CPSC 511 — Fall 2014
Solutions for Question #3 on Midterm Test

This question concerned ***log-space mapping reductions***.

(a) You were first asked to define a ***log-space mapping reduction.***

Definition: If $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma^\star$, for two alphabets Σ_1 and Σ_2, then a ***log-space mapping reduction*** from L_1 to L_2 is a ***total*** function $f : \Sigma_1^\star$ to $\Sigma_2^\star$ that satisfies the following.

1. For every string $\omega \in L_1$, $f(\omega) \in L_2$.
2. For every string $\omega \in \Sigma_1^\star$ such that $\omega \notin L_1$, $f(\omega) \notin L_2$.
3. The function f is computable by a deterministic Turing machine using space in $O(\log n)$, when given an input string with length n, in the worst case.

(b) You were next asked to sketch a proof that log-space mapping reductions are ***transitive:*** That is, if $L_1 \subseteq \Sigma_1^\star$, $L_2 \subseteq \Sigma_2^\star$ and $L_3 \subseteq \Sigma_3^\star$, then **if**

- L_1 is log-space mapping reducible to L_2, and
- L_2 is log-space mapping-reducible to L_3,

then L_1 is log-space mapping reducible to L_3 as well.

Sketch of Proof: Suppose that there is a log-space mapping reduction $f_1 : \Sigma_1^\star \to \Sigma_2^\star$ from L_1 to L_2 as well as a log-space mapping reduction $f_2 : \Sigma_2^\star \to \Sigma_3^\star$ from L_2 to L_3.

Consider the ***composition*** of these functions

$$f = f_2 \circ f_1 : \Sigma_1^\star \to \Sigma_3^\star.$$

Notice first that if $\omega \in L_1$ then

$$\begin{aligned}
\omega \in L_1 &\Rightarrow f_1(\omega) \in L_2 && \text{(since } f_1 \text{ is a log-space mapping reduction from } L_1 \text{ to } L_2\text{)} \\
&\Rightarrow f_2(f_1(\omega)) \in L_3 && \text{(since } f_2 \text{ is a log-space-mapping reduction from } L_2 \text{ to } L_3\text{)} \\
&\Rightarrow f(\omega) \in L_3 && \text{(since } f = f_2 \circ f_1\text{).}
\end{aligned}$$

Notice next that if $\omega \in \Sigma_1^\star$ but $\omega_1 \notin L$ then $f_1(\omega) \in \Sigma_2^\star$ and $f(\omega) = f_2(f_1(\omega)) \in \Sigma_3^\star$. In this case

$$\begin{aligned}
\omega \notin L_1 &\Rightarrow f_1(\omega) \notin L_2 && \text{(since } f_1 \text{ is a log-space mapping reduction from } L_1 \text{ to } L_2\text{)} \\
&\Rightarrow f_2(f_1(\omega)) \notin L_3 && \text{(since } f_2 \text{ is a log-space-mapping reduction from } L_2 \text{ to } L_3\text{)} \\
&\Rightarrow f(\omega) \notin L_3 && \text{(since } f = f_2 \circ f_1\text{).}
\end{aligned}$$

Thus $f : \Sigma_1^\star \to \Sigma_3^\star$ is a total function such that $\omega \in L_1$ if and only if $f(\omega) \in L_3$ for every string $\omega \in \Sigma_1^\star$. It remains only to prove that f is computable by a log-space deterministic Turing machine in order show that f Is a log-space mapping reduction from L_1 to L_3.

Now, since $_1$ and f_2 are log-space mapping reductions from L_1 to L_2 and from L_2 to L_3, respectively,

- there exists a deterministic Turing machine

  $$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{0,1}, q_{\text{accept},1}, q_{\text{reject},1})$$

  that computes f_1 using space in $O(\log n)$ when executed on an input string with length n, in the worst case, and
- there exists a deterministic Turing machine

  $$M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{0,2}, q_{\text{accept},2}, q_{\text{reject},2})$$

  that computes f_2 using space in $O(\log n)$ when executed on an input string with length n, in the worst case, as well.

Unfortunately, a deterministic Turing machine that implements the algorithm

On input $\omega \in \Sigma_1^\star$

1. Run M_1 on the input ω to compute $f_1(\omega)$.
2. Run M_2 on the input $f_1(\omega)$ to produce $f(\omega) = f_2(f_1(\omega))$ as output.

is ***not*** generally a deterministic Turing machine that computes the function f using only a logarithmic amount of work space!

The problem here is that the intermediate value $f_1(\omega)$ is a string whose length is generally *polynomial* in the length of ω, but generally ***not*** logarithmic in the length of ω – it cannot be computed and stored for reuse (as an input string) at a later step!

Something more complicated — but also more space-efficient — is needed instead. One might think of this as a hand-shaking protocol: M_2 drives the computation because this machine produces the required output. Whenever it is necessary for M_2 to read its input tape M_1 is asked to produce the required symbol from $f_1(\omega)$ — storing, and reporting, only the single symbol from Σ, that is currently needed by M_2.

Consider, now, a multi-tape deterministic Turing machine M as follows:

- The input alphabet M is the same as for M_1 — the alphabet, Σ_1, used to define strings in the domain of the function f_1.
- The tape alphabet Σ of M includes

  $$\Sigma_1 \cup \Sigma_2 \cup \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

  so that it includes the tapes alphabets for M_1 and M_2, along with symbols needed to represent the decimal representations of nonnegative integers.
- M is a multi-tape Turing machine whose tapes include
  - the input tape for M_1, which is also the input tape for M;
  - the work tapes for M_1;
  - an address tape used for communication between M_1 and M_2 along with one or two extra "scratch" tapes used to support this;
  - the work tapes for M_2;
  - the output tape for M_2 — which is also the output tape for M.

  Note that these tapes *do not* include either M_1's output tape or M_2's input tape — the string $f_1(\omega)$ is never stored during the computation being described.
- In a brief *initialization* phase, the symbol 1 should be written onto the address tape, to indicate that M_2 would begin by reading the string in position 1 on its input tape. Control should be passed to M_1 — but M's finite control should also be used to remember the current state of M_2 (initially its start state).
- Control should repeatedly pass between M_1 and M_2 as follows.

  M_1 should begin its computation, keeping track of the number of symbols of output that it would generate but not writing any, as long as this number of symbols would be less than the number on the address tape. When the number of output symbols would be equal to this value M_1 writes the symbol requested by M_2 onto the address tape, to the right of the integer that was written by M_2, and cleans up its tapes so that it will be able to begin its computation on input ω if needed again.

  M_2 reads the symbol M_1 has written onto the address tape — erasing this symbol in the process. Using its finite control and information on its tapes, it makes its next move — decrementing the integer on the address tape if this value is greater than one and the move would normally move its input tape head *left*, incrementing this value if the input tape head would normally move *right* and a blank was not just printed by M_1 (so that the input tape head is not already to the right of the input) — and leaving the value unchanged, otherwise.

  This process continues until M_2 enters a halting state, at which point M halts as well.

- Since M_1 halts when executed on input ω every part of this computation in which M_1's computation is being simulated halts too.

  The information provided to M_2 using the address tape is the same as what would be seen if $f_1(\omega)$ was stored on its input tape, so the computation of M_2 halts as well, with $f_2(f_1(\omega)) = f(\omega)$ printed on M's output tape. Thus M computes the function f, as required.

- Since M_1 uses space in $O(\log |\omega|)$ it uses time that is at most polynomial in $|\omega|$, so that the length of the string $f_1(\omega)$ is at most polynomial in $|\omega|$ as well.

  Thus $\log |f_1(\omega)| \in O(\log |\omega|)$, so that the space used by M_2 is in $O(\log |\omega|)$ too. That is, the space used on M_1's and M_2's work tapes is (in total) in $O(\log |\omega|)$.

  It remains only to notice that the address tape is used to store the decimal representation of an integer between 0 and $|f_1(\omega)|+1$, so that the length of the non-blank part of this tape is in $O(\log |\omega|)$ too. The "scratch" tapes used for address calculations store decimal representations of numbers with at most the same length.

  Thus the total space used by M is in $O(\log |\omega|)$ in the worst case — as needed to establish that f is a log-space mapping reduction from L_1 to L_2, as required.

Note: Once again this answer is longer than was was needed to receive full marks for this question on the midterm!

However, in order to get full marks for part (b) it was necessary to clearly identify the composition of the log space mapping reductions from L_1 to L_2 and from L_2 to L_3 as a log-space mapping reduction from L_1 to L_3 — and to explain how this function can be computed without writing a large intermediate value on a work tape, so that this function can also be computed using logarithmic work space.