ARSDIGITA VNIVERSITY

Month 8: Theory of Computation

Problem Set 5 Solutions - Mike Allen

1. Name that Language

a. **DCFL**. $\{0^n1^m0^p1^q \mid n+m = p+q \text{ and } n,m,p,q > 0\}$
b. **DCFL**. $\{0^n1^m0^m1^n \mid n,m > 0\}$
c. **regular** . $\{0^n1^m0^p1^q \mid n,m,p,q > 0\}$
d. **CFL**. The set of strings over alphabet {0,1,2} with an equal number of 0s and 2s or an equal number of 0s and 1s.
e. **regular**. $\{0^m \mid m = 2k+1 \text{ where } k>0\}$
f. **regular**. The set of strings with 3n 0s and 4m 1s for m,n > 0.
g. **DCFL**. The set of strings with at least ten times as many 0s as 1s.
h. **regular**. The set of strings that are either odd length or contain 5 consecutive 1s.
i. **TM**. $\{0^m10^{m!} \mid m>0\}$
j. **DCFL**. The set of stings over alphabet {0,1,2} where the number of 1s equals the number of 2s and every 0 is followed immediately by at least one 1.

3. An Undecidability Problem

Prove that the problem of determining if the languages generated by two CFGs are equal is undecidable.

We show this by reducing the problem of determining whether a CFG accepts everything to the problem of determing if the languages generated by two CFGs are equal by taking our input CFG and comparing it to the CFG that generates everything. Since we know from class that the "everything" problem is undecidable, the "equal" probably must also be.

4. R.E. or not?

Determine for each of the following languages whether whether or not it is recursively enumerable and whether the complement is or is not.

a. **no yes** The language of all TMs that accept nothing.

   We need to try an infinite number of strings in a TM to determine that it accepts nothing, but we only need to find a single string that it accepts to show that it accepts something.

b. **no no** The language of all TMs that accept everything.

   We need to try an infinite number of string in a TM to determine whether it accepts everything, and we may or may not need to if it does not.

c. **no no** The language of all TMs that accept Regular languages.

We can enumerate all possible regular languages, but testing every regular language against every TM would take forever. If fact, even testing a TM against a single infinite regular language would take forever.

d. **no yes** The language of all PDAs that accept everything.

We would have to try every string before declaring that a PDA accepts everything, but since membership in a CFG (equivalent to a PDA) is decidable, we determine that a PDA does not accept anything once it rejects anything.

e. **yes no** The language of all CFGs that are ambiguous.

We can determine that a CFG is ambiguous by finding a single string which has an ambiguous derivation, but we cannot determine if a CFG is unambiguous unless we try everything string in it.

5. A Refutation of the Halting Problem?

Consider the language of all TMs that given no input eventually write a non-blank symbol on their tapes. Explain why this set is decidable. Why does this not conflict with the halting problem?

The state of a Turing machine is determined by the state of its controller (a DFA) and the state of the tape. Since the controller is finite, there are only a finite number of states possible before the TM is forced to either loop or write a symbol, so we simply run the Turing machine for that number of steps and then accept if it has written a symbol and reject otherwise.

This does not conflict with the halting problem because we are considering only a subset of Turing machines (those roughly equivalent to DFAs) and not really addressing the halting problem for Turing machines.

6. PCP for One-Character Strings

Prove that the Post Correspondence Problem is decidable for strings over the alphabet {0}.

We can simplify this problem by assigning each domino a value which is the number of 0s on the top row minus the number of zeros on the bottom row. Our goal, then, is to choose dominos whose values sum to zero. We consider a few cases:

1. If the set contains a "0" domino, that domino alone is a legal solution, so we accept.
2. If the set contains a "+m" and "-n" domino, we construct a solution from n "+m"s and m "+n"s and accept.
3. If the set contains only "+" dominos or "-" dominos we cannot add them to zero,

so we reject.

Since we can accept or reject based on a quick examination of the set of dominos, the problem is decidable.

9. Satisfiablity for DNF Formulas in P

Prove that the problem of determining whether there is a T/F assignment that makes a given DNF true can be solved in polynomial time. How is it possible that CNF is still NP-complete?

A DNF takes the following form

```
( .. ^ .. ^ ..) V (.. ^ .. ^ ..) V ...
```

Since the terms are OR'd together, if any of them are true, the entire expression will be true, so determining whether a DNF is satisfiable is the same as determining if any of the terms is independently satisfiable. A term is satisfiable if it does not contain any incompatible values (ie x_n and not x_n). So, we can check for satisfiability by a simple scan of the input which can easily be accomplished in polynomial time.

Even in light of this property of the DNF, the CNF can still be NP-complete as long as the reduction from CNF to DNF cannot be performed in polynomial time.

10. A Punchcard puzzle that is NP-complete.

(text 7.26) Prove that the Punchcard Puzzle is NP-complete.

We show that the problem is in NP by showing that it is verifiable in polynomial time. To do this, we simply stack the cards according to the answer presented in the certificate to determine if they cover all the holes. This can be accomplished easily in polynomial time.

To show that it is NP-complete, we reduce 3-SAT to it. We create cards $\{x_1,x_2,...\}$ for each variable in the 3-SAT formula and create a hole position in each column for each term of the formula. Then, we punch holes in the left column of the card in every position which corresponds to a term that does not contain that card's variable and in the right column for every term which does not contain that card's variable's complement. The Punchcard problem is only satisfiable if every hole can be covered by one of the card which implies that every term in 3-SAT problem is satisfiable.

11. PSPACE-hard implies NP-hard

(Text 8.6) Show that any PSPACE-hard language is also NP-hard.

First we must show that the language is not in NP. This is trivial since NP is a subset of PSPACE, and therefore, anything outside of PSPACE is also outside of NP.

Then we must show that any problem in NP can be reduced to any PSPACE-hard

language. This is also fairly simple, since any SAT problem can be reduced to a TQBF problem by simply appending "there exists x_n" to the front of the SAT expression for each variable x_n and then solving it using the TQBF algorithm. Then the TQBF problem can be reduced to any PSPACE-hard problem by the definition of PSPACE-hard because TQBF is PSPACE-complete. Thus, any PSPACE-hard problem is also NP-hard.

12. A TIC-TAC-TOE game in PSPACE

    Show that go-moku is in PSPACE.

    A quick examination of the rules of the game makes it obvious that you can solve GM by using the MiniMax search algorithm on the game tree. Since the game tree has depth n^2 and only one branch of computation must be stored at a time, the space necessary is $O(n^2)$ which is in PSPACE.

    NOTE: This is a lot easier than determining whether it is PSPACE-complete.

13. A Punchcard Puzzle which is PSPACE-complete

    Show that the two-person game variation Punchcard Puzzle is PSPACE-complete.

    First we must show that the game is in PSPACE, which is easy, since it can be solved with a MiniMax search of the game tree. Since only one branch of computation is stored at a time, the storage is polynomial and the game is in PSPACE.

    Then, we can show that it is PSPACE-complete by reducing the FORMULA-GAME to it. We begin with our formula and insert quantifiers with dummy variables so the quantifiers take on the repeating form (there-exists, for-all). Now, we create a card for each quantifier/variable and create a hole position in each column for each term of the formula. We punch holes in the left column of the card in every position which corresponds to a term that does not contain that card's variable and in the right column for every term which does not contain that card's variable's complement. Now, we play the Punchcard game with the cards by choosing which way each card is flipped, which corresponds to choosing the T/F value of each variable in the formula game.