Recap: Ch 8-10

Limits of Computation

- "To compute": How can we examine computability of some problem/language? What exactly is an algorithm? Undecidability?
- We need a formal definition of algorithm and of computation in general

Turing Machines

Abstract computing device

- Simple, yet as powerful as any computer
- Made computation easy to describe

Church-Turing Hypothesis

Informally: This model captures anything we can compute.

The TM

- Infinite tape cells containing tape symbols
- Read/write head that can move left or right
- Finite state control

Formally, a TM M $= (Q, \Sigma, \Gamma, \delta, q_0, q_F)$
Transitions $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
Acceptance: w is accepted by TM M if M, when started with w on the tape, eventually enters a final state.

How to Design One

- Take informal description
- Break it down into components
- Describe high-level idea (how its components can be made)
- Build it precisely together with proof

Instantaneous Descriptions, or Configuration of a TM

With $x, y \in \Gamma^*$ and $q \in Q$:

$$\text{If } \delta(q, a) = (p, b, L) \text{ then } xraq\gamma y \vdash xq\gamma by \text{ or } xqay \vdash qxby$$

$$\text{If } \delta(q, a) = (p, b, R) \text{ then for any } r : xrqay \vdash xrbpy$$

Language Accepted by TM M

$L(M) = \{w \in \Sigma^* | q_0 w \vdash^* \alpha q_F \beta \text{ where } \alpha, \beta \in \Gamma^*\}$

Recursively Enumerable Languages

TM is called the recursively enumerable languages

Making/Designing TMs Easier

- Storage in the FSC
- Multiple tracks
- Multiple tapes (can simulate with single tape/single head)
- One head, array under the head (not independent)
- Input transformation

Note: They all can convert back to the basic TM

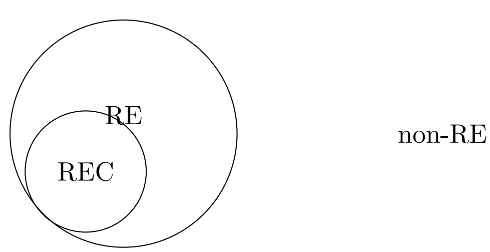
Classes of Languages/Problems

1. **Recursive languages:** Class of languages accepted by TMs that always halt - TM that always halt $\rightarrow$ algorithm
   - decidable problems $\xrightarrow{\text{yes/no}}$ (halt)
2. **Not:** problems/languages that are not RE/enumerable are called undecidable $\rightarrow$ they don't have algorithms
3. **Recursively enumerable languages:** class of languages accepted by TMs (that may not halt) $\rightarrow$ procedure
   - more $\xrightarrow{\text{yes}}$ (halt)
4. **Non-RE languages:** languages/problems for which there is no TM
   - implies $\rightarrow$?

Note: Closure properties (see Folk or other primary text)

Showing Languages to be Undecidable or Non-RE

Use the UTM and via diagonalization, for which we first need a suitable encoding of a TM by means of strings.



Diagonalization Language (for showing non-RE)

Let $\overline{L_d}$ be the set of strings w such that w is a TM and $w \notin L(M_w)$. To show that $\overline{L_d}$ is non-RE:

- When w is a string (i.e., encoded TM) that does not accept itself
- Let D be a characteristic vector of $L(M_i)$, specifying which strings belong to $L(M_i)$
- Let $D_i = \{M_i | M_i \notin L(M_i)\}$
- Then proving by contradiction: $\overline{L_d}$ is non-RE (i.e., there is no TM that accepts $\overline{L_d}$)

Universal TM

U is a RE but not recursive language, i.e.:

- UTM takes over the task for some TM M by having string w and accepts iff M accepts w
- UTM's input: $\langle E(M), w \rangle$ UTM accepts iff M accepts w
- Is simulated by M on w So, $L(U) = \{(M, w) | w \in L(M)\}$ accepts iff
- (undecidability of L_u)

Halting Problem

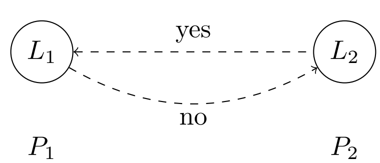
Set of pairs (M, w) s.t. $w \in H(M)$, with:

- $H(M)$ is the set of inputs w s.t. M halts eventually
- Regardless of whether or not M accepts w is RE but not rec.

Reductions

L_1 reduces to L_2 (denoted $L_1 \leq L_2$) if there exists a function R (called the "reduction" from L_1 to L_2) such that:

1. R is computable by some TM M_R that takes as input a string x (an instance of L_1) and halts with a string $R(x)$ (an instance of L_2) on its tape.
2. M_R is an algorithm
3. $x \in L_1 \Leftrightarrow R(x) \in L_2$



Notes on Reductions

- For showing (un)decidability, we don't care how long the reduction takes, but when we work at tractability/intractability (P/NP etc.), we put a bound on T_R to show the reduction has to take in polynomial time or less, denoted as $\leq_p$
- You'll have to understand and be able to explain why configurations at intermediate steps (including those for time bounds) are reasonable

Typical Approach

If you want to prove whether area/problem P_1 (new choice for the complexity class) of P_2, then take a problem P_1 with known class, devise the algorithm R and compute time $O(T_R)$ to demonstrate that P_1 is also element of P_2. (recall also 'lower bound' and 'upper bound' notions)

Rice's Theorem

Every non-trivial property of RE-languages is undecidable.

Tractable Problems

Complexity theory considers tractable all problems with polynomial time algorithms ($T(n) = O(n^k)$)

- $P = \{L | L = L(M)$ for some poly-time DTM $M\}$
- $NP = \{L | L = L(M)$ for some poly-time NDTM $M\}$

Running Time (or Time Complexity) of a TM

A TM has time complexity $T(n)$ if it halts in at most $T(n)$ steps (accepting or not) for all inputs n of length n.

Examples of Problems

In P (tractable)

- Is a given vertex v in a graph G reachable from vertex s?
- Path from node s to y in graph G?

In NP (less tractable)

- Traveling salesman problem
- Clique
- Knapsack

To establish something is in NP:

1. Guess some solution
2. Verify that it is a correct solution

NP-Hardness

Problem Y is NP-hard if $\forall x \in NP$ we have $x \leq_p Y$

NP-Completeness

Y is NP-complete if:

1. Y is NP-hard
2. Y $\in$ NP

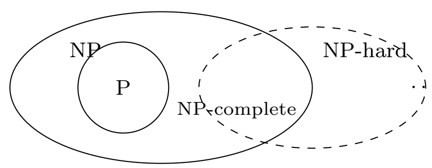
(NP-completeness is a strong evidence of intractability)

co-NP

A problem whose complement is in NP
(Note: P is closed under complementation) e.g., UNSAT
co-NP = $\{\overline{L} | L \in NP\}$

Complexity Class Relationships

Assuming P$\neq$NP, we obtain the following figure:



Additional Notes on Completeness

For completing "completeness" as we have seen with NP, we can apply the same idea to other complexity classes.
e.g., EXPTIME-complete (is EXPTIME-hard and in EXPTIME)

Bounded TMs

Time-bounded
space-bounded (i.e., limited amount of tape)

e.g., PSPACE = $\{L | L(M)$ for some DTM M that uses at most space that is polynomial in its input$\}$
Examples: QSAT, equivalence of CFGs, minimization of DFAs

Relationships Between Classes

EXPTIME $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE $\subseteq$ NEXPSPACE $\subseteq \cdots \subseteq$ EXPTIME
Non-trivial problems in these classes are logic related,
e.g., class satisfiability in ER and UML class diagrams