CS 360 - MODULE 9 - ADDITIONAL NOTES

COLLIN ROBERTS

1. Reductions Are Highly Directional

Here we exhibit a choice of decision problems P_1, P_2 (which are questions of membership in the languages L_1, L_2 respectively over $\Sigma = \{0, 1\}$) such that there exists a reduction from P_1 to P_2, but there does **not** exist a reduction from P_2 to P_1.
Let

$$\begin{aligned} L_1 &= L(0^*) \\ L_2 &= \{M \mid L(M) \text{ is non-regular}\} = L_{nreg}. \end{aligned}$$

Note that:

- L_1 is regular. Therefore L_1 is a DCFL, a CFL, and a decidable language.
- $L_2 = L_{nreg}$ is undecidable. This is proved explicitly in Module 9, and is also easily proved directly using Rice's Theorem.
- Since some TMs have regular languages, while other TMs have non-regular languages, we have that $L_2 \neq \emptyset$ and $L_2 \neq \Sigma^*$.
- Then by Problem 2a on CM A06, there exists a reduction from P_1 to P_2.
- Now for a contradiction, assume that there exists a reduction from P_2 to P_1.
- Then, since P_2 is not decidable, Theorem 9.7 implies that P_1 is undecidable. This contradiction shows that no reduction from P_2 to P_1 exists.

2. If a Language and it Complement are Both CFLs, Does It Follow That the Language a DCFL?

Here we present a counterexample to show that this statement does **not** hold in general. Let

$$\begin{aligned} \Sigma &= \{0, 1\} \\ L &= \{w \in \Sigma^* \mid w^R = w\}, \end{aligned}$$

i.e. L is the language of **palindromes** over Σ.
We have that L is a CFL, generated by the grammar $G : S \to \varepsilon|0|1|0S0|1S1$.
It is an exercise to prove that the complement L' is also a CFL. I suggest constructing a PDA to recognize even-length non-palindromes (an old assignment question for CS 360), and another PDA to recognize odd-length non-palindromes. This shows that both languages are CFLs, and then by the closure rules for CFLs, their union is also a CFL.
We argued informally in class that L is **not** the language of any DPDA, in other words, L is not a DCFL.

3. The Halting Problem Is Recursively Enumerable, But Not Recursive

Recall this definition of the Halting Problem:
$L_{Halt} = \{(e, w) \mid \text{TM represented by } e, \text{ halts when processing } w\}$
Construct the Halting Turing Machine, M_{halt}, to simulate a Turing machine M, on an input w.

(1) Input: a pair, (e, w).
(2) If e does not represent any TM, then M_{halt} rejects (e, w).

(3) Otherwise, if $e = f(M)$ for some Turing machine M, then
   (a) If M accepts w, then M_{halt} accepts (e, w).
   (b) If M rejects w, then M_{halt} accepts (e, w).
   (c) If M runs forever on w, then M_{halt} runs forever on (e, w).

(1) The halting language, L_{Halt}, is **recursively enumerable**.
   (a) M_{halt} will have four tapes:
      (i) Keep (e, w), which really is (M, w)
      (ii) Mimic M's tape
      (iii) Maintain the current state, q, of M
      (iv) Use for scratch work
   (b) To simulate one transition ($\delta(q, a)$) from M, inside of M_{halt}:
      (i) Use tape #1, to look up M's transition $\delta(q, a)$.
         (A) Important: e on tape #1 is the full specification of M.
      (ii) Update M's state on tape #3.
      (iii) Update M's tape contents on tape #2; move the tape head L or R.
      (iv) Repeat as needed.
   (c) M_{halt} is a Turing machine; its language, L_{halt}, is r.e.

(2) L_{halt} is not **recursive**.
   (a) We present a reduction from L_u to L_{halt}.
   (b) Let (e, w) be any candidate for L_u.
   (c) Construct new TM, M'_{halt}, to carry out the following algorithm:
      (i) On any input $x \in \{0, 1\}^*$ (say),
         (A) Ignore x and run U on (e, w).
         (B) If U accepts (e, w), then make M'_{halt} halt and accept x
         (C) If U rejects (e, w), then make M'_{halt} execute an infinite loop
         (D) U can run forever on (e, w), in which case M'_{halt} runs forever on x.
      (ii) By construction, M'_{halt} halts for all inputs x, if and only if U accepts (e, w).
      (iii) Take $(M'_{halt}, 0)$ as the corresponding instance for L_{halt}.

This reduction, plus Theorem 9.7, prove that L_{halt} is not recursive.