CS481F01 Prelim 2 Solutions

A. Demers

7 Nov 2001

1 (30 pts = 4 pts each part + 2 free points). For this question we use the following notation:

$x \preceq y$	means	x is a prefix of y
$m \approx_k n$	means	$\|m - n\| \leq k$

For each of the following languages, tell whether it is (1) regular, (2) a deterministic CFL, (3) a nondeterministic CFL, or (4) not a CFL. Assume the languages are over the alphabet $\Sigma = \{a, b, c\}$.

$$(a) \qquad \{ \; w \mid \sharp a(w) \; = \; \sharp b(w) \; \}$$

DCFL. A deterministic PDA can maintain on its stack the excess $a's$ or $b's$ that have been seen, popping them when complementary characters are read. I simply ignores any $c's$.

$$(b) \qquad \{ \; w \mid (\sharp a(w) \; = \; \sharp b(w)) \vee (\sharp a(w) \; = \; \sharp c(w)) \; \}$$

CFL. This is clearly the union of two DCFLs, and thus is a CFL. To see it is not deterministic, note that taking the complement and then intersecting with the regular set $a^*b^*c^*$ yields the set

$$\{ \; a^i b^j c^k \mid i \neq j \wedge i \neq k \; \}$$

which is easily seen not to be a CFL.

$$(c) \qquad \{ \; w \mid (\sharp a(w) \; = \; \sharp b(w)) \wedge (\sharp a(w) \; = \; \sharp c(w)) \; \}$$

NOT CFL. Intersect with the set $a^*b^*c^*$ and apply the pumping lemma.

$$(d) \qquad \{ \; w \mid (\forall x \preceq w)(\sharp a(x) \; \approx_2 \; \sharp b(x)) \; \}$$

REGULAR. The conditions must hold for every prefix x of w. Thus, the letter counts of $a's$, $b's$ and $c's$ cannot differ by more than 2 at any point, and can be checked continuously by a finite automaton.

$$(e) \qquad \{\ w \mid (\forall x \preceq w)((\sharp a(x)\ \approx_2\ \sharp b(x)) \vee (\sharp a(x)\ \approx_2\ \sharp c(x)))\ \}$$

DCFL. The first edition of this solution set mistakenly called this set regular. To see why it is not regular, consider the numbers

$$\begin{aligned} n_b(w,i) &= \sharp b(w[1..i]) - \sharp a(w[1..i]) \\ n_c(w,i) &= \sharp c(w[1..i]) - \sharp a(w[1..i]) \end{aligned}$$

Then the desired language is exactly

$$\{\ w \mid (\forall i, 1 \le i \le |w|)(|n_b(w,i)| \le 2\ \vee\ |n_c(w,i)| \le 2)\ \}$$

If there were two universal quantifiers, "under" the $\vee$ operator, we would have

$$\begin{aligned} \{\ w \mid ((\forall i, 1 \le i \le |w|)(|n_b(w,i)| \le 2)) \\ \vee\ ((\forall i, 1 \le i \le |w|)(|n_c(w,i) \le 2))\ \} \end{aligned}$$

which is the union of two regular sets, and thus regular. However, with only a single quantifier the language is not regular, because for any i one of $n_b(w,i)$ or $n_c(w,i)$ is allowed to be arbitrarily large, as long as (the absolute value of) the other remains below 2. Thus, roughly,

$$L\ \cap\ (ab)^*c^*(abc)^* \quad \approx \quad \{\ (ab)^i c^i (abc)^j \mid i,j > 0\ \}$$

which is easily seen not to be regular using the Pumping Lemma.

So what is the answer? This is a DCFL. (In fact, it's a deterministic 1-counter language, but that wasn't one of the choices). To see this, consider scanning w from left to right, keeping track of both $n_b(w,i)$ and $n_c(w,i)$. Initially, do this with the finite states. Suppose $n_b(w,i)$ grows outside the range [-2,2]. At that point begin using the stack to maintain $n_b(w,i)$. When the value returns to the range [-2,2], switch back to maintaining $n_b(w,i)$ in the finite states. (This is easy to detect, e.g. by using a different symbol for the bottom two characters of the stack.) Similarly, whenever $n_c(w,i)$ goes out of range, use the stack to maintain $n_c(w,i)$. The language definition requires that at most one of $n_b(w,i)$ and $n_c(w,i)$ can be out of range at any point; thus at most one of them needs the stack. The choice of which counter needs the stack can be made deterministically. If at any point *both* counters need the stack, the machine rejects.

$$(f) \qquad \{\ w \mid (\forall x \preceq w)((\sharp a(x)\ \approx_2\ \sharp b(x)) \wedge (\sharp a(x)\ \approx_2\ \sharp c(x)))\ \}$$

REGULAR, by an argument similar to part (d).

$$(g) \qquad \{\ w \mid (\sharp a(w)\ =\ \sharp b(w))\ \wedge\ (\forall x \preceq w)(\sharp a(x)\ \approx_2\ \sharp c(x))\ \}$$

DCFL. This language can be recognized by a product machine consisting of a deterministic PDA as for part (a) running in parallel with a deterministic finite automaton that makes sure the numbers of $a's$ and $c's$ always remain within 2 of one another.

2 (25 pts). Recall a CFG is *linear* if no production has more than one nonterminal in its right hand side; that is, every production is of the form

$$A\ \rightarrow\ x \qquad \text{or} \qquad A\ \rightarrow\ xBy$$

where x and y are possibly-empty terminal strings. We say L is a *linear language* if $L\ =\ L(G)$ for some linear grammar G.

(a) Does every linear language have a linear grammar in GNF?

No. A linear grammar in GNF has productions of the form

$$A\ \rightarrow\ a \qquad \text{or} \qquad A\ \rightarrow\ aB$$

This is a right-linear grammar; it generates a regular set. There are many non-regular linear languages – the palindromes, for example.

(b) Are the linear languages closed under union; i.e.,

$$L_1, L_2 \text{ linear} \qquad \Rightarrow \qquad L_1 \cup L_2 \text{ linear ?}$$

Yes. Let

$$G_1 = (N_1, \Sigma, P_1, S_1) \qquad G_2 = (N_2, \Sigma, P_2, S_2)$$

Union the nonterminal and production sets, and add a new start symbol S' with the productions

$$S'\ \rightarrow\ S_1 \mid S_2$$

This grammar clearly generates the union language.

(c) Are the linear languages closed under intersection; i.e.,

$$L_1, L_2 \text{ linear} \qquad \Rightarrow \qquad L_1 \cap L_2 \text{ linear ?}$$

No. Consider the two linear grammars

$$\begin{aligned} S &\rightarrow aS \mid B \\ B &\rightarrow bBc \mid bc \end{aligned}$$

$$\begin{aligned} S' &\rightarrow S'c \mid A \\ A &\rightarrow aAb \mid ab \end{aligned}$$

The intersection of these languages is our old friend

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

which is not even context-free.

(d) Are the linear languages closed under complement; i.e.,

$$L \text{ linear} \qquad \Rightarrow \qquad \overline{L} \text{ linear ?}$$

No. Note that

$$L_1 \cap L_2 \quad = \quad \overline{\overline{L_1} \cup \overline{L_2}}$$

Since the linear languages are closed under union but not under intersection, they cannot be closed under complement.

(e) Is every linear language a deterministic CFL?

No. A grammar similar to those in part (b) above can be constructed to show that the language

$$\{\ a^i b^j c^k \mid i \neq j \vee j \neq k\ \}$$

is linear, and its complement is not context-free.

3 (20 pts). Younger's algorithm for a CNF grammar can be expressed as follows. Given an input string $x = a_1 a_2 \ldots a_n$, execute the following code:

```
for $i = 0$ to $n - 1$ do
    $T[i, i+1] \leftarrow \{ A \mid A \rightarrow a_{i+1} \in P \}$
for $d = 2$ to $n$ do
    for $i = 0$ to $n - d$ do
        $j \leftarrow i + d$;  $T[i, j] \leftarrow \emptyset$
        for $k = i + 1$ to $j - 1$ do
            $T[i, j] \leftarrow T[i, j] \cup \{ A \mid (A \rightarrow BC \in P)$
                                   $\wedge (B \in T[i, k]) \wedge (C \in T[k, j]) \}$
if $S \in T[0, n]$ then return true else return false
```

We showed in lecture that a CNF grammar can be converted to an equivalent GNF grammar in which every production right hand side has at most two nonterminals; i.e., every production is of the form

$$A \rightarrow aB_1 \ldots B_m \qquad 0 \leq m \leq 2$$

How would you modify the above algorithm to work on such a GNF grammar? What is the running time of the revised algorithm, as a function of the input length n? The size of the grammar (e.g. the number of productions) may be ignored in your answer.

The change is straightforward. The GNF grammar has productions of three types:

(1)	$A \rightarrow a$
(2)	$A \rightarrow aB$
(3)	$A \rightarrow aBC$

The type (1) productions are just as in CNF, and are handled by the first loop of the algorithm. The type (2) productions are handled by an additional assignment outside the innermost loop. Type (3) productions are handled by a slight modification of the inner loop. Thus, the revised algorithm is

```
for $i = 0$ to $n - 1$ do
    $T[i, i+1] \leftarrow \{ A \mid A \rightarrow a_{i+1} \in P \}$
for $d = 2$ to $n$ do
    for $i = 0$ to $n - d$ do
        $j \leftarrow i + d$
        $T[i, j] \leftarrow \{ A \mid (A \rightarrow aB \in P)$
                        $\wedge (a = a_{i+1}) \wedge (B \in T[i+1, j]) \}$
        for $k = i + 2$ to $j - 1$ do
            $T[i, j] \leftarrow T[i, j] \cup \{ A \mid (A \rightarrow aBC \in P)$
                        $\wedge (a = a_{i+1})$
                        $\wedge (B \in T[i+1, k]) \wedge (C \in T[k, j]) \}$
if $S \in T[0, n]$ then return true else return false
```

The running time is still dominated by the inner loop, which is easily seen to be $O(n^3)$.

4 (25 pts = 12 + 13). Let B_k be the alphabet of k different kinds of parentheses:

$$B_k = \{ [_1, [_2, \ldots, [_k,]_1,]_2, \ldots,]_k \}$$

Recall $DS_k \subseteq B_k^*$, the *Dyck Set* of order k, is the set of all balanced strings of parentheses of k kinds.

In lecture we proved the Chomsky-Schutzenberger Theorem: every CFL L can be expressed as

$$L = h(DS_m \cap R)$$

for suitably chosen natural number m, homomorphism h, and regular set R. Here we examine how many kinds of parentheses are really necessary.

(a) Let

$$\Sigma = \{0, 1\} \qquad \Gamma = \{a_1, a_2, \ldots, a_m\}$$

Give a homomorphism $h : \Gamma^* \rightarrow \Sigma^*$ that is 1-to-1:

$$(\forall x, y \in \Gamma^*)(h(x) = h(y) \Rightarrow x = y)$$

Your homomorphism need not be *onto* Σ^* – it is okay if

$$(\exists z \in \Sigma^*)(\ h^{-1}(z) \ = \ \emptyset\ )$$

But your h must be 1-to-1 and must be defined on every element of Γ^*. You should explain why your answer is correct; a formal proof is not necessary.

Here is one simple approach.

$$h(a_i) \quad = \quad 0^i 1^{(m-i)}$$

This is like a fixed-length unary encoding of i. It ensures that, for all $i \neq j$

$$h(a_i) \ \neq \ h(a_j) \qquad \text{but} \qquad |h(a_i)| \ = \ |h(a_j)| \ = \ m$$

Thus, the image of any word in Γ^* can be decoded uniquely by breaking it into blocks of exactly m symbols each, then counting the leading 0's in each block.

Of course, there is a similar solution that uses binary rather than unary encoding of a_i.

(b) Show how your solution to part (a) can be extended to give a 1-to-1 homomorphism $g_m : B_m^* \rightarrow B_2^*$ such that

$$g_m^{-1}(\ DS_2\ ) \quad = \quad DS_m$$

Explain your answer.

Hint: Your solution for g_m does not need to be onto B_2^*, or even onto DS_2. But it does need to preserve the property of being a balanced string. If you can show

$$\begin{array}{l} w \in DS_m \quad \Rightarrow \quad g_m(w) \in DS_2 \\ w \in (B_m^* \ - \ DS_m) \quad \Rightarrow \quad g_m(w) \in (B_2^* \ - \ DS_2) \end{array}$$

and g_m is 1-to-1, then it is a correct solution. Why?

As in part (a), we map each symbol to a fixed length block of symbols. We map left brackets to blocks of left brackets, and right brackets to blocks of right brackets:

$$\begin{array}{lcl} g_m(\ [_i\ ) & = & [_0^i\ [_1^{m-i} \\ g_m(\ ]_i\ ) & = &]_1^{m-i}\ ]_0^i \end{array}$$

The argument that this homomorphism is 1-to-1 goes just as in part (a). For the rest, observe that

$$g_m(\ [_i\ )\ \text{matches}\ g_m(\ ]_j\ ) \qquad \Leftrightarrow \qquad i\ =\ j$$

Note the left-bracket and right-bracket mappings need to be "reversed" in order to achieve this matching.

From this observation it is easily seen that g_m preserves the property of being a balanced string as described above. A string x in DS_2 is the image of a string w in DS_m iff the m-symbol blocks of x are all of consistent handedness (left or right), and each block consists of a (possibly empty) sequence of $[_0$ (or $]_1$) characters followed by a (possibly empty) sequence of $[_1$ (or $]_0$) characters. For each such block you can determine the inverse image by counting the number of leading $[_0$ (or $]_1$) characters.

You can now conclude from the Chomsky-Schutzenberger Theorem that every CFL L is

$$L\ =\ h(\ g_m^{-1}(DS_2)\ \cap\ R\ )$$

for suitably chosen natural number m, homomorphism h, and regular set R.