Reductions

Given two languages $L_1 \subseteq \Sigma^*_1$, and $L_2 \subseteq \Sigma^*_2$.

A **reduction** of $L_1$ to $L_2$ is a **computable** function

$$f : \Sigma^*_1 \rightarrow \Sigma^*_2$$

such that for every string $x \in \Sigma^*_1$,

$$f(x) \in L_2 \text{ if, and only if, } x \in L_1$$

Resource Bounded Reductions

If $f$ is computable by a polynomial time algorithm, we say that $L_1$ is **polynomial time reducible** to $L_2$.

$$L_1 \leq_P L_2$$

If $f$ is also computable in $\text{SPACE}(\log n)$, we write

$$L_1 \leq_L L_2$$

Reductions 2

If $L_1 \leq_P L_2$ we understand that $L_1$ is no more difficult to solve than $L_2$, at least as far as polynomial time computation is concerned.

That is to say,

If $L_1 \leq_P L_2$ and $L_2 \in \text{P}$, then $L_1 \in \text{P}$

We can get an algorithm to decide $L_1$ by first computing $f$, and then using the polynomial time algorithm for $L_2$. 
Complexity Theory

Completeness

The usefulness of reductions is that they allow us to establish the relative complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in \( \text{NP} \) that are maximally difficult.

A language \( L \) is said to be \( \text{NP-hard} \) if for every language \( A \in \text{NP} \), \( A \leq_p L \).

A language \( L \) is \( \text{NP-complete} \) if it is in \( \text{NP} \) and it is \( \text{NP-hard} \).

SAT is NP-complete

Cook showed that the language \( \text{SAT} \) of satisfiable Boolean expressions is \( \text{NP-complete} \).

To establish this, we need to show that for every language \( L \) in \( \text{NP} \), there is a polynomial time reduction from \( L \) to \( \text{SAT} \).

Since \( L \) is in \( \text{NP} \), there is a nondeterministic Turing machine

\[
M = (Q, \Sigma, s, \delta)
\]

and a bound \( n^k \) such that a string \( x \) is in \( L \) if, and only if, it is accepted by \( M \) within \( n^k \) steps.

Boolean Formula

We need to give, for each \( x \in \Sigma^* \), a Boolean expression \( f(x) \) which is satisfiable if, and only if, there is an accepting computation of \( M \) on input \( x \).

\( f(x) \) has the following variables:

\[
\begin{align*}
S_{i,q} & \quad \text{for each } i \leq n^k \text{ and } q \in Q \\
T_{i,j,\sigma} & \quad \text{for each } i, j \leq n^k \text{ and } \sigma \in \Sigma \\
H_{i,j} & \quad \text{for each } i, j \leq n^k
\end{align*}
\]

Intuitively, these variables are intended to mean:

- \( S_{i,q} \) – the state of the machine at time \( i \) is \( q \).
- \( T_{i,j,\sigma} \) – at time \( i \), the symbol at position \( j \) of the tape is \( \sigma \).
- \( H_{i,j} \) – at time \( i \), the tape head is pointing at tape cell \( j \).

We now have to see how to write the formula \( f(x) \), so that it enforces these meanings.
Initial state is $s$ and the head is initially at the beginning of the tape.

$$S_{1,s} \land H_{1,1}$$

The head is never in two places at once

$$\bigwedge_i \bigwedge_j (H_{i,j} \rightarrow \bigwedge_{j' \neq j} (\neg H_{i,j'}))$$

The machine is never in two states at once

$$\bigwedge_i \bigwedge_q (S_{i,q} \rightarrow \bigwedge_{q' \neq q} (\neg S_{i,q'}))$$

Each tape cell contains only one symbol

$$\bigwedge_i \bigwedge_q \bigwedge_{\sigma} (T_{i,j,\sigma} \rightarrow \bigwedge_{\sigma' \neq \sigma} (\neg T_{i,j,\sigma'}))$$

where $\Delta$ is the set of all triples $(q', \sigma', D)$ such that $((q, \sigma), (q', \sigma', D)) \in \delta$ and

$$j' = \begin{cases} j & \text{if } D = S \\ j - 1 & \text{if } D = L \\ j + 1 & \text{if } D = R \end{cases}$$

Finally, the accepting state is reached

$$\bigvee_i S_{i, \text{acc}}$$

The initial tape contents are $x$

$$\bigwedge_j T_{1,j,x_j} \land \bigwedge_{n<j} T_{1,j,\bot}$$

The tape does not change except under the head

$$\bigwedge_i \bigwedge_j \bigwedge_{\sigma} (H_{i,j} \land T_{i,j,\sigma}) \rightarrow T_{i+1,j',\sigma}$$

Each step is according to $\delta$.

$$\bigwedge_i \bigwedge_j \bigwedge_{\sigma} \bigwedge_{q} (H_{i,j} \land S_{i,q} \land T_{i,j,\sigma})$$

$$\rightarrow \bigvee_{\Delta} (H_{i+1,j'} \land S_{i+1,q'} \land T_{i+1,j',\sigma'})$$

A Boolean expression is in conjunctive normal form if it is the conjunction of a set of clauses, each of which is the disjunction of a set of literals, each of these being either a variable or the negation of a variable.

For any Boolean expression $\phi$, there is an equivalent expression $\psi$ in conjunctive normal form.

$\psi$ can be exponentially longer than $\phi$.

However, CNF-SAT, the collection of satisfiable CNF expressions, is NP-complete.
**3SAT**

A Boolean expression is in 3CNF if it is in conjunctive normal form and each clause contains at most 3 literals.

3SAT is defined as the language consisting of those expressions in 3CNF that are satisfiable.

3SAT is NP-complete, as there is a polynomial time reduction from CNF-SAT to 3SAT.

**Composing Reductions**

Polynomial time reductions are clearly closed under composition. So, if \( L_1 \leq_P L_2 \) and \( L_2 \leq_P L_3 \), then we also have \( L_1 \leq_P L_3 \).

Note, this is also true of \( \leq_L \), though less obvious.

If we show, for some problem \( A \) in NP that

\[
\text{SAT} \leq_P A
\]

or

\[
\text{3SAT} \leq_P A
\]

it follows that \( A \) is also NP-complete.