Two Proofs of Ladner's Theorem

We give two proofs of Ladner's Theorem in this note. This note is adapted from the appendix of the paper "Uniformly Hard Sets" by Fortnow and Downey.

Theorem 1 (Ladner) *If $P \neq NP$ then there exists an incomplete set A in $NP - P$.*

Both proofs have a similar set up. First we assume that P $\neq$ NP. Every NP-complete language is not in P with this assumption and we will focus on one of them, namely SAT, the language of satisfiable Boolean formula.

We have two sets of requirements to fulfill. Let $M_1, M_2, \ldots$ be an enumeration of deterministic Turing machines clocked so that the machine $M_i(x)$ runs in time $|x|^i$ and captures all of the languages in P. We also have a similar list f_i of the polynomial-time computable functions.

1. R_i: $A \neq L(M_i)$.
2. S_i: For some x, $x \in$ SAT and $f_i(x) \notin A$ or $x \notin$ SAT and $f_i(x) \in A$.

In addition we need to guarantee that A is in NP.

Proof by blowing holes in SAT

This proof is based on the original proof of Richard Ladner.

Our set A will be defined using a function f by

$$A = \{x \mid x \in \text{ SAT and } f(|x|) \text{ is even}\}.$$

Note that if we make $f(n)$ computable in polynomial in n time then A will be in NP.

The function f will be set to the current stage of the construction. Intuitively in stage $2i$, we keep $f(n) = 2i$ for large enough n until condition R_i is fulfilled. If R_i is never fulfilled then the set A will be equal to $L(M_i)$ and a finite difference from SAT contradicting the assumption that P $\neq$ NP.

In stage $2i + 1$ we keep $f(n) = 2i + 1$ until condition S_i is fulfilled. If S_i is never fulfilled then A will be finite and SAT reduces to A via f_i which would put SAT in P, again contradicting the fact that P $\neq$ NP.

The trick is to do this while keeping f polynomial-time computable. We do this by *delayed diagonalization*, i.e., we do not start a new stage until we see the requirement for the previous stage has been fulfilled on inputs so small we can test it. Thus we do not start a new stage until well after the old requirements are fulfilled.

We now formally define $f(n)$ inductively in n. Let $f(0) = f(1) = 2$. For $n \geq 1$ we define $f(n+1)$ as follows: If $\log^{f(n)} n \geq n$ then let $f(n+1) = f(n)$. Otherwise we have two cases:

1. $f(n) = 2i$: Check to see if there is an input x, $|x| \leq \log n$ such that either
   (a) $M_i(x)$ accepts and either $f(|x|)$ is odd or x is not in SAT, or
   (b) $M_i(x)$ rejects and $f(|x|)$ is even and x is in SAT.

   If such an x exists then let $f(n+1) = f(n) + 1$ otherwise we let $f(n+1) = f(n)$.
2. $f(n) = 2i + 1$: Check to see if there is an input x, $|x| \leq \log n$ such that either
   (a) x is in SAT and either $f(|f_i(x)|)$ is odd or $f_i(x)$ is not in SAT, or
   (b) x is not in SAT and $f(|f_i(x)|)$ is even and $f_i(x)$ is in SAT.

   If such an x exists then let $f(n+1) = f(n) + 1$ otherwise we let $f(n+1) = f(n)$.

Since to compute $f(n)$ we only examine x with $|x| \leq \log n$ and

$$|x|^i \leq \log^i n \leq \log^{f(n)} n < f(n),$$

we can compute $f(n)$ in time polynomial in n. It is straightforward to check that $f(n)$ does not increase until the corresponding requirements if fulfilled and that if $f(n)$ remains constant for all large n then we will have violated the P $\neq$ NP assumption.

Proof by Padding SAT

This proof is based on an unpublished proof of Russell Impagliazzo.

Here the idea is to encode SAT questions of length n on inputs of length $f(n)$. Define the language L as

$$L = \{\phi 0 1^{f(n)-|n|-1} \mid \phi \text{ in SAT, and } |\phi| = n\}.$$

We will create a polynomial-time computable in n function f large enough so that L is not NP-complete but not so large as to make L in P.

We will keep $f(n) = n^i$ long enough to fulfill R_i and then let $f(n) = n^{i+1}$.

We define formally define an algorithm for computing $f(n)$. Let $i = 1$ initially. For each n in order we do the following: Let $f(n) = n^i$. Check to see if there is an input x, $|x| \leq \log n$ such that either

1. $M_i(x)$ accepts and x is not in L, or

2. $M_i(x)$ rejects and x is in L.

If so let $i = i + 1$ otherwise leave i unchanged. Go onto the next n.

Since we are only checking very small x, we can compute f in polynomial time in n.

Suppose that L is in P. We then have that $L = L(M_i)$ for some i so $f(n) = n^i$ for suitably large n. But then we have an easy reduction from SAT to L and SAT would also be in P, violating our assumption.

So we have fulfilled all of the R_i requirements and i goes to infinity. Suppose some requirement S_j is not fulfilled. We then have a function f_j that reduces SAT to L. We want to show that we can now compute whether ϕ is in SAT in polynomial time.

Since f_j runs in time bounded by n^j we have that for all ϕ, $|f_j(\phi)| \leq |\phi|^j$. There must be some n_0 such that for all $n \geq n_0$, $f(n) = n^k$ for some $k > j$. We hardwire satisfiability for all inputs of length up to n_0.

Suppose we have a formula ϕ with $|\phi| > n_0$. If $f_j(\phi)$ is not in the range of f then $f_j(\phi)$ is not in L so ϕ is not in SAT. Otherwise, $f_j(\phi) = \psi 01^{f(m)-m-1}$ where $m = |\psi|$ and ϕ in SAT if and only ψ is in SAT. We have $f(m) = |f_j(\phi)| \leq |\phi|^j$ so $|\psi| = m \leq |\phi|^{j/k}$ if $f(m) = m^k$. Since $|\phi| > n_0$ we have $k > j$ so $|\psi| < |\phi|$. If $|\psi| \leq n_0$ then we know whether ψ and thus ϕ is in SAT. Otherwise we apply this algorithm recursively to ψ. Since $|\psi|$ gets smaller each step the algorithm runs in polynomial time.