6.840
Theory of Computation
2004/10/21

Dan Ports
drkp@mit.edu

10

Problem 3-1

Let L be the language of all Turing machine representations that contain no useless states. We will show that L is undecidable. We begin with a lemma.

Lemma 1. *Any Turing machine can be converted to a Turing machine that accepts the same language and either has no useless states, or the only useless state is the accept state.*

Proof. Suppose M is a Turing machine that accepts some language L_M. We construct a Turing machine M' that accepts L_M and satisfies the useless-state property described above. We begin by making M' a copy of M; clearly, it therefore accepts L_M.

We augment the input alphabet of M' with an additional symbol $\wr$. Let S be the set of states of M, except for the accept state q_{accept}. Put S in any order. We add transitions to M' such that if $\wr$ is the current tape symbol, the machine moves from the current state to the next state in the ordering of S (or back to the first state in the ordering if the current state was the last state in the ordering), without moving the tape head.

The result of this construction is a machine M' that has the same states as M, and accepts the same language L_M, since we did not change any of the transitions except for those involving our new symbol. Moreover, if the input string is simply the symbol $\wr$, then M' will loop through all states except for q_{accept}. This ensures that all states except q_{accept} are non-useless. This proves the lemma. □

We now prove by contradiction that L is undecidable. Suppose L is decidable, and R is a TM that decides L. We give a machine $\aleph$ that solves A_{TM}.

$\aleph$ = " On input $\langle M, w \rangle$,

1. Construct the Turing machine $\mathscr{M}$ that ignores its input and simulates M on w.
2. Construct the Turing machine $\mathfrak{M}$ that accepts the same language as $\mathscr{M}$ and either has no useless states, or the only useless state is the accept state, as described above.
3. Run R on $\mathfrak{M}$. If $\mathfrak{M}$ has any useless states, the useless state is q_{accept}, so M does not accept w. So *reject*. If $\mathfrak{M}$ has no useless states, M accepts w. So *accept*. "

$\aleph$ decides A_{TM}. But A_{TM} is undecidable. This is a contradiction $\leftrightarrow$. So L is undecidable.

6.840
Theory of Computation
2004/10/21

Dan Ports
drkp@mit.edu

Problem 3-2

Let L be any Turing-recognizable language. We show that L is mapping reducible to A_{TM}. Let M be a recognizer for L.

Define $f(w) = \langle M, w \rangle$. Then f maps L to A_{TM}: if w is in L, then M accepts w, and so $\langle M, w \rangle$ is in A_{TM}; conversely, if w is not in L, then M does not accept w, and so $\langle M, w \rangle$ is not in A_{TM}.

To see that f is a computable function, consider the following machine that generates f:

F = " On input w,

1. Output $\langle M, w \rangle$ "

6.840
Theory of Computation
2004/10/21

Dan Ports
drkp@mit.edu

10

Problem 3-3

Let $BB(k)$ be the busy beaver function. We show that $BB(k)$ is not computable. Suppose it is. Then there is a machine $\mathfrak{B}$ that computes $BB(k)$. Assume without loss of generality that $\mathfrak{B}$ produces its output in a unary representation (if it does not, we can create a new machine that simulates $\mathfrak{B}$ and then converts the output to unary).

Then we create the following machine:

$M =$ " On a blank tape,

1. Obtain own description $\langle M \rangle$ via the recursion theorem. Let x be the number of states in $\langle M \rangle$.
2. Compute $BB(x)$ by simulating $\mathfrak{B}$ on input x, and write the (unary) result to the tape.
3. Write an additional 1 to the tape, then halt. "

Note that M is a Turing machine with x states. It outputs $BB(x) + 1$ 1s to the tape, since the unary representation of $BB(x)$ is simply $BB(x)$ 1s. But $BB(x)$ is defined to be the maximum number of 1s producible by a x-state TM. This is a contradiction $\lightning$. So BB is uncomputable.

10

6.840

Theory of Computation

2004/10/21

Dan Ports

drkp@mit.edu

Problem 3-4

Let $AMBIG_{\mathsf{CFG}}$ be the ambiguity problem for CFGs. We show that $AMBIG_{\mathsf{CFG}}$ is undecidable by reduction from PCP. Let

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

be a PCP instance. Define G to be a CFG over the terminal symbols $\{t_i, b_i, a_i \ : \ 1 \leq 1 \leq k\}$ as follows

$$G = \begin{cases} S \to T|B \\ T \to t_1 T a_1 | \cdots | t_k T a_k | t_1 a_1 | \cdots | t_k a_k \\ B \to b_1 B a_1 | \cdots | b_k B a_k | b_1 a_1 | \cdots | b_k a_k \end{cases}$$

We show that G is ambiguous if and only if P is solvable.

Suppose P is solvable. Then there is a sequence of dominoes $i_1, i_2, \ldots, i_n$ such that

$$t_{i_1} t_{i_2} \cdots t_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}$$

Consider the string $t_{i_1} t_{i_2} \cdots t_{i_n} a_{i_n} a_{i_{n-1}} \cdots a_{i_1}$. This string can be derived in the CFG G by repeatedly applying the T production rule. However, since the string is equal to $b_{i_1} b_{i_2} \cdots b_{i_n} a_{i_n} a_{i_{n-1}} \cdots a_{i_1}$, it can also be generated by repeatedly applying the B production rule. This gives two different parse trees, so G is ambiguous. ✓

Next, suppose G is ambiguous. We show that P is solvable. Since G is ambiguous, there is some string s that has two different parse trees. Observe that s must end in some sequence of a symbols equal in length to the number of non-a symbols before it; write it as $a_{i_n} a_{i_{n-1}} \cdots a_{i_1}$.

Note that, if we consider the T production rule alone as a grammar, it is unambiguous, since for a given string, there is only one clause in the rule that can be used to parse it: the one corresponding to the a_i at the end of the string. The same is true of the B production rule. Because of this, one of the parse trees for s must use the T production rule and the other must use the B production rule. So, using the two parse trees, s can be written as either $t_{i_1} t_{i_2} \cdots t_{i_n} a_{i_n} a_{i_{n-1}} \cdots a_{i_1}$ or $b_{i_1} b_{i_2} \cdots b_{i_n} a_{i_n} a_{i_{n-1}} \cdots a_{i_1}$. This means that

$$t_{i_1} t_{i_2} \cdots t_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}$$

and so $i_1, i_2, \ldots, i_n$ is a solution to the PCP. ✓

We have shown that PCP is reducible to $AMBIG_{\mathsf{CFG}}$. So there can be no TM that decides $AMBIG_{\mathsf{CFG}}$, since such a machine would also be able to decide PCP, which is undecidable.

6.840

Theory of Computation

2004/10/21

(10)

Dan Ports

drkp@mit.edu

Problem 3-5

a) Let A_{2DFA} be the acceptance problem for two-headed finite automata. We show that A_{2DFA} is decidable.

Note that a 2DFA can be simulated by a Turing machine. We quadruple the size of the input alphabet by two mark bits, one representing the position of each head. Then the TM can scan back and forth to find the mark bits and identify the symbols under each head in the 2DFA. It can then move the mark bits to an adjacent symbol in order to simulate a transition.

In order to show that A_{2DFA} is recognizable, we also need to show that the number of configurations of a 2DFA is bounded.

Lemma 2. *A 2DFA with q states has $q(n+2)^2$ possible configurations on an input of length n.*

Proof. Since the input has length n, the tape of the 2DFA has length exactly $n+2$. Therefore, there are $n+2$ possible positions for each of the two heads, and q states. The state the 2DFA is in and the position of each head defines a configuration, since the tape's contents are static. □

This gives us a procedure for deciding A_{2DFA}. The following TM M decides A_{2DFA}:

M = " On input $\langle D, w \rangle$, where D is a 2DFA and w an input string,

1. Write w and the two delimiter characters to the tape, placing both mark bits at the leftmost symbol in w
2. Simulate D on w for at most $q(n+2)^2$ transitions, or until it halts
3. If D accepts, *accept*. If D rejects or does not halt, *reject*. "

b) Let E_{2DFA} be the emptiness problem for two-headed finite automata. We show that E_{2DFA} is undecidable. We do so by showing that for a TM M, we can create a 2DFA $D_{M,w}$ that verifies a TM computation history for M on input w and accepts if it is a valid accepting computation history.

The procedure for doing so is essentially identical to the procedure for doing so with a LBA (Sipser, proof of theorem 5.9), except that we use the second head instead of marking the tape in verifying the configuration transitions.

2DFA $D_{M,w}$ = " On input $C = \#C_1\#C_2\# \cdots \#C_l\#$,

1. Using the first head, verify that C_1 is the start configuration for M on w
2. Using the first head, verify that C_l is an accept configuration for M.
3. For each i from 1 to $l-1$, verify the transition from configuration i to $i+1$:
   1. Scan to the beginning of configuration i with the first head and the beginning of configuration $i+1$ with the second head.
   2. Scan over C_i and C_{i+1} with the two heads in lockstep, and verify that all tape symbols are identical except for those under or next to M's head in C_i.
   3. Verify that C_{i+1} is a valid configuration transition from C_i by checking that the state, head position, and symbol under the head were updated according to the transition function of M.
4. If all of these conditions are met, *accept*. Otherwise, *reject*. "

6.840
Theory of Computation
2004/10/21

10/10

Dan Ports
drkp@mit.edu

Problem 3-6

The following program, written in Scheme, produces itself as output.

```
((lambda (x) (list x (list (quote quote) x)))
 (quote (lambda (x) (list x (list (quote quote) x)))))
```

The output of this program produced by running it through the MIT Scheme interpreter is below. (It is also above!)

```
((lambda (x) (list x (list (quote quote) x)))
 (quote (lambda (x) (list x (list (quote quote) x)))))
```

6.840
Theory of Computation
2004/10/21

Dan Ports
drkp@mit.edu

Problem 3-7

On the universe of real numbers $\mathbb{R}$, a model for the relation ϕ_{lt} is

$$\phi_{\text{lt}}(x, y) = \text{TRUE if } x < y$$

Call this model $(\mathbb{R}, <)$.

Recall that ϕ_{eq} is the three conditions of an equivalence relation; on $\mathbb{R}$, the standard equality relation satisfies ϕ_{eq}. So the first condition holds. The second condition holds because $x = y$ implies $x \not< y$. If $x \neq y$, then either $x < y$ or $y < x$, so the third condition holds. The fourth condition is transitivity: if $x < y$ and $y < z$, then $x < z$; this also holds under $(\mathbb{R}, <)$. The final condition states that for every x there exists some $y < x$; this is true in $\mathbb{R}$. So the model $(\mathbb{R}, <)$ satisfies the statement ϕ_{lt}.