CS 361 Meeting 22 — 11/2/16

Announcements
1. Pizza/course information session tonight, 6PM, Bio 211?

2. Homework 6 due Monday
   - Answers to questions 1, 2 and 5 should involve detailed, formal specifications of the behavior of the machines/grammars involved.
   - Answers to questions 3 and 4 should be precise but more high-level.

3. Midterms will be graded this year!

Languages about Automata

1. We are getting close to the goal of showing an example of a problem (i.e., language) that is not computable (i.e., decidable). The first example of such a language (and many of the examples of such languages) is a language that involves statements about automata.

2. To get ready for such languages, it makes sense to spend a little time talking about languages that make simpler statements about automata. That is, to talk about decidable languages about automata.

3. A few examples of the sorts of languages I have in mind include:
   - $A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA and } w \in L(B) \}$
   - $A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression and } w \in L(R) \}$
   - $E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}$
   - $E_{QDFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$
   - $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG and } w \in L(G) \}$
   - $E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \}$
   - $E_{QCFG} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(A) = L(B) \}$

4. In the descriptions of all of these sets, the angle brackets, $\langle \rangle$ are included to suggest that descriptions of the grammars, automata, etc. that are included in these languages must somehow be encoded in a precise way.
   - To determine that any of these languages is Turing-recognizable or Turing-decidable, we would need to describe some Turing machine that recognized the language in questions. This machine would have some fixed alphabet $\Sigma$, but we wish to include machines and grammars over all alphabets in these languages. Accordingly, we will have to somehow encode arbitrary finite alphabets in some single alphabet.

5. The exact encoding scheme used is usually unimportant and rarely explicitly discussed in the proof of a language’s decidability, but before we start ignoring these details, I thought it would be helpful to think concretely about how we might represent one of these languages. So, let’s think about how we might represent the strings in $A_{CFG}$.
   - This mainly boils down to how do we represent an arbitrary CFG in some fixed alphabet.
   - I would like you all to take a few minutes to design a scheme for representing arbitrary CFGs given a fixed alphabet like:
     \[ \Sigma = \{ \to, \#, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, V, T \} \]
    - As an example, consider the grammar $G =$
      \[ L \to 1L1 \mid +R \]
      \[ R \to 1R1 \mid = \]
     That is, be prepared to show how to construct the representation of $\langle G, 1+1=1 \rangle$.
   - There are many ways to do this. For example, we could number the elements of the alphabets of terminals and variables (non-terminals) and then use “$Tn$” to encode the $n$th terminal and “$Vn$” for the $n$th variable.
• We would then give up on the $|$ symbol and just list all of the productions separated by # as in:

$$V_1 \rightarrow T_1V_1T_1#V_1 \rightarrow T_2V_2#V_2 \rightarrow T_1V_2T_1#V_2 \rightarrow T_3$$

• We could complete the representation by adding $w$ separated by the remaining symbols with a pair of #’s.

$$V_1 \rightarrow T_1V_1T_1#V_1 \rightarrow T_2V_2#V_2 \rightarrow T_1V_2T_1#V_2 \rightarrow T_3##T_1T2T3T1$$

• Of course, we could have used binary numbers instead of decimal, or unary, or ...

6. All of the language we are interested in should only contain strings that are valid given the representation scheme we have chosen. For example, given our scheme for CFGs, a string that contained two consecutive T’s should immediately be rejected.

7. Fortunately, for reasonable representation schemes, the language of syntactically correct encodings is regular, context-free, or at worst decidable, so we can assume that any TM we describe to recognize one of these languages begins by rejecting anything with invalid format.

A Quick Flashback on Nondeterministic Turing Machines

1. At the end of last class, one student requested that I plan to spend a bit of time reviewing and answering questions about the scheme I presented for simulating a nondeterministic Turing machine using a deterministic Turing Machine.

2. The first fact I want to share is that my approach is quite different from the one in the book. I said that explicitly about my presentation about multi-tape Turing machines, but never really warned you that I was doing the same with nondeterminism.

3. Next, I want to make sure that you all understand the relationship between the nondeterministic machine I used as an example:

The contents of the nodes of the tree are configurations of the Turing Machine.

The root is labeled with the start configuration.
• If a is a child of b, then a yields b.
• When nondeterminism is present in the machines transition function, a node will have multiple children.

4. If the structure of the tree is clear, the key to understanding the way the deterministic Turing machine simulates the nondeterministic one is that it is trying to build the tree breadth-first. Of course, building a tree on a one-dimensional Turing machine tape is a bit hard. So, what it is really trying to do is list all the configurations that will appear in the tree on its tape in the order they would be encountered by making a breadth first traversal of the tree.

5. The final detail of the process that I fear may have confused some is the way the configurations from the nodes of the tree are encoded on the Turing machine tape. That is why I saved this flashback for after we talked about encodings.
   • Each configuration $c_i = (p_i, q_i, t_i), q_i \in Q, p_i, t_i \in \Gamma^*$ is encoded by basically dropping the parentheses.
   • The sequence of configurations $c_0, c_1, c_2, \ldots$ is encoded as $\langle c_0 \rangle \langle c_1 \rangle \langle c_2 \rangle \cdots$ using dollar signs to separate the configurations whose children have already been written on the tape and ‡’s around configurations whose children still need to be computed.

Decidable Properties of DFAs/Reg Exps.

1. OK. Enough of that! Let’s get back to talking about language based on questions about automata, grammars, etc.

2. Assuming now that the particular representation we use is not critical, let’s consider whether some of the problems mentioned above are decidable.

3. Consider the first language:

$$ A_{DFA} = \{ \langle B, w \rangle \mid D \text{ is a DFA and } w \in \mathcal{L}(B) \} $$

4. We can show that this language is decidable by arguing that a TM can read in a description of a DFA from its input tape and then simulate that DFA. We will assume that the representation used for a DFA is essentially a list of triples of the form $(q, x, \delta(q, x))$ describing the machine’s transition function together with descriptions of its initial state, its final states and the number of states and symbols in its alphabet.

To make the explanation of the simulation as simple as possible, we will assume a 3-tape machine.
   • First, the machine will find $w$ at the end of its first/input tape, copy it to its second tape, and erase it from the first tape, leaving mainly a list of transition function triples on the first tape.
   • Next, the machine will write the description of the initial state on its third tape.
   • Mainly, the machine will repeatedly (until it reaches the last input symbol):
     - Scan its first tape to find a $(q, x, \delta(q, x))$ with $x$ matching the symbol encoded starting with the position of the second tape head and $q$ matching the symbol on the third tape.
     - Move the second tape head to the beginning of the next encoded symbol
     - Copy the state $\delta(q, x)$ from the triple found on the first tape over the state that had been stored on the third tape.

5. I would like you to take a little class time to try to show that two of the other languages listed above are decidable, but first, let’s talk about one more so that you have a better sense just how much you can assume.

6. The language I have in mind is

$$ A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression and } w \in \mathcal{L}(R) \} $$

   • In some sense, this is the same as $A_{DFA}$, since the language recognized by DFAs are exactly the same as those described by regular expressions.
• In particular, we know that there is an algorithm that can convert a regular expression into a NFA and then the subset construction can convert this NFA into a DFA.

• Turing machines can be used to implement algorithms!

• Therefore, we can say that $A_{REX}$ is decidable because we can build a TM that uses the algorithms we studied earlier to convert the regular expression in its input into a DFA and then uses the machine we described above for $A_{DFA}$ to finish the job!

7. Next, let’s think a bit about how we could argue the following two languages are decidable.

• $E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}$, and

• $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG and } w \in L(G) \}.$

8. Both of these can be shown using results we showed (much) earlier.

9. For $E_{DFA}$, we can use the pumping lemma to argue that if $L(A) \neq \emptyset$, then there must be some $w \in L(A)$ with $|w| \leq |Q|$ (where $Q$ is the DFA’s set of states). We can easily construct a TM that enumerates all strings over the DFA’s alphabet that are shorter than $|Q| + 1$ in alphabetical order and uses the machine for $A_{DFA}$ as a function/method to check if any of these strings belong to $L(A)$.

10. Alternately, we can treat $E_{DFA}$ as a graph search problem. Viewing the machine’s state diagram as a graph, if there is any path from the start symbol to a final state in the graph, then the machine must accept the string whose symbols label the edges of the path. A breadth-first search of the graph will find all reachable states.