B62005Y-02 理论计算机科学基础 2017 年 6 月 14 日

Solutions to Homework # 7

主讲教师：杨光 张远航 2015K8009929045

Sipser 8.1 *Proof.* ($\Rightarrow$) We can simulate a SPACE($f(n)$) single-tape TM with $f(n) \geq n$ on a SPACE($f(n)$) two-tape read-only TM. First, scan the read-only tape, copying its contents to the work tape. Then simulate the remaining computations, treating the read/write work tape as the input tape of a single-tape TM. We can copy contents of the counter, using only log space to keep track of our position in the input. We can write all the input on our work tape since $f(n) \geq n$.

($\Leftarrow$) We can simulate a SPACE($f(n)$) two-tape read-only TM with $f(n) \geq n$ on a SPACE($f(n)$) single-tape TM. We make sure not to write over the first n input symbols and use the rest of the tape as our work area. This only adds n extra space and this increases our space by at most a constant, so the space complexity remains the same. □

Sipser 8.4

- *Union*: Suppose $A, B \in$ PSPACE. Let M_1 and M_2 be $f_1(n) = O(n^k)$ and $f_2(n) = O(n^l)$ space deciders for the two languages. Running M_1 and M_2 respectively we get $f_1(n)+f_2(n)$ which takes polynomial space. (Alternatively, we can run them simultaneously, and we get $2 \cdot \max(f_1(n), f_2(n))$ which also takes polynomial space.)
- *Complementation*: The following Turing machine decides the complement of a language L, decided by a poly-space decider M: run M on input w. *Accept* if M rejects and *reject* otherwise. This machine is obviously a poly-space decider for $\overline{L}$.
- *Star*: Assume the input is w. A decider for the star language starts by calculating whether each substring of w is in L. Then it tries all possible ways to split w into substrings and see whether in any case all substrings are in L. Extra storage is polynomial. □

Sipser 8.6 *Proof.* Consider a language L which is PSPACE-hard. Then for any language $A \in$ PSPACE, A is polynomial time reducible to L. We know that NP$\subset$ PSPACE.

Therefore, for any language $A' \in \text{NP}$, A' is polynomial time reducible to L. Therefore L is NP-hard. This means that any PSPACE-hard language is also NP-hard. □

Sipser 8.16 *Proof.* First we show that *STRONGLY-CONNECTED* $\in$ NL. Consider the following machine which decides $\overline{\textit{STRONGLY-CONNECTED}}$.

$M =$ "On input $\langle G \rangle$:

1. Nondeterministically select two nodes a and b.
2. Run *PATH*(a, b). If it rejects, then the graph is not strongly connected, so *accept*. Otherwise, *reject*."

Since storing the node numbers a and b only takes log space, and *PATH* uses only log space, $\overline{\textit{STRONGLY-CONNECTED}} \in$ NL. Finally, since NL = co-NL, *STRONGLY-CONNECTED* $\in$ NL.

Next we show that every other language in L is log space reducible to *STRONGLY-CONNECTED* by reducing *PATH* to *STRONGLY-CONNECTED*. Consider the following reduction.

$R =$ "On input $\langle G, s, t \rangle$:

1. Copy all of G onto the output tape.
2. Additionally, for each node i in G:
3. Output an edge from i to s;
4. Output an edge from t to i."

If there is a path from s to t, the the constructed graph is indeed strongly connected, because every node can now get to every other node by going through this path. If there is not a path from s to t, then the constructed graph is not strongly connected because the only additional edges in the constructed graph go into s and out of t, so there can be no new ways of getting from s to t.

Finally, we must verify that the reduction can indeed be performed by a log space transducer. Indeed, though the output has size $O(n)$, essentially the only space necessary to perform the reduction is that used to keep track of the node number i in the "for loop" above. □

Sipser 8.31 *Proof.* First we must show that the game is in PSPACE, which is easy, since it can be solved with a mini-max 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 *FORMULA-GAME* to it. We begin with our formula and insert quantifiers with "dummy" variables so the quantifiers take on the repeating form $(\exists, \forall)$. Now, we create a card for each quantifier and 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 *PUZZLE* 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. □

Sipser 8.33 *Proof.* Any string consisting of parentheses would be a properly nested one if and only if the number of left and right parentheses are equal and for any prefix of it the the number of left parentheses is no less than the right parentheses. So to decide whether a string w is properly nested, we can simply scan the string from left to right to maintain the number of occurrences of left and right parentheses and check whether they have violated the property. The number of occurrences is in $O(|w|)$ so we need only logarithmic spaces, which implies that A is in L. □