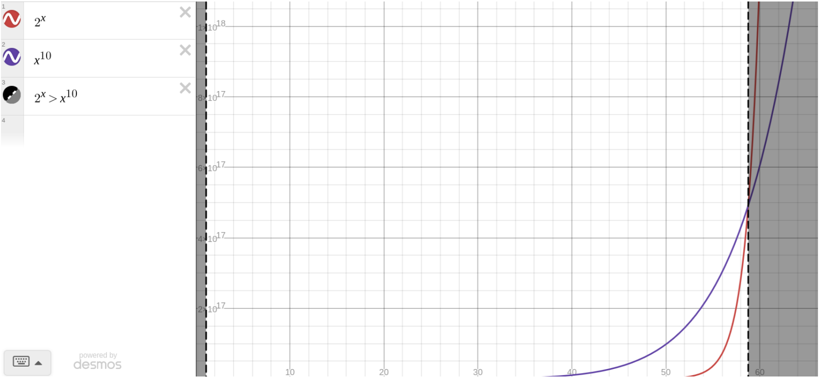
Intractability

Textbook: Chapter 9

Tractability vs. Intractability

Def: A language A is **tractable** ("realistically" solvable) if $A \in P$. It is **intractable** (theoretically solvable, but not in "realistic" time) if $A \notin P$.

- In reality, there are plenty of language $\in P$ that are infeasible to decide IRL: $O(n^{999999})$ is technically "tractable"



- Exponential cost will eventually overtake **any** polynomial

Space Constructability

- Recall: A function runs with deterministic **space** complexity $O(f(n))$ if it uses space $f(n)$ on input length n
- **Def:** A function f is **space-constructable** if a TM can map any string of length x to $f(x)$ in space $O(f(x))$
- Space-constructable function examples
  - $\lg(n)$
  - $n \lg(n)$
  - n^2

Space Hierarchy Theorems

- Clearly, $SPACE(n) \subseteq SPACE(n^2)$ since n is $O(n^2)$
- How can we prove that two space complexity classes are different, e.g. $SPACE(n) \subsetneq SPACE(n^2)$?

Thm: Space hierarchy theorem. For any space-constructable f, a language A exists that is decidable in $O(f(n))$ space but **not** in $o(f(n))$ space.

- A is decidable with space bounded by $f(n)$, but cannot be decided with space insignificant to $f(n)$

Corollary: For any nonnegative integers $\epsilon_1, epsilon_2$ where $0 \leq \epsilon_1 < \epsilon_2$,

$$SPACE(n^{\epsilon_1}) \subsetneq SPACE(n^{\epsilon_2})$$

- Finally, a proper hierarchy! This is very useful!

Space Hierarchy Proof pt. 1

Pf: By construction. Let A be the language of the following Turing Machine.

```python
# Returns whether M rejects M in space f(n)
def D(inp) -> bool: # inp is of form <M>10*
    M, w = decode_TM_input_pair(inp) # w is 10*
    n = len(inp)
    number_of_steps = 0
    while number_of_steps < 2 ** f(n):
        # True, False, or None
        result = M.one_step_with_input(inp)
        if M.memory_usage > f(n):
            return False
        elif result is None:
            number_of_steps += 1
        else:
            return not result
    return False
```

Space Hierarchy Proof pt. 2

- D clearly uses at most $O(f(n))$ space
- Now, the space hierarchy theorem is proven only if D's language is not decidable in space $o(f(n))$

Lemma: A is not decidable in space $o(f(n))$.

Pf: Assume to the contrary that some TM M decides A in space $g(n)$ where $g(n)$ is $o(f(n))$.

What if we run D from before on this new M?

Because $g(n)$ is $o(f(n))$, there will always exist some n_0 for any constant d such that, for all $n > n_0$, $dg(n) < f(n)$. Therefore, D can correctly simulate M as input as long as $n > n_0$. We can force this condition by giving it input $\langle M \rangle 10^{n_0}$.

Space Hierarchy Proof pt. 3

When D takes in $\langle M \rangle\, 10^{n_0}$, it will decode and run it. Since the input is long enough, it will not exceed the time limit. Since $g(n)$ is $o(f(n))$, it will not exceed the space limit. By assumption, D must decide A (the same language as M): However, $D(\langle M \rangle\, 10^{n_0})$ does the **opposite** of $M(\langle M \rangle\, 10^{n_0})$! Contradiction.

Time Constructability

- Time complexities give us a lot more trouble than space ones
- If we could prove that $P \subsetneq NP$ like we can prove $PSPACE = NPSPACE$, we could settle the P/NP debate
- Thus, we try to do the same with time

Def: A function t where $t(n)$ is at least $O(n \log n)$ is called **time constructible** if a TM can map input of length n to $t(n)$ in deterministic time $O(t(n))$.

- Examples: $n \log n$, $n\sqrt{n}$, n^2, 2^n

Time Hierarchy Theorems

Thm: Time hierarchy theorem. For any time constructible function t, a language A exists that is decidable in $O(t(n))$ time but not in time

$$o\left(\frac{t(n)}{\log t(n)}\right)$$

- Subtly different, and therefore less strong!

Corollary: For functions t_1, t_2 where t_2 is time constructable and t_1 is $o\left(\frac{t_2(n)}{\log t_2(n)}\right)$,

$$TIME(t_1(n)) \subsetneq TIME(t_2(n))$$

- Pf. excluded, but follows the same process as for space

EXPSPACE-Completeness

- Recall: *EXPSPACE* is the set of all problems solvable given exponential space
- $EXPSPACE = \bigcup_k SPACE(a^{n^k})$
- Any algorithm in P must be in *PSPACE*, since we can't use more than a constant number of memory "units" per time step
- A polynomial-time reduction cannot use more than polynomial space

Def: A language B is *EXPSPACE*-**complete** if both:

1. $B \in EXPSPACE$
2. If language A is in *EXPSPACE*, A is polynomial-time reducible to B

Relativization

- We previously used oracle Turing Machines to do reductions
- We briefly mentioned that an oracle call is assumed to take a single computation
- What if we redefine a problem's time complexity with respect to an oracle?

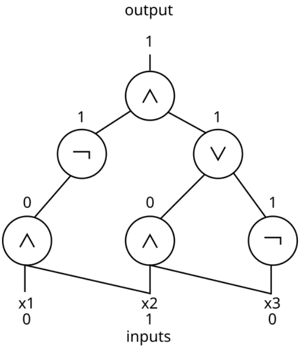
Def: For some language A, let T^A denote a TM that can decide A in a single step (equivalently, a TM that has an oracle for A). P^A is the set of all languages decidable in polynomial time by a TM with an oracle for A. Similarly, NP^A is the set of all languages decidable in nondeterministic polynomial time given an oracle for A.

Relativization Example

- We know SAT is NP-complete: Any $A \in NP$ is polynomial time decidable if SAT is
- Therefore, $NP \subseteq P^{SAT}$
- Indeed, we know of languages A and B such that $P^A \neq NP^A$ and $P^B = NP^B$

Circuit Complexity

Def: A Boolean circuit is a Directed **Acyclic** Graph (DAG) from some set of inputs $\{x_1, x_3, \ldots, x_i\}$ to a single output, where every node has an in-degree of 1 or 2 an out-degree of 1. Every node represents $\wedge$, $\vee$, or $\neg$.



Circuit Families and Attributes

A circuit's **size** is the number of nodes, whereas a circuit's **depth** is the length of the longest path from an input to the output. Every circuit with n inputs describes some $f : \{0,1\}^n \to \{0,1\}$.

A **circuit family** $C = (C_0, C_1, C_2, \ldots)$ is an infinite list of circuits where C_i has i inputs. We say C decides $A \subseteq \mathcal{P}(\{0,1\})$ if for every string w, $w \in A$ iff $C_{|w|}(w) = 1$.

A circuit is **size minimal** if no smaller circuit does what it does. The same holds for depth. The **size complexity** of a circuit family C is a function f such that $f(n)$ is the size of C_n. The **depth complexity** is a function f such that $f(n)$ is the depth of C_n.

Ex: A is the language of strings that contain an odd number of 1s. A has circuit complexity $O(n)$.

Since circuits simulate Boolean formulae and SAT is NP-complete, circuit satisfiability is also NP-complete. Boolean circuits are known to be Turing-equivalent by the same methods of the Cook-Levin theorem / TM Gödel-ization

Next up: Advanced Complexity Analysis

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME \subseteq EXPSPACE$$