Assignment #3

CAS 705

Computability and Complexity

Jason Jaskolka
0546444
Dr. Michael Soltys
November 10, 2009

Question 1

This exercise is related to the Immerman-Szelepcsényi theorem.

Suppose that the membership in some language L can be determined by a nondeterministic TM M with time complexity $t_M(n)$ and space complexity $s_M(n)$.

Furthermore, suppose that the number of strings in L of a given length is given by $f : \mathbb{N} \rightarrow \mathbb{N}$, that requires time $t_f(n)$ and space $s_f(n)$.

Under these assumptions give a nondeterministic TM for $\bar{L}$, and determine upper bounds for the time and space complexity for such a TM.

We want to build a nondeterministic TM, N that will accept $\bar{L}$. Suppose we can encode configurations of M (the TM that decides L) over a finite alphabet, denoted Σ, such that each input string of length n can be represented as a string in $\Sigma^{s_M(n)}$. Let $|\Sigma| = k$.

Assume that when M wishes to accept, its work-tape is erased and enters a unique state q_{accept}. Due to this assumption, we can say that there is a unique accepting configuration $c_{accept} \in \Sigma^{s_M(n)}$ for inputs of length n. We will denote $c_{init} \in \Sigma^{s_M(n)}$ as the initial configuration on an input z, where $|z| = n$.

We know that M has at most $k^{s_M(n)}$ configurations for z which means that if z is accepted by M then there is an accepting path in the computation graph of length at most $k^{s_M(n)}$. Let us define C_i as the set of configurations in $\Sigma^{s_M(n)}$ that are reachable from c_{init} in at most i steps. Thus, we have that $f(n) = |C_n|$. So, we have that $C_0 = \{c_{init}\}$ and we know that M accepts if and only if $c_{accept} \in C_{k^{s_M(n)}}$.

How does the machine N decide $\bar{L}$? Well, it begins by using the inductive counting technique to compute f. We know that $f(0) = |C_0| = 1$. Suppose we have computed $f(i)$ and we have written the result on the work-tape of N. We then compute $f(i+1)$ by writing each $x \in \Sigma^{s_M(n)}$ one-by-one. For each $x \in \Sigma^{s_M(n)}$, we check if $x \in C_{i+1}$ by guessing the computation path of length i. If we succeed, then we increase a *counter* by 1. If any path contains c_{accept} we reject. If the *counter* reached $f(i)$ but we have not yet reached c_{accept}, then we accept at it is not reachable. It at the end we have *counter* $< f(i)$, then we reject.

Once we have computed $f(i+1)$, we need to check if $c_{accept} \notin C_{k^{s_M(n)}}$.We can do this by nondeterministically guessing the $f(s_M(n))$ elements of $C_{s_M(n)}$, while verifying that each guess is in $C_{s_M(n)}$ by guess ing the path of computation. Finally, we check that each guessed element is not equal to c_{accept}.

Thus, we have provided a a nondeterministic TM for $\bar{L}$. We can see that this machine can be programmed to work in space $O(s_M(n))$ since we know that $\mathsf{NSPACE}(f(n)) = \mathsf{co\text{-}NSPACE}(f(n))$ by the Immerman-Szelepcsényi theorem for $f(n) \geq \log n$. We can also see that this machine can be programmed to work in time $O(t_M(n) \times t_f(n))$.

Question 2

Suppose that the language L in the previous question is context-sensitive. Let $T_{n,i} := \{x : |x| \leq n, S \overset{i}{\Rightarrow} x\}$, so that for some m we will have $T_{n,m} = T_{n,m+1}$. Let $g(n) := |T_{n,m}|$ for this m.

Show that g can be computed in nondeterministic linear space.

Conclude that if L is context-sensitive, so is $\bar{L}$.

We assume that the language L is context-sensitive meaning that L is generated by a grammar G, with variables denoted by V and terminal alphabet denoted by Σ, where all the rules are of the form $\alpha \to \beta$, with $|\alpha| \leq |\beta|$. Recall that a *sentential form* is the start symbol S of a grammar or any string in $(V \cup \Sigma)^*$ that can be derived from S [2].

We know that we have an m such that $T_{n,m} = T_{n,m+1}$. This means that $T_{n,m}$ will be the set of all the strings with length at most n that can be derived from S. Thus, we have $k = g(n)$ such that k is the number of strings of length at most n that can be derived from S. Knowing this, we can decide if a given string z is not in the language by the following algorithm which follows from the proof of the Immerman-Szelepcsényi theorem:

For each $z \in (V \cup \Sigma)^{\leq n}$, we guess a derivation (which can be done with linear space in the length of the string if it is done step-by-step). In the event of a success, we decrease k by 1. If we end up guessing a derivation for z, then we reject. If at the end we have $k > 0$ then we reject since this means that we missed some string which had a derivation, which may have been z. It at the end we have $k = 0$, then we accept.

Now we need to show how to compute $g(n)$. We have that $T_{n,0} = \{S\}$ so, $g(0) = 1$. We can now use the inductive counting method to compute $g(i + 1)$ from $g(i)$ using the following algorithm:

For each $z \in (V \cup \Sigma)^{\leq n}$, we check if $z \in T_{n,i+1}$ by going through all of the strings derived from S of length at most i, denoted by $y \in (V \cup \Sigma)^{\leq n}$. We check if $y \in T_{n,i}$. If we find that $y \in T_{n,i}$ then by implication we have that $z \in T_{n,i+1}$ and since we know $g(i)$ then we will be able to determine if we have missed any y. We can stop when we find an i such that $T_{n,i} = T_{n,i+1}$.

The above algorithm runs in nondeterministic linear space. This is because we only consider derivations of strings that are not longer than the number of sentential forms of length n or less. Since the number of sentential forms of length n or less can be encoded in $O(n)$ many-bits, we have that we compute $g(n)$ in nondeterministic linear space.

We have shown that there exists an algorithm for deciding the language $\bar{L}$. The existence of such an algorithm follows from the Immerman-Szelepcsényi theorem, since $\mathsf{CSL} = \mathsf{LBA} = \mathsf{NSPACE}(n) = \text{co-}\mathsf{NSPACE}(n)$. Therefore, if L is context-sensitive, so is $\bar{L}$.

Question 3

Exercise 3.14. Show that $\text{value}(p_\alpha) > 0$ **iff** α **is true.**

We will show that $\text{value}(p_\alpha) > 0$ iff α is true by structural induction on α.

Case: $\alpha = x$ then $p_\alpha = x$, therefore

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & x = \mathtt{T} \\ \iff & p_\alpha = 1 \\ \iff & p_\alpha > 0 \end{aligned}$$

Case: $\alpha = \neg x$ then $p_\alpha = (1 - x)$, therefore

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & x = \mathtt{F} \\ \iff & p_\alpha = (1 - 0) \\ \iff & p_\alpha > 0 \end{aligned}$$

Case: $\alpha = \alpha_1 \wedge \alpha_2$ then $p_\alpha = p_{\alpha_1} \cdot p_{\alpha_2}$ and it follows by induction that

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & \alpha_1 \text{ is true and } \alpha_2 \text{ is true} \\ \iff & p_{\alpha_1} > 0 \text{ and } p_{\alpha_2} > 0 \\ \iff & p_\alpha > 0 \end{aligned}$$

Case: $\alpha = \alpha_1 \vee \alpha_2$ then $p_\alpha = p_{\alpha_1} + p_{\alpha_2}$ and it follows by induction that

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & \alpha_1 \text{ is true or } \alpha_2 \text{ is true} \\ \iff & p_{\alpha_1} > 0 \text{ or } p_{\alpha_2} > 0 \\ \iff & p_\alpha > 0 \end{aligned}$$

Case: $\alpha = \forall x \alpha_1(x)$ then $p_\alpha = \prod_{x \in \{0,1\}} p_{\alpha_1}$ and it follows by induction that

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & \alpha_1(x/\mathtt{T}) \text{ is true and } \alpha_1(x/\mathtt{F}) \text{ is true} \\ \iff & p_{\alpha_1}(1) > 0 \text{ and } p_{\alpha_1}(0) > 0 \\ \iff & p_\alpha > 0 \end{aligned}$$

Case: $\alpha = \exists x \alpha_1(x)$ then $p_\alpha = \sum_{x \in \{0,1\}} p_{\alpha_1}$ and it follows by induction that

$$\begin{aligned} & \alpha \text{ is true} \\ \iff & \alpha_1(x/\mathtt{T}) \text{ is true or } \alpha_1(x/\mathtt{F}) \text{ is true} \\ \iff & p_{\alpha_1}(1) > 0 \text{ or } p_{\alpha_1}(0) > 0 \\ \iff & p_\alpha > 0 \end{aligned}$$

Therefore, value$(p_\alpha) > 0$ iff α is true.

Exercise 3.15. Show that value$(p_\alpha) < 2^{2^{|p_\alpha|}}$**.**

We will show that value$(p_\alpha) < 2^{2^{|p_\alpha|}}$ by structural induction on α.

Case: $\alpha = x$ then $|p_\alpha| = 1$, so it holds that

$$\text{value}(p_\alpha) \leq 1 < 2^{2^1}$$

Case: $\alpha = \neg x$ then $|p_\alpha| = 1$, so it holds that

$$\text{value}(p_\alpha) \leq 1 < 2^{2^1}$$

Case: $\alpha = \alpha_1 \wedge \alpha_2$ then it follows by induction that $|p_{\alpha_1}| < 2^{2^m}$ and $|p_{\alpha_2}| < 2^{2^n}$ where $m + n \leq |p_\alpha|$, so value(p_α) can be at most

$$\text{value}(p_\alpha) \leq 1 < 2^{2^m} \cdot 2^{2^n} = 2^{2^{m+n}} \leq 2^{2^{|p_\alpha|}}$$

Case: $\alpha = \alpha_1 \vee \alpha_2$ then it follows by induction that $|p_{\alpha_1}| < 2^{2^m}$ and $|p_{\alpha_2}| < 2^{2^n}$ where $m + n \leq |p_\alpha|$, so value(p_α) can be at most

$$\text{value}(p_\alpha) \leq 1 < 2^{2^m} + 2^{2^n} \leq 2^{2^{m+n}} \leq 2^{2^{|p_\alpha|}}$$

Case: $\alpha = \forall x \alpha_1(x)$ then it follows by induction that $|p_{\alpha_1}| < 2^{2^m}$ where $m < |p_\alpha|$, so value(p_α) can be at most

$$\text{value}(p_\alpha) \leq p_\alpha = \prod_{x \in \{0,1\}} p_{\alpha_1} \leq 2^{2^m} \cdot 2^{2^m} \leq 2^{2 \cdot 2^m} \leq 2^{2^{m+1}} \leq 2^{2^{|p_\alpha|}}$$

Case: $\alpha = \forall x \alpha_1(x)$ then it follows by induction that $|p_{\alpha_1}| < 2^{2^m}$ where $m < |p_\alpha|$, so value(p_α) can be at most

$$\text{value}(p_\alpha) \leq p_\alpha = \sum_{x \in \{0,1\}} p_{\alpha_1} \leq 2^{2^m} + 2^{2^m} \leq 2^{2^{m+1}} \leq 2^{2^{|p_\alpha|}}$$

Therefore, value$(p_\alpha) < 2^{2^{|p_\alpha|}}$.

Exercise 3.16. Show that for all a, such that $0 < a < 2^{2^n}$ there exists a prime number $k \in [2^n, 2^{3n}]$ such that $a \not\equiv_k 0$.

To show that for all a, such that $0 < a < 2^{2^n}$ there exists a prime number $k \in [2^n, 2^{3n}]$ such that $a \not\equiv_k 0$, we will use the following theorems:

Theorem (Prime Number Theorem [4]). *For every m, there is at least $\sqrt{m}$ prime numbers $\leq m$.*

Theorem (Chinese Remainder Theorem [4]). *Given two sets of numbers of equal size, $r_1, r_2, ..., r_n$ and $m_1, m_2, ..., m_n$ such that*

$$0 \leq r_i < m_i \qquad 0 \leq i \leq n$$

and $\gcd(m_i, m_j) = 1$ for $i \neq j$, then there exists an r such that $r \equiv_{m_i} r_i$ for $0 \leq i \leq n$.

Let $n = |p_\alpha|$ and let $a = \text{value}(p_\alpha)$. Using the Prime Number Theorem we have that the number of prime numbers between 2^n and 2^{3n} is at least 2^n which are $k \in [2^n, 2^{3n}]$. By exercise 3.15, we have that $a < 2^{2^n}$ and since the product of these primes is greater than 2^{2^n}, by the Chinese Remainder Theorem, we have that at least one of these primes which we will call does not divide n. Therefore, for all a, such that $0 < a < 2^{2^n}$ there exists a prime number $k \in [2^n, 2^{3n}]$ such that $a \not\equiv_k 0$.

References

1. Kozen, Dexter. *Theory of Computation*. Springer, 2006.

2. Matuszek, David. *CSC 4170-50: Theory of Computation: Context-Free Grammars*. University of Pennsylvania. Available: `http://www.seas.upenn.edu/~cit596/notes/dave/cfg7.html`. Last Modified: Feb. 26, 1996.

3. Procter, Rob. *Computational Complexity Supplementary Note 1: The Immerman-Szelepcsényi Theorem*. University of Edinburgh.
Available: `http://www.dcs.ed.ac.uk/teaching/cs4/www/cc/immerman-szelepcsenyi.ps`. 1988-1989.

4. Soltys, Michael. *An Introduction to Computational Complexity*. Jagiellonian University Press. 2009.