CS 516—Software Foundations via Formal Languages—Spring 2022

Problem Set 6

Model Answers

Problem 1

Suppose, toward a contradiction, that X is regular. Thus there is an $n \in \mathbb{N} - \{0\}$ with the property of the Pumping Lemma, where X has been substituted for L. Suppose $z = \mathsf{0}^n\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n$. Since $n + n \leq n + n$, we have that $z \in X$. Thus, since $|z| = 4n \geq n$, it follows there are $u, v, w \in \mathbf{Str}$ such that $z = uvw$ and properties (1)–(3) of the lemma hold. Since $uvw = z = \mathsf{0}^n\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n$, (1) tells us that there are $i, j, k \in \mathbb{N}$ such that

$$u = \mathsf{0}^i, \quad v = \mathsf{0}^j, \quad w = \mathsf{0}^k\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n, \quad i + j + k = n.$$

By (2), we have that $j \geq 1$, and thus that $n + j > n$. By (3), we have that $\mathsf{0}^{i+j+j+k}\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n = \mathsf{0}^i\mathsf{0}^j\mathsf{0}^j\mathsf{0}^k\mathsf{1}^n\mathsf{2}^n\mathsf{3}^n = uvvw = uv^2w \in X$. Thus $(i+j+j+k)+n \leq n+n$, so that $n+j = i+j+k+j = i + j + j + k \leq n$. But $n + j \leq n$ contradicts $n + j > n$. Thus X is regular.

Problem 2

(a) G is

$$\begin{aligned}
&\mathsf{A} \to \mathsf{0A3} \mid \mathsf{A3} \mid \mathsf{B} \mid \mathsf{C}\\
&\mathsf{B} \to \mathsf{1B3} \mid \mathsf{D}\\
&\mathsf{C} \to \mathsf{0C2} \mid \mathsf{D}\\
&\mathsf{D} \to \% \mid \mathsf{1D2} \mid \mathsf{D2}
\end{aligned}$$

(b) We put the expression

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

of G in the file `ps6-p2-gram`, and load G into Forlan, calling it `gram`:

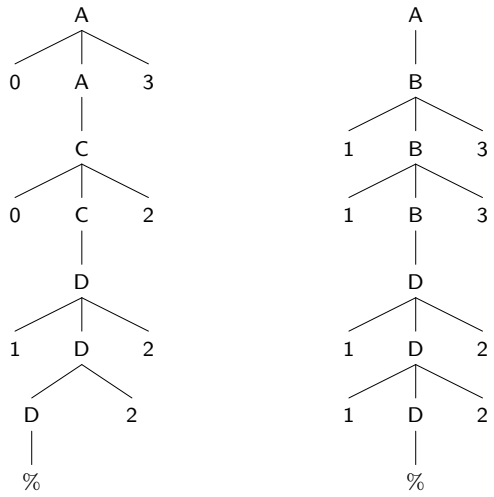
```
- val gram = Gram.input "ps6-p2-gram";
val gram = - : gram
```

Then we compute and display the alphabet of `gram`, find parse trees `pt1` and `pt2` showing why 0012223 and 11112233 are generated by `gram`, and then display those parse trees:

```
- SymSet.output("", Gram.alphabet gram);
```

```
0, 1, 2, 3
val it = () : unit
- val pt1 = Gram.parseAlphabet gram (Str.fromString "0012223");
val pt1 = - : pt
- val pt2 = Gram.parseAlphabet gram (Str.fromString "11112233");
val pt2 = - : pt
- PT.output("", pt1);
A(0, A(C(0, C(D(1, D(D(%), 2), 2)), 2)), 3)
val it = () : unit
- PT.output("", pt2);
A(B(1, B(1, B(D(1, D(1, D(%), 2), 2)), 3), 3))
val it = () : unit
```

Here are drawings of the two parse trees:



(pt_1) (pt_2)

(c) Continuing our Forlan session, we put our testing code

```
(* 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, bs) counts the number of occurrences of a in bs *)
```

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

(* val inX : str -> bool

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

fun inX (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 i + j <= k + l end;

(* 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 set * str set

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

   xs is all elements of upto n that are in X; and

   ys is all elements of upto n that are not in X *)

fun partition n = Set.partition inX (upto n);

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

   if n >= 0, then test n 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 in X 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 in X but is generated by gram, then yOpt = SOME y
      for some such y; otherwise, yOpt = NONE. *)

fun test n =
      let val (goods, bads) = partition n
      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;
```

in the file `ps6-p2-testing.sml`, and load it into Forlan:

```
- use "ps6-p2-testing.sml";
[opening ps6-p2-testing.sml]
val inOrder = fn : sym list -> bool
val count = fn : sym * sym list -> int
val inX = fn : str -> bool
val upto = fn : int -> str set
val partition = fn : int -> str set * str set
val test = fn : int -> gram -> str option * str option
val it = () : unit
```

Then we carry out the required testing:

```
- test 9 gram;
val it = (NONE,NONE) : str option * str option
```

Problem 3

(a) First, we give these definitions:

$$\begin{aligned}\mathbf{minAndRen} &= \mathbf{renameStatesCanonically} \circ \mathbf{minimize},\\ \mathbf{faToDFAmar} &= \mathbf{minAndRen} \circ \mathbf{nfaToDFA} \circ \mathbf{efaToNFA} \circ \mathbf{faToEFA},\\ \mathbf{regToDFAmar} &= \mathbf{faToDFAmar} \circ \mathbf{regToFA},\\ \mathbf{allStrDFA} &= \mathbf{regToDFAmar}((0+1+2)^*).\end{aligned}$$

Thus $\mathbf{minAndRen} \in \mathbf{DFA} \to \mathbf{DFA}$, $\mathbf{faToDFAmar} \in \mathbf{FA} \to \mathbf{DFA}$, $\mathbf{regToDFAmar} \in \mathbf{Reg} \to \mathbf{DFA}$ and $\mathbf{allStrDFA} \in \mathbf{DFA}$.

We define $\mathbf{lenDFA} \in \mathbb{N} \to \mathbf{DFA}$ by: $\mathbf{lenDFA}\, m = \mathbf{regToDFAmar}((0+1+2)^m)$.

We define $\mathbf{hasStrDFA} \in \mathbf{Str} \to \mathbf{DFA}$ and $\mathbf{notHasStrDFA} \in \mathbf{Str} \to \mathbf{DFA}$ by:

$$\mathbf{hasStrDFA}\, x = \mathbf{regToDFAmar}((0+1+2)^*(\mathbf{strToReg}\, x)(0+1+2)^*),$$
$$\mathbf{notHasStrDFA}\, x = \mathbf{minAndRen}(\mathbf{minus}(\mathbf{allStrDFA}, \mathbf{hasStrDFA}\, x)).$$

We let the DFA **hasAllSymsDFA** be

$$\mathbf{minAndRen}(\mathbf{inter}(\mathbf{hasStrDFA}\, 0, \mathbf{inter}(\mathbf{hasStrDFA}\, 1, \mathbf{hasStrDFA}\, 2))),$$

and let the DFA **notHasAllSymsDFA** be

$$\mathbf{minAndRen}(\mathbf{minus}(\mathbf{allStrDFA}, \mathbf{hasAllSymsDFA})).$$

We define $\mathbf{lenAndNotHasAllSymsDFA} \in \mathbb{N} \to \mathbf{DFA}$ by:

$$\mathbf{lenAndNotHasAllSymsDFA}\, m =$$
$$\mathbf{minAndRen}(\mathbf{inter}(\mathbf{lenDFA}\, m, \mathbf{notHasAllSymsDFA})).$$

We then define $\mathbf{someLenNotHasAllSymsFA} \in \mathbb{N} \to \mathbf{FA}$ by:

$$\mathbf{someLenNotHasAllSymsFA}\, m =$$
$$\mathbf{concat}(\mathbf{allStrDFA}, \mathbf{concat}(\mathbf{lenAndNotHasAllSymsDFA}\, m, \mathbf{allStrDFA})).$$

We then define $\mathbf{someLenNotHasAllSymsDFA} \in \mathbb{N} \to \mathbf{DFA}$ by:

$$\mathbf{someLenNotHasAllSymsDFA}\, m = \mathbf{faToDFAmar}(\mathbf{someLenNotHasAllSymsFA}\, m).$$

And we define $\mathbf{allLenHasAllSymsDFA} \in \mathbb{N} \to \mathbf{DFA}$ by:

$$\mathbf{allLenHasAllSymsDFA}\, m =$$
$$\mathbf{minAndRen}(\mathbf{minus}(\mathbf{allStrDFA}, \mathbf{someLenNotHasAllSymsDFA}\, m)).$$

We let the DFA **noBadDFA** be

$$\mathbf{minAndRen}(\mathbf{inter}(\mathbf{notHasStrDFA}\, 02, \mathbf{inter}(\mathbf{notHasStrDFA}\, 10, \mathbf{notHasStrDFA}\, 21)).$$

Finally, we define $\mathbf{ansDFA} \in \mathbb{N} \to \mathbf{DFA}$ by:

$$\mathbf{ansDFA}\, m = \mathbf{minAndRen}(\mathbf{inter}(\mathbf{noBadDFA}, \mathbf{allLenHasAllSymsDFA}\, m)).$$

(b) We put our definition

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

val allStrReg = Reg.fromString "(0 + 1 + 2)*";
val allStrDFA = regToDFAmar allStrReg;
val allStrFA  = injDFAToFA allStrDFA;
```

```
fun lenDFA m = regToDFAmar(Reg.power(Reg.fromString "0 + 1 + 2", m));

fun hasStrReg x    =
      Reg.concat(allStrReg, Reg.concat(strToReg x, allStrReg));
fun hasStrDFA x    = regToDFAmar(hasStrReg x);
fun notHasStrDFA x = minAndRen(DFA.minus(allStrDFA, hasStrDFA x));

val hasAllSymsDFA    =
      minAndRen
      (DFA.inter
       (hasStrDFA(Str.fromString "0"),
        DFA.inter
        (hasStrDFA(Str.fromString "1"),
         hasStrDFA(Str.fromString "2"))));
val notHasAllSymsDFA = minAndRen(DFA.minus(allStrDFA, hasAllSymsDFA));

fun lenAndNotHasAllSymsDFA m =
      minAndRen(DFA.inter(lenDFA m, notHasAllSymsDFA));
fun lenAndNotHasAllSymsFA m  = injDFAToFA(lenAndNotHasAllSymsDFA m);

fun someLenNotHasAllSymsFA m  =
      FA.concat(allStrFA, FA.concat(lenAndNotHasAllSymsFA m, allStrFA));
fun someLenNotHasAllSymsDFA m = faToDFAmar(someLenNotHasAllSymsFA m);

fun allLenHasAllSymsDFA m =
      minAndRen(DFA.minus(allStrDFA, someLenNotHasAllSymsDFA m));

val noBadDFA =
      minAndRen
      (DFA.inter
       (notHasStrDFA(Str.fromString "02"),
        DFA.inter
        (notHasStrDFA(Str.fromString "10"),
         notHasStrDFA(Str.fromString "21"))));

fun ansDFA m = minAndRen(DFA.inter(noBadDFA, allLenHasAllSymsDFA m));
```

of `ansDFA` in the file `ps6-p3.sml`, and load it into Forlan:

```
- use "ps6-p3.sml";
[opening ps6-p3.sml]
val minAndRen = fn : dfa -> dfa
val faToDFAmar = fn : fa -> dfa
val regToDFAmar = fn : reg -> dfa
val allStrReg = - : reg
val allStrDFA = - : dfa
val allStrFA = - : fa
val lenDFA = fn : int -> dfa
```

```
val hasStrReg = fn : str -> reg
val hasStrDFA = fn : str -> dfa
val notHasStrDFA = fn : str -> dfa
val hasAllSymsDFA = - : dfa
val notHasAllSymsDFA = - : dfa
val lenAndNotHasAllSymsDFA = fn : int -> dfa
val lenAndNotHasAllSymsFA = fn : int -> fa
val someLenNotHasAllSymsFA = fn : int -> fa
val someLenNotHasAllSymsDFA = fn : int -> dfa
val allLenHasAllSymsDFA = fn : int -> dfa
val noBadDFA = - : dfa
val ansDFA = fn : int -> dfa
val it = () : unit
```

Next we put our testing code

```
(* val hasBad2 : str -> bool

   hasBad2 x tests whether x has one or more of 02, 10 and 21 as a
   substring *)

fun hasBad2 x =
      Str.substr(Str.fromString "02", x) orelse
      Str.substr(Str.fromString "10", x) orelse
      Str.substr(Str.fromString "21", x)

(* val hasSym sym * str -> bool

   hasSym(a, x) tests whether a is a symbol in x *)

fun hasSym(a, x) = Str.substr([a], x);

(* val hasAllSyms : str -> bool

   hasAllSyms x tests whether x has at least one occurrence of
   all of 0, 1 and 2 *)

fun hasAllSyms x =
     hasSym(Sym.fromString "0", x) andalso
     hasSym(Sym.fromString "1", x) andalso
     hasSym(Sym.fromString "2", x)

(* val inX : int -> str -> bool

   if m >= 0 and x is in {0, 1, 2}^*, inX m x tests whether x is in X_m *)

fun inX m x =
      not(hasBad2 x) andalso
      Set.all
```

```
      (fn y => if length y = m then hasAllSyms y else true)
      (StrSet.substrings x);

(* val upto : int -> str set

   if n >= 0, then upto n returns all strings over alphabet {0, 1, 2} 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", ys))
      end;

(* val partition : int -> int -> str set * str set

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

   xs is all elements of upto n that are in X_m; and

   ys is all elements of upto n that are not in X_m *)

fun partition m n = Set.partition (inX m) (upto n);

(* val test : int -> int -> dfa -> str option * str option

   if m >= 0 and n >= 0, then test m n returns a function f such that,
   for all DFAs dfa, f dfa returns a pair (xOpt, yOpt) such that:

     If there is an element of {0, 1, 2}^* of length no more than n that
     is in X_m but is not accepted by dfa, then xOpt = SOME x for some
     such x; otherwise, xOpt = NONE.

     If there is an element of {0, 1, 2}^* of length no more than n that
     is not in X_m but is accepted by dfa, then yOpt = SOME y for some
     such y; otherwise, yOpt = NONE. *)

fun test m n =
      let val (goods, bads) = partition m n
      in fn dfa =>
              let val determAccepted = DFA.determAccepted dfa
                  val goodNotAccOpt  = Set.position (not o determAccepted) goods
                  val badAccOpt      = Set.position determAccepted bads
              in ((case goodNotAccOpt of
                        NONE   => NONE
                      | SOME i => SOME(ListAux.sub(Set.toList goods, i))),
```

```
                (case badAccOpt of
                      NONE   => NONE
                    | SOME i => SOME(ListAux.sub(Set.toList bads, i))))
            end
      end;

(* doit m prints the required information about ansDFA m *)

fun doit m =
      let val dfa = ansDFA m
          val bs  = DFA.alphabet dfa
          val n   = DFA.numStates dfa
          val res =
                case test m 12 dfa of
                     (NONE, NONE) => "test succeeded"
                   | _            => "test failed"
      in print "m = "; print(Int.toString m); print ": ";
         print "alphabet is {"; print(SymSet.toString bs); print "}; ";
         print "number of states is "; print(Int.toString n); print "; ";
         print res; print "\n"
      end;
```

in the file `ps6-p3-testing.sml`, and load it into Forlan:

```
- use "ps6-p3-testing.sml";
[opening ps6-p3-testing.sml]
val hasBad2 = fn : str -> bool
val hasSym = fn : sym * str -> bool
val hasAllSyms = fn : str -> bool
val inX = fn : int -> str -> bool
val upto = fn : int -> str set
val partition = fn : int -> int -> str set * str set
val test = fn : int -> int -> dfa -> str option * str option
val doit = fn : int -> unit
val it = () : unit
```

Finally, we apply the function `doit` to each of the required values of m, printing the required information about each `ansDFA` m:

```
- app doit [0, 1, 2, 3, 4, 5];
m = 0: alphabet is {}; number of states is 1; test succeeded
m = 1: alphabet is {}; number of states is 1; test succeeded
m = 2: alphabet is {0, 1, 2}; number of states is 3; test succeeded
m = 3: alphabet is {0, 1, 2}; number of states is 9; test succeeded
m = 4: alphabet is {0, 1, 2}; number of states is 18; test succeeded
m = 5: alphabet is {0, 1, 2}; number of states is 30; test succeeded
val it = () : unit
```

Note that the alphabets of the DFAs when m is 0 and 1 are empty. ($X_0 = \emptyset$, $X_1 = \{\%\}$ and $X_2 = \{\%, 0, 1, 2\}$).

(c) Easy calculations using the functions' specifications show that:

- $L(\mathbf{minAndRen}\, M) = L(M)$, for all DFAs M;
- $L(\mathbf{faToDFAmar}\, M) = L(M)$, for all FAs M;
- $L(\mathbf{regToDFAmar}\, \alpha) = L(\alpha)$, for all regular expressions α;
- $L(\mathbf{allStrDFA}) = \{0,1,2\}^*$.

Define $\mathbf{Len} \in \mathbb{N} \to \mathbf{Lan}$ by

$$\mathbf{Len}\, m = \{\, w \in \{0,1,2\}^* \mid |w| = m \,\}.$$

Because $\{0,1,2\}^m = \mathbf{Len}\, m$, for all $m \in \mathbb{N}$, we have that $L(\mathbf{lenDFA}\, m) = \mathbf{Len}\, m$, for all $m \in \mathbb{N}$.

Define $\mathbf{HasStr} \in \mathbf{Str} \to \mathbf{Lan}$ and $\mathbf{NotHasStr} \in \mathbf{Str} \to \mathbf{Lan}$ by:

$$\mathbf{HasStr}\, x = \{\, w \in \{0,1,2\}^* \mid x \text{ is a substring of } w \,\},$$
$$\mathbf{NotHasStr}\, x = \{\, w \in \{0,1,2\}^* \mid x \text{ is not a substring of } w \,\}.$$

Because $\mathbf{HasStr}\, x = \{0,1,2\}^* \{x\} \{0,1,2\}^*$, for all $x \in \{0,1,2\}^*$, we have that $L(\mathbf{hasStrDFA}\, x) = \mathbf{HasStr}\, x$, for all $x \in \{0,1,2\}^*$. And, because $\{0,1,2\}^* - \mathbf{HasStr}\, x = \mathbf{NotHasStr}\, x$, for all $x \in \{0,1,2\}^*$, and complementation corresponds to negation, we have that $L(\mathbf{notHasStrDFA}\, x) = \mathbf{NotHasStr}\, x$, for all $x \in \{0,1,2\}^*$.

Let

$$\mathbf{HasAllSyms} = \{\, w \in \{0,1,2\}^* \mid \{0,1,2\} \subseteq \mathbf{alphabet}\, w \,\}.$$

Then we have that $L(\mathbf{hasAllSymsDFA}) = \mathbf{HasStr}\, 0 \cap \mathbf{HasStr}\, 1 \cap \mathbf{HasStr}\, 2 = \mathbf{HasAllSyms}$, because intersection corresponds to conjunction ("and"). Let

$$\mathbf{NotHasAllSyms} = \{\, w \in \{0,1,2\}^* \mid \{0,1,2\} \not\subseteq \mathbf{alphabet}\, w \,\}.$$

Since $\{0,1,2\}^* - \mathbf{HasAllSyms} = \mathbf{NotHasAllSyms}$, we have that $L(\mathbf{notHasAllSymsDFA}) = \mathbf{NotHasAllSyms}$.

Define $\mathbf{LenAndNotHasAllSyms} \in \mathbb{N} \to \mathbf{Lan}$ by

$$\mathbf{LenAndNotHasAllSyms}\, m = \{\, w \in \{0,1,2\}^* \mid |w| = m \text{ and } \{0,1,2\} \not\subseteq \mathbf{alphabet}\, w \,\}.$$

Because $\mathbf{LenAndNotHasAllSyms}\, m = \mathbf{Len}\, m \cap \mathbf{NotHasAllSyms}$, for all $m \in \mathbb{N}$, we have that $L(\mathbf{lenAndNotHasAllSymsDFA}\, m) = \mathbf{LenAndNotHasAllSyms}\, m$, for all $m \in \mathbb{N}$.

Define $\mathbf{SomeLenNotHasAllSyms} \in \mathbb{N} \to \mathbf{Lan}$ by:

$$\mathbf{SomeLenNotHasAllSyms}\, m =$$
$$\{\, w \in \{0,1,2\}^* \mid \text{there is a substring } v \text{ of } w \text{ such that } |v| = m \text{ and } \{0,1,2\} \not\subseteq \mathbf{alphabet}\, v \,\}.$$

Because

$$\mathbf{SomeLenNotHasAllSyms}\, m = \{0,1,2\}^* \, (\mathbf{LenAndNotHasAllSyms}\, m) \, \{0,1,2\}^*,$$

for all $m \in \mathbb{N}$, we have that $L(\mathbf{someLenNotHasAllSymsDFA}\, m) = \mathbf{SomeLenNotHasAllSyms}\, m$, for all $m \in \mathbb{N}$.

Define **AllLenHasAllSyms** $\in \mathbb{N} \to$ **Lan** by:

$$\textbf{AllLenHasAllSyms}\, m =$$
$$\{\, w \in \{0,1,2\}^* \mid \text{for all substrings } v \text{ of } w,\ \text{if } |v| = m,\ \text{then } \{0,1,2\} \subseteq \textbf{alphabet}\, v \,\}.$$

Because $\{0,1,2\}^* - \textbf{SomeLenNotHasAllSyms}\, m = \textbf{AllLenHasAllSyms}$, for all $m \in \mathbb{N}$, we have that $L(\textbf{allLenHasAllSymsDFA}\, m) = \textbf{AllLenHasAllSyms}\, m$, for all $m \in \mathbb{N}$.

Let

$$\textbf{NoBad} = \{\, w \in \{0,1,2\}^* \mid \text{neither } 02, 10 \text{ nor } 21 \text{ are substrings of } w \,\}.$$

Since $\textbf{NoBad} = \textbf{NotHasStr}\, 02 \cap \textbf{NotHasStr}\, 10 \cap \textbf{NotHasStr}\, 21$, we have that $L(\textbf{noBadDFA}) = \textbf{NoBad}$.

Finally, because $\textbf{NoBad} \cap \textbf{AllLenHasAllSyms}\, m = X_m$, for all $m \in \mathbb{N}$, we have that $L(\textbf{ansDFA}\, m) = X_m$, for all $m \in \mathbb{N}$.