CS 361 Meeting 28 — 5/4/20

Linear Bounded Automata

(Click for video)

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 quite powerful.

7. Many of the languages we have determined are decidable by Turing machines are clearly decidable by LBAs:

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

     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 δ 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.

   - $E_{DFA} = \{\langle M\rangle \mid M \text{ is a DFA and } \mathcal{L}(M) = \emptyset\}$

     When we originally considered this question, I suggested two approaches. One, was too exhaustively search all strings of length less than the pumping length for a string that the DFA accepted.
     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.
     The other approach is to search that set of states as a graph to see if there is any path from the start state to any final state. This can be implemented by an algorithm that simply places markers on the states it has found to be reachable from the start state. This can easily be done in place.

   - $ALL_{DFA} = \{\langle M\rangle \mid M \text{ is a DFA and } \mathcal{L}(A) = \Sigma^*\}$

     $\langle M\rangle \in ALL_{DFA}$ 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 E_{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}.

Click here to view the slides for this class

- $E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } \mathcal{L}(G) = \emptyset\}$

  The algorithm we (briefly) discussed that decides E_{CFG} (see the notes for lecture 23) 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.

8. If we think of computable functions instead of languages, many important algorithms can be implemented on LBAs:

   - Sorting
   - Dijkstra's algorithm
   - Matrix multiplication

A Decidable Question about LBAs

(Click for video)

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

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

   are decidable when applied to LBAs:

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

   item 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 (and in at least one interesting case, decidable).

2. Recall that a Turing machine configuration is a triple (x, q, 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 $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ and a string $w \in \Sigma^*$, we define the language of computation histories of M on w as
   >
   > $$\begin{aligned} L_{Computation-history}(M, w) = \\ \{w_0 w_1 ... w_n \mid \; & \text{each } w_i \text{ is a configuration for } M, \\ & w_0 \text{ is the initial configuration for } w, \\ & w_n \text{ is an accept configuration, and} \\ & \text{each } w_i \text{ yields } w_{i+1} \text{ according to } \delta \;\} \end{aligned}$$

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

   - If $w_0 w_1 ... w_n$ 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:

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

   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 $|Q| \times (|w| + 1) \times |\Gamma|^{|w|}$ steps and accepts only if the simulated machine reaches an accept state.

Some Undecidable Questions about LBAs

https://williams.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d6998f06-94b2-4c60-9a84-abac00221908

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

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

   is undecidable.

   - First, it should be clear that for any Turing machine, the language $L_{Computation-history}(M, w)$ is decidable.
     - Given a string h on its input tape, a Turing machine designed to decide $L_{Computation-history}(M, w)$ would need to verify that h is a string of the form $w_0 w_1 \ldots w_n$ where:
       (a) w_0 is an encoding of the configuration (q_0, ϵ, w).
       (b) w_n is a valid encoding where the Turing machine is in q_{accept}.
       (c) Each w_i yields w_{i+1} according to the Turing machines' transition function.
   - Now, observe that a Turing machine designed to decide $L_{Computation-history}(M, w)$ 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 $L_{Computation-history}(M, w)$.
     - Provide $< M' >$ as an input to $M_{E_{LBA}}$ and either accept if it rejects or reject if it accepts.

     This machine would decide $M_{A_{TM}}$, so the assumption that $M_{E_{LBA}}$ exists must be false and E_{LBA} must be undecidable.
   - It should be clear that $\overline{E_{LBA}}$ 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 $|Q| \times (|w| + 1) \times |\Gamma|^{|w|}$ steps.
   - If E_{LBA} were also recognizable, then both languages would be decidable. Therefore, we can conclude that E_{LBA} is not recognizable.

2. Given that E_{LBA} is unrecognizable, the language

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

   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.

3. Finally,

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

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

$$\overline{L_{Computation-history}(M,w)}$$

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

Resource Restrictions and Big-O

(Click for video)

1. Over the last several weeks, we have happily discussed Turing machine algorithms that required vast amounts of time and tape for relatively simple problems. For the remainder of the semester, we will examine the power of Turing machines when the amount of tape or the number of steps they can use while processing an input is limited.

2. Today, we talked about a model of computation in which one of the resources the machine could use, memory, was related to the size of its input:

   > **Definition:** A *linear bounded automaton* is a restricted type of Turing machine wherein the tape head is not permitted to move off the portion of the tape containing the input.

3. Suppose that we instead decided to place a restriction on the number of steps a TM could take to process its input:

   > **Definition:** A *linear-time bounded automaton* is a restricted type of Turing machine wherein the tape head is not permitted to move more than once for each symbol on the tape containing its input. (Assume that if the TM tries to take too many steps, the input is rejected by default.)

4. This restriction changes the beast quite dramatically.

   - Such machines cannot recognize even simple context-free languages like $w\$w^R$.
   - Such machines can clearly still recognize all regular languages.
   - In fact, they can only recognize regular languages!

5. Changing from $|w|$ steps to $k \times |w|$ steps does not help:

   - Still cannot recognize simple context-free languages like $w\$w^R$.

6. On the other hand, changing from $|w|$ steps to $|w|^2$ steps makes a big difference. With $|w|^2$ a TM can recognize many languages that are not context-free.

   - $\{w\#w \mid w \in \Sigma^*\}$
   - $\{n\#w\#w_1\#w_2\#...\#w_k \mid n, w, w_i \in \{0,1\}^*, n \leq k, \text{ and } w = w_n\}$
   - $\{a^n b^n c^n \mid n \geq 0\}$ — this language is not context-free.

7. This provides a handy refresher on and justification for big-O notation:

   > **Definition:** Let f and g be functions $f, g : \mathcal{N} \rightarrow \mathcal{R}^+$. We say that $f(n) = O(g(n))$ if for some positive integers c and n_0
   >
   > $$f(n) < cg(n)$$
   >
   > for all $n \geq n_0$. We say that $g(n)$ is an asymptotic upper bound for $f(n)$.

8. Basically, when we explore what can be computed by TMs where the amount of space they can use or the number of steps they take is limited to be less than some function of the size of the input, we only discuss the highest order term of the function that describes the amount of time/space used by an algorithm, ignoring both constant multipliers and lower-order terms

9. Based on this notion, we can define classes of languages recognized by TMs in numbers of steps bounded asymptotically bounded by any function:

   > **Definition:** Let $t : \mathcal{N} \rightarrow \mathcal{R}^+$ be a function. Define the **time complexity class,** $TIME(t(n))$ to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Why P?

(Click for video)

1. Consider how many steps are required to recognize the languages below on a multi-tape TM:

   - $\{w\#w \mid w \in \Sigma^*\}$ — $O(n)$
   - $\{n\#w\#w_1\#w_2\#...\#w_k \mid n, w, w_i \in \{0,1\}^*, n \leq k, \text{ and } w = w_n\}$ — $O(n)$
   - $\{w_0\#w_1\#w_2\#...\#w_k \mid \exists i, j \leq k \text{ with } w_i = w_k\}$ — $O(n \log n)$

2. Now do the same thing for a single-tape TM. In general, you should recognize that for every step made by a multi-tape TM, a single tape TM may have to make a complete pass over its tape whose length can be as large as the number of steps executed. As a result, the running time will be at most the square of that of a single tape machine.

3. A similar pattern emerges if you compare the expected running time of an algorithm when that algorithm is implemented on a conventional computer (using a standard programming language). If the running time on a conventional machine is bounded by a polynomial, then the same algorithm can be implemented on a Turing machine and its running time will just be bounded by a polynomial of a higher degree.

4. The key observation is that for all the variations in our computing model, any bound on one model that is a polynomial remains a polynomial on most other reasonable models.

5. This inspires:

   Definition: P is the class of language that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words

   $$P = \bigcup_k TIME(n^k)$$

6. Of course, the biggest variation in model we can make is to switch from a deterministic model to a non-deterministic model. For DFAs and TMs with no limits on the durations of their computations, we have shown nondeterministic modes and deterministic modes are equivalent. We might well wonder whether this is the case for TMs limited by some time bound.

7. As you should already know, it is not known whether this is the case.

8. To allow us to explore this question, we consider the execution time of a nondeterministic TM to be the length of the longest path in its computation tree and say

   Definition: $NTIME(t(n)) = \{L \mid L$ is a language decided by an $O(t(n))$ time nondeterministic Turing machine $\}$.

   Definition: NP is the class of language that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words

   $$NP = \bigcup_k NTIME(n^k)$$