A Hard Language

We give a careful construction of a language $L$ that is not recognizable in polynomial time. The construction is based on diagonalization.

The basic idea presented in class was to simulate each Turing machine $M_x$ with description $x$ for $2^{|x|}$ steps, and to put $x$ in $L$ if $M_x$ did not accept $x$ within $2^{|x|}$ steps. Unfortunately, we cannot quite prove that the language $L$ so constructed is not in $P$. The argument was supposed to go as follows: Suppose there were a polynomial time Turing machine $M_\ell$ with description $\ell$ that accepted $L$. Let $T_L(\cdot)$ be a polynomial bound on its running time. Consider what $M_\ell$ does on input $\ell$. If $M_\ell$ halts within $2^{|\ell|}$ steps and accepts $\ell$, then by definition of $\ell$, $\ell \notin L$. But this contradicts the assumption that $M_\ell$ accepts $L$. Similarly, if $M_\ell$ halts within $2^{|\ell|}$ steps and rejects $\ell$, then by definition of $\ell$, $\ell \in L$, which also contradicts the assumption that $M_\ell$ accepts $L$. We conclude that $M_\ell(\ell)$ runs for more than $2^{|\ell|}$ steps without halting, from which we can conclude $T(|\ell|) > 2^{|\ell|}$. This was supposed to contradict the assumption that $T(\cdot)$ is a polynomial. However, all we can say about the relation between a polynomial $T(n)$ and the function $2^n$ is that $2^n > T(n)$ for all sufficiently large $n$.$^1$ This doesn’t preclude having $T(n) > 2^n$ for the particular value $n = |\ell|$.

To fix the proof, we need to be able to argue that each candidate polynomial time machine $M_\ell$ fails to correctly identify membership in $L$ for infinitely many inputs $x$, not just for the single value $x = \ell$ as we showed above. If we can do this, the above argument will allow us to conclude that $M_\ell(x)$ fails to halt in $2^{|x|}$ steps for infinitely many $x$. It will then follow that $T(n) > 2^n$ for infinitely many $n$, which does contradict the assumption that $T(\cdot)$ is a polynomial.

How do we construct such a language $L$? In the previous construction, we considered the machine $M_\ell$ only when trying to decide whether or not to include the string $\ell$ in $L$. In the new construction, we will consider the machine $M_\ell$ when considering many different inputs $x$. To this end, we need a function $\pi(x)$ on strings that maps infinitely many $x$ onto each string $y$, that is, $\forall y \exists x (\pi(x) = y)$.\(^2\)

Now, to determine whether to put $x$ in $L$, we simulate the machine $M_{\pi(x)}$ on input $x$ for $2^{|x|}$ steps. As before, we put $x$ in $L$ if $M_{\pi(x)}$ does not accept $x$ with $2^{|x|}$ steps. This ensures that if $M_\ell$ is any machine that accepts $L$, then it runs for more than $2^{|x|}$ steps on all inputs $x$ for which $\pi(x) = \ell$. Since there are infinitely many such $x$, we conclude that $M_\ell$ does not run in polynomial time. Hence, $L \notin P$.

How can we construct such a function $\pi$? There are many ways, and you might want to think about how to do this before reading further. One way is by using a pairing function, that is, a one-to-one function $\sigma(x, y)$ that maps pairs of strings to strings. Concatenation doesn’t quite work, since it isn’t one-to-one. (For example $\text{concatenate}(00, 00) = \text{concatenate}(00, 00)$.) However, we can make it work by doubling each bit of $x$ and $y$, and using the pair “01” as a marker to separate the two strings. Thus, $\sigma(00, 1) = 00000111$ and $\sigma(0, 01) = 00010011$. We then define $\pi(\sigma(x, y)) = x$, and for $z$ not in the range of $\sigma$, we define $\pi(z) = e$ (the null string). Computing $\pi(z)$ is straightforward: Divide $z$ into pairs of bits. See if all pairs are in $\{00, 01, 11\}$ and exactly one pair

---

$^1$More precisely, $\exists N(\forall n > N)(2^n > T(n))$. The quantifier combination “$\exists N(\forall n > N)$” occurs so often that we define a new symbol $\forall \infty$ to stand for it. This lets us write the condition more succinctly as $\forall n(2^n > T(n))$, which we read variously as “for all sufficiently large $n$”, or “for almost all $n$”, or “for all but finitely many $n$”.

$^2$$\exists x^{\infty}$ means “there exists infinitely many $x$”. It is the dual of $\forall x$, so $\exists x P(x) \iff \neg \forall x \neg P(x)$. 

is 01. If so, then take the first bit from each pair until the occurrence of 01. If not, or if $|z|$ is odd, then output $e$.

The fact that we have a language $L$ that is not in $\mathcal{P}$ is not so remarkable by itself, since any non-decidable language such as the halting set $H$ is also not in $\mathcal{P}$. However, $L$ is decidable, and in fact, $L$ can be computed in exponential time by simply implementing the algorithm that is implicit in its definition. Namely, to test whether or not $x \in L$, one computes $z = \pi(x)$ and then simulates the Turing machine $M_z$ on input $x$ for at most $2^{|x|}$ steps. This requires decoding the description $z$ into a form that the simulator can use followed by $2^{|x|}$ simulation cycles. Without going into details, we claim that all of this can be done on a Turing machine in time that is polynomial in $|x| + 2^{|x|}$, which is $O(c^{|x|})$ for some constant $c$. So $L$ cannot by accepted by any Turing machine that runs in time $2^{|x|}$, but it can be accepted by a machine that runs in time $O(c^{|x|})$. 