CS 361 Meeting 26 — 4/29/20

Lot's of Languages

(Click for video)

1. We have now identified specific examples of languages that are decidable, recognizable but not decidable, and not recognizable but have recognizable complements. I would like to round things out by showing a specific language such that neither the language nor its complement is recognizable. There are lots of of such languages and lots of languages for us to think about!

2. REALLY! LOTS!

3. If you think about the counting argument I discussed in our first class this semester, you will realize in that discussion, I suggested that there had to be functions that were not computable because there are uncountably infinitely many functions and only countably infinitely many programs (now known as Turing machines).

4. You may mistakenly think that the undecidable or unrecognizable languages we have identified are examples of these uncomputable functions I was discussing during our first class. They aren't.

5. Consider how many languages there are that are recognizable (including the ones that are decidable).

   - Every such language can be "paired" with a Turing machine.
   - Every Turing machine can be encoded as a string over some preselected alphabet.
   - There are only countably infinitely many strings over any language.
   - This implies that there are only countably many recognizable languages (including all the undecidable ones).

6. A similar argument says that the elements of the set of all languages that are not recognizable but have recognizable complements can be paired with the TMs that recognize their complements. Therefore the set of all such languages is also countable.

7. Recall! The set of all languages over a given alphabet is uncountable.

8. So, there must be uncountably many languages that are not associated with any Turing machine. These are the language that are not recognizable and have complements that are also not-recognizable. We still haven't identified a single language that belongs to this much larger group.

Estimating the Computational Challenge of a Language

(Click for video)

1. Before you try to prove that a language is decidable, recognizable or not, it is a good idea to get an intuition for how computationally challenging the language is by considering relatively naive approaches to identifying its members.

2. Let's start by trying to develop some intuition about the correct categories for the following eight languages in the hope that we might find some candidates for the languages that are neither recognizable or co-recognizable:

   - $EQ_{CFG} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are CFGs and } \mathcal{L}(A) = \mathcal{L}(B)\}$
   - $\overline{EQ_{CFG}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are not CFGs or } \mathcal{L}(A) \neq \mathcal{L}(B)\}$
   - $ALL_{CFG} = \{\langle M\rangle \mid M \text{ is a CFG and } \mathcal{L}(A) = \Sigma^*\}$
   - $\overline{ALL_{CFG}} = \{\langle M\rangle \mid M \text{ is not a CFG or } \mathcal{L}(A) \neq \Sigma^*\}$
   - $EQ_{TM} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are TMs and } \mathcal{L}(A) = \mathcal{L}(B)\}$
   - $\overline{EQ_{TM}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are not TMs or } \mathcal{L}(A) \neq \mathcal{L}(B)\}$
   - $ALL_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } \mathcal{L}(A) = \Sigma^*\}$
   - $\overline{ALL_{TM}} = \{\langle M\rangle \mid M \text{ is not a TM or } \mathcal{L}(A) \neq \Sigma^*\}$

3. With a little thought (and luck) you should be able to guess that:

   - None of these examples are decidable.

Click here to view the slides for this class

- $\overline{EQ_{CFG}}$ and $\overline{ALL_{CFG}}$ are recognizable since we can convert the grammars into Chomsky normal form and then enumerate all strings checking to see if some string is in one language but not the other (for $\overline{EQ_{CFG}}$) or that a given string is not in $\overline{ALL_{CFG}}$ and accepting as soon as we find an example.
- EQ_{CFG} and ALL_{CFG} are probably not recognizable (since if there is no counter-example, the procedure described in the preceding bullet point loops forever). Their complements, however, seem recognizable.
- Neither EQ_{TM}, ALL_{TM} or their complements are clearly recognizable because a TM may loop forever on a single a string if it is not in its language so we may reach a string that is a counter-example but never be sure it is because the machine just loops rather than reaching a reject state.

4. So, the best candidates for our goal of finding examples of a pair of a language and its complement that are both not recognizable seem to be EQ_{TM}, ALL_{TM}.

A Language that is not Recognizable or co-Recognizable

(Click for video)

1. Let's consider ALL_{TM}. Our hunch is that this language and its complement are both not recognizable. That means that proving they are undecidable won't help much. That would tell us that one of them must not be recognizable, but we would not know which one was not recognizable or that both of them are actually not recognizable.

2. Instead, we will seek two separate reduction proofs that ALL_{TM} and its complement are not recognizable.

3. To start, we will first assume that $\overline{ALL_{TM}}$ is recognizable and try to show that if this was true we could construct a TM to recognize some language like $\overline{A_{TM}}$ or E_{TM} that we already know is not recognizable.

   - We will use $\overline{A_{TM}} = \{\langle M, w\rangle \mid M \text{ is not a TM or } w \notin \mathcal{L}(M)\}$
   - Since we will be using proof by contradiction, we begin by assuming that $\overline{ALL_{TM}}$ is recognized by some TM $M_{\overline{ALL}}$.
   - Next, we construct a machine $M_{\overline{A_{TM}}}$ that will use $M_{\overline{ALL}}$ to recognize $\overline{A_{TM}}$. This machine behaves as follows:
     - Accept the input if it is not a valid TM + input description
     - On input $\langle M, w\rangle$, construct a description, $\langle M'\rangle$ of a TM M' that behaves as follows:
       * On input w', simulate M on input w. Accept w' if M accepts w. Otherwise, reject.
     - Run $M_{\overline{ALL}}$ on $\langle M'\rangle$ and accept if it does.
   - The way we construct M' in our $M_{\overline{A_{TM}}}$ ensures that $\mathcal{L}(M') = \Sigma^*$ exactly when $w \in \mathcal{L}(M)$ and $\mathcal{L}(M') = \emptyset$ exactly when $w \notin \mathcal{L}(M)$. Therefore, $< M' > \in \overline{ALL_{TM}}$ exactly when $w \notin \mathcal{L}(M)$.

4. If you are having deja vu, that is reasonable. The details of this proof (notably the construction of M' should seem very familiar from our argument that E_{TM} was undecidable.

ALL_{TM} is not Recognizable Either

(Click for video)

1. Next, we will assume that ALL_{TM} is recognizable and try to show that if this was true we could construct a TM to recognize some language like $\overline{A_{TM}}$ or E_{TM} that we already know is not recognizable.

2. This isn't as easy as the first proof. A first guess would be that we could just reverse the way the M' used in out argument for $\overline{ALL_{TM}}$ behaved.

   - That is given $\langle M, w\rangle$, construct a description, $\langle M'\rangle$ of a TM M' that behaves as follows:
     - On input w', simulate M on input w. Accept w' if M rejects w. Otherwise, reject.
   - Unfortunately, this machine cannot do what its description says (accept w') in the case that M rejects w by looping because it can never be certain that M is looping rather than just running for a long time.

3. We need to somehow turn M looping (or rejecting) into M' accepting all strings and M accepting into not accepting all strings.

4. We can do this by making M' accepts strings as long as M takes more steps to decide how to handle w than some function that grows without bounds as a function of w. Length of w is such a function.

   - That is given $\langle M, w\rangle$, construct a description, $\langle M'\rangle$ of a TM M' that behaves as follows:
     - On input w', simulate M on input w for $|w'|$ steps. Reject w' if M accepts w within the period of simulation. Otherwise, accept w'.
   - If M rejects or loops on w, $L(M') = \Sigma^*$. If M accepts w, then $L(M')$ is the finite set of strings whose length is less than the number of steps M executes before accepting w.

5. The complete proof looks like:

   - We will use $\overline{A_{TM}} = \{\langle M, w\rangle \mid M \text{ is not a TM or } w \notin \mathcal{L}(M)\}$
   - Since we will be using proof by contradiction, we begin by assuming that ALL_{TM} is recognized by some TM M_{ALL}.
   - Next, we construct a machine $M_{\overline{A_{TM}}}$ that will use M_{ALL} to recognize $\overline{A_{TM}}$. This machine behaves as follows:
     - Accept the input if it is not a valid TM + input description
     - On input $\langle M, w\rangle$, construct a description, $\langle M'\rangle$ of a TM M' that behaves as follows:
       - On input w', simulate M on input w for $|w'|$ steps. Accept w' if M has not accepted w during the simulation. Otherwise, reject.
     - Run M_{ALL} on $\langle M'\rangle$ and accept if it does.
   - The machine we have described will accept $\langle M, w\rangle$ iff $w \notin \mathcal{L}(M)$ since if $w \notin \mathcal{L}(M)$, M' will accept all input since no matter how long the input is, simulating M will not accept w if simulated for as many steps as the length of the input. On the other hand, if M does eventually accept w, then M''s language will be finite.

6. It should be clear that the fact that neither ALL_{TM} nor its complement is recognizable puts EQ_{TM} in the same category.

   - If EQ_{TM} was recognizable by $M_{EQ_{TM}}$, we could use $M_{EQ_{TM}}$ to recognize ALL_{TM} by building a machine that would take its input $\langle M\rangle$ and convert it into an input of the form $\langle M, M_{\Sigma^*}\rangle$ for $M_{EQ_{TM}}$ where M_{Σ^*} is a machine that accepts Σ^* by simply consisting of a single state that is both a start state and the accept state.
   - We could do the same trick for $\overline{EQ_{TM}}$ by using a machine that could recognize $\overline{EQ_{TM}}$ to build a machine that could recognize $\overline{ALL_{TM}}$.

Computation Histories

(Click for video)

1. Some very interesting proofs of undecidability rely on the technique of constructing a language that describes the possible computations of a TM on one or more inputs.

2. Examples of such proofs include showing that two of the CFG-related languages from our list: EQ_{CFG} and ALL_{CFG} are not recognizable.

3. Recall that a TM configuration is a triple (q, u, v) with $q \in Q$ representing the current state of the control, $u \in \Gamma^*$ representing the contents of the tape to the left of the current head position, and $v \in \Gamma^*$ representing the tape contents from the head to the right end of the non-blank tape.

4. We can use a sequence of strings that describe configurations to describe the complete computation of a TM.

   Definition: Given a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ and a string $w \in \Sigma^*$, we define the language of accepting computation histories of M on w as

   $$L_{Computation-history}(M, w) = \{w_0 w_1 ... w_n \mid \begin{array}{l}\text{each } w_i \text{ is an encoding of a configuration for } M \\ w_0 \text{ is the initial configuration for input } w \\ w_n \text{ is a final/accept configuration} \\ \text{each } w_i \text{ yields } w_{i+1} \text{ according to } \delta\ \}\end{array}$$

5. Interestingly, if M is deterministic, then $L_{Computation-history}(M, w)$ is actually a very simple language. It is either equal to $\emptyset$ if $w \notin L(M)$ or it contains a single string if $w \in M$.

   - This is interesting because the empty language and any language containing just 1 (or any finite number) of strings must be regular and therefore context-free.
   - If given the description of M and w we could build a DFA that accepted $L_{Computation-history}(M, w)$ then since emptiness of a regular language is decidable, we could decide whether $w \in L(M)$. This question is better known as A_{TM} and we know that it is undecidable!
   - This implies that even though $L_{Computation-history}(M, w)$ must be regular if M is deterministic, there is no algorithm to find a DFA for the language given a description of M. The DFA is uncomputable!

6. This reflect the idea that the fragments of Turing machines we called transducers have their limitations too! We can formalize the idea of what a transducer can do with a definition.

   > Definition: A function $f : \Sigma^* \to \Sigma^*$ is computable iff there is a Turing machine M such that on every input w, M halts with $f(w)$ on its tape.

   - The function $f(\langle M, w\rangle) = \langle M'\rangle$ where $\langle M'\rangle$ is an encoded description for a PDA or DFA with $L(M') = L_{Computation-History}(M, w)$ must not be computable function!