CS481F01 Solutions 6 – PDAS

A. Demers

2 November 2001

1. Give a NPDAs that recognize the following languages:

(a) The set of all strings in $\{0,1\}^*$ that contain twice as many $1s$ as $0s$.

(answer a) We build a machine that keeps on its stack the $0s$ or $1s$ that need to be matched to complete the input word successfully.

$$\begin{aligned} M &= (Q, \{0,1\}, \Gamma, \delta, s, \bot, \emptyset) \\ Q &= \{s, s'\} \\ \Gamma &= \{0, C, 1, \bot\} \end{aligned}$$

The difficulty here is that after reading a 1 from the input you need to push "half" a 0. That's what the symbol C in the stack alphabet is supposed to represent.

The machine will accept by empty stack.

$$\begin{aligned} (s, \epsilon, \bot) &\rightsquigarrow (s, \epsilon) \\ (s, 0, \bot) &\rightsquigarrow (s, 11\bot) \\ (s, 1, \bot) &\rightsquigarrow (s, C\bot) \end{aligned}$$

$$\begin{aligned} (s, 1, C) &\rightsquigarrow (s, 0) \\ (s, 1, 0) &\rightsquigarrow (s, C0) \\ (s, 1, 1) &\rightsquigarrow (s, \epsilon) \end{aligned}$$

$$\begin{aligned} (s, 0, 0) &\rightsquigarrow (s, \epsilon) \\ (s, 0, C) &\rightsquigarrow (s', \epsilon) \\ (s', \epsilon, 0) &\rightsquigarrow (s, C) \end{aligned}$$

Correctness follows by showing that if

$$(s, w, \bot) \ \leadsto \ (s, \alpha)$$

then

$$\alpha \ \ \in \ \ (0 + C)0^*\bot \ + \ 1^*\bot$$

and for all such α

$$(s, w, \bot) \ \leadsto \ (s, \alpha) \qquad \text{iff} \qquad \sharp 1(w\alpha) = 2\sharp 0(w\alpha) + \sharp C(w\alpha)$$

The proof is a routine induction.

(b) The set

$$\{ \ x^r\$y \mid (\exists n)(x = \mathbf{binary}(n) \ \wedge \ y = \mathbf{binary}(n+1) \ \}$$

where $\mathbf{binary}(n)$ is the binary encoding of natural number n. For example, this set contains 0$1, 1101$1100 and 001$101 but not 1$1 or 11$10.

(answer b) The machine will read the binary encoding of n from the input, from least-significant digit to most-significant digit, at the same time pushing the binary encoding of $n + 1$ onto its stack. When it encounters the $, it will match the stack against the remaining input.

$$M \ \ = \ \ (\{p_0, p_1, q\}, \{0, 1, \$\}, \{0, 1, \bot\}, \delta, p_1, \emptyset)$$

$$\begin{array}{l}
(p_1, 0, A) \ \leadsto \ (p_0, 1A) \\
(p_1, 1, A) \ \leadsto \ (p_1, 0A) \\
(p_0, 0, A) \ \leadsto \ (p_0, 0A) \\
(p_0, 1, A) \ \leadsto \ (p_0, 1A)
\end{array}$$

$$\begin{array}{l}
(p_1, \$, A) \ \leadsto \ (q, 1A) \\
(p_0, \$, A) \ \leadsto \ (q, A)
\end{array}$$

$$\begin{array}{l}
(q, 1, 1) \ \leadsto \ (q, \epsilon) \\
(q, 0, 0) \ \leadsto \ (q, \epsilon) \\
(q, \epsilon, \bot) \ \leadsto \ (q, \epsilon)
\end{array}$$

The two states p_0, p_1 represent a "carry" value of 0 or 1, respectively; the machine transitions to state q after reading $; in state q it matches the stack against the remaining input.

2. An NPDA

$$M \;=\; (\; Q, \Sigma, \Gamma, \delta, s, \bot, F \;)$$

is a *Binary-Stack* NPDA if $|\Gamma| = 2$. M is a *Unary-Stack* NPDA if $|\Gamma| = 1$.

(a) Prove that every CFL is $L_{es}(M)$ for some Binary-Stack NPDA M.

(answer a) Suppose

$$M \;=\; (Q, \Sigma, \Gamma, \delta, s, \bot, F)$$

is a NPDA that accepts by final state. We construct a Binary-Stack NPDA recognizing the same language, again by final state.

$$M' \;=\; (Q', \Sigma, \{0,1\}, \delta', s', 0, F')$$

Let

$$k \;=\; \lceil (\log |\Gamma|) \rceil$$

so that any symbol from Γ can be encoded in k bits. The states of M' will be

$$Q' \;=\; \{\; s' \;\} \;\cup\; \{q_u \mid q \in Q,\; u \in \{0,1\}^{*k} \;\}$$

The machine works by treating a group of k bits at the top of the stack as if it were a single symbol from a larger stack alphabet, by popping the k bits from the stack and remembering them in the state. This is similar to the technique used by a bottom-up parser to read the top two symbols from the stack and compare them to the right hand side of a production.

For any $A \in \Gamma$, let $\mathbf{rep}(A)$ be a binary string of length k that we use as the representation of A. The initial move of M' is

$$(s', \epsilon, 0) \;\rightsquigarrow\; (s_\epsilon, \mathbf{rep}(\bot)0)$$

This pushes the representation of $\bot$ onto the stack. Now, for each state $q \in Q$ and each $u \in \Sigma^{*(k-1)}$ we have

$$(q_u, \epsilon, 0) \;\rightsquigarrow\; (q_{u0}, \epsilon)$$
$$(q_u, \epsilon, 0) \;\rightsquigarrow\; (q_{u0}, \epsilon)$$

These states have the effect of popping the top k symbols from the stack and remembering them (in the subscript of the state name). Now suppose that in M we had

$$(q, \epsilon, A) \rightsquigarrow (r, B_1 \ldots B_m)$$

In M' we would add

$$(q_{\mathbf{code}(A)}, \epsilon, z) \rightsquigarrow (r_\epsilon, \mathbf{code}(B_1) \ldots \mathbf{code}(B_m)z$$

for all $z \in \{0, 1\}$. That is, the state $q_{\mathbf{code}(A)}$ that "remembers" having popped $\mathbf{code}(A)$ from the stack simulates the move that M would have made with A as the top of the stack. Similarly, if in M we had

$$(q, a, A) \rightsquigarrow (r, B_1 \ldots B_m)$$

In M' we would add

$$(q_{\mathbf{code}(A)}, a, z) \rightsquigarrow (r_\epsilon, \mathbf{code}(B_1) \ldots \mathbf{code}(B_m)z$$

For the final states of M' we chose

$$F' = \{ q_\epsilon \mid q \in F \}$$

A formal proof of correctness is a tedious but straightforward argument.

(b) Give a language L that is not regular but is $L_{es}(M)$ for some Unary-Stack NPDA M.

(answer b) It should be clear that the non-regular language

$$\{ 0^i 1 0^i \mid i > 0 \}$$

can be recognized by a (deterministic) Unary-Stack NPDA by empty stack.

(c) Does there exist a CFL L that is not $L_{es}(M)$ for any Unary-Stack NPDA M? Argue convincingly for your answer. A detailed proof is not necessary.

(answer c) This is a much simpler argument in the deterministic case, sigh. But here's a reasonably careful argument for the nondeterministic case.

First of all, observe that a unary stack is just a counter. We might as well call a configuration (q, w, n) where n, the stack content, is a natural number. The initial stack symbol is the *only* stack symbol. The initial configuration of a machine with input w is just $(s, w, 1)$. Further, the machine cannot recognize the bottom of its stack. Thus, if

$$(p, w, m) \quad \rightarrow^* \quad (q, \epsilon, n)$$

then, for all $k \geq 0$,

$$(p, w, m + k) \quad \rightarrow^* \quad (q, \epsilon, n + k)$$

and for all $k < m$ either

$$(p, w, m - k) \quad \rightarrow^* \quad (q, \epsilon, n - k)$$

or

$$(\exists q')(\exists w' \neq \epsilon)(\ (p, w, m - k) \quad \rightarrow^* \quad (q', w', 0)\ )$$

That is, if a computation starting from a given state p with stack n accepts input w, then from the same state given a shorter stack the machine must accept some prefix of w.

Now here's a related property: the stack can't grow more than linearly in the following sense:

For any Unary-Stack NPDA there exists a constant c such that

$$\begin{array}{lll} (p, w, 1) & \rightarrow^* & (q, \epsilon, n) \\ \Rightarrow & (p, w, 1) & \rightarrow^* \ (q, \epsilon, n') \qquad n' \leq c(|w| + 1) \end{array}$$

To prove this, let

$$c = (|Q| + 1)(m) \qquad m = \text{ max } \sharp \text{ syms pushed in a transition of } M$$

Now we use induction on $|w|$.

Basis: $|w| = 0$. Clearly a computation pushing more than c symbols must perform more than $|Q|$ moves, all of them ϵ-transitions, so by the pigeonhole

principle some state must repeat, and the resulting loop can be removed yielding a computation that pushes fewer symbols.

Induction: Consider $w = ax$. The computation must look like

$$(p, ax, 1) \ \rightarrow^* \ (r, x, m) \ \rightarrow^* \ (q, \epsilon, n)$$

where m is at most c by the same argument used in the basis case. Now consider the last point in this computation at which the stack is shorter than m:

$$(r, x, m) \ \rightarrow^* (r', x', m') \ \rightarrow^* \ (q, \epsilon, n)$$

The induction hypothesis applies to allow us to say there exists n' such that

$$(r', x', m') \ \rightarrow^* \ (q, \epsilon, n') \qquad (n' - m' + 1) \leq c(|x'| + 1)$$

This gives us

$$m \leq c \qquad m' \leq m \qquad ((n' - m' + 1) \leq c(|x'| + 1)$$

from which the claim follows.

A similar claim, based on the same sort of looping argument, is

$$\begin{array}{rl} ((p, \epsilon, n) & \rightarrow^* (q, \epsilon, n')) \ \wedge \ (n - n' \ > \ c) \\ & \Rightarrow \ (p, \epsilon, n) \ \rightarrow^* \ (r, \epsilon, 0) \end{array}$$

That is, if a machine can shrink its stack very much without reading any input, then it can completely empty its stack (thereby accepting whatever input it has read up to that point.

We are almost there. Consider the language

$$L \ = \ \{ \ w\$w^r \mid w \in \{0,1\}^* \ \}$$

I claim this language cannot be recognized by a Unary-Stack machine. To see this, suppose M recognizes L and consider the sets

$$\mathcal{C}_w \ = \ \{(q, n) \mid (s, w\$w^r, 1) \ \rightarrow^* \ (q, w^r, n) \ \rightarrow^* \ (r, \epsilon, 0) \ \}$$

These are the states and stack lengths in accepting computations just after the \$ symbol is read. Now note that

$$\mathcal{C}_w \ \cup \ \mathcal{C}_x \ = \ \emptyset \qquad\qquad w \neq x$$

because otherwise the machine would accept strings of the form $w\$x^r$, which would be incorrect. But note there are 2^k distinct strings of length k. From this it follows that, for at least one w of length k, the shortest stack in $\mathcal{C}_w$ has length $O(2^k)$. Consider such a stack – it corresponds to a pair (q, n) in $\mathcal{C}_w$. By the lemma we proved above, there is another computation

$$(s, w\$w^r, 1) \ \rightarrow^* \ (q, w^r, n') \qquad\qquad n' \leq c(|w| + 1)$$

Since (q, n') is not in $\mathcal{C}_w$, it follows that this computation does not continue on to accept $w\$w^r$. But since the computation reaches the same state q, the only way it can fail to continue on and accept after reading w^r is by emptying its stack before reaching the end of w^r. But that implies some prefix of w^r leads to acceptance:

$$w = xy \qquad (s, xy\$y^r, 1) \ \rightarrow \ (q, y^r, n') \ \rightarrow^* (p, \epsilon, 0)$$

contradicting our assumption that M recognizes exactly the set of palindromes L. Phew! Your argument didn't need to be nearly this detailed.

3. This is a "cumulative" problem – each part develops on the previous parts. We derive some properties of the Deterministic CFLs (DCFLs), i.e. languages that are $L(M)$ for some Deterministic PDA M.

(a) Show the set

$$\{ \ a^i b^i c^i \mid i > 0 \ \}$$

is not a CFL.

(answer a) This is a simple pumping lemma application. Consider the word

$$a^k b^k c^k$$

where k is the constant of the pumping lemma. By the p.l. we can conclude

$$a^k b^k c^k \quad = \quad uvwxy$$

where vwx can be pumped and the length of vwx is at most k. The upper bound on the length of vwx implies that vx can contain at most two different letters. Thus, any pump cannot contain equal numbers of a, b and c, and so cannot be in the set.

(b) Show the set

$$\{\ a^i b^j c^i \mid i, j > 0\ \}$$

is a DCFL.

(answer b) We essentially did this in lecture. The machine pushes all the $a's$ it sees, reads and skips all $b's$, and then matches the remaining $c's$ against the stacked $a's$. It rejects if its input is not in $a^*b^*c^*$. This is easily done without nondeterminism.

(c) Show the DCFLs are not closed under intersection: give DCFLs L_1 and L_2 such that $L_1 \cap L_2$ is not a DCFL.

(answer c) By a technique similar to part (b), we can show that

$$\{\ a^i b^i c^j \mid i, j > 0\ \}$$

is a DCFL. But the intersection of this DCFL with the one from part (b) is

$$\{\ a^i b^j c^i \mid i, j > 0\ \} \cup \{\ a^i b^i c^j \mid i, j > 0\ \} \quad = \quad \{\ a^i b^i c^i \mid i > 0\ \}$$

which we showed in part (a) was not a CFL.

(d) In lecture and in the text we show that DCFLs are closed under complement:

$$L \text{ is a } DCFL \quad \Rightarrow \quad \bar{L} \ \equiv\ (\Sigma^* - L) \text{ is a } DCFL$$

It follows that the DCFLs cannot be closed under union (Why?).

(answer Why) Because

$$L_1 \cap L_2 \quad = \quad \overline{(\bar{L_1} \cup \bar{L_2})}$$

so closure under union would imply closure under intersection.

Give an example; that is, give two DCFLs L_1 and L_2 such that $L_1 \cup L_2$ is not a DCFL. (Note that $L_1 \cup L_2$ is certainly a CFL; it's just not a deterministic one).

(answer d) Let

$$\begin{array}{rcl} L_1 & = & \{\ a^i b^j c^k \mid i \neq j\ \} \\ L_2 & = & \{\ a^i b^j c^k \mid j \neq k\ \} \end{array}$$

Each of these is a DCFL by an argument analogous to part (b). Now consider

$$\overline{(L_1\ \cup\ L_2)}\ \cap\ a^*b^*c^*) \quad = \quad \{a^i b^i b^i \mid i > 0\ \}$$

This is not a CFL. But it is the intersection of

$$\overline{L_1 \cup L_2}$$

with a regular set, We know the CFLs are closed under intersection with a regular set. We conclude that

$$\overline{L_1 \cup L_2}$$

is not a CFL, and thus $L_1 \cup L_2$ cannot be a DCFL.

(e) Let the *tagged union* of two languages be defined by

$$L_0 \cup_t L_1 \quad \equiv \quad \{\ 0w \mid w \in L_0\ \}\ \cup\ \{\ 1w \mid w \in L_1\ \}$$

Prove the DCFLs are closed under tagged union.

(answer e) This is a straightforward machine construction. Suppose we have DPDAs M_0 and M_1 recognizing L_0 and L_1 respectively. Assume the state sets of the two machines are disjoint, but they have the same initial stack symbol $\perp$. We construct a new machine with

$$Q = Q_0 \cup Q_1 \cup \{s\}$$

The new machine has all the states and transitions of M_0 and M_1. Its initial state is s. From the initial state there are two possible transitions:

$$\begin{array}{l} (s, 0, \perp)\ \rightsquigarrow\ (s_0, \perp) \\ (s, 1, \perp)\ \rightsquigarrow\ (s_1, \perp) \end{array}$$

Clearly this machine simply reads its first input symbol and (deterministically) decides to simulate either M_0 or M_1 as appropriate.

(f) Are the DCFLs closed under homomorphism? Explain your answer.

(answer f) No. Let L_0 and L_1 be DCFLs over $\{a, b, c\}$ such that

$$L_0 \cup L_1 \quad \text{is not a DCFL}$$

Now define the homomorphism h by

$$h(0) = h(1) = \epsilon \qquad h(a) = a \qquad h(b) = b \qquad h(c) = c$$

Now if L is the tagged union of L_0 and L_1, clearly

$$h(L) = L_0 \cup L_1 \quad \text{is not a DCFL}$$

as required.