CS 516—Software Foundations via Formal Languages—Spring 2022

Problem Set 7

Model Answers

Problem 1

Suppose, toward a contradiction, that X is context-free. Thus there is an $n \in \mathbb{N} - \{0\}$ with the property of the Pumping Lemma for Context-free Languages, where X has been substituted for L. Let $z = \mathsf{0}^n\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n$. Then $z \in X$ and $|z| = 4n \geq n$. Thus the property of the lemma tells us there are $u, v, w, x, y \in \textbf{Str}$ such that $z = uvwxy$ and

(1) $|vwx| \leq n$; and

(2) $vx \neq \%$; and

(3) $uv^iwx^iy \in X$, for all $i \in \mathbb{N}$.

Because $\mathsf{0}^n\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n = z = uvwxy$, (1) tells us that:

- **alphabet**(vwx) does not include both $\mathsf{0}$ and $\mathsf{2}$; and
- **alphabet**(vwx) does not include both $\mathsf{1}$ and $\mathsf{3}$.

By (2), we have that **alphabet**(vx) is a nonempty subset of $\{\mathsf{0}, \mathsf{1}, \mathsf{2}, \mathsf{3}\}$. And by (3), we have that $uwy = uv^0wx^0y \in X$. Thus there are four cases to consider.

- ($\mathsf{0} \in$ **alphabet**(vx)) Then $\mathsf{2} \notin$ **alphabet**(vx). Thus uwy has less-than n occurrences of $\mathsf{0}$, but n occurrences of $\mathsf{2}$, contradicting $uwy \in X$.
- ($\mathsf{1} \in$ **alphabet**(vx)) Then $\mathsf{3} \notin$ **alphabet**(vx). Thus uwy has less-than n occurrences of $\mathsf{1}$, but n occurrences of $\mathsf{3}$, contradicting $uwy \in X$.
- ($\mathsf{2} \in$ **alphabet**(vx)) Then $\mathsf{0} \notin$ **alphabet**(vx). Thus uwy has less-than n occurrences of $\mathsf{2}$, but n occurrences of $\mathsf{0}$, contradicting $uwy \in X$.
- ($\mathsf{3} \in$ **alphabet**(vx)) Then $\mathsf{1} \notin$ **alphabet**(vx). Thus uwy has less-than n occurrences of $\mathsf{3}$, but n occurrences of $\mathsf{1}$, contradicting $uwy \in X$.

Because we obtained a contradiction in each case, we have an overall contradiction. Thus X is not context-free.

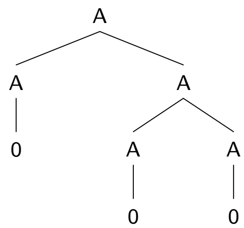
Problem 2

From the assumptions, we know that L is a regular language, G is a grammar in Chomsky Normal Form that generates $L - \{\%\}$, k is the number of variables of G, $n = 2^k$, $z \in L$ has length at least n, pt is a valid parse tree for G of height at least $k + 1$, where **rootLabel** $pt = s_G$ and **yield** $pt = z$, and pat is a path through pt whose length is the height of pt.

It is consistent with these assumptions that L is $\{\, \mathsf{0}^n \mid n \in \mathbb{N} \text{ and } n \geq 1 \,\}$, G is the grammar

$$\mathsf{A} \rightarrow \mathsf{AA} \mid \mathsf{0},$$

$k = 1$, $n = 2^k = 2^1 = 2$, $z = \mathsf{000}$, pt is



and pat is $[2, 1, 1]$. Thus the first repetition of variables as we follow pat through pt happens immediately.

Continuing the proof, this means that $pt' = pt$ and $pt'' = \mathsf{A}(\mathsf{A}(\mathsf{0}), \mathsf{A}(\mathsf{0}))$. Thus $u = \%$, $v = \mathsf{0}$, $w = \mathsf{00}$, $x = \%$ and $y = \%$. But this means that $|vwx| = |\mathsf{0}(\mathsf{00})\%| = |\mathsf{000}| = 3 > 2 = n$, violating the property (1) we needed to prove.

Problem 3

We define languages Y, Z and W by:

$$\begin{aligned} Y &= \{\, \mathsf{1}^n \mathsf{1}^j \mathsf{2}^k \mathsf{3}^n \mid n, j, k \in \mathbb{N} \text{ and } j \leq k \,\}, \\ Z &= \{\, \mathsf{0}^n \mathsf{1}^j \mathsf{2}^k \mathsf{2}^n \mid n, j, k \in \mathbb{N} \text{ and } j \leq k \,\}, \\ W &= \{\, \mathsf{1}^j \mathsf{2}^k \mid j, k \in \mathbb{N} \text{ and } j \leq k \,\}. \end{aligned}$$

We will show that $\Pi_{\mathsf{A}} = X$, $\Pi_{\mathsf{B}} = Y$, $\Pi_{\mathsf{C}} = Z$ and $\Pi_{\mathsf{D}} = W$. Thus we will be able to conclude $L(G) = \Pi_{\mathsf{A}} = X$.

Lemma PS7.3.1

(A) For all $w \in \Pi_{\mathsf{A}}$, $w \in X$.

(B) For all $w \in \Pi_{\mathsf{B}}$, $w \in Y$.

(C) For all $w \in \Pi_{\mathsf{C}}$, $w \in Z$.

(D) For all $w \in \Pi_{\mathsf{D}}$, $w \in W$.

Proof. By induction on Π. There are eleven productions to consider.

($\mathsf{A} \rightarrow \mathsf{0A3}$) Suppose $w \in \Pi_{\mathsf{A}}$, and assume the inductive hypothesis: $w \in X$. Thus $w = \mathsf{0}^i \mathsf{1}^j \mathsf{2}^k \mathsf{3}^l$ for some $i, j, k, l \in \mathbb{N}$ such that $i + j \leq k + l$. Hence $\mathsf{0}w\mathsf{3} = \mathsf{00}^i \mathsf{1}^j \mathsf{2}^k \mathsf{3}^l \mathsf{3} = \mathsf{0}^{i+1} \mathsf{1}^j \mathsf{2}^k \mathsf{3}^{l+1} \in X$, because $(i + 1) + j = i + j + 1 \leq k + l + 1 = k + (l + 1)$.

($\mathsf{A} \rightarrow \mathsf{A3}$) Suppose $w \in \Pi_{\mathsf{A}}$, and assume the inductive hypothesis: $w \in X$. Thus $w = \mathsf{0}^i \mathsf{1}^j \mathsf{2}^k \mathsf{3}^l$ for some $i, j, k, l \in \mathbb{N}$ such that $i + j \leq k + l$. Hence $w\mathsf{3} = \mathsf{0}^i \mathsf{1}^j \mathsf{2}^k \mathsf{3}^l \mathsf{3} = \mathsf{0}^i \mathsf{1}^j \mathsf{2}^k \mathsf{3}^{l+1} \in X$, because $i + j \leq k + l \leq k + l + 1 = k + (l + 1)$.

($\mathsf{A} \to \mathsf{B}$) Suppose $w \in \Pi_{\mathsf{B}}$, and assume the inductive hypothesis: $w \in Y$. Thus $w = \mathsf{1}^n\mathsf{1}^j\mathsf{2}^k\mathsf{3}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Hence $w = \mathsf{0}^0\mathsf{1}^n\mathsf{1}^j\mathsf{2}^k\mathsf{3}^n = \mathsf{0}^0\mathsf{1}^{n+j}\mathsf{2}^k\mathsf{3}^n \in X$, because $0 + (n + j) = n + j \leq n + k = k + n$.

($\mathsf{A} \to \mathsf{C}$) Suppose $w \in \Pi_{\mathsf{C}}$, and assume the inductive hypothesis: $w \in Z$. Thus $w = \mathsf{0}^n\mathsf{1}^j\mathsf{2}^k\mathsf{2}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Hence $w = \mathsf{0}^n\mathsf{1}^j\mathsf{2}^k\mathsf{2}^n\mathsf{3}^0 = \mathsf{0}^n\mathsf{1}^j\mathsf{2}^{k+n}\mathsf{3}^0 \in X$, because $n + j \leq n + k = (k + n) + 0$.

($\mathsf{B} \to \mathsf{1B3}$) Suppose $w \in \Pi_{\mathsf{B}}$, and assume the inductive hypothesis: $w \in Y$. Thus $w = \mathsf{1}^n\mathsf{1}^j\mathsf{2}^k\mathsf{3}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Hence $\mathsf{1}w\mathsf{3} = \mathsf{1}\mathsf{1}^n\mathsf{1}^j\mathsf{2}^k\mathsf{3}^n\mathsf{3} = \mathsf{1}^{n+1}\mathsf{1}^j\mathsf{2}^k\mathsf{3}^{n+1} \in Y$, because $j \leq k$.

($\mathsf{B} \to \mathsf{D}$) Suppose $w \in \Pi_{\mathsf{D}}$, and assume the inductive hypothesis: $w \in W$. Thus $w = \mathsf{1}^j\mathsf{2}^k$ for some $j, k \in \mathbb{N}$ such that $j \leq k$. Hence $w = \mathsf{1}^0\mathsf{1}^j\mathsf{2}^k\mathsf{3}^0 \in Y$, because $j \leq k$.

($\mathsf{C} \to \mathsf{0C2}$) Suppose $w \in \Pi_{\mathsf{C}}$, and assume the inductive hypothesis: $w \in Z$. Thus $w = \mathsf{0}^n\mathsf{1}^j\mathsf{2}^k\mathsf{2}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Hence $\mathsf{0}w\mathsf{2} = \mathsf{0}\mathsf{0}^n\mathsf{1}^j\mathsf{2}^k\mathsf{2}^n\mathsf{2} = \mathsf{0}^{n+1}\mathsf{1}^j\mathsf{2}^k\mathsf{2}^{n+1} \in Z$, because $j \leq k$.

($\mathsf{C} \to \mathsf{D}$) Suppose $w \in \Pi_{\mathsf{D}}$, and assume the inductive hypothesis: $w \in W$. Thus $w = \mathsf{1}^j\mathsf{2}^k$ for some $j, k \in \mathbb{N}$ such that $j \leq k$. Hence $w = \mathsf{0}^0\mathsf{1}^j\mathsf{2}^k\mathsf{2}^0 \in Z$, because $j \leq k$.

($\mathsf{D} \to \%$) We have that $\% = \mathsf{1}^0\mathsf{2}^0 \in W$, because $0 \leq 0$.

($\mathsf{D} \to \mathsf{1D2}$) Suppose $w \in \Pi_{\mathsf{D}}$, and assume the inductive hypothesis: $w \in W$. Thus $w = \mathsf{1}^j\mathsf{2}^k$ for some $j, k \in \mathbb{N}$ such that $j \leq k$. Hence $\mathsf{1}w\mathsf{2} = \mathsf{1}\mathsf{1}^j\mathsf{2}^k\mathsf{2} = \mathsf{1}^{j+1}\mathsf{2}^{k+1} \in W$, because $j + 1 \leq k + 1$.

($\mathsf{D} \to \mathsf{D2}$) Suppose $w \in \Pi_{\mathsf{D}}$, and assume the inductive hypothesis: $w \in W$. Thus $w = \mathsf{1}^j\mathsf{2}^k$ for some $j, k \in \mathbb{N}$ such that $j \leq k$. Hence $w\mathsf{2} = \mathsf{1}^j\mathsf{2}^k\mathsf{2} = \mathsf{1}^j\mathsf{2}^{k+1} \in W$, because $j \leq k \leq k + 1$.

□

Lemma PS7.3.2

(1) For all $n \in \mathbb{N}$, $\mathsf{2}^n \in \Pi_{\mathsf{D}}$.

(2) For all $w \in \Pi_{\mathsf{D}}$ and $n \in \mathbb{N}$, $\mathsf{1}^n w \mathsf{2}^n \in \Pi_{\mathsf{D}}$.

(3) For all $w \in \Pi_{\mathsf{D}}$ and $n \in \mathbb{N}$, $\mathsf{1}^n w \mathsf{3}^n \in \Pi_{\mathsf{B}}$.

(4) For all $w \in \Pi_{\mathsf{D}}$ and $n \in \mathbb{N}$, $\mathsf{0}^n w \mathsf{2}^n \in \Pi_{\mathsf{C}}$.

(5) For all $w \in \Pi_{\mathsf{A}}$ and $n \in \mathbb{N}$, $w\mathsf{3}^n \in \Pi_{\mathsf{A}}$.

(6) For all $w \in \Pi_{\mathsf{B}}$ and $n \in \mathbb{N}$, $\mathsf{0}^n w \mathsf{3}^n \in \Pi_{\mathsf{A}}$.

(7) For all $w \in \Pi_{\mathsf{C}}$ and $n \in \mathbb{N}$, $\mathsf{0}^n w \mathsf{3}^n \in \Pi_{\mathsf{A}}$.

Proof.

(1) We proceed by mathematical induction.

(Basis Step) We have $\mathsf{2}^0 = \% \in \Pi_{\mathsf{D}}$, because of the production $\mathsf{D} \to \%$.

(Inductive Step) Suppose $n \in \mathbb{N}$, and assume the inductive hypothesis: $\mathsf{2}^n \in \Pi_{\mathsf{D}}$. Then $\mathsf{2}^{n+1} = \mathsf{2}^n\mathsf{2} \in \Pi_{\mathsf{D}}$, because of the inductive hypothesis and the production $\mathsf{D} \to \mathsf{D2}$.

(2) Suppose $w \in \Pi_{\mathsf{D}}$. We must show that, for all $n \in \mathbb{N}$, $\mathsf{1}^n w \mathsf{2}^n \in \Pi_{\mathsf{D}}$. We proceed by mathematical induction.

(Basis Step) We have $\mathsf{1}^0 w \mathsf{2}^0 = w \in \Pi_{\mathsf{D}}$, by the assumption.

(Inductive Step) Suppose $n \in \mathbb{N}$, and assume the inductive hypothesis: $\mathsf{1}^n w \mathsf{2}^n \in \Pi_{\mathsf{D}}$. Then $\mathsf{1}^{n+1} w \mathsf{2}^{n+1} = \mathsf{1}(\mathsf{1}^n w \mathsf{2}^n)\mathsf{2} \in \Pi_{\mathsf{D}}$, because of the inductive hypothesis and the production $\mathsf{D} \to \mathsf{1D2}$.

(3) Suppose $w \in \Pi_{\mathsf{D}}$. We must show that, for all $n \in \mathbb{N}$, $\mathsf{1}^n w \mathsf{3}^n \in \Pi_{\mathsf{B}}$. We proceed by mathematical induction.

(Basis Step) We have $\mathsf{1}^0 w \mathsf{3}^0 = w \in \Pi_{\mathsf{B}}$, because of the assumption and the production $\mathsf{B} \to \mathsf{D}$.

(Inductive Step) Suppose $n \in \mathbb{N}$, and assume the inductive hypothesis: $\mathsf{1}^n w \mathsf{3}^n \in \Pi_{\mathsf{B}}$. Then $\mathsf{1}^{n+1} w \mathsf{3}^{n+1} = \mathsf{1}(\mathsf{1}^n w \mathsf{3}^n)\mathsf{3} \in \Pi_{\mathsf{B}}$, because of the inductive hypothesis and the production $\mathsf{B} \to \mathsf{1B3}$.

(4) Follows similarly to the preceding parts, using productions $\mathsf{C} \to \mathsf{0C2}$ and $\mathsf{C} \to \mathsf{D}$.

(5) Follows similarly to the preceding parts, using the production $\mathsf{A} \to \mathsf{A3}$.

(6) Follows similarly to the preceding parts, using the productions $\mathsf{A} \to \mathsf{0A3}$ and $\mathsf{A} \to \mathsf{B}$.

(7) Follows similarly to the preceding parts, using the productions $\mathsf{A} \to \mathsf{0A3}$ and $\mathsf{A} \to \mathsf{C}$.

□

Lemma PS7.3.3
$W \subseteq \Pi_{\mathsf{D}}$.

Proof. Suppose $w \in W$, so that $w = \mathsf{1}^j\mathsf{2}^k$ for some $j, k \in \mathbb{N}$ such that $j \leq k$. Since $j \leq k$, we have that $k = n + j$ for some $n \in \mathbb{N}$. Thus $w = \mathsf{1}^j\mathsf{2}^{n+j} = \mathsf{1}^j\mathsf{2}^n\mathsf{2}^j$. By Lemma PS7.3.2(1), we have that $\mathsf{2}^n \in \Pi_{\mathsf{D}}$. Thus $w = \mathsf{1}^j\mathsf{2}^n\mathsf{2}^j \in \Pi_{\mathsf{D}}$ by Lemma PS7.3.2(2). □

Lemma PS7.3.4
$Y \subseteq \Pi_{\mathsf{B}}$.

Proof. Suppose $w \in Y$, so that $w = \mathsf{1}^n\mathsf{1}^j\mathsf{2}^k\mathsf{3}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Since $j \leq k$, we have that $\mathsf{1}^j\mathsf{2}^k \in W \subseteq \Pi_{\mathsf{D}}$, by Lemma PS7.3.3. Thus $w = \mathsf{1}^n(\mathsf{1}^j\mathsf{2}^k)\mathsf{3}^n \in \Pi_{\mathsf{B}}$, by Lemma PS7.3.2(3).
□

Lemma PS7.3.5
$Z \subseteq \Pi_{\mathsf{C}}$.

Proof. Suppose $w \in Z$, so that $w = \mathsf{0}^n\mathsf{1}^j\mathsf{2}^k\mathsf{2}^n$ for some $n, j, k \in \mathbb{N}$ such that $j \leq k$. Since $j \leq k$, we have that $\mathsf{1}^j\mathsf{2}^k \in W \subseteq \Pi_{\mathsf{D}}$, by Lemma PS7.3.3. Thus $w = \mathsf{0}^n(\mathsf{1}^j\mathsf{2}^k)\mathsf{2}^n \in \Pi_{\mathsf{C}}$, by Lemma PS7.3.2(4). $\square$

Lemma PS7.3.6
$X \subseteq \Pi_{\mathsf{A}}$.

Proof. Suppose $w \in X$, so that $w = \mathsf{0}^i\mathsf{1}^j\mathsf{2}^k\mathsf{3}^l$ for some $i, j, k, l \in \mathbb{N}$ such that $i + j \leq k + l$. There are two cases to consider.

- Suppose $i \leq l$. Thus $l = i + n$ for some $n \in \mathbb{N}$, so that $w = \mathsf{0}^i\mathsf{1}^j\mathsf{2}^k\mathsf{3}^{i+n}$. Since $i + j \leq k + l = k + i + n$, it follows that $j \leq k + n$. There are two subcases to consider.
  - Suppose $n \leq j$. Thus $j = n + m$ for some $m \in \mathbb{N}$. Hence $w = \mathsf{0}^i\mathsf{1}^{n+m}\mathsf{2}^k\mathsf{3}^{i+n} = \mathsf{0}^i(\mathsf{1}^n\mathsf{1}^m\mathsf{2}^k\mathsf{3}^n)\mathsf{3}^i$. Since $j \leq k + n$, we have that $n + m \leq k + n$, and thus $m \leq k$. Hence $\mathsf{1}^n\mathsf{1}^m\mathsf{2}^k\mathsf{3}^n \in Y \subseteq \Pi_{\mathsf{B}}$, by Lemma PS7.3.4. Thus $w \in \Pi_{\mathsf{A}}$ by Lemma PS7.3.2(6).
  - Suppose $j < n$. Thus $n = j + m$ for some $m \in \mathbb{N} - \{0\}$. Hence $w = \mathsf{0}^i\mathsf{1}^j\mathsf{2}^k\mathsf{3}^{i+j+m} = (\mathsf{0}^i(\mathsf{1}^j\mathsf{2}^k\mathsf{3}^j)\mathsf{3}^i)\mathsf{3}^m = (\mathsf{0}^i(\mathsf{1}^j\mathsf{1}^0\mathsf{2}^k\mathsf{3}^j)\mathsf{3}^i)\mathsf{3}^m$. Since $0 \leq k$, we have that $\mathsf{1}^j\mathsf{1}^0\mathsf{2}^k\mathsf{3}^j \in Y \subseteq \Pi_{\mathsf{B}}$, by Lemma PS7.3.4. By Lemma PS7.3.2(6), we have that $\mathsf{0}^i(\mathsf{1}^j\mathsf{1}^0\mathsf{2}^k\mathsf{3}^j)\mathsf{3}^i \in \Pi_{\mathsf{A}}$. Thus $w = (\mathsf{0}^i(\mathsf{1}^j\mathsf{1}^0\mathsf{2}^k\mathsf{3}^j)\mathsf{3}^i)\mathsf{3}^m \in \Pi_{\mathsf{A}}$, by Lemma PS7.3.2(5).
- Suppose $l < i$. Thus $i = l + n$ for some $n \in \mathbb{N} - \{0\}$. Hence $w = \mathsf{0}^{l+n}\mathsf{1}^j\mathsf{2}^k\mathsf{3}^l$. Since $l + n + j = i + j \leq k + l$, it follows that $n + j \leq k$, so that $k = n + j + m$ for some $m \in \mathbb{N}$. Thus $w = \mathsf{0}^{l+n}\mathsf{1}^j\mathsf{2}^{n+j+m}\mathsf{3}^l = \mathsf{0}^l(\mathsf{0}^n\mathsf{1}^j\mathsf{2}^{j+m}\mathsf{2}^n)\mathsf{3}^l$. Since $j \leq j + m$, we have that $\mathsf{0}^n\mathsf{1}^j\mathsf{2}^{j+m}\mathsf{2}^n \in Z \subseteq \Pi_{\mathsf{C}}$, by Lemma PS7.3.5. Thus $w = \mathsf{0}^l(\mathsf{0}^n\mathsf{1}^j\mathsf{2}^{j+m}\mathsf{2}^n)\mathsf{3}^l \in \Pi_{\mathsf{A}}$, by Lemma PS7.3.2(7).

$\square$

By Lemmas PS7.3.1, PS7.3.3, PS7.3.4, PS7.3.5 and PS7.3.6, we have that $L(G) = \Pi_{\mathsf{A}} = X$, $\Pi_{\mathsf{B}} = Y$, $\Pi_{\mathsf{C}} = Z$ and $\Pi_{\mathsf{D}} = W$.

Problem 4

First we load the grammar

```
{variables} A, B, C, D {start variable} A
{productions}
A -> 0A3 | A3 | B | C;
B -> 1B3 | D;
C -> 0C2 | D;
D -> % | 1D2 | D2
```

of Problem 3 (generating the language X) into Forlan, calling it `old`:

```
- val old = Gram.input "ps7-p3-gram";
val old = - : gram
```

Next, we load our Forlan/SML code `ps7-p4-gen.sml`

```
val minAndRen = DFA.renameStatesCanonically o DFA.minimize;
val regToDFA  = nfaToDFA o efaToNFA o faToEFA o regToFA;

fun elimVars(gram, nil)     = gram
  | elimVars(gram, q :: qs) =
      elimVars(Gram.eliminateVariable(gram, Sym.fromString q), qs);

(* DFA accepting all elements of {0, 1, 2, 3}^* of even length *)

val evenLenDFA =
      minAndRen(regToDFA(Reg.fromString "((0 + 1 + 2 + 3)(0 + 1 + 2 + 3))*"));

(* initial grammar generating Y *)

val new0 =
      Gram.restart
      (Gram.renameVariablesCanonically(Gram.minus(old, evenLenDFA)));

(* better grammar generating Y, resulting from variable elimination *)

val new1 = elimVars(new0, ["Q", "O", "J", "L", "F", "H", "C", "E"]);

(* renaming of variables so as to make the symmetry clear: <A>/A,
   <B>/B, <C>/C, <D>/D *)

val new =
      Gram.renameVariables
      (new1,
       SymRel.fromString
       ("(D, <A>), (B, A)," ^
        "(G, <B>), (I, B)," ^
        "(K, <C>), (M, C)," ^
        "(P, <D>), (N, D)"));
```

for generating a grammar `new` generating Y into Forlan:

```
- use "ps7-p4-gen.sml";
[opening ps7-p4-gen.sml]
val minAndRen = fn : dfa -> dfa
val regToDFA = fn : reg -> dfa
val elimVars = fn : gram * string list -> gram
val evenLenDFA = - : dfa
val new0 = - : gram
val new1 = - : gram
val new = - : gram
val it = () : unit
```

And then we output `new`:

```
- Gram.output("", new);
{variables} A, B, C, D, <A>, <B>, <C>, <D> {start variable} <A>
{productions}
A -> D | <B>3 | <C>3 | 0B3 | 0C2 | 0C3 | 1B3 | A33 | 0<A>33 | 00A33;
B -> % | <D>2 | 1D2 | 1D3 | 11B33; C -> % | <D>2 | 0D2 | 1D2 | 00C22;
D -> % | 12 | D22 | 1<D>22 | 11D22;
<A> -> <D> | B3 | C3 | 0<B>3 | 0<C>2 | 0<C>3 | 1<B>3 | <A>33 | 0A33 | 00<A>33;
<B> -> D2 | 1<D>2 | 1<D>3 | 11<B>33; <C> -> D2 | 0<D>2 | 1<D>2 | 00<C>22;
<D> -> 2 | <D>22 | 1D22 | 11<D>22
val it = () : unit
```

When producing this grammar, we renamed the variables so as to emphasize the connection between pairs of variables: $\langle A\rangle$ (the start variable) and A; $\langle B\rangle$ and B; $\langle C\rangle$ and C; and $\langle D\rangle$ and D.

We can make an educated guess as to what the languages generated by these variables are. To confirm our guess we wrote the Forlan/SML code `ps7-p4-testing.sml`

```
(* val inOrder : sym list -> bool

   inOrder x tests whether an element of {0, 1, 2, 3}^* is in
   {0}^*{1}^*{2}^*{3}^* *)

fun inOrder (b :: c :: ds) =
      Sym.compare(b, c) <> GREATER andalso
      inOrder(c :: ds)
  | inOrder _              = true;

(* val count : sym * sym list -> int

   count(a, x) counts the number of occurrences of a in x *)

fun count(_, nil)     = 0
  | count(a, b :: bs) =
      (if Sym.equal(a, b) then 1 else 0) + count(a, bs);

(* val inLan : (int * int * int * int -> bool) -> str -> bool

   inLan f x tests whether x is in {0}^*{1}^*{2}^*{3}^* and f(i, j, k,
   l) holds, where i, j, k and l, respectively, are the numbers of 0s,
   1s, 2s and 3s, respectively, in x *)

fun inLan (f : int * int * int * int -> bool) (x : str) =
      inOrder x andalso
      let val i = count(Sym.fromString "0", x)
          val j = count(Sym.fromString "1", x)
          val k = count(Sym.fromString "2", x)
          val l = count(Sym.fromString "3", x)
      in f(i, j, k, l) end;

(* val even : int -> bool
```

```
   even n tests whether n is even *)

fun even (n : int) = n mod 2 = 0

(* val odd : int -> bool

   odd n tests whether n is odd *)

fun odd (n : int) = n mod 2 = 1

(* val inYgen : bool -> str -> bool *)

fun inYgen (b : bool) =
      inLan
      (fn (i, j, k, l) =>
            i + j <= k + l andalso
            ((if b then odd else even) (i + j + k + l)))

(* val inY     : str -> bool
   val inYeven : str -> bool

   inY tests for membership of Y
   inYeven tests for membership of Y, but where the length is even *)

val inY     = inYgen true
val inYeven = inYgen false

(* val in123gen : bool -> str -> bool *)

fun in123gen (b : bool) =
      inLan
      (fn (i, j, k, l) =>
            i = 0 andalso l <= j andalso j - l <= k andalso
            ((if b then odd else even) (j + k + l)))

(* val in123     : str -> bool
   val in123even : str -> bool

   in123 tests for membership in {1^n1^j2^k3^n | j <= k and n + j + k + n
   is odd};
   in123even tests for membership in {1^n1^j2^k3^n | j <= k and n + j + k + n
   is even} *)

val in123     = in123gen true
val in123even = in123gen false

(* val in012gen : bool -> str -> bool *)
```

```
fun in012gen (b : bool) =
      inLan
      (fn (i, j, k, l) =>
            l = 0 andalso i <= k andalso j <= k - i andalso
            ((if b then odd else even) (i + j + k)))

(* val in012     : str -> bool
   val in012even : str -> bool

   in012 tests for membership in {0^n1^j2^k2^n | j <= k and n + j + k + n
   is odd};
   in012even tests for membership in {0^n1^j2^k2^n | j <= k and n + j + k + n
   is even} *)

val in012     = in012gen true
val in012even = in012gen false

(* val in12gen : bool -> str -> bool *)

fun in12gen (b : bool) =
      inLan
      (fn (i, j, k, l) =>
            i = 0 andalso l = 0 andalso j <= k andalso
            ((if b then odd else even) (j + k)))

(* val in12     : str -> bool
   val in12even : str -> bool

   in12 tests for membership in {1^j2^k | j <= k and j + k is odd};
   in12even tests for membership in {1^j2^k | j <= k and j + k is even} *)

val in12     = in12gen true
val in12even = in12gen false

(* val upto : int -> str set

   if n >= 0, then upto n returns all strings over alphabet {0, 1, 2,
   3} of length no more than n *)

fun upto 0 : str set = Set.sing nil
  | upto n           =
      let val xs = upto(n - 1)
          val ys = Set.filter (fn x => length x = n - 1) xs
      in StrSet.union
         (xs, StrSet.concat(StrSet.fromString "0, 1, 2, 3", ys))
      end;
```

```
(* val partition : int -> (str -> bool) -> str set * str set

   if n >= 0, then partition n p returns (xs, ys) where:

   xs is all elements of upto n that are satisfied by p; and

   ys is all elements of upto n that are not satisfied by p *)

fun partition n (p : str -> bool) = Set.partition p (upto n);

(* val test : int -> (str -> bool) -> gram -> str option * str option

   if n >= 0, then test n p returns a function f such that, for all
   grammars gram, f gram returns a pair (xOpt, yOpt) such that:

     If there is an element of {0, 1, 2, 3}* of length no more than n
     that is satisfied by p but is not generated by gram, then xOpt =
     SOME x for some such x; otherwise, xOpt = NONE.

     If there is an element of {0, 1, 2, 3}* of length no more than n
     that is not satisfied by p but is generated by gram, then yOpt =
     SOME y for some such y; otherwise, yOpt = NONE. *)

fun test n (p : str -> bool) =
      let val (goods, bads) = partition n p
      in fn gram =>
              let val generated     = Gram.generated gram
                  val goodNotGenOpt = Set.position (not o generated) goods
                  val badGenOpt     = Set.position generated bads
              in ((case goodNotGenOpt of
                         NONE   => NONE
                       | SOME i => SOME(ListAux.sub(Set.toList goods, i))),
                  (case badGenOpt of
                         NONE   => NONE
                       | SOME i => SOME(ListAux.sub(Set.toList bads, i))))
              end
      end;

(* val changeStartVariable : gram * sym -> gram

   if q is a variable of gram, then changeStartVariable(gram, q)
   returns the simplification of the grammar formed by changing gram's
   start variables to be q; otherwise, it raises an exception *)

fun changeStartVariable(gram, q) =
      let val {vars, start, prods} = Gram.toConcr gram
      in if SymSet.memb(q, vars)
         then Gram.simplify
```

```
              (Gram.fromConcr{vars = vars, start = q, prods = prods})
         else raise Fail "symbol must be variable of grammar"
      end;

(* doit : int -> (str -> bool) -> gram -> sym -> str option * str option *)

fun doit n p gram q = test n p (changeStartVariable(gram, q));
```

which we now load into Forlan:

```
- use "ps7-p4-testing.sml";
[opening ps7-p4-testing.sml]
val inOrder = fn : sym list -> bool
val count = fn : sym * sym list -> int
val inLan = fn : (int * int * int * int -> bool) -> str -> bool
val even = fn : int -> bool
val odd = fn : int -> bool
val inYgen = fn : bool -> str -> bool
val inY = fn : str -> bool
val inYeven = fn : str -> bool
val in123gen = fn : bool -> str -> bool
val in123 = fn : str -> bool
val in123even = fn : str -> bool
val in012gen = fn : bool -> str -> bool
val in012 = fn : str -> bool
val in012even = fn : str -> bool
val in12gen = fn : bool -> str -> bool
val in12 = fn : str -> bool
val in12even = fn : str -> bool
val upto = fn : int -> str set
val partition = fn : int -> (str -> bool) -> str set * str set
val test = fn : int -> (str -> bool) -> gram -> str option * str option
val changeStartVariable = fn : gram * sym -> gram
val doit = fn : int -> (str -> bool) -> gram -> sym -> str option * str option
val it = () : unit
```

We then use the function `doit` to verify the connections between the variables of `new` and their languages on all strings over the alphabet $\{0,1,2,3\}^*$ of length no more than 9:

```
- doit 9 inY new (Sym.fromString "<A>");
val it = (NONE,NONE) : str option * str option
- doit 9 inYeven new (Sym.fromString "A");
val it = (NONE,NONE) : str option * str option
- doit 9 in123 new (Sym.fromString "<B>");
val it = (NONE,NONE) : str option * str option
- doit 9 in123even new (Sym.fromString "B");
val it = (NONE,NONE) : str option * str option
- doit 9 in012 new (Sym.fromString "<C>");
val it = (NONE,NONE) : str option * str option
- doit 9 in012even new (Sym.fromString "C");
```

```
val it = (NONE,NONE) : str option * str option
- doit 9 in12 new (Sym.fromString "<D>");
val it = (NONE,NONE) : str option * str option
- doit 9 in12even new (Sym.fromString "D");
val it = (NONE,NONE) : str option * str option
```

Working outside of Forlan, we then formulate the grammar

```
{variables} <A>, <B>, <C>, <D>, A, B, C, D
{start variable} <A>
{productions}
<A> -> 0<A>3 |  A 3 | <B> | <C>;
 A  -> 0 A 3 | <A>3 |  B  |  C;
<B> -> 1<B>3 | <D>;
 B  -> 1 B 3 |  D;
<C> -> 0<C>2 | <D>;
 C  -> 0 C 2 |  D;
<D> -> 2 | 1<D>2 |  D 2;
 D  -> % | 1 D 2 | <D>2
```

that is inspired by `new`, and which we put in the file `ps7-p4-gram`. We load this grammar into Forlan, calling it `final`:

```
- val final = Gram.input "ps7-p4-gram";
val final = - : gram
```

Finally, we check that its variables generate the same languages as the variables of `new`, when we restrict our attention to strings over the alphabet $\{0, 1, 2, 3\}^*$ of length no more than 9:

```
- doit 9 inY final (Sym.fromString "<A>");
val it = (NONE,NONE) : str option * str option
- doit 9 inYeven final (Sym.fromString "A");
val it = (NONE,NONE) : str option * str option
- doit 9 in123 final (Sym.fromString "<B>");
val it = (NONE,NONE) : str option * str option
- doit 9 in123even final (Sym.fromString "B");
val it = (NONE,NONE) : str option * str option
- doit 9 in012 final (Sym.fromString "<C>");
val it = (NONE,NONE) : str option * str option
- doit 9 in012even final (Sym.fromString "C");
val it = (NONE,NONE) : str option * str option
- doit 9 in12 final (Sym.fromString "<D>");
val it = (NONE,NONE) : str option * str option
- doit 9 in12even final (Sym.fromString "D");
val it = (NONE,NONE) : str option * str option
```