Ladner's Theorem, Sparse and Dense Languages

Vasiliki Velona

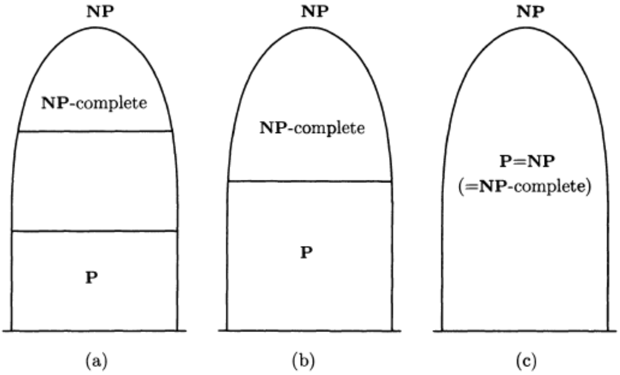
$\mu\prod\lambda\forall$

November 2014

Part 1, Ladner's Theorem

Part 1: Ladner's theorem

(Ladner, 1975): If $P \neq NP$, then there is a language in P which is neither in P or is NP complete



The second scenario is impossible.

Ladner's theorem proof

Preliminaries

- We can compute an enumeration of all polynomialy bounded TMs $(M_1, M_2, M_3, ...)$ and all logarithmic space reductions $(R_1, R_2, R_3, ...)$
- Why? One way is to use a polynomial "clock" on every M_i that will allow it to run for no more than $|x|^i$ steps for input x. Similarly, we can do this for logarithmic space reductions.

Ladner's theorem proof

- The wanted language is described in terms of the machine K that decides it:

  $L(K) = \{x | x \in SAT \text{ and } f(|x|) \text{ is even}\}$

  $f(n)$ will be described later.

- The demands for this language are the following:

1. $\forall i L(K) \neq L(M_i)$ (out of P)
2. $\forall i \exists w : K(R_i(w)) \neq S(w)$ (out of NP-complete)

and we'll prove that they're met.

Ladner's theorem proof

We want to check our conditions one after the other:
$M_1, R_1, M_2, R_2, \ldots$

Let F be the turing machine that computes f. For $n = 1$ F makes two steps and outputs 2. For $n \geq 2$ F proceeds this way:

1. Computes $f(1), f(2), \ldots$ as many of them as it can for n steps.
2. If the last value of f thus computed was $f(i) = k$ then

   - If k=2i we check our conditions for M_i versus K with inputs z, ranging lexicografically over all Σ^*. (for n steps)
     If a such z is found then $f(n) = 2i + 1$. Else $f(n) = 2i$
   - If k=2i+1 we check our conditions for $K(R_i)$ versus S with inputs z, ranging lexicografically over all Σ^* (for n steps)
     If we find such a z then $f(n) = 2i$. Else $f(n) = 2i + 1$

Ladner's theorem proof

Comments on the construction

- Obviously F is $\mathcal{O}(n)$ and thus K is in NP.

  Reminder: $K = \{x | x \in SAT \text{ and } f(|x|) \text{ is even}\}$

- The function f is a *very slowly* growing function: Suppose that n(k) is the smallest number for which $f(n) = k$. Then the smallest number for which f(n) has a chance at becoming k+1 is at least $\frac{n(k)^2}{2}$ (in fact it is even bigger). It follows that $f(n) = \mathcal{O}(\log \log n)$

- This technique is often called "lazy" diagonalization. It will be more clear in the final arguments.

Ladner's theorem proof

Final arguments

$$L(K) = \{x | x \in SAT \text{ and } f(|x|) \text{ is even}\}$$

- Suppose first that $L(K) \in P$, and so is accepted by some polynomial-time machine in our enumeration, let's say M_i. Then f(n)=2i for all $n \geq n_0$ for some n_0 and thus L(K) coincides with SAT on all but finitely many strings. But this contradicts the assumptions $P \neq NP$ and $L(K) \in P$
- Suppose that L is NP-complete, and so there is a reduction, let's say R_i in our enumeration, from SAT to L(K).It follows that f(n)=2i+1 for all $n \geq n_0$ for some n_0. But then L(K) is a finite language and this contradicts with the assumption that L(K) is NP-complete.
- End of proof

Problems conjured to be NP-intermediate

The language constructed in the proof is artificial. The question is whether any "natural" decision problems are intermediate. Some candidates:

1. GRAPH ISOMORPHISM: Given (simple, undirected) graphs $\{G_1\}$ and $\{G_2\}$, are they isomorphic?
2. FACTORING: Given natural numbers $\{m < n\}$, does $\{n\}$ have a prime factor greater than $\{m\}$?
3. DISCRETE LOGARITHM: Given natural numbers $g, h, k < n$, does there exist $\{e \leq k\}$ such that $\{g^e = h\}$ modulo $\{n\}$?
4. CIRCUIT MINIMIZATION: Given a string $\{x \in \{0, 1\}^n\}$ where $\{n = 2^k\}$ for some $\{k\}$, and $\{s > 0\}$ (in binary), is there a $\{k\}$-input Boolean circuit $\{C\}$ of size at most $\{s\}$ such that for all $\{i\}$, $\{0 \leq i < n\}$, $\{C(i) = x_i\}$?

Part 2, Dense and Sparse Languages

Part II: Density

Let $L \subset \Sigma^*$ be a language. We define its density to be the following:

$$dens_L(n) = |\{x \in L : |x| \leq n\}|$$

Sparse languages are languages with polynomially bounded density functions.
Dense languages are languages with superpolynomial densities.

Density

Definition: We say that two languages $K, L \in \Sigma^*$are polynomially isomorphic if there iis a function h from Σ^* to itself such that:

- h is a bijection
- For each $x \in \Sigma^*$, $x \in K$ if and only if $h(x) \in L$
- Both h and its inverse h_{-1} are polynomial-time computable

Proposition: If $K, L \subset \Sigma^*$ are polynomially isomorphic, then $dens_K$ and $dens_L$ are polynomially related.

Proof: All strings in K of length at most n are mapped by the polynomial isomorphism into strings of L of length at most $p_1(n)$, where p_1 is the polynomial bound of the isomorphism. Since the mapping must be one-to-one, $dens_K(n) \leq dens_L(p_1(n))$. Similarly, $dens_L(n) \leq dens_K(p_2(n))$ where p_2 is the polynomial bound of the inverse isomoprhism.

Sparse Language Facts

- It known that there is a polynomial-time Turing reduction from any language in P to a sparse language.
- Fortune showed in 1979 that if any sparse language is co-NP-complete, then P $=$ NP.
- Mahaney used this to show in 1982 that if any sparse language is NP-complete, then P $=$ NP.

Unary Languages and the $P \neq NP$ question

A familiar kind of sparse languages are the unary languages, the subsets of $\{0\}^*$. Intrestingly, there is a direct argument that proves the last for unary languages:

- Suppose a unary language $U \subset \{0\}^*$ is NP-complete. Then P=NP.

  Proof: It suffices to show that $SAT \in P$ if there exist a reduction from SAT to U.

  - Given a boolean expression ϕ with n variables $x_1, x_2, \ldots$ we consider a partial truth assignment $t \in \{0,1\}^j$.
  - $t_i = 1$ means $x_i = true$ and $t_i = 0$ means $x_i = false$.
  - $\phi[t]$ is the expression resulting from ϕ if we substitute the truth assignements of t in ϕ. (omiting any **false** literals from a clause, and omitting any clause with a **true** literal)

Example

For $t = 001$

and $\phi = (x_1 \vee x_2 \vee \neg x_1) \wedge (x_5 \vee x_4 \vee x_3) \wedge (x_5 \vee x_4)$

we have $\phi[t] = (x_5 \vee x_4)$

It's clear that if $|t| = 5$ then $\phi[t]$ is either **true** and has no clauses, or **false** and has an empty clause.

The algorithm

A resonable algorithm for SAT:

- **If $|t| = n$, then return "yes" if $\phi[t]$ has no clauses, else return "no"**
  Otherwise return "yes" if and only if either $\phi[t0]$ or $\phi[t1]$ returns "yes"

A better one:

- **If $|t| = n$, then return "yes" if $\phi[t]$ has no clauses, else return "no"**
  Otherwise look up $H(t)$ in the table; if a pair $(H(t), v)$ is found return v.
  Otherwise return "yes" if either $\phi[t0]$ or $\phi[t1]$ returns "yes"; return no otherwise.
  In either case, update the table by inserting $(H(t), v)$

What about H?

We need a function H that

1. maintains satisfiability: if $H(t) = H(t')$ for two partial truth assignments t and t' then $\phi[t]$ and $\phi[t']$ must be both satisfiable or both unsatisfiable.
2. has a small range, so that the table can be searched efficiently

The reduction R from SAT to U has these two properties. So we can define $H(t) = R(\phi(t))$

[All values of $H(t)$ must be of length at most p(n), the polynomial bound on R, when applied to an expression of n variables. But since U is unary there are at most $p(n)$ such values.]

The Complexity of the algorithm

- On each recursive call the algorithm takes at most p(n) time. So, the total time is $\mathcal{O}(Mp(n))$, where M is the total number of the algorithm invocations.

- **Claim:** We can pick a set $T = \{t_1, t_2, t_3, \ldots\}$ of invocations, such that:
  1. $|T| \geq \frac{M}{2n}$
  2. All invocations in T are recursive
  3. None of the elements of T is a prefix of another element in T.

- All the invocations in T are mapped to different H values. But there are only $p(n)$ such values. So $\frac{M}{2n} \leq p(n)$, and the running time is $\mathcal{O}(np(n)^2)$

Thank you!