CS 591 S2—Formal Language Theory: Integrating Experimentation and Proof—Fall 2019

Problem Set 5

Model Answers

Problem 1

Because h is a bijection from $Q_{N'}$ to Q_N, there are three things we must prove:

- We must show that $h\, s_{N'} = s_N$. We have that $h\, s_{N'} = h(\delta_{N'}(s_{N'}, \%)) = \delta_N(s_N, \%) = s_N$.
- We must show that $\{\, h\, q \mid q \in A_{N'} \,\} = A_N$. It will suffice to show that $\{\, h\, q \mid q \in A_{N'} \,\} \subseteq A_N \subseteq \{\, h\, q \mid q \in A_{N'} \,\}$.

  ($\{\, h\, q \mid q \in A_{N'} \,\} \subseteq A_N$) Suppose $r \in \{\, h\, q \mid q \in A_{N'} \,\}$, so that $r = h\, q$ for some $q \in A_{N'}$. Because q is reachable in N', there is a $w \in \Sigma^*$ such that $\delta_{N'}(s_{N'}, w) = q$. Thus $w \in L(N') = L(N)$, so that $r = h\, q = h(\delta_{N'}(s_{N'}, w)) = \delta_N(s_N, w) \in A_N$.

  ($A_N \subseteq \{\, h\, q \mid q \in A_{N'} \,\}$) Suppose $r \in A_N$. Because r is reachable in N, there is a $w \in \Sigma^*$ such that $\delta_N(s_N, w) = r$. Thus $w \in L(N) = L(N')$, so that $r = \delta_N(s_N, w) = h(\delta_{N'}(s_{N'}, w))$ and $\delta_{N'}(s_{N'}, w) \in A_{N'}$. Thus $r \in \{\, h\, q \mid q \in A_{N'} \,\}$.
- We must show that $\{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\} = T_N$. It will suffice to show that $\{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\} \subseteq T_N \subseteq \{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\}$.

  ($\{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\} \subseteq T_N$) Suppose $t \in \{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\}$, so that $t = (h\, q, x, h\, r)$ for some $q, r \in Q_{N'}$ and $x \in \mathbf{Str}$ such that $q, x \to r \in T_{N'}$. Because N' is a DFA with alphabet Σ, we have that $x = a$, for some $a \in \Sigma$, and $\delta_{N'}(q, a) = r$. Since q is reachable in N', there is a $y \in \Sigma^*$ such that $\delta_{N'}(s_{N'}, y) = q$. Thus $\delta_{N'}(s_{N'}, ya) = \delta_{N'}(\delta_{N'}(s_{N'}, y), a) = \delta_{N'}(q, a) = r$, so that $\delta_N(h\, q, a) = \delta_N(h(\delta_{N'}(s_{N'}, y)), a) = \delta_N(\delta_N(s_N, y), a) = \delta_N(s_N, ya) = h(\delta_{N'}(s_{N'}, ya)) = h\, r$. Thus $t = (h\, q, x, h\, r) = (h\, q, a, h\, r) \in T_N$.

  ($T_N \subseteq \{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\}$) Suppose $t \in T_N$, so that $t = (q', a, r')$ for some $q', r' \in Q_N$ and $a \in \Sigma$. Because $\mathbf{range}\, h = Q_N$, there are $q, r \in Q_{N'}$ such that $h\, q = q'$ and $h\, r = r'$. Thus $t = (h\, q, a, h\, r)$, so that $\delta_N(h\, q, a) = h\, r$. Because q is reachable in N', there is an $y \in \Sigma^*$ such that $\delta_{N'}(s_{N'}, y) = q$. Thus $h(\delta_{N'}(s_{N'}, ya)) = \delta_N(s_N, ya) = \delta_N(\delta_N(s_N, y), a) = \delta_N(h(\delta_{N'}(s_{N'}, y)), a) = \delta_N(h\, q, a) = h\, r$. Since h is injective, it follows that $\delta_{N'}(q, a) = \delta_{N'}(\delta_{N'}(s_{N'}, y), a) = \delta_{N'}(s_{N'}, ya) = r$. Thus $t = (h\, q, a, h\, r)$ and $(q, a, r) \in T_{N'}$, showing that $t \in \{\, (h\, q), x \to (h\, r) \mid q, x \to r \in T_{N'} \,\}$.

Problem 2

(a) First we give some standard definitions:

$$\begin{aligned}\mathbf{minAndRen} &= \mathbf{renameStatesCanonically} \circ \mathbf{minimize},\\ \mathbf{efaToDFA} &= \mathbf{nfaToDFA} \circ \mathbf{efaToNFA},\\ \mathbf{strToEFA} &= \mathbf{faToEFA} \circ \mathbf{strToFA},\\ \mathbf{allStrEFA} &= \mathbf{closure}(\mathbf{union}(\mathbf{symToNFA}\,0, \mathbf{symToNFA}\,1)), \text{ and}\\ \mathbf{allStrDFA} &= \mathbf{minAndRen}(\mathbf{efaToDFA}\,\mathbf{allStrEFA}).\end{aligned}$$

Thus $\mathbf{minAndRen} \in \mathbf{DFA} \to \mathbf{DFA}$, $\mathbf{efaToDFA} \in \mathbf{EFA} \to \mathbf{DFA}$, $\mathbf{strToEFA} \in \mathbf{Str} \to \mathbf{EFA}$, $\mathbf{allStrEFA} \in \mathbf{EFA}$ and $\mathbf{allStrDFA} \in \mathbf{DFA}$.

Next, we define $\mathbf{hasSubEFA} \in \{0,1\}^* \to \mathbf{EFA}$ by: for all $x \in \{0,1\}^*$,

$$\mathbf{hasSubEFA}\,x = \mathbf{concat}(\mathbf{allStrDFA}, \mathbf{concat}(\mathbf{strToEFA}\,x, \mathbf{allStrDFA})).$$

Define $\mathbf{hasSubDFA} \in \{0,1\}^* \to \mathbf{DFA}$ by:

$$\mathbf{hasSubDFA} = \mathbf{minAndRen} \circ \mathbf{efaToDFA} \circ \mathbf{hasSubEFA}.$$

Define $\mathbf{hasNotSubDFA} \in \{0,1\}^* \to \mathbf{DFA}$ by: for all $x \in \{0,1\}^*$,

$$\mathbf{hasNotSubDFA}\,x = \mathbf{minAndRen}(\mathbf{minus}(\mathbf{allStrDFA}, \mathbf{hasSubDFA}\,x)).$$

Define $\mathbf{someUnmatchedEFA} \in \{0,1\}^* \times \{0,1\}^* \to \mathbf{EFA}$ by: for all $x, y \in \{0,1\}^*$,

$$\begin{aligned}&\mathbf{someUnmatchedEFA}(x,y)\\ &= \mathbf{concat}(\mathbf{hasNotSubDFA}\,y, \mathbf{concat}(\mathbf{strToEFA}\,x, \mathbf{hasNotSubDFA}\,y)).\end{aligned}$$

Define $\mathbf{someUnmatchedDFA} \in \{0,1\}^* \times \{0,1\}^* \to \mathbf{DFA}$ by:

$$\mathbf{someUnmatchedDFA} = \mathbf{minAndRen} \circ \mathbf{efaToDFA} \circ \mathbf{someUnmatchedEFA}.$$

Define $\mathbf{allMatchedDFA} \in \{0,1\}^* \times \{0,1\}^* \to \mathbf{DFA}$ by: for all $x, y \in \{0,1\}^*$,

$$\mathbf{allMatchedDFA}(x,y) = \mathbf{minAndRen}(\mathbf{minus}(\mathbf{allStrDFA}, \mathbf{someUnmatchedDFA}(x,y))).$$

Finally, define $\mathbf{dcsDFA} \in \{0,1\}^* \times \{0,1\}^* \to \mathbf{DFA}$ by: for all $x, y \in \{0,1\}^*$,

$$\mathbf{dcsDFA}(x,y) = \mathbf{minAndRen}(\mathbf{inter}(\mathbf{allMatchedDFA}(x,y), \mathbf{allMatchedDFA}(y,x))).$$

(b) Our definition of `dcsDFA` is in the file `ps5.sml`:

```
val zero      = Sym.fromString "0";
val one       = Sym.fromString "1";
val minAndRen =
      DFA.renameStatesCanonically o DFA.minimize;
val efaToDFA  = nfaToDFA o efaToNFA;
```

```
val strToEFA  = faToEFA o strToFA;
val allStrEFA =
      EFA.closure
      (EFA.union(injNFAToEFA(symToNFA zero), injNFAToEFA(symToNFA one)));
val allStrDFA = minAndRen(efaToDFA allStrEFA);

fun hasSubEFA x =
      EFA.concat
      (injDFAToEFA allStrDFA,
       EFA.concat(strToEFA x, injDFAToEFA allStrDFA));

val hasSubDFA = minAndRen o efaToDFA o hasSubEFA;

fun hasNotSubDFA x = minAndRen(DFA.minus(allStrDFA, hasSubDFA x));

fun someUnmatchedEFA(x, y) =
      EFA.concat
      (injDFAToEFA(hasNotSubDFA y),
       EFA.concat(strToEFA x, injDFAToEFA(hasNotSubDFA y)));

val someUnmatchedDFA = minAndRen o efaToDFA o someUnmatchedEFA;

fun allMatchedDFA(x, y) =
      minAndRen(DFA.minus(allStrDFA, someUnmatchedDFA(x, y)));

fun dcsDFA(x, y) =
      minAndRen(DFA.inter(allMatchedDFA(x, y), allMatchedDFA(y, x)));
```

We load it into Forlan as follows:

```
- use "ps5.sml";
[opening ps5.sml]
val zero = - : sym
val one = - : sym
val minAndRen = fn : dfa -> dfa
val efaToDFA = fn : efa -> dfa
val strToEFA = fn : str -> efa
val allStrEFA = - : efa
val allStrDFA = - : dfa
val hasSubEFA = fn : str -> efa
val hasSubDFA = fn : str -> dfa
val hasNotSubDFA = fn : str -> dfa
val someUnmatchedEFA = fn : str * str -> efa
val someUnmatchedDFA = fn : str * str -> dfa
val allMatchedDFA = fn : str * str -> dfa
val dcsDFA = fn : str * str -> dfa
val it = () : unit
```

Our testing harness for `dcsDFA` is in the file `ps5-test.sml`:

```
(* val upto : int -> str set

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

(* val occs : str * str -> (str * str)list

   if x, w in {0, 1}*, then occs(x, w) returns the list of all pairs
   (u, v) such that w = u @ x @ v *)

fun occs(ds : str, nil : str) : (str * str)list =
      if null ds
      then [(nil, nil)]
      else nil
  | occs(ds,       cs)                          =
      let val b  = hd cs
          val bs = tl cs
      in (if Str.prefix(ds, cs)
          then [(nil, List.drop(cs, length ds))]
          else nil) @
         map (fn (es, fs) => (b :: es, fs)) (occs(ds, bs))
      end;

(* val inDCS : str * str -> str -> bool

   if x, y, w in {0, 1}*, inDCS (x, y) w tests whether w is in
   DCS(x, y) *)

fun inDCS (x, y) w =
      List.all
      (fn (u, v) => Str.substr(y, u) orelse Str.substr(y, v))
      (occs(x, w))
      andalso
      List.all
      (fn (u, v) => Str.substr(x, u) orelse Str.substr(x, v))
      (occs(y, w));

(* val all2 : ('a * 'a -> bool) -> 'a set -> bool

   all2 f xs tests whether f(x, y) for all x, y in xs *)
```

```
fun all2 f xs =
      Set.all
      (fn x =>
            Set.all
            (fn y => f(x, y))
            xs)
      xs

(* val test : int * int -> int * int * ((str * str -> dfa) -> bool)

   if n, m >= 0, then test(n, m) binds xs and ys to upto n and upto m,
   respectively, and then returns

     (Set.size xs * Set.size xs, Set.size ys, f),

   where f is the function that, when called with argument
   g : str * str -> dfa, tests whether, for all x, y in xs, g(x, y)
   returns a DFA dfa such that dfa is minimized (isomorphic to
   the minimization of itself) and, for all w in ys,

     if w is in DCS(x, y), then dfa accepts w; and

     if w is not in DCS(x, y), then dfa does not accept w *)

fun test(n, m) =
      let val xs = upto n
          val ys = upto m
      in (Set.size xs * Set.size xs,
          Set.size ys,
          fn g =>
               all2
               (fn (x, y) =>
                     let val dfa           = g(x, y)
                         val accepted      = DFA.determAccepted dfa
                         val (goods, bads) = Set.partition (inDCS(x, y)) ys
                     in DFA.isomorphic(DFA.minimize dfa, dfa) andalso
                        Set.all accepted goods                andalso
                        Set.all (not o accepted) bads
                     end)
               xs)
      end;
```

We load it into Forlan, and demonstrate how some of its functions work, as follows:

```
- use "ps5-test.sml";
[opening ps5-test.sml]
val upto = fn : int -> str set
val occs = fn : str * str -> (str * str) list
```

```
val inDCS = fn : str * str -> str -> bool
val all2 = fn : ('a * 'a -> bool) -> 'a set -> bool
val test = fn : int * int -> int * int * ((str * str -> dfa) -> bool)
val it = () : unit
- val ps = occs(Str.fromString "01", Str.fromString "00101101");
val ps = [([-],[-,-,-,-,-]),([-,-,-],[-,-,-]),([-,-,-,-,-,-],[])]
  : (str * str) list
- map (fn (u, v) => (Str.toString u, Str.toString v)) ps;
val it = [("0","01101"),("001","101"),("001011","%")] : (string * string) list
- val f = inDCS(Str.fromString "0011", Str.fromString "1100");
val f = fn : str -> bool
- f(Str.fromString "010");
val it = true : bool
- f(Str.fromString "001100");
val it = false : bool
- f(Str.fromString "00111100");
val it = true : bool
- f(Str.fromString "0011001100");
val it = false : bool
- f(Str.fromString "001100110011");
val it = false : bool
- f(Str.fromString "00110011001100");
val it = true : bool
```

Finally, we apply `dcsDFA` to all 961 pairs of strings in $\{0,1\}^*$ of length no more than 4, and check that each of the resulting DFAs works correctly on all 65535 elements of $\{0,1\}^*$ of length no more than 15, as follows:

```
- val (n, m, f) = test(4, 15);
val n = 961 : int
val m = 65535 : int
val f = fn : (str * str -> dfa) -> bool
- f dcsDFA;
val it = true : bool
```

(This took about 40 minutes on my laptop.)

(c) First, we note that, because **renameStatesCanonically** and **minimize** preserve the meaning of DFAs, for all DFAs M,

$$\begin{aligned} L(\mathbf{minAndRen}\, M) &= L(\mathbf{renameStatesCanonically}(\mathbf{minimize}\, M)) \\ &= L(\mathbf{minimize}\, M) = L(M), \end{aligned}$$

and thus $\mathbf{minAndRen}\, M \approx M$.

Lemma PS5.2.1
For all DFAs M, $\mathbf{minimize}(\mathbf{minAndRen}\, M)$ *is isomorphic to* $\mathbf{minAndRen}\, M$.

Proof. We have that $\mathbf{minAndRen}\, M = \mathbf{renameStatesCanonically}(\mathbf{minimize}\, M)$ is isomorphic to $\mathbf{minimize}\, M$. Thus it will suffice to show that $\mathbf{minimize}(\mathbf{minAndRen}\, M)$ is isomorphic to $\mathbf{minimize}\, M$. By Theorem 3.13.12, it will suffice to show that

(1) $\mathbf{minimize}(\mathbf{minAndRen}\, M) \approx M$;

(2) $\mathbf{alphabet}(\mathbf{minimize}(\mathbf{minAndRen}\, M)) = \mathbf{alphabet}(L(M))$; and

(3) $|Q_{\mathbf{minimize}(\mathbf{minAndRen}\, M)}| \leq |Q_{\mathbf{minimize}\, M}|$.

For (1), we have that $\mathbf{minimize}(\mathbf{minAndRen}\, M) \approx \mathbf{minAndRen}\, M \approx M$.
For (2), by Theorem 3.13.12, we have that

$$\mathbf{alphabet}(\mathbf{minimize}(\mathbf{minAndRen}\, M)) = \mathbf{alphabet}(L(\mathbf{minAndRen}\, M)) = \mathbf{alphabet}(L(M)).$$

For (3), by Theorem 3.13.12, we have that

$$\begin{aligned} |Q_{\mathbf{minimize}(\mathbf{minAndRen}\, M)}| &\leq |Q_{\mathbf{minAndRen}\, M}| \\ &= |Q_{\mathbf{renameStatesCanonically}(\mathbf{minimize}\, M)}| = |Q_{\mathbf{minimize}\, M}|. \end{aligned}$$

□

Define $\mathbf{HasSub} \in \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ by: for all $x \in \{0,1\}^*$, $\mathbf{HasSub}\, x = \{\, w \in \{0,1\}^* \mid x$ is a substring of $w \,\}$.

Clearly:

Lemma PS5.2.2
For all $x \in \{0,1\}^*$, $\mathbf{HasSub}\, x = \{0,1\}^* \{x\} \{0,1\}^*$.

Lemma PS5.2.2 and easy calculations show:

Lemma PS5.2.3
(1) For all $x \in \{0,1\}^*$, $L(\mathbf{hasSubEFA}\, x) = \mathbf{HasSub}\, x$.

(2) For all $x \in \{0,1\}^*$, $L(\mathbf{hasSubDFA}\, x) = \mathbf{HasSub}\, x$.

Define $\mathbf{HasNotSub} \in \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ by: for all $x \in \{0,1\}^*$, $\mathbf{HasNotSub}\, x = \{\, w \in \{0,1\}^* \mid x$ is not a substring of $w \,\}$.

Because complementation corresponds to negation, we have:

Lemma PS5.2.4
For all $x \in \{0,1\}^*$, $\mathbf{HasNotSub}\, x = \{0,1\}^* - \mathbf{HasSub}\, x$.

Lemmas PS5.2.3 and PS5.2.4, and an easy calculation show:

Lemma PS5.2.5
For all $x \in \{0,1\}^*$, $L(\mathbf{hasNotSubDFA}\, x) = \mathbf{HasNotSub}\, x$.

Define $\mathbf{SomeUnmatched} \in \{0,1\}^* \times \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ by: for all $x, y \in \{0,1\}^*$, $\mathbf{SomeUnmatched}(x,y)$ is the set of all $w \in \{0,1\}^*$ such that there are $u, v \in \{0,1\}^*$ such that $w = uxv$, y is not a substring of u, and y is not a substring of v.

It is easy to show:

Lemma PS5.2.6
For all $x, y \in \{0,1\}^*$, $\mathbf{SomeUnmatched}(x,y) = \mathbf{HasNotSub}\, y \,\{x\}\, \mathbf{HasNotSub}\, y$.

Lemmas PS5.2.5 and PS5.2.6, and easy calculations show:

Lemma PS5.2.7

(1) For all $x, y \in \{0,1\}^*$, $L(\mathbf{someUnmatchedEFA}(x,y)) = \mathbf{SomeUnmatched}(x,y)$.

(2) For all $x, y \in \{0,1\}^*$, $L(\mathbf{someUnmatchedDFA}(x,y)) = \mathbf{SomeUnmatched}(x,y)$.

Define $\mathbf{AllMatched} \in \{0,1\}^* \times \{0,1\}^* \to \mathcal{P}(\{0,1\}^*)$ by: for all $x, y \in \{0,1\}^*$, $\mathbf{AllMatched}(x,y)$ is the set of all $w \in \{0,1\}^*$ such that, for all $u, v \in \{0,1\}^*$, if $w = uxv$, then y is a substring of u, or y is a substring of v.

Lemma PS5.2.8

For all $x, y \in \{0,1\}^*$, $\mathbf{AllMatched}(x,y) = \{0,1\}^* - \mathbf{SomeUnmatched}(x,y)$.

Proof. Follows from the relationship between complementation and negation, since, if $w \in \{0,1\}^*$, then:

> there do not exist $u, v \in \{0,1\}^*$ such that $w = uxv$, and y is not a substring of u, and y is not a substring of v

iff

> for all $u, v \in \{0,1\}^*$ it is not the case that: $w = uxv$, and y is not a substring of u, and y is not a substring of v

iff

> for all $u, v \in \{0,1\}^*$, $w \neq uxv$, or y is a substring of u, or y is a substring of v

iff

> for all $u, v \in \{0,1\}^*$, $w \neq uxv$, or: y is a substring of u, or y is a substring of v

iff

> for all $u, v \in \{0,1\}^*$, if $w = uxv$, then y is a substring of u, or y is a substring of v.

□

Lemmas PS5.2.7 and PS5.2.8, and an easy calculation show:

Lemma PS5.2.9

For all $x, y \in \{0,1\}^*$, $L(\mathbf{allMatchedDFA}(x,y)) = \mathbf{AllMatched}(x,y)$.

Because intersection corresponds to conjunction, we have:

Lemma PS5.2.10

For all $x, y \in \{0,1\}^*$, $\mathbf{DCS}(x,y) = \mathbf{AllMatched}(x,y) \cap \mathbf{AllMatched}(y,x)$.

Lemmas PS5.2.9 and PS5.2.10, and an easy calculation, show:

Lemma PS5.2.11

For all $x, y \in \{0,1\}^*$, $L(\mathbf{dcsDFA}(x,y)) = \mathbf{DCS}(x,y)$.

Finally, Lemma PS5.2.1 tells us that:

Lemma PS5.2.12

For all $x, y \in \{0,1\}^*$, $\mathbf{minimize}(\mathbf{dcsDFA}(x,y))$ *is isomorphic to* $\mathbf{dcsDFA}(x,y)$.