Assignment 11: Grammars and Turing Machines

Due: Wednesday, November 23

- Emails about this assignment should be directed to `cs81help@cs.hmc.edu`.
- The usual collaboration rules apply. You may *discuss* an exercise with any other student(s) currently taking CS 81 as long as:
  - You contribute equally;
  - You come away from this discussion only with *understanding in your head* — no written materials or computer notes may be retained;
  - Your submission is authored solely by you, on a separate occasion.
- You should refer only to materials from this semester of CS 81 (lecture notes, handouts, textbooks, grutors, profs, etc.).
- Bring a writeup/printout to class. Illegible answers will get no credit.
- Make sure your submission includes your name!

1. Read Chapters 19 and 20 of Rich.

   Then read (or at least skim) Chapter 17 on Turing Machines. Be careful, however, not to get bogged down in the precise details of *programming* Turing Machines to solve specific problems; the goal here is to make sure you have a decent understanding of the capabilities of Turing Machines.

   Come up with two questions about the reading; these should be questions where you're not sure of the answer. These may relate to points where the book is confusing, or simply to some related question or conjecture that occurs to you while doing the reading.

2. Do Problem 1(a-c) on page 445 of Rich. You may assume that the decidable languages in question are provided by giving descriptions of Turing Machines that always halt. You may also assume the Church-Turing Thesis, i.e., that if you can describe an algorithm in C/Python/Pseudocode/etc. then a Turing Machine could do the work.

   For example, here is an argument that decidable languages are closed under intersection:

For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We can construct a TM to decide $L_1 \cap L_2$ by following this algorithm:

> Given an input x, first simulate M_1 running on x and then simulate M_2 running on x. If both accept, we accept x. If either machine rejects, we reject x.

3. Do Problem 2(a,e) on page 445 of Rich.

4. Describe (clearly, but informally) how a finite state machine like a PDA but with *two* stacks could simulate a Turing Machine. (Big hint: consider the part of the tape to the left of the read/write head, and the part of the tape to the right.)

5. Some things would become much, much easier if halting were decidable.

   Goldbach's Conjecture states that every even integer greater than 2 can be expressed as the sum of two primes. It has not yet been proved, but (according to Wikipedia) all even numbers up to 1.609×10^{18} have been checked so far, and each has turned out to be a sum of primes.

   If we had a way of determining in finite time whether an arbitrary program will halt or not, we could use it to determine the truth of Goldbach's Conjecture once and for all. Explain how.

6. **The Elusiveness of Undecidability.** It is widely known that Halting (membership in $H = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$) is not decidable. But:

   (a) Consider the following machine, which we will call $\texttt{TM}_1$. It first confirms that the tape contains a number expressed in binary (and if not, rejects). It then enters the following loop:

   - All digits are zero, it halts and accepts (ending the loop).
   - Otherwise, it decrements the binary number by 1 (by completely overwriting it.)

   Does the machine $\texttt{TM}_1$ halt on the input 10000000000000?

   (b) Consider a similar machine, which we will call TM_2. It is exactly the same as TM_1, except that each time around the loop it *increments* the binary number by 1. Does the machine $\texttt{TM}_2$ halt on the input 10000000000000?

   (c) Why don't your answers to Parts (a) and (b) contradict the undecidability of Halting?

(d) $\texttt{TM}_1$ and $\texttt{TM}_2$ are admittedly not very interesting. You are probably wondering, since Halting is undecidable, where the hard Turing Machines are. Suppose I were to design a very complicated $\texttt{TM}_3$, and prove mathematically that "there is no series of steps by which you can determine whether $\texttt{TM}_3$ terminates on the input 10000000000000" (essentially, that "you will not and cannot know, even in principle, whether $\texttt{TM}_3$ terminates on this input"). Show that this supposition leads to a contradiction.

(It follows that, although there's no algorithm to solve the halting problem in general, we cannot point to any *specific* instance as the undecidable input.)

7. **LL(1) Grammars.**

For any $z \in (\Sigma \cup V)^*$ (i.e, z is a sequence of terminals and nonterminals) we can define the set $\textsc{First}(z) \subseteq \Sigma$ as follows:

$$\textsc{First}(z) := \{\, \sigma \in \Sigma \mid \exists w \in \Sigma^*.\, z \Rightarrow^* \sigma w \,\}$$

That is, $\textsc{First}(z)$ contains all *terminals* that can begin a string derivable from z.

In class, we defined an $\mathrm{LL}(1)$ grammar to be one in which predictive parsing can be made to work without guessing, making unambiguous predictions by peeking ahead only at 1 upcoming input symbol.

It turns out that a grammar that does not mention ε is $\mathrm{LL}(1)$ if and only if the following condition holds:

Condition 1 *For every nonterminal A and every pair of distinct productions $A \to z_1$ and $A \to z_2$ we have $\textsc{First}(z_1) \cap \textsc{First}(z_2) = \emptyset$.*

(a) Give two reasons that the following grammar is not LL(1):

$$\begin{array}{lcl}
S & \to & \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S \\
 & \mid & \texttt{begin}\ \texttt{end} \\
 & \mid & \texttt{begin}\ L\ \texttt{end} \\
 & \mid & \texttt{print}\ E \\
L & \to & S \\
 & \mid & L\ \texttt{;}\ S \\
E & \to & \texttt{true} \\
 & \mid & \texttt{false}
\end{array}$$

(b) The grammar

$$\begin{array}{lcl} S & \to & E\,; \\ E & \to & T\,E' \\ E' & \to & +\,T\,E' \\ & \mid & -\,T\,E' \\ & \mid & \varepsilon \\ T & \to & 0 \mid 1 \end{array}$$

contains ε and is LL(1). Demonstrate this by building a 2-D table of predictions indexed by nonterminals in $\{S, E, E', T\}$ and terminals in $\{;, +, -, 0, 1\}$. For each combination of nonterminal V and terminal x, state which grammar rule we should use as our prediction when (1) we expect the next segment of the input to be derivable from V, and (2) we peek ahead and see that the upcoming input character is x. (E.g., if we are expecting T and see 1, we probably want to predict $T \to 1$.) In some cases (e.g., we expect T, the next character is $+$) no prediction can work regardless of what follows the first character of the input; the answer for such combinations should be "error."

(c) The grammar

$$\begin{array}{lcl} S & \to & A\,a \\ A & \to & a \\ & \mid & \varepsilon \end{array}$$

satisfies Condition 1 but is not $\mathrm{LL}(1)$. Show that knowing only the first terminal in the input is not always enough to guarantee you are making the "right" prediction.

[For grammars containing ε, there is a more complicated condition that needs to be checked before we can be sure that a grammar is $\mathrm{LL}(1)$.]