CS 154 Section 3—Homework #4

SJSU Students

May 8, 2006

Exercise 3.9

Let a k-PDA be a pushdown automata that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs

a. Show that 2-PDAs are more powerful than 1-PDAs.

b. Show that 3-PDAs are not more powerful than 2-PDAs. (Hint: Simulate a Turing machine tape with two stacks.)

(a.) By Example 2.36, no PDA recognizes

$$B = \{a^n b^n c^n | n >= 0\}.$$

The following high level description of a 2-PDA recognizes B: While reading the string, push all the a's that appear onto stack 1. If we see b's or c's before a' we immediately reject. Then push all the b's that follow the a's in the input onto stack2. If it sees any a's at this stage, reject. When it sees the first c, pop stack 1 and stack 2 at the same time. After this, reject the input if it sees any a's or b's in the input. Pop stack 1 and stack 2 for each input character c reads. If the input ends at the same time both stacks become empty, accept. Otherwise, reject.

b. We show that a 2-PDA can simulate a Turing Machine (TM). In addition, a 3-tape Nondeterministic Turing Machine (NTM) can simulate a 3-PDA, and an ordinary deterministic 1-tape TM can simulate a 3-tape NTM.

Therefore a 2-PDA can simulate a 3-PDA, and so they are equivalent in power. The simulation of a TM by a 2-PDA is as follows: Record the tape of the TM onto two stacks, stack 1 stores the characters on the left of the head, with the bottom of stack storing the leftmost character of the tape in the TM. Stack 2 stores the characters on the right of the head, with the bottom of the stack storing the rightmost non-blank character on the tape in the TM. In other words, if a TM configuration is adxb, the corresponding 2-PDA is in state dx, with stack 1 storing the string a and stack 2 storing the string b to power of R, both from bottom to top. For each transition in the TM, the corresponding PDA transition pops cx off stack 2, pushes cy into stack 2, then pops stack 1 and pushes the character into stack 2, and goes from state dx to dy pops ci off stack 2 and pushes cy into stack 1, and goes from state xi to xj. So we have simulated the Turing machine using 2-pda

Exercise 3.11

```
A Turing machine with doubly infinite tapes are similar to an ordinary
Turing machine, but its tape is infinite to the left as well as to
the right.  The tape is initially defined as usual except that the
head never encounters an end to the tape as it moves leftward.  Show
that this type of Turing machine recognizes the class of Turing-recognizable
languages.
```

A TM with doubly infinite tape can simulate an ordinary TM. It marks the square to the left of the input to detect and prevent the head from moving off of the end of the tape. To simulate the doubly infinite tape TM by an ordinary TM, we show how to simulate it with a 2-tape TM, which was already shown to be equivalent in power to an ordinary TM by the book. The first tape of the 2-tape TM is written with the input string and the second tape is initially blank. As a first step of the simulation we write a new symbol X on the left hand squares of each tape. For the first tape we do this after shifting the contents over by one. We leave the second tape head over the X on the second tape. We then step by step simulate the doubly infinite tape machine. While doubly infinite tape machine stays on the input or to the right, we only move the first tape head of our simulating machine leaving the second tape on the X. If the doubly infinite tape machine tries to move left off the input, our machine moves the first tape head onto its X and then moves its second tape head to the right off of its X. The second tape is

then used to simulate the doubly infinite tape machines actions to the left of the input. Here move lefts (move rights) on the doubly infinite tape machine correspond to move rights (move lefts) on our simulator on tape 2. When the doubly infinite tape machine moves right back onto the input again, in our simulation we move the second tape back onto the X and resume simulating on the first tape.

Exercise 3.15

a. Let L1 and L2 be decided by TM1 and TM2. We then make a new Turing machine that we call TM3 which can decide the union between L1 and L2:
On input w:
1. Run TM1 on L1 and if it accepts, then TM1 accepts, if not, TM1 rejects L1.
2. Run TM2 on L2 and if it accepts, TM2 accepts. If it does not accept,then TM2 reject.

If either TM1 or TM2 accepts, then TM3 accepts. If both TM1 and TM2 rejects, then TM3 rejects. So we have a machine TM3, which can decide the union of two decidable languages. So we have shown that two decidable languages is closed under the union operation because the union is also a decidable language.

b. We next prove the decidable languages are closed under concatenation. Given two decidable languages L1 and L2 , let TM1 and TM2 decide them. We will build a deterministic Turing machine TM3 which will recognize the concatenation of the two languages: "On input string w, which is a concatenation between two strings:

1. Our decision procedure cycles over the $|w|$ many ways to split w into $w1 \circ w2$.
2. Run TM1 on w1
3. Run TM2 on w2
4. If both of the machine accepts, then TM3 accepts, if not then we will try with another w1,w2 (This combination produces w).
5. If all the possible w1 and w2 has been tried on the machine, and none give any accept states for TM1 and TM2 on the same possible cut of w1 and

w2, then reject.

c. We prove that the decidable languages are closed under star operation... Suppose we have a Turing Machine TM1 that recognizes the decidable language L1. The following procedure decides $(L1)^*$
"On input w:
1. We cycle over each way to cut w so that w = w1w2w3...wn
2. Run TM1 on each w_k for k=1,....n
If TM1 accepts each of this strings w_k,then Accepts
3. If TM1 runs on all possible way to cut w to w1w2w3...wn, but has not yet no accepted, then we reject.

d. To prove that the decidable languages are closed under the complement, we will do the following. Given a language L1 decided by Turing machine TM1, we construct a new Turing Machine TM2 for the complement as follows:
On input string w:
1. We will run TM1 on the string w, if TM1 accepts TM2 rejects, if TM1 rejects, TM2 accepts.

e. To prove that the decidable languages are closed under the intersection operation, we will do the following. Suppose we have two languages L1 and L2 decided by TM1 and TM2. We construct a new Turing Machine TM3 that decides the intersection between L1 and L2.

On the input string w:
1. Run TM1 on w, if TM1 rejects, TM3 rejects
2. Run TM2 on w, if it accepts, and TM1 accepts, then TM3 accepts. If TM2 itself rejects, then TM3 rejects.

Exercise 4.2

Let $EQ_{DFA,REX}$ = {(A,R)| A a DFA, R is a regular expression and L(A)=L(R)}.
The following TM E decides $EQ_{DFA,REX}$:

E=On input (A,R):
1. Convert regular expression R into an equivalent DFA B using the procedures given in Theorem 1.54.
2. Use the TM C for deciding EQ_{DFA} given in Theorem 4.5, on input (A,B).
3. If R accepts, accept. Otherwise, reject.

Exercise 4.12

We observe that L(R) $\subseteq$ L(S) if and only if $\overline{L(S)} \bigcap$ L(R) = $\emptyset$. The following TM X decides A.

X = "On input (R,S) where R and S are regular expressions:
1. Construct a DFA E such that L(E) =$\overline{L(S)} \bigcap$ L(R).
2. Run TM T on (E), where T decides E_{DFA} from Theorem 4.4.
3. If T accepts, accept. Otherwise, reject."

Exercise 4.27

Given a CFG G and a number k, first scan the encoding of G to determine the number m of nonterminals and b the maximum number of nonterminal on the right hand side of a rule. Then cycle over all possible derivations of length $\leq b^m + 1$ where each possible variable is used as a start variable. If it is possible that a variable can go to a string of terminals it must be possible to do it with a derivation of less than this size since after length $\leq b^m + 1$ some variable must repeat. If it is possible let D_A be a derivation $D \longrightarrow w$. Now for each possible derivation beginning with the start variable in G of length $\leq b^m + 1$, try continuing the derivation by deriving from the remaining variables A in the output string with their D_A. If this gets a string of terminals $\geq b^m + 1$ and $k = \infty$ then accept, this string will be pumpable. If there is no string $\geq b^m + 1$ and $k = \infty$ reject. If there is no string $\geq b^m + 1$ and k is some other number and then count the number of strings which are derivable with derivations of length $\leq b^m + 1$, if this equals k accepts otherwise reject.