CS 591 S2—Formal Language Theory: Integrating Experimentation and Proof—Fall 2018

Problem Set 7

Model Answers

Problem 1

First, we put the Forlan code

```
val regToGram' = faToGram o regToFA;
val faToGram'  = regToGram o faToReg Reg.weaklySimplify;
```

in the file `ps7-p1-convert.sml`. Then we load `ps7-p1-convert.sml` into Forlan:

```
- use "ps7-p1-convert.sml";
[opening ps7-p1-convert.sml]
val regToGram' = fn : reg -> gram
val faToGram' = fn : fa -> gram
val it = () : unit
```

To compare `regToGram'` with `regToGram`, we proceed as follows:

```
- val reg = Reg.fromString "(01*)*0";
val reg = - : reg
- val gram1 = Gram.renameVariablesCanonically(regToGram reg);
val gram1 = - : gram
- val gram1' = Gram.renameVariablesCanonically(regToGram' reg);
val gram1' = - : gram
- Gram.output("", gram1);
{variables} A, B, C, D, E, F, G {start variable} A
{productions} A -> BC; B -> % | DB; C -> 0; D -> EF; E -> 0; F -> % | GF; G -> 1
val it = () : unit
- Gram.output("", gram1');
{variables} A, B, C, D, E, F, G, H {start variable} A
{productions}
A -> B | D; B -> 0C; C -> %; D -> 0E; E -> F; F -> A | G; G -> 1H; H -> F
val it = () : unit
```

To compare `faToGram'` with `faToGram`, we put the text

```
{states} A, B {start state} A {accepting states} B
{transitions} A, 0 -> B; A, 1 -> A; B, 0 -> A; B, 1 -> B
```

in the file `ps7-p1-fa.txt`, and the proceed as follows:

```
- val fa = FA.input "ps7-p1-fa.txt";
val fa = - : fa
- val gram2 = Gram.renameVariablesCanonically(faToGram fa);
```

```
val gram2 = - : gram
- val gram2' = Gram.renameVariablesCanonically(faToGram' fa);
val gram2' = - : gram
- Gram.output("", gram2);
{variables} A, B {start variable} A {productions} A -> 0B | 1A; B -> % | 0A | 1B
val it = () : unit
- Gram.output("", gram2');
{variables} A, B, C, D, E, F, G, H, I, J, K, L, M, N {start variable} A
{productions}
A -> BC; B -> % | DB; C -> EF; D -> 1; E -> 0; F -> % | GF; G -> H | I; H -> 1;
I -> JK; J -> 0; K -> LM; L -> % | NL; M -> 0; N -> 1
val it = () : unit
```

Problem 2

Suppose, toward a contradiction, that X is context-free. Thus there is an $n \in \mathbb{N}$ with the property of the Pumping Lemma for Context-free Languages, where X has been substituted for L. Let $z = \mathsf{0}^n\mathsf{2}^n\mathsf{1}^n\mathsf{3}^n$. Since $z \in \Sigma^*$ and $\mathbf{zeros}\, z = n = \mathbf{ones}\, z$ and $\mathbf{twos}\, z = n = \mathbf{threes}\, z$, we have that $z \in X$. Because $z \in X$ and $|z| = 4n \geq n$, the property of the lemma tells us there are $u, v, w, x, y \in \mathbf{Str}$ 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 $\mathsf{0}^n\mathsf{2}^n\mathsf{1}^n\mathsf{3}^n = z = uvwxy$, (1) tells us that vwx does not have both $\mathsf{0}$'s and $\mathsf{1}$'s, and vwx does not have both $\mathsf{2}$'s and $\mathsf{3}$'s. Thus vx does not have both $\mathsf{0}$'s and $\mathsf{1}$'s, and vx does not have both $\mathsf{2}$'s and $\mathsf{3}$'s. By (2), it will suffice to consider the following cases:

($\mathbf{zeros}(vx) \geq 1$) Thus $\mathbf{ones}(vx) = 0$, so that $\mathbf{ones}(uwy) = \mathbf{ones}(uvwxy) - \mathbf{ones}(vx) = \mathbf{ones}\, z - 0 = n - 0 = n$. And $\mathbf{zeros}(uwy) = \mathbf{zeros}(uvwxy) - \mathbf{zeros}(vx) = \mathbf{zeros}\, z - \mathbf{zeros}(vx) = n - \mathbf{zeros}(vx) < n$. But (3) tells us that $uwy = uv^0wx^0y \in X$, so that $n > \mathbf{zeros}(uwy) = \mathbf{ones}(uwy) = n$—contradiction.

($\mathbf{ones}(vx) \geq 1$) Thus $\mathbf{zeros}(vx) = 0$, so that $\mathbf{zeros}(uwy) = \mathbf{zeros}(uvwxy) - \mathbf{zeros}(vx) = \mathbf{zeros}\, z - 0 = n - 0 = n$. And $\mathbf{ones}(uwy) = \mathbf{ones}(uvwxy) - \mathbf{ones}(vx) = \mathbf{ones}\, z - \mathbf{ones}(vx) = n - \mathbf{ones}(vx) < n$. But (3) tells us that $uwy = uv^0wx^0y \in X$, so that $n > \mathbf{ones}(uwy) = \mathbf{zeros}(uwy) = n$—contradiction.

($\mathbf{twos}(vx) \geq 1$) Thus $\mathbf{threes}(vx) = 0$, so that $\mathbf{threes}(uwy) = \mathbf{threes}(uvwxy) - \mathbf{threes}(vx) = \mathbf{threes}\, z - 0 = n - 0 = n$. And $\mathbf{twos}(uwy) = \mathbf{twos}(uvwxy) - \mathbf{twos}(vx) = \mathbf{twos}\, z - \mathbf{twos}(vx) = n - \mathbf{twos}(vx) < n$. But (3) tells us that $uwy = uv^0wx^0y \in X$, so that $n > \mathbf{twos}(uwy) = \mathbf{threes}(uwy) = n$—contradiction.

($\mathbf{threes}(vx) \geq 1$) Thus $\mathbf{twos}(vx) = 0$, so that $\mathbf{twos}(uwy) = \mathbf{twos}(uvwxy) - \mathbf{twos}(vx) = \mathbf{twos}\, z - 0 = n - 0 = n$. And $\mathbf{threes}(uwy) = \mathbf{threes}(uvwxy) - \mathbf{threes}(vx) = \mathbf{threes}\, z - \mathbf{threes}(vx) =$

$n - \mathbf{threes}(vx) < n$. But (3) tells us that $uwy = uv^0wx^0y \in X$, so that $n > \mathbf{threes}(uwy) = \mathbf{twos}(uwy) = n$—contradiction.

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

Problem 3

(a) First, we put the text

```
{variables} A, B, <1>, <3> {start variable} A
{productions}
A -> 0A3 | B<3>;
B -> 1B2 | <1>;
<1> -> 1 | 1<1>;
<3> -> 3 | 3<3>
```

for a grammar generating $\{\, 0^i1^j2^k3^l \mid i,j,k,l \in \mathbb{N} \text{ and } i < l \text{ and } j > k \,\}$ in the file `ps7-p3-orig-gram.txt`. Next we put the text

```
{states} A, B {start state} A {accepting states} A
{transitions}
A, 0 -> B; A, 1 -> B; A, 2 -> A; A, 3 -> A;
B, 0 -> A; B, 1 -> A; B, 2 -> B; B, 3 -> B
```

for a DFA accepting all elements of $\{0,1,2,3\}^*$ in which the sum of the numbers of 0's and 1's is even in the file `ps7-p3-even-01-dfa.txt`. Then we load the grammar and DFA into Forlan:

```
- val origGram = Gram.input "ps7-p3-orig-gram.txt";
val origGram = - : gram
- val even01DFA = DFA.input "ps7-p3-even-01-dfa.txt";
val even01DFA = - : dfa
```

Next, we put the Forlan program

```
(* standard definitions *)

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

(* the alphabet {0, 1, 2, 3} *)

val syms0123 = SymSet.fromString "0, 1, 2, 3";

(* regular expression and DFA generating/accepting {0, 1, 2, 3}* *)

val allStrReg = Reg.closure(Reg.allSym syms0123);
val allStrDFA = minAndRen(regToDFA allStrReg);

(* symbolic relation swapping 2/3 for 0/1 *)
```

```
val swap23for01 = SymRel.fromString "(0, 2), (1, 3), (2, 0), (3, 1)";

(* DFA accepting all elements of {0, 1, 2, 3}* in which the sum of
   the numbers of 2s and 3s is odd *)

val odd23DFA =
      minAndRen
      (DFA.minus
       (allStrDFA,
        DFA.renameAlphabet(even01DFA, swap23for01)));

(* DFA accepting all elements of {0, 1, 2, 3}* in which sum of the
   numbers of 0s and 1s is even, and sum of the numbers of 2s and 3s
   is odd *)

val paritiesDFA = minAndRen(DFA.inter(even01DFA, odd23DFA));

(* grammar generating X *)

val gram0 =
      Gram.renameVariablesCanonically
      (Gram.inter(origGram, injDFAToEFA paritiesDFA));
```

in the file `ps7-p3-find.sml`, and proceed as follows:

```
- use "ps7-p3-find.sml";
[opening ps7-p3-find.sml]
val regToDFA = fn : reg -> dfa
val minAndRen = fn : dfa -> dfa
val syms0123 = - : sym set
val allStrReg = - : reg
val allStrDFA = - : dfa
val swap23for01 = - : sym_rel
val odd23DFA = - : dfa
val paritiesDFA = - : dfa
val gram0 = - : gram
val it = () : unit
- Gram.output("", gram0);
{variables} A, B, C, D, E, F, G, H, I, J, K, L, M {start variable} A
{productions}
A -> B; B -> DK | EM | 0C3; C -> FJ | GL | 0B3; D -> H | 1G2; E -> 1F2;
F -> I | 1E2; G -> 1D2; H -> 1I; I -> 1 | 1H; J -> 3L; K -> 3 | 3M; L -> 3 | 3J;
M -> 3K
val it = () : unit
- Gram.numVariables gram0;
val it = 13 : int
- Gram.numProductions gram0;
val it = 22 : int
```

In the grammar `gram0`, there are opportunities for hand-simplification using Forlan. We put the text

```
fun elimVars(gram, nil)     = gram
  | elimVars(gram, x :: xs) =
      elimVars(Gram.eliminateVariable(gram, Sym.fromString x), xs);

(* hand-simplified grammar generating X *)

val gram1 =
      Gram.renameVariablesCanonically
      (elimVars(Gram.restart gram0, ["E", "G", "H", "J", "M"]));
```

in the file `ps7-p3-elim.sml`, and proceed as follows:

```
- use "ps7-p3-elim.sml";
[opening ps7-p3-elim.sml]
val elimVars = fn : gram * string list -> gram
val gram1 = - : gram
val it = () : unit
- Gram.output("", gram1);
{variables} A, B, C, D, E, F, G {start variable} A
{productions}
A -> CF | 0B3 | 1D23F; B -> 0A3 | D3G | 1C2G; C -> 1E | 11C22; D -> E | 11D22;
E -> 1 | 11E; F -> 3 | 33F; G -> 3 | 33G
val it = () : unit
- Gram.numVariables gram1;
val it = 7 : int
- Gram.numProductions gram1;
val it = 16 : int
```

We finish up by making some final improvements to `gram1`, working outside of Forlan. First, note that in `gram1`, F and its productions, and G and its productions, have the same form. There is no reason to have both of them, and so we can remove G and its productions, replacing all occurrences of G in the remaining productions by F. This gives us the grammar:

$$\begin{aligned}
&\mathsf{A} \to \mathsf{CF} \mid \mathsf{0B3} \mid \mathsf{1D23F},\\
&\mathsf{B} \to \mathsf{0A3} \mid \mathsf{D3F} \mid \mathsf{1C2F},\\
&\mathsf{C} \to \mathsf{1E} \mid \mathsf{11C22},\\
&\mathsf{D} \to \mathsf{E} \mid \mathsf{11D22},\\
&\mathsf{E} \to \mathsf{1} \mid \mathsf{11E},\\
&\mathsf{F} \to \mathsf{3} \mid \mathsf{33F}.
\end{aligned}$$

Next, we note that

$$\begin{aligned}
\Pi_{\mathsf{E}} &= \{\, 1^m \mid m \in \mathbb{N} \text{ and } m \text{ is odd} \,\},\\
\Pi_{\mathsf{D}} &= \{\, 1^{2n}1^m2^{2n} \mid n, m \in \mathbb{N} \text{ and } m \text{ is odd} \,\},\\
\Pi_{\mathsf{C}} &= \{\, 1^{2n}11^m2^{2n} \mid n, m \in \mathbb{N} \text{ and } m \text{ is odd} \,\} = \{\, 11^{2n}1^m2^{2n} \mid n, m \in \mathbb{N} \text{ and } m \text{ is odd} \,\} = \{1\}\Pi_{\mathsf{D}}.
\end{aligned}$$

Thus we can remove C and its productions, replacing all occurrences of C by 1D:

$$\begin{aligned}
&\mathsf{A} \to \mathsf{1DF} \mid \mathsf{0B3} \mid \mathsf{1D23F},\\
&\mathsf{B} \to \mathsf{0A3} \mid \mathsf{D3F} \mid \mathsf{11D2F},\\
&\mathsf{D} \to \mathsf{E} \mid \mathsf{11D22},\\
&\mathsf{E} \to \mathsf{1} \mid \mathsf{11E},\\
&\mathsf{F} \to \mathsf{3} \mid \mathsf{33F}.
\end{aligned}$$

Since E is only used in a production of D, we can combine the productions of D and E, yielding

$$\mathsf{D} \to \mathsf{1} \mid \mathsf{11D} \mid \mathsf{11D22}.$$

Then, because B's productions don't involve self-recursion, we can get rid of B and its productions, by substituting its right sides into the only production using B, $\mathsf{A} \to \mathsf{0B3}$, obtaining:

$$\begin{aligned}
&\mathsf{A} \to \mathsf{1DF} \mid \mathsf{00A33} \mid \mathsf{0D3F3} \mid \mathsf{011D2F3} \mid \mathsf{1D23F},\\
&\mathsf{D} \to \mathsf{1} \mid \mathsf{11D} \mid \mathsf{11D22},\\
&\mathsf{F} \to \mathsf{3} \mid \mathsf{33F}.
\end{aligned}$$

And, because $\Pi_{\mathsf{F}} = \{\, 3^m \mid m \in \mathbb{N} \text{ and } m \text{ is odd} \,\}$, we can replace F3 by 3F in the right sides of productions, yielding:

$$\begin{aligned}
&\mathsf{A} \to \mathsf{1DF} \mid \mathsf{00A33} \mid \mathsf{0D33F} \mid \mathsf{011D23F} \mid \mathsf{1D23F},\\
&\mathsf{D} \to \mathsf{1} \mid \mathsf{11D} \mid \mathsf{11D22},\\
&\mathsf{F} \to \mathsf{3} \mid \mathsf{33F}.
\end{aligned}$$

To rename the variables of this grammar canonically, we can return to Forlan:

```
- val gram = Gram.renameVariablesCanonically(Gram.input "");
@ {variables} A, D, F {start variable} A
@ {productions}
@ A -> 1DF | 00A33 | 0D33F | 011D23F | 1D23F;
@ D -> 1 | 11D | 11D22;
@ F -> 3 | 33F
@ .
val gram = - : gram
- Gram.output("", gram);
{variables} A, B, C {start variable} A
{productions}
A -> 1BC | 00A33 | 0B33C | 1B23C | 011B23C; B -> 1 | 11B | 11B22; C -> 3 | 33C
val it = () : unit
- Gram.numVariables gram;
val it = 3 : int
- Gram.numProductions gram;
val it = 10 : int
```

The grammar `gram` is our answer, but `gram0` will be useful in part (b).

(b) We put the Forlan program

```
(* regular expression generating all elements of {0, 1, 2, 3}* of
   given length *)

fun lenReg n = Reg.power(Reg.allSym syms0123, n);

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

val lenDFA = minAndRen o regToDFA o lenReg;

(* genLen n returns all elements of X of length n

   note the use of gram0, instead of gram *)

fun genLen n =
      let val interGram =
                Gram.renameVariablesCanonically
                (Gram.inter(gram0, injDFAToEFA(lenDFA n)))
      in Gram.toStrSet interGram end;
```

in the file `ps7-p3-gen-len.sml`. We use `gram0` rather than `gram` in the function `genLen`, because its productions have shorter right sides, and so `genLen`'s use of `Gram.inter` runs much faster. We then continue the Forlan session of (a) as follows:

```
- use "ps7-p3-gen-len.sml";
[opening ps7-p3-gen-len.sml]
val lenReg = fn : int -> reg
val lenDFA = fn : int -> dfa
val genLen = fn : int -> str set
val it = () : unit
- val xs = genLen 25;
val xs = - : str set
- Set.size xs;
val it = 364 : int
- StrSet.output("", xs);
0000000000013333333333333, 0000000000111133333333333, 0000000000112333333333333,
0000000000113333333333333, 0000000001111123333333333, 0000000001111133333333333,
0000000001112233333333333, 0000000001112333333333333, 0000000001113333333333333,
0000000001333333333333333, 0000000011111111333333333, 0000000011111122333333333,
0000000011111123333333333, 0000000011111133333333333, 0000000011112223333333333,
0000000011112233333333333, 0000000011112333333333333, 0000000011113333333333333,
0000000011233333333333333, 0000000011333333333333333, 0000000111111111233333333,
0000000111111111333333333, 0000000111111122233333333, 0000000111111122333333333,
0000000111111123333333333, 0000000111111133333333333, 0000000111112222333333333,
0000000111112223333333333, 0000000111112233333333333, 0000000111112333333333333,
0000000111113333333333333, 0000000111223333333333333, 0000000112333333333333333,
0000000113333333333333333, 0000000133333333333333333, 0000001111111111113333333,
0000001111111111223333333, 0000001111111111233333333, 0000001111111111333333333,
0000001111111122223333333, 0000001111111122233333333, 0000001111111122333333333,
0000001111111123333333333, 0000001111111133333333333, 0000001111112222233333333,
```

000000111111222233333333, *000000111111222333333333*, *000000111111223333333333*,
000000111111233333333333, *000000111111333333333333*, *000000111122233333333333*,
000000111223333333333333, *000000111233333333333333*, *000000111333333333333333*,
000000123333333333333333, *000000133333333333333333*, *000001111111111112333333*,
000001111111111113333333, *000001111111111222333333*, *000001111111111223333333*,
000001111111111233333333, *000001111111111333333333*, *000001111111122222333333*,
000001111111122223333333, *000001111111122233333333*, *000001111111122333333333*,
000001111111123333333333, *000001111111133333333333*, *000001111112222223333333*,
000001111112222233333333, *000001111112222333333333*, *000001111112223333333333*,
000001111112233333333333, *000001111112333333333333*, *000001111113333333333333*,
000001111222233333333333, *000001111222333333333333*, *000001111223333333333333*,
000001111233333333333333, *000001111333333333333333*, *000001122333333333333333*,
000001123333333333333333, *000001133333333333333333*, *000001333333333333333333*,
000011111111111111133333, *000011111111111112233333*, *000011111111111112333333*,
000011111111111113333333, *000011111111111222333333*, *000011111111111222333333*,
000011111111111223333333, *000011111111111233333333*, *000011111111111333333333*,
000011111111122222233333, *000011111111122222333333*, *000011111111122223333333*,
000011111111122233333333, *000011111111122333333333*, *000011111111123333333333*,
000011111111133333333333, *000011111112222222333333*, *000011111112222223333333*,
000011111112222233333333, *000011111112222333333333*, *000011111112223333333333*,
000011111112233333333333, *000011111113333333333333*, *000011111222222233333333*,
000011111222222333333333, *000011111222223333333333*, *000011111222233333333333*,
000011111223333333333333, *000011111233333333333333*, *000011111333333333333333*,
000011112222333333333333, *000011112233333333333333*, *000011112333333333333333*,
000011113333333333333333, *000011233333333333333333*, *000011333333333333333333*,
000111111111111111123333, *000111111111111111133333*, *000111111111111112223333*,
000111111111111112233333, *000111111111111112333333*, *000111111111111113333333*,
000111111111111222223333, *000111111111111222233333*, *000111111111111222333333*,
000111111111111223333333, *000111111111111233333333*, *000111111111111333333333*,
000111111111122222223333, *000111111111122222233333*, *000111111111122222333333*,
000111111111122223333333, *000111111111122233333333*, *000111111111122333333333*,
000111111111123333333333, *000111111111133333333333*, *000111111112222222233333*,
000111111112222222333333, *000111111112222223333333*, *000111111112222233333333*,
000111111112222333333333, *000111111112223333333333*, *000111111112233333333333*,
000111111112333333333333, *000111111113333333333333*, *000111112222222333333333*,
000111112222333333333333, *000111112223333333333333*, *000111112233333333333333*,
000111112333333333333333, *000111113333333333333333*, *000111222333333333333333*,
000111233333333333333333, *000111333333333333333333*, *000133333333333333333333*,
001111111111111111111333, *001111111111111111122333*, *001111111111111111123333*,
001111111111111111133333, *001111111111111112222333*, *001111111111111112223333*,
001111111111111112233333, *001111111111111112333333*, *001111111111111113333333*,
001111111111111222222333, *001111111111111222223333*, *001111111111111222233333*,
001111111111111222333333, *001111111111111223333333*, *001111111111111233333333*,
001111111111111333333333, *001111111111122222222333*, *001111111111122222223333*,
001111111111122222233333, *001111111111122222333333*, *001111111111122223333333*,
001111111111122233333333, *001111111111122333333333*, *001111111111123333333333*,
001111111111133333333333, *001111111112222222233333*, *001111111112222222333333*,
001111111112222223333333, *001111111112222233333333*, *001111111112222333333333*,
001111111112223333333333, *001111111112233333333333*, *001111111112333333333333*,
001111111113333333333333, *001111111222222233333333*, *001111111222222333333333*,
001111111222223333333333, *001111111222233333333333*, *001111111222333333333333*,
001111111223333333333333, *001111111233333333333333*, *001111111333333333333333*,
001111112222333333333333, *001111112223333333333333*, *001111112233333333333333*,
001111112333333333333333, *001111113333333333333333*, *001111222333333333333333*,
001111233333333333333333, *001111333333333333333333*, *001133333333333333333333*,
001111111111111111111333, *001111111111111111122333*, *001111111111111111123333*,
001111111111111111133333, *001111111111111112222333*, *001111111111111112223333*,
001111111111111112233333, *001111111111111112333333*, *001111111111111113333333*,
001111111111111222222333, *001111111111111222223333*, *001111111111111222233333*,
001111111111111222333333, *001111111111111223333333*, *001111111111111233333333*,
001111111111111333333333, *001111111111122222222333*, *001111111111122222223333*,
001111111111122222233333, *001111111111122222333333*, *001111111111122223333333*,
001111111111122233333333, *001111111111122333333333*, *001111111111123333333333*,

00111111111111133333333333, 00111111111112222222223333, 00111111111112222222233333,
00111111111112222222333333, 00111111111112222223333333, 00111111111112222233333333,
00111111111112222233333333, 00111111111112223333333333, 00111111111112233333333333,
00111111111112333333333333, 00111111111113333333333333, 00111111112222222233333333,
00111111112222222333333333, 00111111112222223333333333, 00111111112222233333333333,
00111111112222333333333333, 00111111112223333333333333, 00111111112233333333333333,
00111111113333333333333333, 00111111222222333333333333, 00111111222223333333333333,
00111111222233333333333333, 00111111222333333333333333, 00111111233333333333333333,
00111111333333333333333333, 00111112223333333333333333, 00111112233333333333333333,
00111112333333333333333333, 00111133333333333333333333, 00112333333333333333333333,
00113333333333333333333333, 01111111111111111111111233, 01111111111111111111111333,
01111111111111111111122233, 01111111111111111111122333, 01111111111111111111123333,
01111111111111111111133333, 01111111111111111112222233, 01111111111111111112222333,
01111111111111111112223333, 01111111111111111112233333, 01111111111111111112333333,
01111111111111111113333333, 01111111111111111222222233, 01111111111111111222222333,
01111111111111111222223333, 01111111111111111222233333, 01111111111111111222333333,
01111111111111111223333333, 01111111111111111233333333, 01111111111111111333333333,
01111111111111122222222233, 01111111111111122222222333, 01111111111111122222223333,
01111111111111122222233333, 01111111111111122222333333, 01111111111111122223333333,
01111111111111122233333333, 01111111111111122333333333, 01111111111111123333333333,
01111111111111133333333333, 01111111111112222222222333, 01111111111112222222223333,
01111111111112222222233333, 01111111111112222222333333, 01111111111112222223333333,
01111111111112222233333333, 01111111111112222333333333, 01111111111112223333333333,
01111111111112233333333333, 01111111111112333333333333, 01111111111113333333333333,
01111111112222222223333333, 01111111112222222233333333, 01111111112222222333333333,
01111111112222223333333333, 01111111112222233333333333, 01111111112222333333333333,
01111111112223333333333333, 01111111112233333333333333, 01111111113333333333333333,
01111111222222233333333333, 01111111222222333333333333, 01111111222223333333333333,
01111111222233333333333333, 01111111222333333333333333, 01111111233333333333333333,
01111112333333333333333333, 01111112333333333333333333, 01111113333333333333333333,
01112333333333333333333333, 01112333333333333333333333, 01113333333333333333333333,
01333333333333333333333333, 11111111111111111111111113, 11111111111111111111111223,
11111111111111111111111233, 11111111111111111111111333, 11111111111111111111122223,
11111111111111111111122233, 11111111111111111111122333, 11111111111111111111123333,
11111111111111111111133333, 11111111111111111112222223, 11111111111111111112222233,
11111111111111111112222333, 11111111111111111112223333, 11111111111111111112233333,
11111111111111111112333333, 11111111111111111113333333, 11111111111111111222222223,
11111111111111111222222233, 11111111111111111222222333, 11111111111111111222223333,
11111111111111111222233333, 11111111111111111222333333, 11111111111111111223333333,
11111111111111111233333333, 11111111111111111333333333, 11111111111111122222222223,
11111111111111122222222233, 11111111111111122222222333, 11111111111111122222223333,
11111111111111122222233333, 11111111111111122222333333, 11111111111111122223333333,
11111111111111122233333333, 11111111111111122333333333, 11111111111111123333333333,
11111111111111133333333333, 11111111111112222222222233, 11111111111112222222222333,
11111111111112222222223333, 11111111111112222222233333, 11111111111112222223333333,
11111111111112222223333333, 11111111111112222233333333, 11111111111112222333333333,
11111111111112223333333333, 11111111111112233333333333, 11111111111112333333333333,

```
111111111113333333333333, 111111111222222222333333, 111111111222222223333333,
111111111222222233333333, 111111111222222233333333, 111111111222223333333333,
111111111222223333333333, 111111111223333333333333, 111111111233333333333333,
111111111333333333333333, 111111112222222233333333, 111111112222223333333333,
111111112222223333333333, 111111112223333333333333, 111111112233333333333333,
111111113333333333333333, 111111222223333333333333, 111111222233333333333333,
111111222333333333333333, 111111223333333333333333, 111111233333333333333333,
111111333333333333333333, 111122233333333333333333, 111223333333333333333333,
111233333333333333333333, 111333333333333333333333, 112333333333333333333333,
113333333333333333333333
val it = () : unit
```

On my laptop, it took about five minutes for `genLen` to return. I don't know how long the computation using `gram` would have taken—maybe too long to get an answer?