Announcements

1. Homework 9 due today

Rice’s Theorem

1. To catch up on a bit of the gap between where we are and where the syllabus says we should be, I am going to almost skip one topic that I normally cover. That is, I am going to tell you about a useful theorem about decidability but instead of proving it, let you read the proof in the text if you are interested.

2. 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:

- \( \text{REGULAR}_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular} \} \)
- \( \text{CONTEXT-FREE}_{TM} = \{ \langle M \rangle \mid L(M) \text{ is context-free} \} \)
- \( \text{REVERSIBLE}_{TM} = \{ \langle M \rangle \mid w \in L(M) \iff w^R \in L(M) \} \)
- \( \text{EVEN}_{TM} = \{ \langle M \rangle \mid w \in L(M) \Rightarrow |w| \text{ mod } 2 = 0 \} \)
- \( \text{PRIME}_{TM} = \{ \langle M \rangle \mid w \in L(M) \Rightarrow |w| \text{ is prime} \} \)

3. 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.
- It is presented as an exercise in Sipser (including a solution).

4. 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.

5. 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 \( L \) and \( \overline{L} \) are undecidable.

6. Even if you aren’t curious enough to read the proof in the text, this result can serve as a quick rule that will let you immediately identify a large class of languages that are not decidable.

Connecting Decidability to Mathematics

1. To conclude our exploration of questions about what is and is not computable, I would like to give a simple example that might help you realize how hard some of these problems really are.

2. Consider the **Goldbach’s Conjecture**: Every even integer greater than 2 can be written as the sum of two primes.

This simple statement has been an open question since June 7, 1742.

3. We can easily describe an algorithm/Turing machine to test whether a number is a counter-example to the Goldbach Conjecture:

\( M_{GB} = \)

- On input \( n \):
  - if \( n \) is odd, reject
  - For all \( 1 < i < n \)
    - if \( i \) is prime and \( n - i \) is prime
      - reject
    - accept

4. The language of this machine is the set of numbers that are counterexamples to the Goldbach conjecture. The language is clearly clearly decidable (the algorithm always halts).
5. On the other hand, if $E_{TM}$ were decidable, applying its solution to $\langle M_{GB} \rangle$ would solve the open mathematical question.

6. To determine if $\langle M_{GB} \rangle \in E_{TM}$ we would have to build a machine that could prove or disprove the Goldbach Conjecture (and any other open mathematical program for which we could write a similar algorithm to search for counter examples). Thus, deciding or recognizing $E_{TM}$ is probably very hard!

**Linear Bounded Automata**

1. Turing machines are a ridiculous model of computation. Real machines do not have infinite memory (and what they have is easier to use).

2. Deterministic Finite Automata are even sillier. The assumption of a fixed memory leaves them too weak for most interesting problems.

3. An alternative worth thinking about is a machine whose memory size grows with its input size in some predictable way. That is, the memory size is determined by some fixed function of the input size.

4. One very simple approach is to limit the memory to be some fixed multiple of the input size.

5. If we limit a Turing machine so that it can never write past the end of its input, we get a model that accomplishes this:
   - Since we can give the machine an alphabet that simulates multiple tapes, the memory available can be $k$ times the input size for any fixed $k$.
   - We call such machines Linear Bounded Automata or LBAs for short.

6. Despite the memory limitation, Linear Bounded Automata are actually fairly powerful.

7. Many of the languages we have determined are decidable by Turing machines are clearly decidable by LBAs:
   - $A_{DFA} = \{ \langle M, w \rangle \mid M$ is a DFA and $w \in L(M) \} \quad \text{Imagine 3 tapes where the machine keeps its input (starting with the description of the DFA) on its first tape, initially copies the input $w$ to the second tape and the start state of $M$ to the third tape. It then repeatedly searches the list of tuples that describe $\delta$ on the description of $M$ on the first tape to determine the correct next state. Once the next state is determined, it gets copied to the third tape and the head on the second tape moves to the next input character. Then the head on the second tape reaches the end of $w$, the machine checks to see if the state on the third tape is in the list of final states on the first tape. The contents of the “extra” tapes will always be no longer than the original input, so this is an LBA.}
   - $E_{DFA} = \{ \langle M \rangle \mid M$ is a DFA and $L(M) = \emptyset \}$
     - If there is any $w$ in $L(M)$, we can argue using the pumping lemma that there must be at least one such input whose length is no greater than the size of $M$’s set of states (which must be less than the length of the input $\langle M \rangle$). So, the same simulation technique described for $A_{DFA}$ can be applied to all strings of size up to the number of states.
   - $ALL_{DFA} = \{ \langle M \rangle \mid M$ is a DFA and $L(A) = \Sigma^* \}$
     - $\langle M \rangle \in ALL_{DFA}$ iff a machine that accepts the complement of $L(M)$ is in $E_{DFA}$. We can easily convert our simulation technique described for $A_{DFA}$ and $ALL_{DFA}$ so that it treats the list of final states as a list of non-final states. With this change, the algorithm for $E_{DFA}$ decides $ALL_{DFA}$.
   - $E_{CFG} = \{ \langle G \rangle \mid G$ is a CFG and $L(G) = \emptyset \}$
     - The algorithm we (briefly) discussed that decides $E_{CFG}$ (see the notes for lecture 24) can be implemented by including enough symbols in the tape alphabet of a machine to enable it to mark the elements of the set of variables known to be useless on the encoding of the CFG provided as input.
3. We can define languages that encode the complete computations of a TM.

- Sorting
- Dijkstra’s algorithm
- Matrix multiplication

9. Given the power of these machines, it seems interesting to ask whether members of our generic set of language questions:

- $A_{??}$ = \{ \langle M, w \rangle | M is a ??? and $w \in \mathcal{L}(M)$ \}
- $E_{??}$ = \{ \langle M \rangle | M is a ??? and $\mathcal{L}(M) = \emptyset$ \}
- $EQ_{??}$ = \{ \langle A, B \rangle | A and B are ???s and $\mathcal{L}(A) = \mathcal{L}(B)$ \}
- $ALL_{??}$ = \{ \langle M \rangle | M is a ??? and $\mathcal{L}(A) = \Sigma^*$ \}

are decidable when applied to LBAs:

- $A_{LBA}$ = \{ \langle M, w \rangle | M is an LBA and $w \in \mathcal{L}(M)$ \}
- $E_{LBA}$ = \{ \langle M \rangle | M is an LBA and $\mathcal{L}(M) = \emptyset$ \}
- $EQ_{LBA}$ = \{ \langle A, B \rangle | A and B are LBAs and $\mathcal{L}(A) = \mathcal{L}(B)$ \}
- $ALL_{LBA}$ = \{ \langle M \rangle | M is a LBA and $\mathcal{L}(A) = \Sigma^*$ \}

Machine Configurations and Computation Histories

1. The technique we used to show $ALL_{CFG}$ depended on a language that encoded computation histories of Turing machines. Similar techniques can be used to show that many properties of LBAs are undecidable (or in at least one case, decidable).

2. Recall that a Turing machine configuration is a triple $(q, x, y)$ where $q$ represents the machine’s current state, $x \in \Gamma^*$ represents the contents of the tape preceding the current position of the read head, and $y \in \Gamma^*$ represent the contents of the tape starting at the read head and continuing until the last non-blank symbol.

3. We can define languages that encode the complete computations of a TM.

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

$$L_{\text{Computation-history}}(M, w) = \{w_0w_1...w_n | \text{ each } w_i \text{ is a configuration for } M, w_0 \text{ is the initial configuration for } w, w_n \text{ is an accept configuration, and each } w_i \text{ yields } w_{i+1} \text{ according to } \delta \}$$

4. While this definition applies to all Turing machines, the computation histories of LBAs have some special properties.

- If $w_0w_1...w_n$ is a computation history of an LBA, then for all $i$ and $j$, $|w_i| = |w_j| = |w| + k$ where $k$ is a small constant that depends on the encoding used.
  - Most likely, $k = 2$, one symbol for the current state embedded in $w$ and another for a separator that appears between configurations.
- Since all the configurations in an LBA computation history must be of the same length, there are only finitely many different configurations.
  - Each of the $|w|$ cells on the tape can hold any of the $|\Gamma|$ tape alphabet symbols, the head can be at any of the $|w|$ tape cells or looking at the space at the end of the input, and the machine can be in any of its $|Q|$ states leading to a total of $|Q| \times (|w| + 1) \times |\Gamma|^{|w|}$ distinct configurations.
- No configuration can appear more than once in a computation history.
  - Configuration histories must end in the machine’s accept state, so they must be finite.
  - If an LBA every re-entered a previous configuration, it would be trapped in a loop and could never terminate.
- If an LBA takes more than $|Q| \times (|w| + 1) \times |\Gamma|^{|w|}$ steps on input $w$, it will never halt.
5. The acceptance problem for LBAs:

\[ A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA and } w \in L(M) \} \]

is therefore decidable!

- We can build a Turing machine which when given an input \( \langle M, w \rangle \) where \( M \) is an LBA, simulates \( M \) on \( w \) for at most \(|Q| \times |w| \times |\Gamma|^{\vert w \vert} \) steps and accepts only if the simulated machine reaches an accept state.

6. Computation histories also give us a way to show that LBAs are powerful enough that the question of whether an LBAs language is empty:

\[ E_{LBA} = \{ \langle M \rangle \mid M \text{ is an LBA and } L(M) = \emptyset \} \]

is undecidable.

- First, it should be clear that for any Turing machine, the language \( L_{\text{Computation-history}}(M, w) \) is decidable.
- Given a string \( h \) on its input tape, a Turing machine designed to decide \( L_{\text{Computation-history}}(M, w) \) would need to verify that \( h \) is a string of the form \( w_0 w_1 \ldots w_n \) where:
  - \( w_0 \) is an encoding of the configuration \( (q_0, \epsilon, w) \).
  - \( w_n \) is a valid encoding where the Turing machine is in \( q_{\text{accept}} \).
  - Each \( w_i \) yields \( w_{i+1} \) according to the Turing machines’ transition function.
- Now, observe that a Turing machine designed to decide \( L_{\text{Computation-history}}(M, w) \) could do all its work by marking up symbols on the original input. That is, this machine would be an LBA.
- We know that \( A_{TM} \) is undecidable. Suppose \( E_{LBA} \) was decided by \( M_{E_{LBA}} \). Then, we could built a machine \( M_{A_{TM}} \) as follows:
  - On input \( < M, w > \), build a description of an LBA \( < M' > \) that decides the language \( L_{\text{Computation-history}}(M, w) \).
  - Provide \( < M' > \) as an input to \( M_{E_{LBA}} \) and either accept if it rejects or reject if it accepts.

This machine would decide \( A_{TM} \), so the assumption that \( M_{E_{LBA}} \) exists must be false and \( E_{LBA} \) must be undecidable.

- It should be clear that \( E_{LBA} \) is recognizable. We just run the machine on all possible inputs until one is accepted and then accept. We don’t even have to dovetail since we can stop the simulation on any input after executing \(|Q| \times (|w| + 1) \times |\Gamma|^{\vert w \vert} \) steps.
- If \( E_{LBA} \) were also recognizable, then both languages would be decidable. Therefore, we can conclude that \( E_{LBA} \) is not recognizable.

7. Given that \( E_{LBA} \) is unrecognizable, the language

\[ EQ_{LBA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are LBAs and } L(A) = L(B) \} \]

must also be unrecognizable since if it were recognizable, we could recognize \( E_{LBA} \) by asking if a given LBA was equivalent to a trivial LBA designed to accept no inputs.

8. Finally,

\[ ALL_{LBA} = \{ \langle M \rangle \mid M \text{ is a LBA and } L(A) = \Sigma^* \} \]

must be unrecognizable because we can equally easily built an LBA that decides

\[ L_{\text{Computation-history}}(M, w) \]

and the machine that decided this language would belong to \( ALL_{LBA} \) exactly when \( w \notin L(M) \).