CS 361 Meeting 27 — 5/1/20

Almost Computable Histories

(Click for video)

1. While descriptions of the languages of computation histories of a TM is not computable, there is a related family of languages with computable descriptions.

2. To get a bit closer to being able to find a way to compute $L_{Computation-history}(M, w)$, we can define a slight variation of this language:

   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 reversed computation histories of M on w as

   $$\begin{array}{l} L^R_{Computation-history}(M, w) = \\ \qquad \{w_0 w_1^R ... w_n \mid \begin{array}{l} \text{each } w_i \text{ is a configuration for } M \\ w_0 \text{ is the initial configuration for } w \\ w_n \text{ is a final/accept configuration} \\ \text{each } w_i \text{ yields } w_{i+1} \text{ according to } \delta \\ \text{every other } w_i \text{ is written backwards } \} \end{array} \end{array}$$

3. Again, if M is deterministic, this language is finite and therefore both regular and context-free.

4. You can imagine that it would be possible to construct a CFG or a PDA that ensures that all even-odd pairs of strings or all odd-even pairs of configurations in such strings satisfied the "yields" requirement.

   - Since the orientations of consecutive configurations alternate, a PDA can check any pair by
     - Pushing the contents of the first configuration on its stack.
     - As it reads the next configuration in the input, pop the symbols of the preceding configuration from the stack and make sure that they either match the current configuration, or
     - Check that the differences are consistent with the transition function, δ, of the Turing machine considered.
   - Alas, if configurations i and $i + 1$ are checked in this way it is impossible for the PDA to check $i + 1$ agains $i + 2$ since its stack is now empty!

5. Somewhat amazingly the complement of $L^R_{Computation-history}(M, w)$ which is far from finite is context-free and we can effectively construct a PDF or CFG that describes it given a description of M and w.

   - Fundamentally, to recognize a string belongs to the complement, a PDA just needs to find one thing "wrong" with it. One pair of mismatched consecutive configurations is sufficient. We don't need to check all pairs.
   - We can describe a PDA that uses its non-determinism to guess where the feature that would make a string invalid as a computation history occurs and then use the stack of the PDA to validate the guess.
   - The PDA guesses one of the following issues:
     - The whole string is not formatted properly,
     - The first configuration is not a valid initial configuration for w,
     - The last configuration is not an accepting configuration.
     - For any pair of consecutive configurations, the earlier configuration does not yield the following configuration (a PDA could nondeterministically guess which pair didn't match, push the first configuration on its stack and then verify the mismatch as it read the next configuration),
   - In the first three cases, checking the guess only requires recognizing that the input belongs to a regular languages. This is easiest to see in the second case. There is one, known, correct initial configuration. We know that a DFA can recognize the set of strings that do not start with any given prefix. So, if our PDA guesses this is what is wrong, it takes a transition to a set of states that implement this DFA.

Click here to view the slides for this class

- For the last option, the PDA has to compare a pair of consecutive configurations as suggested above to make sure it can find a mistake,

6. Using this fact, we can show that ALL_{CFG} is not recognizable by reducing $\overline{A_{TM}}$ to ALL_{CFG}. To do this, we will assume the existence of a TM $M_{ALL_{CFG}}$ that recognizes ALL_{CFG} and build a TM $M_{\overline{A_{TM}}}$ that recognizes $\overline{A_{TM}}$.

   - $M_{\overline{A_{TM}}}$ should accept its input if it is not a valid encoding of a TM description and input.
   - Next, $M_{\overline{A_{TM}}}$ should create a CFG $G_{\overline{M}}$ for $\overline{L^R_{Computation-history}(M, w)}$.
   - Finally, $M_{\overline{A_{TM}}}$ should run $M_{ALL_{CFG}}$ on $\langle G_{\overline{M}} \rangle$ and accept if $M_{ALL_{CFG}}$ accepts.
   - This machine recognizes $\overline{A_{TM}}$ because M has an accepting computation history on w iff $w \in L(M)$. As a result, $\overline{L^R_{Computation-history}(M, w)} = L(G_{\overline{M}}) = \Sigma^*$ exactly when $w \notin L(M)$.
   - Since it would be a contradiction if we could recognize $\overline{A_{TM}}$, it must be impossible to recognize ALL_{CFG}.

7. Since we can recognize $\overline{ALL_{CFG}}$ by simply checking all strings looking for one that cannot be derived from the grammar, this implies that $\overline{ALL_{CFG}}$ is recognizable but not decidable.

8. If we could decide EQ_{CFG}, we could decide ALL_{CFG} by taking the input CFG and comparing it to a CFG that generates Σ^*. Thus, we can also conclude that $\overline{EQ_{CFG}}$ is recognizable but not decidable.

Mapping Reductions

(Click for video)

1. We have observed that many of the proofs of undecidability and non-recognizability we have explored have a very similar structure.

2. We can formalize these similarities in the notion of mapping reducibility and then use this idea to "simplify" the proofs for many results involving decidability and recognizability.

3. First we should review the idea of a computable function.

   Definition: A function f from $\Sigma^* \rightarrow \Sigma^*$ is *computable* if and only if some Turing machine M on every input w, halts with $f(w)$ on its tape.

4. This definition is mainly an admission that Turing machines can do interesting things other than just accept and reject.

   - This is not new. One of the first TMs we considered implemented a computable function. It took input strings and did its best to insert a # in the middle of them.
   - In each of the non-recognizability proofs we have given, we have embedded such a computable function.
     - All of the computations that generated some $\langle M' \rangle$ given some $\langle M, w \rangle$ were examples of computable functions.
     - In the argument for ALL_{CFG}, the conversion of $\langle M, w \rangle$ into a description of a PDA for the complement of the languages of computation histories of M on w is another computable function.

5. Given the notion of computable functions, we can capture the essence of what our M's and similar concoctions are really about.

   Definition: Language A is *many-to-one reducible* to language B (written $A \leq_m B$) if there exists a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that for every $w \in \Sigma^*$

   $$w \in A \Leftrightarrow f(w) \in B.$$

   In this case we call f a reduction.

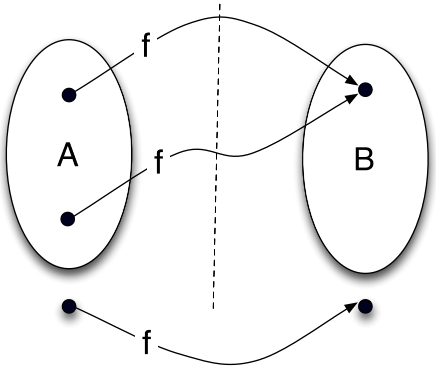
6. Let me share two handy memory aids for dealing with the notation.

(a) The point of the $\leq$ goes in the direction opposite the function arrow.

(b) It helps to read the $\leq$ as "is easier than" rather than "is less than".

7. The following diagram illustrates what this definition requires and allows.

   - It must map members of A to members of B.
   - It must map strings that are not in A to strings that are not in B.
   - It can map multiple input strings to the same output.



8. As a simple example, the computable function

$$f(< M, w >) \;=\; < M' > \quad \text{where } M' \text{ is a TM that ignores its input, runs } M \text{ on } w \text{ and accepts its input if } M \text{ accepts } w.$$

   shows that $A_{TM} \leq_m ALL_{TM}$ since it maps any $< M, w >$ that belongs in A_{TM} to an $< M' >$ that belongs in ALL_{TM}.

9. Note that there is nothing that inherently ties this computable function to A_{TM} and ALL_{TM}. As long as we can identify two languages A and B such that $w \in A \Leftrightarrow f(w) \in B$, the function f justifies the claim that $A \leq_M B$.

   - As an easy example of this, note that this function also shows that $\overline{A_{TM}} \leq_m \overline{ALL_{TM}}$. In general $A \leq_M B \Leftrightarrow \overline{A} \leq_M \overline{B}$.
   - As a more daring example, this function also shows that $\overline{A_{TM}} \leq_m E_{TM}$ since it maps any $< M, w >$ that belongs in $\overline{A_{TM}}$ to an $< M' >$ that belongs in E_{TM}.

Using Mapping Reductions

(Click for video)

1. With these definitions, we can succinctly formalize the technique we have been using in all our proofs for the last few classes:

   Theorem: If $A \leq_m B$ and ...

   (a) B is decidable, then A is decidable,

   (b) A is undecidable, then B is undecidable,

   (c) B is recognizable, then A is recognizable, and

   (d) A is not recognizable, then B is not recognizable

   - We won't give a detailed proof of these claims, but they are all just obvious applications of the proof techniques we have been employing.

2. As an example of how we might use such a mapping reduction, consider the language

$$DISJOINT_{TM} = \{\langle M, M\rangle \mid \text{ M \& N are TMs and } L(M) \cap L(N) = \emptyset\}$$

.

   - We will show that $DISJOINT_{TM}$ is not recognizable by showing that $E_{TM} \leq_M DISJOINT_{TM}$.
   - To do this, we need a mapping that will take any w of the form $\langle M\rangle$ to a pair of TM descriptions $\langle N, N'\rangle$ in such a way that $L(N)$ and $L(N')$ are disjoint if and only if $L(M)$ is empty.
   - Technically, we also have to make sure that in the case that w is a string that is not of the form $\langle M\rangle$ (i.e. it is not a valid

TM encoding) that $f(w)$ is still defined and that in this case, $f(w) \notin DISJOINT_{TM}$. We can do this by either making $f(w)$ be a badly formed string (i.e., not of the form $\langle M_1, M_2 \rangle$) or a string of the form $\langle M_1, M_1 \rangle$ where $L(M_1)$ is non-empty.

- Now, for w of the form $\langle M \rangle$:
  - let $EVERY$ be a TM that accepts all strings over Σ.
  - Consider the function $f(w) = \langle M, EVERY \rangle$ where $w = \langle M \rangle$.
  - This is clearly computable.
  - It is also clear that $f(\langle M \rangle) = \langle M, EVERY \rangle \in C$ if and only if $\langle M \rangle \in E_{TM}$.
- Given that we know that E_{TM} is not recognizable, we can conclude that $DISJOINT_{TM}$ is not recognizable.

3. One can give a similar, short argument that EQ_{CFG} is not recognizable by reducing it to ALL_{CFG}.

   - Let $EVERY$ be a CFG that generates all strings over an alphabet.
   - Consider the computable function $f(w) = \langle G, EVERY \rangle$ when w = $\langle G \rangle$ for some CFG G, and $f(w)$ = any string that is not in ALL_{CFG} otherwise.
   - Given this function, it is clear that $ALL_{CFG} \leq_M EQ_{CFG}$.
   - This implies that EQ_{CFG} is "harder than" ALL_{CFG} in the sense that EQ_{CFG} must not be recognizable since we already know that ALL_{CFG} is not recognizable.

Turing Equivalence

(Click for video)

1. It is important to notice that the symbol we use for Turing-reducability is $\leq_M$ rather than $<_M$. It is "less than or equal" rather than "strictly less than".

2. This suggest, that there may be examples of languages such that $A \leq_M B$ while at the same time $B \leq_M A$. In this case, we say that A and B are Turing-equivalent and write $A \equiv_M B$.

3. As an example of this, consider the languages A_{TM} and $\overline{E_{TM}}$. We already know that these language are of similar difficulty since both are recognizable but both of their complements are not recognizable. They fall in the same bubble of our favorite Venn diagram!

4. Consider the reductions that are possible between these two languages.

5. We can easily show that $A_{TM} \leq_M \overline{E_{TM}}$ using one of our favorite constructions.

   Let $f(w) = \langle M' \rangle$ where:

   - if w is a string of the form $\langle M, w \rangle$ (i.e. a suitable input for A_{TM}), then M' is a Turing machine that on input w' runs M on w and accepts w iff M accepts w.
   - if w is not a string of the form $\langle M, w \rangle$, then M' is a Turing machine that rejects all inputs.

6. With a bit more effort, we can show that $\overline{E_{TM}} \leq_M A_{TM}$.

   Let $f(w) = \langle M', \epsilon \rangle$ where

   - If w is of the form $\langle M \rangle$ for some Turing machine M, then M' is a Turing machine which ignores its own input and instead uses dovetailing to simulate M on all w over its alphabet looking for some $w \in L(M)$. If such a w is found, M' accepts its own input.
   - If w is not a valid input for E_{TM}, the M' should be a Turing machine that accepts all inputs.

7. Given these reductions, we can conclude that $\overline{E_{TM}} \equiv_M A_{TM}$ and, that $E_{TM} \equiv_M \overline{A_{TM}}$.

Rice's Theorem

(Click for video)

1. While we have categorized a large number of languages as decidable, recognizable, or not recognizable, there are still plenty of additional examples we could consider:

   - $REGULAR_{TM} = \{\langle M \rangle \mid L(M) \text{ is regular } \}$

- $CONTEXT\text{-}FREE_{TM} = \{\langle M\rangle \mid L(M) \text{ is context-free }\}$
- $PALINDROME_{TM} = \{\langle M\rangle \mid w \in L(M) \iff w^R \in L(M)\}$
- $EVEN_{TM} = \{\langle M\rangle \mid w \in L(M) \Rightarrow \ |w| = 2\}$
- $PRIME_{TM} = \{\langle M\rangle \mid w \in L(M) \Rightarrow \ |w| \text{ is prime }\}$

2. There is a single theorem that will quickly allow us to show that all of the languages listed above are undecidable.

   - This result is known as Rice's Theorem.

3. Informally, Rice's Theorem says that any nontrivial property of a Turing machine's language is undecidable.

   - Nontrivial means that the languages of some but not all TMs have this property.
   - The fact that it is a property of the language rather than the TMs means that it must be based strictly on the set of strings a given TM accepts rather than on how the TM is designed or operates.

4. We can formalize this notion as:

   Rice's Theorem. Suppose that L is a language with

   $$\emptyset \subset L \subset \{\langle M\rangle \mid \langle M\rangle \text{ is a valid Turing machine}\}$$

   such that if $L(M) = L(N)$ then $\langle M\rangle \in L \iff \langle N\rangle \in L$ then L is undecidable.

   - The first requirement:

     $$\emptyset \subset L \subset \{\langle M\rangle \mid \langle M\rangle \text{ is a valid Turing machine}\}$$

     captures the idea that the property is non-trivial by saying that some TM languages have to have the property while others don't.
   - The requirement

     $$L(M) = L(N) \text{ then } \langle M\rangle \in L \iff \langle N\rangle \in L$$

     captures the idea that all TMs with the same language have to share having the property or not having it. A property like "being a TM with an even number of states" would violate this because one could easily imagine machines with even and odd state counts that recognized the same languages.

5. Recall that the set of decidable sets is closed under complement. So if L is decidable, then $\overline{L}$ is also decidable and vice versa. Therefore, we could restate the theorem as:

   Rice's Theorem'. Suppose that L is a language with

   $$\emptyset \subset L \subset \{\langle M\rangle \mid \langle M\rangle \text{ is a valid Turing machine}\}$$

   such that if $L(M) = L(N)$ then $\langle M\rangle \in L \iff \langle N\rangle \in L$ then L and $\overline{L}$ are undecidable.

6. This minor change in the statement of the theorem is useful, because it allows us to add an assumption about L that will make the proof easier without reducing the strength of the theorem. In particular, we would like to assume that Turing machines whose languages are empty are in $\overline{L}$ rather than L.

   Rice's Theorem: Suppose that L is a language with

   $$\emptyset \subset L \subset \{\langle M\rangle \mid \langle M\rangle \text{ is a valid Turing machine}\}$$

   such that if $L(M) = L(N)$ then $\langle M\rangle \in L \iff \langle N\rangle \in L$ and $L(M) = \emptyset \Rightarrow \langle M\rangle \notin L$, then L and $\overline{L}$ are undecidable.

7. The proof of this theorem is a good opportunity to utilize the notion of mapping reducibility we discussed last class. In particular, to prove this theorem we just need to show that some undecidable set like A_{TM} is reducible to the L in the statement of the theorem. That is, we must show that

   $$A_{TM} \leq_m L$$

   .

8. To do this, we need to find a computable function $f(\langle M, w\rangle) = \langle M'\rangle$ such that

- if $w \in L(M)$ then $\langle M' \rangle \in L$, and
- if $w \notin L(M)$ then $\langle M' \rangle \notin L$

9. Let $\langle M_{inL} \rangle$ be any member of L. Since we assumed that L was not empty we know such a machine description must exist.

10. Define f as follows. Given $\langle M, w \rangle$, f will produce a description of a machine M' that first ignores its input w' and simulates M on w and loops if M either loops or rejects. If, however, the simulation of M would have accepted w, M' will then run M_{inL} on w' and accept or reject w' as M_{inL} would.

11. If $w \notin M$, then $L(M') = \emptyset$ and therefore $f(\langle M, w \rangle) \notin L$ as desired.

12. if $w \in M$, then $L(M') = L(M_{inL}) \Rightarrow \langle M' \rangle \in L$.

13. This completes the proof that $A_{TM} \leq_m L$ and therefore shows that neither L nor $\overline{L}$ is decidable.