Homework 5

SJSU Students

May 18, 2006

Exercise 5.4

If $A \leq_m B$ and B is a regular language, does that imply that A is a regular language? Why or why not?

No, it does not. For example, $\{a^n b^n | n >= 0\} \leq m\{a\}$. The reduction first tests whether its input is a member of $\{a^n b^n | n >= 0\}$. This can easily be done by a Turing computable function. If it is of this form, it outputs the string a, and if not, it outputs the string b.

Exercise 5.13

Let $U = \{\langle M, q\rangle \mid \texttt{q is a useless state in TM } M\}$
Suppose U is decidable and TM T decides it.
Then construct TM Z that uses T to decide E_{TM}
where $E_{TM} = \{\langle M\rangle \,| M \texttt{ is a TM and } L(M) = 0\}$
$Z =$ " $\texttt{On input }\langle M\rangle :$
1. Run TM T on input $\langle M, q_{accept}\rangle$ where q_{accept} is the accept state of M
2. If T accepts, reject. If T rejects, accept.

Exercise 5.21

AMBIG$_{CFG}$=$\{\langle$G$\rangle|$G is an ambiguous CFG$\}$
Assume AMBIG$_{CFG}$ is decidable then there exists some TM to decide it.
Define TM M$_{PCP}$ to do the following on input $\langle$P$\rangle$ with P=$\{$[t$_1$/b$_1$]...[t$_n$/b$_n$]$\}$ a set of dominoes with t$_1$...t$_n$, b$_1$...b$_n$ $\in \Sigma$.
1. Construct a CFG G = (V,Γ, R, S) with
V=$\{$S, T, B$\}$

$\Gamma=\Sigma\bigcup\{a_1,...,a_n\}$

Rules:

$S \to T|B$

$T \to t_1Ta_1|...|t_nTa_n|t_1a_1|...|t_na_n$

$B \to b_1Ba_1|...|b_nBa_n|b_1a_1|...|b_na_n$

2. Run M on input G, accept if M accepts, reject if M reject

a decider for PCP. But we know PCP is undecidable.

Need to show P$\in$PCP $\Leftrightarrow$ G$\in$AMBIG$_{CFG}$

"$\Rightarrow$"' Assume P has a match: $t_{i1}t_{i2}...t_{ik} = b_{i1}b_{i2}...b_{ik}$. Then the string $t_{i1}t_{i2}...t_{ik}a_{ik}a_{ik-1}...a_{i1}$=$b_{i1}b_{i2}...b_{ik}a_{ik}a_{ik-1}...a_{i1}$ has two diffirent leftmost derivation, one from T and one from B. "$\Leftarrow$"' Assume G is ambiguous, then some string w is at least two different leftmost derivations.

$\to$ All string generated by G have the from $w_1a_{ik}a_{ik-1}....a_{i1}$ where $w_1 \in \Sigma^*$.

$\to$ Except from the application of the rules containing the start variable on the left side, all other steps in the derivation are uniquely determined by the sequence $a_{ik}a_{ik-1}...a_{i1}$

$\to$Therefore w_1 has at most 2 leftmost derivations:

1. $S \Rightarrow T \Rightarrow t_{i1}Ta_{i1} \Rightarrow... \Rightarrow t_{i1}...t_{ik}a_{ik}a_{ik-1}...a_{i1}$
2. $S \Rightarrow B \Rightarrow b_{i1}Ta_{i1} \Rightarrow... \Rightarrow b_{i1}...b_{ik}a_{ik}a_{ik-1}...a_{i1}$

$\to$ Since $t_{i1}t_{i2}...t_{ik}a_{ik}a_{ik-1}...a_{i1}$=$b_{i1}b_{i2}...b_{ik}a_{ik}a_{ik-1}...a_{i1}$ it follows $t_{i1}...t_{ik}$=$b_{i1}...b_{ik}$ and P has a match.

Exercise 5.34

Consider the problem of determining whether a PDA accepts some string of the form $\{ww|w \in \{0,1\}^*\}$. Use the computation history method to show this problem is undecidable.

Proof. Suppose it was decidable given $\langle M\rangle$ whether $L(M)$ contains such a string. Let R be a decision for this, we will show how R could be used to get a decision procedure for A_{TM}. Give an input $\langle M, x\rangle$ for A_{TM}, consider the following language $L := \{ww|\ w$ an appropriate computation history$\}$. Here we require w be in the format $\#C_0\#C_1^R\#C_2\#c_3^R\#\cdots\#C_{2m}\#C_{2m+1}^R\#$. Given $\langle M, x\rangle$ we could build a PDA $\langle P\rangle$ which recognizes L. P checks C_0 is a start configuration using its hard-coded value x in its states. P pushes all of w onto the stack, nondeterministically guessing w endpoint. It then pops the characters of w off one by one as it reads the second copy. While doing this it can check if a given C_i is followed by the appropriate next configuration. Finally, it checks the last configuration is accepting. So given $\langle M, x\rangle$, we can make a decision procedure for A_{TM} by next building $\langle P \rangle$, then running R on $\langle P \rangle$, if R accepts, our procedure accepts; if R rejects our procedure rejects.

Exercise 6.13

For each $m > 1$ let $\mathcal{Z}_m = \{0, 1, 2, \ldots, m-1\}$, and let $\mathcal{F}_m = (\mathcal{Z}_m, +, \times)$ be the model whose universe is $\mathcal{Z}_m$ and that has relations corresponding to the $+$ and $\times$ relations computed modulo m. Show that for each m the $Th(\mathcal{F}_m)$ is decidable.

Proof. The proof is by induction of the complexity of the sentence ϕ we are trying to check is in $Th(\mathcal{F}_m)$. To keep things general, we will give an algorithm which works on a formula ϕ together with an assignment ν of the variables to elements of $\mathcal{Z}_m$. In the case, where ϕ is a sentence there will be no unbound variables and so ν can be empty. In the base, we have an atomic formula, which consists of an equation $t(x_1, \ldots, x_n) = s(x_1, \ldots, x_n)$ where t and s are terms over $+$ and $\times$. A TM can look at the codes for these two terms and substitute in the values of the variables assignment ν and then compute the values of the terms. This is possible since $+$ and $\times$ are computable functions. If the values of t and s are the same the machine would *accept*; otherwise, it would *reject*. Checking if $\not\phi$, $\phi \wedge \psi$, $\phi \vee \psi$ are true based on the the values obtained for ϕ and ψ is easily Turing computable. To check the truth of $\exists y \phi(y, \vec{x})$, the TM could check each variable assignment $0, 1, \ldots m-1$ for y to see if any make $\phi(y, \vec{x})$ true, if so *accept*, otherwise *reject*. To check the truth of $\forall y \phi(y, \vec{x})$, the TM could check each variable assignment $0, 1, \ldots m-1$ for y to see if that all of them make $\phi(y, \vec{x})$ true, if so *accept*, otherwise *reject*. This completes the induction.

Exercise 6.22

Show that the function $K(x)$ is not a computable function.

Proof. Suppose $K(x)$ is computable by some Turing Machine. Then the function $f(x)$ which computes:

$$\min\{y | K(y) > 2^x\}$$

would also be Turing computable. Now consider $K(f(x))$, by the definition of $f(x)$ it should by of size $> 2^x$. On the hand the description $\langle f, x \rangle$ has length $|\langle f \rangle| + |x|$, as x grows this value will be strictly less than 2^x giving a contradiction.