Chapter 11: A Hierarchy of Formal Languages & Automata*

Peter Cappello
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
cappello@cs.ucsb.edu

- Please read the corresponding chapter before attending this lecture.
- These notes are supplemented with figures, and material that arises during the lecture in response to questions.
- Please report any errors in these notes to `cappello@cs.ucsb.edu`. I'll fix them immediately.

*Based on **An Introduction to Formal Languages and Automata**, 3rd Ed., Peter Linz, Jones and Bartlett Publishers, Inc.

11.1 Recursive & Recursively Enumerable Languages

- **Def. 11.1**: A language L is **recursively enumerable** if there is a TM that accepts it. That is, there is a TM M such that,

$$\forall w \in L, \exists x, y \in \Gamma_M^*, q_0\ w \vdash_M^* x\ q_f\ y, \text{ where } q_f \in F_M.$$

- M may not even halt, if $w \notin L$.
- **Def. 11.2**: A language L on Σ is **recursive** if there is a TM M that *halts for all* $w \in \Sigma^+$ and that accepts L.

- If L is recursive, then there is an enumeration procedure for it:
  - Let M_L be a TM that accepts L and always halts.
  - Let StringΣ.next() be an iterator, enumerating Σ^+ in proper order.
  - Define StringΣ string = new StringΣ();
  - Construct TM M_{E_L} that repeats the following steps forever:
    1. StringΣ w = string.next();
    2. Run M_L on w;
    3. If (M_L accepts w) tape.write(w);
- Is there a way to enumerate a recursively enumerable language?

- Let M_L be a TM that accepts recursively enumerable language L.
- Let IterateΣ be an iterator, enumerating Σ^+ in proper order.
- Let Iterator$\Sigma \times \mathbf{N}$, be an iterator, enumerating $\Sigma \times \mathbf{N}$.
  Its next() method is defined according to the following illustration:
- The following procedure enumerates L.

  1. Iterator iterator = new Iterator$\Sigma \times \mathbf{N}$();
  2. Repeat forever:
     (a) (w, n) = iterator.next();
     (b) Run M_L on w for n steps;
     (c) If (M_L accepts w in n steps) tape.write(w);
- This procedure never loops infinitely on any w, even though M_L may loop infinitely on some w.

We begin with a powerful idea: **diagonalization**, due to G. Cantor.
Thm. 11.1: Let S be an infinite countable set. Then, 2^S is not countable.
Proof:

1. Let S be enumerable in order $s_1, s_2, ...$
2. Any subset of S can be characterized by the enumeration indices of the elements of the subset.

   For example, if P is the set of primes, then $3, 5, 7$ can stand for the subset $\{5, 11, 17\}$ if they are the 3rd, 5th, and 7th primes.

   Since infinite sets have infinite subsets, index sets may be infinite.
3. Assume 2^S is countable. Then, it can be enumerated: $S_1, S_2, \ldots$.
4. Construct a table, where the ith row represents S_i.

Row i's jth column is 1 if $s_j \in S_i$.

It is 0 otherwise. Illustrate the table.

5. Construct an infinite subset $T = \{s_i \in T \Leftrightarrow s_i \notin S_i\}$.
6. Since T is a subset of S, $T = S_j$, for some j.
7. But, T cannot be any S_j because, by its definition, its elements *differ* from each S_j on s_j.
8. Therefore, T, a subset of S, is not in the enumeration of 2^S.
9. This contradicts the assumption that 2^S is countable.

Thm. 11.2: For any nonempty Σ, there are languages that are not recursively enumerable.

Proof:

1. Every subset of Σ^* is a language.
2. Since Σ^* is infinite, 2^{Σ^*} is *uncountably* infinite. That is, there are *uncountably* infinitely many languages over Σ.
3. The set of TMs is *countably* infinite.
4. Therefore, there are languages over Σ that are not accepted by any TM. They are not recursively enumerable.

A Language That Is Not Recursively Enumerable

Thm. 11.3: There is a recursively enumerable language L such that $\overline{L}$ is not recursively enumerable.

Proof:

1. Let $\Sigma = \{a\}$.
2. Let T be the set of all TMs with this input alphabet.
3. T is countable. Let $M_1, M_2, \ldots$ be an enumeration of T.
   Define $L = \{a^i \in L \Leftrightarrow a^i \in L(M_i)\}$.
4. L is recursively enumerable:
   (a) Enumerate ordered pairs (w_i, n);
   (b) For each (w_i, n), run M_i on w_i for n steps;
   (c) if M_i accepts w_i in n steps, tape.write(w_i);

5. $\overline{L} = \{a^i : a^i \notin L(M_i)\}$.
6. Assume that $\overline{L}$ is recursively enumerable.
7. Then, there is a TM M_k that accepts it.
8. Is $a^k \in \overline{L}$?
9. If $a^k \in L(M_k)$, then $a^k \notin \overline{L}$, by the definition of $\overline{L}$: A contradiction.
10. If $a^k \notin L(M_k)$, then $a^k \in \overline{L}$, by the definition of $\overline{L}$: A contradiction.
11. Therefore, the assumption $\overline{L}$ is recursively enumerable is false.

Thm. 11.4: 1. If a language L and $\overline{L}$ are recursively enumerable, then both are recursive. 2. If L is recursive, then $\overline{L}$ is recursive.
Proof:

1. We give a TM T that halts on all inputs and accepts L:

   Let M accept L and $\overline{M}$ accept $\overline{L}$.

   Given w, iterate over i until w is either accepted or rejected:

   (a) Run M until w_i is produced;
   (b) If ($w_i = w$) accept w [and halt];
   (c) Run $\overline{M}$ until $\overline{w}_i$ is produced;
   (d) If ($\overline{w}_i = w$) reject w and halt.

2. Since TM T accepts L and halts on all inputs, L is recursive.

Then, $\overline{L}$ is recursive: construct TM $\overline{T}$ that simulates T, accepting w if only if T rejects it.

Thm. 11.5: There is a recursively enumerable language that is not recursive.

Proof:

1. Consider $L = \{a^i \in L \Leftrightarrow a^i \in L(M_i)\}$.
2. L is recursively enumerable, but $\overline{L}$ is not.
3. Therefore, by Thm 11.4, L is not recursive.

- Thus, there is no membership algorithm for the class of languages accepted by Turing machines: recursively enumerable languages.
- Illustrate the containment relations for regular languages, CFLs, recursive languages, and recursively enumerable languages.
- Note that each containment is strict.