f_1$ in logarithmic space:

  calculate the first bit of $f_1(x)$
  keep a counter to say which bit this is—initially 1
  start a simulation of $f_2(f_1(x))$, using the calculated bit
  if the simulation of $f_2$ asks to move the read head to the right
    calculate next bit of $f_1(x)$
    write it on top of the current bit
    update the output bit counter
  if the simulation of $f_2$ asks to move the read head to the left
    restart the calculation of $f_1(x)$
  continue until the required output bit is calculated
  write it on top of the current bit
  update the output bit counter
• A weaker notion of reduction is commonly encountered in textbooks (e.g. Sipser).

• Denote by $\mathbf{P}$ the set of functions $\{n^c \mid c > 0\}$.

• Let $P_1, P_2$ be problems over alphabets $\Sigma_1, \Sigma_2$, respectively.

• We say $P_1$ is (many-one polytime) reducible to $P_2$ if there is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$, in $\text{TIME}(\mathbf{P})$ such that, for all $x \in \Sigma_1^*$, $x \in P_1$ if and only if $f(x) \in P_2$.

• In this case, we write

$$P_1 \leq^p_m P_2$$
Many-one logspace reducibility is at least as strong as many-one polytime reducibility.

Many-one polytime reducibility is obviously transitive. (Ask if you do not understand this.)

However, many-one logspace reducibility is theoretically a bit more useful.

In practice, most encountered instances of many-one polytime reducibility are in fact instances of many-one logspace reducibility.

We shall always use many-one logspace reducibility unless explicitly stated otherwise.
Outline

Reductions and hardness
  Reductions
  Transitivity of reductions
  Hardness and completeness

Cook’s theorem
  Cook’s theorem

Some easy reductions
  3-SAT
  Integer linear programming
• It turns out that, for certain complexity classes $\mathcal{C}$, and certain problems $P$, every problem $P' \in \mathcal{C}$ is reducible to $P$.
• That is, $P$ is at least as hard as every problem in $\mathcal{C}$.
• Of particular interest is where the problem $P$ is itself a member of $\mathcal{C}$.
• Much of the attraction of complexity theory arises from the existence of such problems.
Definition
Let $C$ be a complexity class and $P$ a problem. We say that $P$ is $C$-hard (under many-one logspace reducibility) if, for all $P' \in C$, $P' \leq_m^\log P$.
We say that $P$ is $C$-complete (umolsr) if, $P \in C$ and $P$ is $C$-hard (umolsr).
Outline

Reductions and hardness
  Reductions
  Transitivity of reductions
  Hardness and completeness

Cook’s theorem
  Cook’s theorem

Some easy reductions
  3-SAT
  Integer linear programming
Theorem (Cook)

*SAT* *is* **NPTIME-complete**.
Proof.
Suppose $\mathcal{P}$ is any problem in $\text{NPTime}$. Let $M$ be a TM accepting $\mathcal{P}$, with running time bounded by $p(n)$. For simplicity, let us assume $M$ has just one tape. Thus, $M$ has the form

$$\langle \Sigma, Q, s^*, T \rangle,$$

where $\Sigma$ is the alphabet of $\mathcal{P}$, $Q$ is the set of states, $s^*$ the halting state and $T$ the set of transitions.

Each transition $\tau \in T$ has the form

$$\tau = \langle s, a, t, b, \delta \rangle,$$

where $s, t \in Q$ are states, $a, b \in \Sigma \cup \{\sqcup, \triangleright\}$, and $\delta \in \{-1, 0, 1\}$ indicating ‘left’, ‘stay’ or ‘right’. 

\[\square\]
Proof.
We picture the operation of $M$ as

and encode any run using the proposition letters

$p_{i,j}^a$: tape square $i$ contains symbol $a$ at time $j$
$h_{i,j}$: the head is over tape square $i$ at time $j$
$q_j^s$: the state is $s$ at time $j$.
$t_{i,j}^\tau$: transition $\tau$ is executed at time $j$ with head on tape square $i$. 
Proof.

We write clauses saying that, at each time, the head is somewhere

$$\{ h_{1,j} \lor \cdots \lor h_{p(n),j} \mid 1 \leq j \leq p(n) \}$$

and is not in two places at once

$$\{ \neg h_{i,j} \lor \neg h_{i',j} \mid 1 \leq i < i' \leq p(n), 1 \leq j \leq p(n) \}$$

and so on. We write clauses saying that the input is $x[1], \ldots, x[n]$ (remember $\square$ is the blank symbol):

$$\{ p_{i,1}^x \mid 1 \leq i \leq n \}$$

$$\{ p_{i,1}^\square \mid n + 1 \leq i \leq p(n) \}$$

and so on. (proof TBC . . . )
Proof.
Further, we write clauses specifying when a transition of $M$ may be executed. For all $i, j$ ($1 \leq i, j \leq p(n)$), and for all $a \in \Sigma \cup \{\sqcup, \sqsupset\}$, we take $\Gamma_x$ to contain the (big) clause

$$
\neg q^s_j \lor \neg h_{i,j} \lor \neg p^a_{i,j} \lor \bigvee \{ t^T_{i,j} \mid \tau = \langle s, a, t, b, \delta \rangle \in T \}
$$

listing the allowed transitions $M$ may make. Note that $M$ is a non-deterministic TM!
Proof.
And we write clauses specifying the effects of transitions:

\[
\begin{align*}
\{ \neg t_{i,j}^T \lor p_{i,j+1}^b & \mid 1 \leq i, j \leq p(n), \tau = \langle s, a, t, b, \delta \rangle \} \\
\{ \neg t_{i,j}^T \lor q_{j+1}^t & \mid 1 \leq i, j \leq p(n), \tau = \langle s, a, t, b, \delta \rangle \} \\
\{ \neg t_{i,j}^T \lor h_{i+\delta,j+1} & \mid 1 \leq i, j \leq p(n), \tau = \langle s, a, t, b, \delta \rangle \}.
\end{align*}
\]

Actually, there are some complications here when the tape head is over the leftmost square. Can you fix this formula?
Proof.
And we write clauses saying that $M$ accepts the input:

$$\{q_{p(n)}^{s^*}, p_{1,p(n)}^\gamma \} \cup \{p_{i,p(n)}^\cup \mid 2 \leq i \leq p(n)\},$$

where $s^*$ is the halting state.

Call the resulting set of clauses $\Gamma_x$.
There are a few additional clauses in $\Gamma_x$ that I have not mentioned; but it is routine to fill them in. (proof TBC . . . )
Proof.

It is easy to see that $\Gamma_x$ is satisfiable iff $M$ accepts $x$; hence $\Gamma_x$ is satisfiable iff $x \in P$.

It is also ‘easy’ to see that, from a description of $x$, we can compute the set of clauses $\Gamma_M$ using at most $\log n$ amount of workspace, where $n = |x|$. (Remember: the parameters of $M$ are constant here; the only variable input is $x$.)

Thus, the function $x \mapsto \Gamma_x$ shows that $P \leq_{m}^{\log} \text{SAT}$, as required.
• It is completely trivial that 3-SAT is no harder than SAT.
• Slightly surprising is that the reverse condition holds: SAT is no harder than 3-SAT!
• Notice that this means that 3-SAT is \( \text{NPTime} \)-complete.
• For suppose \( \mathcal{P} \) is a problem in \( \text{NPTime} \). We have

\[
\mathcal{P} \leq^\log_m \text{SAT} \leq^\log_m \text{3-SAT}
\]

and the result follows by the transitivity of \( \leq^\log_m \).
Outline

Reductions and hardness
  Reductions
  Transitivity of reductions
  Hardness and completeness

Cook’s theorem
  Cook’s theorem

Some easy reductions
  3-SAT
  Integer linear programming
Theorem
3-SAT is \( \text{NPTime-complete} \)

Proof.
We show that \( \text{SAT} \leq^\log_m 3\text{-SAT} \).

Suppose we are given a set of clauses \( \Gamma \). We show how to compute a set of 3-literal clauses \( \Gamma' \) such that \( \Gamma \) is satisfiable iff \( \Gamma' \) is satisfiable.

Pick any \( (\ell_1 \lor \cdots \lor \ell_m) \in \Gamma \) with \( m \geq 4 \). (proof TBC . . .)
Proof.
Let $p$ be a new proposition letter, and let $\Gamma''$ be the result of replacing $\gamma$ in $\Gamma$ with the pair of clauses:

$$p \lor \ell_3 \lor \cdots \lor \ell_m$$
$$\neg p \lor \ell_1 \lor \ell_2$$

These clauses entail $\gamma$, so if $\Gamma''$ is satisfiable, $\Gamma$ certainly is. On the other hand, if the assignment $\theta$ satisfies $\Gamma$, then setting $\theta(p) = \theta(\ell_1 \lor \ell_2)$ clearly satisfies $\Gamma''$.

Proceeding in this way, we eventually obtain the required $\Gamma'$. \qed
Outline

Reductions and hardness
  Reductions
  Transitivity of reductions
  Hardness and completeness

Cook’s theorem
  Cook’s theorem

Some easy reductions
  3-SAT
    Integer linear programming
• **Integer linear programming (ILP)** is the problem of determining the existence of a solution (over \( \mathbb{N} \)) to a system of linear Diophantine equations.

\[
\text{ILP}
\]

*Given:* a system of l.d. equations \( \mathcal{E} : A\mathbf{x} = \mathbf{b} \).

*Return:* Yes if \( \mathcal{E} \) has a solution over \( \mathbb{N} \), and No otherwise.

• We are also interested in the special case where the solutions are limited to values 0 and 1

• For \( k \geq 2 \), we have the problem

\[
\text{ILP}(0/1)
\]

*Given:* a system of l.d. equations \( \mathcal{E} : A\mathbf{x} = \mathbf{b} \).

*Return:* Yes if \( \mathcal{E} \) has a solution over \( \{0, 1\} \), and No otherwise.
Theorem

\[ ILP(0/1) \text{ is } \text{NPTime}-\text{complete} \]

Proof.

We show that \(3\text{-SAT} \leq^\text{log} \text{ ILP}(0/1)\).

Suppose we are given a set of 3-literal clauses \(\Gamma\). We show how to compute system of linear Diophantine equations \(\mathcal{E}\) such that \(\mathcal{E}\) has a solution over \(\{0, 1\}\) iff \(\Gamma\) is satisfiable.

For every proposition letter \(p\) mentioned in \(\Gamma\), let \(x_p\) and \(x_{\neg p}\) be variables and write the equation

\[ x_p + x_{\neg p} = 1. \]
Proof.
For every clause $\gamma := (\ell_1 \lor \ell_2 \lor \ell_3) \in \Gamma$, let $y_1^\gamma$, $y_2^\gamma$ be variables, and write the equation

$$x_{\ell_1} + x_{\ell_2} + x_{\ell_3} + y_1^\gamma + y_2^\gamma = 3.$$ 

Call the resulting system of equations $E_\Gamma$.

Suppose $\theta$ is a truth-value assignment for the proposition letters in $\Gamma$. Now define

$$x_p = \begin{cases} 
1 & \text{if } \theta(p) = \top \\
0 & \text{otherwise.}
\end{cases}$$

and define $x_{\neg p} = 1 - x_p$. 

□
Proof.
If \( \theta \) makes \( \gamma := (\ell_1 \lor \ell_2 \lor \ell_3) \) true, then we can certainly find \( y_1^\gamma \), \( y_2^\gamma \) satisfying.

\[
x_{\ell_1} + x_{\ell_2} + x_{\ell_3} + y_1^\gamma + y_2^\gamma = 3.
\]

So all the equations in \( \mathcal{E}_\Gamma \) are satisfied.

Conversely, given any assignment of values in \( \{0, 1\} \) to the variables \( x_\ell \) and \( y_j^\gamma \), define the truth-value assignment

\[
\theta(p) = \begin{cases} \top & \text{if } x_p = 1 \\ \bot & \text{otherwise.} \end{cases}
\]

If the various equations \( x_p + x_{\neg p} = 1 \) hold, then, for all literals \( \ell \), \( \theta(p) = \top \) iff \( x_\ell = 1 \). Hence, if the remaining equations in \( \mathcal{E}_\Gamma \) hold, every clause in \( \Gamma \) is made true by \( \theta \).