Hierarchy Theorems: **If** $f(n)$ is a **C**-constructible function;

C is DSPACE, NSPACE, DTIME, or NTIME; and,

if $g(n)$ is sufficiently smaller than $f(n)$

Then $\mathbf{C}[g(n)]$ is strictly contained in $\mathbf{C}[f(n)]$.

$g(n)$ **sufficiently smaller** than $f(n)$ means:

$$\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = 0$$

$\mathbf{C}$ = DSPACE, NSPACE, NTIME

$$\lim_{n\to\infty}\left(\frac{g(n)\log(g(n))}{f(n)}\right) = 0$$

$\mathbf{C}$ = DTIME

Definition 11.1 Function $f : \mathbf{N} \to \mathbf{N}$ is **C-constructible** if the map

$$1^n \;\mapsto\; f(n)$$

is computable in the complexity class $\mathbf{C}[f(n)]$. $\square$

For example a function $f(n)$ is DSPACE-constructible if the function $f(n)$ can be deterministically computed from the input 1^n, using space at most $O[f(n)]$.

Fact: All reasonable functions greater than or equal to $\log n$ are DSPACE-constructible, and all reasonable functions greater than or equal to n are DTIME-constructible.

Theorem 11.2 (Space Hierarchy Thm:) *If $f \geq \log n$ is space constructible and* $\lim_{n\to\infty}\left(\frac{g(n)}{f(n)}\right) = 0$, *Then* $\mathrm{DSPACE}[g(n)] \subsetneq \mathrm{DSPACE}[f(n)]$.

Proof: Build $\mathrm{DSPACE}[f(n)]$ machine, D, on input: w, $n = |w|$

1. Mark off $6f(n)$ tape cells, (f space constructible)
2. Simulate $M_w(w)$ using space $3f(n)$, time $\leq 2^{3f(n)}$
3. **if** ($M_w(w)$ needs more space or time) **then return**(17)
4. **else if** ($M_w(w)$ = **accept**) **then reject**
5. **else accept** // ($M_w(w)$ = **reject**)

$$\underbrace{\boxed{\text{space to simulate } M_w(w)}}_{3f(n)}\underbrace{\boxed{\qquad\quad\text{counter}\qquad\quad}}_{3f(n)}$$

Claim 11.3 $\mathcal{L}(D) \in \text{DSPACE}[f(n)] - \text{DSPACE}[g(n)]$

Proof: $\mathcal{L}(D) \in \text{DSPACE}[f(n)]$ by construction.

Suppose $\mathcal{L}(D) \in \text{DSPACE}[g(n)]$.

Let $\mathcal{L}(M_w) = \mathcal{L}(D)$, M_w uses $cg(n)$ space.

Choose N s.t. $\forall n > N\ (cg(n) < f(n))$.

Choose w', $M_{w'}(\cdot) = M_w(\cdot)$, $|w'| > N$

On input w', D successfully simulates $M_{w'}(w')$ in $3f(n)$ space and $2^{3f(n)}$ time.

$$w' \in \mathcal{L}(D) \Leftrightarrow w' \notin \mathcal{L}(M_{w'}) \Leftrightarrow w' \notin \mathcal{L}(M_w) \Leftrightarrow w' \notin \mathcal{L}(D)$$

$$\Rightarrow\Leftarrow$$

$\square$

Theorem 11.4 (Ladner) *If* P $\neq$ NP *then there exists an intermediate problem* $I \in$ NP $-$ P *that is not* NP *complete.*

Proof: Assume that P $\neq$ NP.

We will construct I by a method called "delayed diagonalization".

The construction will make sure that:

- I is **not hard**: SAT $\not\leq I$. $R_1, R_3, R_5, \ldots$
- I is **not easy**: $I \notin$ P. $R_2, R_4, R_6, \ldots$

R_{2k+1} : "M_k isn't a DSPACE$[k \log n]$ reduction from SAT to I"

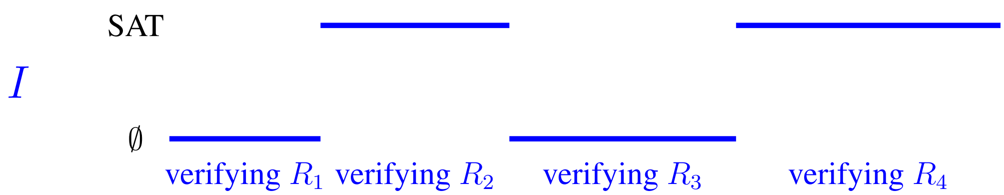
R_{2k+2} : "M_k isn't a DTIME$[kn^k]$ recognizer of I"

Observation: If all the R_i's are met, then we're done.

Conditions to Satisfy: $R_i, \quad i = 1, \ldots \infty$

R_{2k+1} : "M_k isn't a DSPACE$[k \log n]$ reduction from SAT to I"

R_{2k+2} : "M_k isn't a DTIME$[kn^k]$ recognizer of I"



On input w, recursively $I(w)$ does following:

1. **do** for a total of $|w|$ steps {
2. **for** $i = 1 \ldots \infty$ **do** {
3. **for** $x = 1 \ldots \infty$ **do** {
4. **if** (R_i verified on input x) **then** next i
5. } } }
6. **if** (i is even and $w \in$ SAT) **then ACCEPT**
7. **else** REJECT

Note: In line 4, I simulates itself **deterministically**. Thus, to check if an input is in SAT it might need exponential time. Thus, it might only find out exponentially later that condition R_i has been met. That's why this method is called **delayed diagonalization**. The key idea, is that if i is even we are simulating SAT, so if we do this long enough we cannot be in P, whereas if i is odd then we are rejecting all inputs, so if we do this long enough we cannot be NP complete. □