CS 361 Meeting 20 — 4/15/20

Announcements

1. None.

Thinking Conditionally

(Click for video)

1. The initial machine we considered (the one from Sipser that recognized the language $w\#w$ over binary strings w), was not designed as a transducer.

2. Suppose, however, if we turn this into a form of transducer by removing the accept and reject states from the machine (we didn't even show the reject state in the machine's diagram) and instead directed transitions to these two states to distinct states in some larger TM.

3. While $w\$w$ is just one of our silly, theoretical languages, real programs often include statements of the form

```
if ( x = y ) {
                  . . .
} else {
         . . .
}
```

4. The Sipser $w\#w$ "transducer" is in some sense equivalent to such an if statement.

5. We can even consider more complex conditional decisions.

   - Remember all those weird languages with binary strings separated by #'s?
   - Let's think about how to construct a Turing machine to recognize the language:

     $$\{i\#x\#w_1\#w_2\#...\#w_k \mid i, x, w_i \in \{0,1\}^*, 0 < i \leq k, \text{ and } x = w_n\}$$

   - I claim we can convert such a machine into a transducer that can be used in a larger machine to perform the computation associated with a statement like "if (x = w[i]) ..."
   - The good news here is that instead of worrying about all the details, we will view the machine we have looked at in detail as enough to give us confidence that we can create sub-modules that
     - subtract 1 from a sub-string of the input interpreted as a binary number,
     - determine if two clearly delimited substrings are identical (as we did for $w\#w$).
   - The machine we would design would first turn the first two #'s into some other marker.
   - Next, it would repeatedly scan out to the second special marker, replace it with a plain old #, replace the next # to the right with the special marker
   - Then it would go back and replace n by $n - 1$ and repeat the marker moving trick over and over until n became 0.
   - After this was complete, the two substrings that should be compared would be marked with special markers.
   - It should be clear we could adapt Sipser's machine to check that these two substrings were identical.

6. Even though we haven't even precisely defined what a Turing machine is, the examples I have been presenting are designed to do more than familiarize you with the informal rules by which TM's function.

   In addition, I have been trying to help you understand how Turing machines can possibly be as powerful as real computers. In a real machine, memory is divided into units (registers and words of memory) and computations are divided into units (statements/instructions). By dividing our TMs tape into subsections with appropriate markers (think #'s) and building TM transducers that perform simple operations on information encoded on the tape, we can accomplish computations similar to "real" computers.

Click here to view the slides for this class

Getting Formal

(Click for video)

1. We can formalize our understanding Turing Machines, with a few exciting definitions:

   Definition: A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where

   Q is a finite set of states,

   Σ is a finite input alphabet (not containing the blank symbol),

   Γ is a finite tape alphabet which is a superset of Σ including the blank symbol,

   $\delta : Q \times \Gamma \to Q \times \Gamma \times \{Left, Right\}$ is the transition function,

   q_0 is the start state,

   q_{accept} is the accept state, and

   $q_{reject} \neq q_{accept}$ is the reject state.

2. Next, to capture the way we expect a Turing Machine to compute, we need a precise way of describing a snapshot of a point in some computation.

   Definition: A **configuration** of a Turing machine is a triple (u, q, v) where $q \in Q$ is the current state, uv is the contents of the non-blank portion of the tape with u being the portion to the left of the current head position and u being the portion from the symbol currently under the head to the end of the non-blank tape.

Yielding a Configuration

(Click for video)

1. Given one configuration of a deterministic Turing machine, the transition function δ determines what configuration the machine will move to next..

   Definition: We say the configuration (u, q, av) **yields** configuration (u', q', v') for $q, q' \in Q, a \in \Gamma$, and $u, v, u', v' \in \Gamma^*$ if for some b and $c \in \Gamma$:

   - $\delta(q, a) = (q', c, Left), u = u'b$, and $v' = bcv$, or
   - $\delta(q, a) = (q', c, Right)$, $u' = uc$ and $v' = v$, or
   - $\delta(q, a) = (q', c, Left), u = u' = epsilon$, and $v' = cv$, or
   - $\delta(q, a) = (q', c, Right)$, $u' = uc$, $v = \epsilon$, and $v' = \_$.

2. This definition of "yield" is admittedly complicated.

   - First, it helps to temporarily ignore the last two cases. They handle the special situations that crop up if the machine moves off either end of the used section of its tape.
   - Each of the first two rules describes the details of what must happen if the machine moves one position in the middle of its tape. The first is for left moves, the second for right.
   - Think of all of the primed and un-primed variables as patterns matching the contents of the tape. For example, for the second rule:
     - $u' = uc'$ is just a way of saying that the contents of the tape to the left of the tape head after a right move (u'), must be equivalent to the contents of the tape to the left of the tape head before the head moved (u) with the symbol written following the transition rules (c) appended to it.
     - The third sub-piece of the description of the configuration before the move is made (av) is a mathematical way of associating two names with values at the same time. It indicates that a is the name of the symbol under the tape head and the v is the name of the symbols to the right of the tape head.
     - av together represents the symbol under the tape head (a) and the symbols to the right of the tape head (v). So, $v' = v$ says that after a right move, the symbols after the tape head become the symbols under and to the right of the tape head.

Forms of Acceptance

(Click for video)

1. Given the definition of "yield", the definition of acceptance is not that different from the definitions of acceptance for other automata. We require a sequence showing the step-by-step progress of the machine from the start state to the accept state where each state yields the next.

   > **Definition:** A TM **accepts** a string $w \in \Sigma^*$ if there is a sequence of configurations that begins with (ϵ, q_0, w) and ends in (w', q_{accept}, w'') for some $w', w'' \in \Gamma^*$ where each configuration yields the following configuration in the series.

2. If Turing machines were like DFAs or PDAs, we would be done now. Turing machines, however, have a new option. A Turing machine can accept a string, reject a string, or just run forever without making a decision. So we need to distinguish two ways in which a language L can be "the language of a Turing machine".

   > **Definition:** A language L is **Turing-recognizable** if some Turing machine accepts w if and only if $w \in L$. We call these languages *recursively-enumerable.*

   > **Definition:** A language L is Turing-decidable if some Turing machine that halts on all inputs accepts w if and only if $w \in L$. We call these languages *recursive.*