CS 516—Software Foundations via Formal Languages—Spring 2025

Problem Set 2

Model Answers

Problem 1

(a) We use induction on X to show that, for all $w \in X$, $w \in Y$. There are four steps to show.

(1) We must show that $\% \in Y$. Clearly $\% \in \{\mathsf{0}, \mathsf{1}\}^*$. Since the only prefix of $\%$ is itself, and $\mathbf{diff}\, \% = 0 \geq 0$, it follows that $\% \in Y$.

(2) We must show that $\mathsf{1} \in Y$. Clearly $\mathsf{1} \in \{\mathsf{0}, \mathsf{1}\}^*$. Since $\%$ and $\mathsf{1}$ are the only prefixes of $\mathsf{1}$, $\mathbf{diff}\, \% = 0 \geq 0$ and $\mathbf{diff}\, \mathsf{1} = 1 \geq 0$, we have that $\mathsf{1} \in Y$.

(3) Suppose $x, y \in X$, and assume the inductive hypothesis: $x, y \in Y$. We must show that $\mathsf{1}x\mathsf{1}y\mathsf{0} \in Y$. Since $x, y \in \{\mathsf{0}, \mathsf{1}\}^*$, we have that $\mathsf{1}x\mathsf{1}y\mathsf{0} \in \{\mathsf{0}, \mathsf{1}\}^*$. Suppose that v is a prefix of $\mathsf{1}x\mathsf{1}y\mathsf{0}$. We must show that $\mathbf{diff}\, v \geq 0$. There are four cases to consider.

- Suppose $v = \%$. Then $\mathbf{diff}\, v = \mathbf{diff}\, \% = 0 \geq 0$.
- Suppose $v = \mathsf{1}u$, for some prefix u of x. Because $x \in Y$ and u is a prefix of x, we have that $\mathbf{diff}\, u \geq 0$. Thus

$$\mathbf{diff}\, v = \mathbf{diff}(\mathsf{1}u) = \mathbf{diff}\, \mathsf{1} + \mathbf{diff}\, u = 1 + \mathbf{diff}\, u \geq 1 + 0 \geq 0.$$

- Suppose $v = \mathsf{1}x\mathsf{1}u$, for some prefix u of y. Because $x \in Y$ and x is a prefix of itself, we have that $\mathbf{diff}\, x \geq 0$. Because $y \in Y$ and u is a prefix of y, we have that $\mathbf{diff}\, u \geq 0$. Thus

$$\begin{aligned} \mathbf{diff}\, v = \mathbf{diff}(\mathsf{1}x\mathsf{1}u) &= \mathbf{diff}\, \mathsf{1} + \mathbf{diff}\, x + \mathbf{diff}\, \mathsf{1} + \mathbf{diff}\, u \\ &= 1 + \mathbf{diff}\, x + 1 + \mathbf{diff}\, u = 2 + \mathbf{diff}\, x + \mathbf{diff}\, u \geq 2 + 0 + 0 \geq 0. \end{aligned}$$

- Suppose $v = \mathsf{1}x\mathsf{1}y\mathsf{0}$. Because $x \in Y$ and x is a prefix of itself, we have that $\mathbf{diff}\, x \geq 0$. Because $y \in Y$ and y is a prefix of itself, we have that $\mathbf{diff}\, y \geq 0$. Thus

$$\begin{aligned} \mathbf{diff}\, v = \mathbf{diff}(\mathsf{1}x\mathsf{1}y\mathsf{0}) &= \mathbf{diff}\, \mathsf{1} + \mathbf{diff}\, x + \mathbf{diff}\, \mathsf{1} + \mathbf{diff}\, y + \mathbf{diff}\, \mathsf{0} \\ &= 1 + \mathbf{diff}\, x + 1 + \mathbf{diff}\, y + -2 = \mathbf{diff}\, x + \mathbf{diff}\, y \geq 0 + 0 \geq 0. \end{aligned}$$

(4) Suppose $x, y \in X$, and assume the inductive hypothesis: $x, y \in Y$. We must show that $xy \in Y$. Since $x, y \in \{\mathsf{0}, \mathsf{1}\}^*$, we have that $xy \in \{\mathsf{0}, \mathsf{1}\}^*$. Suppose that v is a prefix of xy. We must show that $\mathbf{diff}\, v \geq 0$. There are two cases to consider.

- Suppose v is a prefix of x. Since $x \in Y$, it follows that $\mathbf{diff}\, v \geq 0$.
- Suppose $v = xu$, for some prefix u of y. Since $x \in Y$ and x is a prefix of itself, we have that $\mathbf{diff}\, x \geq 0$. And, since $y \in Y$ and u is a prefix of y, we have that $\mathbf{diff}\, u \geq 0$. Thus $\mathbf{diff}\, v = \mathbf{diff}(xu) = \mathbf{diff}\, x + \mathbf{diff}\, u \geq 0 + 0 \geq 0$.

(b) We begin by proving a useful lemma.

Lemma PS2.1.1
For all $w \in \{\mathtt{0},\mathtt{1}\}^*$, if $\mathbf{diff}\, w \geq 1$ and $w \in Y$, then there are $x, y \in Y$ such that $w = x\mathtt{1}y$.

Proof. Suppose $w \in \{\mathtt{0},\mathtt{1}\}^*$, $\mathbf{diff}\, w \geq 1$ and $w \in Y$. Let x be the longest prefix of w such that $\mathbf{diff}\, x \leq 0$ (x is well-defined because $\%$ is a prefix of w and $\mathbf{diff}\, \% = 0 \leq 0$). Let $z \in \{\mathtt{0},\mathtt{1}\}^*$ be such that $w = xz$. Because $\mathbf{diff}\, w \geq 1$ and $\mathbf{diff}\, x \leq 0$, we have that $z \neq \%$, so that $z = by$ for some $b \in \{\mathtt{0},\mathtt{1}\}$ and $y \in \{\mathtt{0},\mathtt{1}\}^*$. Thus $w = xz = xby$. Because x is a prefix of w and $w \in Y$, it follows that $\mathbf{diff}\, x \geq 0$. Thus $\mathbf{diff}\, x = 0$.

Suppose, toward a contradiction, that $b = \mathtt{0}$. Thus $w = x\mathtt{0}y$. Because $w \in Y$ and $x\mathtt{0}$ is a prefix of w, it follows that $-2 = 0 + -2 = \mathbf{diff}\, x + -2 = \mathbf{diff}(x\mathtt{0}) \geq 0$—contradiction. Thus $b = \mathtt{1}$, so that $w = x\mathtt{1}y$. It remains to show that $x, y \in Y$.

To see that $x \in Y$, suppose v is a prefix of x. We must show that $\mathbf{diff}\, v \geq 0$. Because v is a prefix of x, and x is a prefix of $x\mathtt{1}y = w$, it follows that v is a prefix of w. And $w \in Y$, so that $\mathbf{diff}\, v \geq 0$.

To see that $y \in Y$, suppose v is a prefix of y. We must show that $\mathbf{diff}\, v \geq 0$. Suppose, toward a contradiction, that $\mathbf{diff}\, v \leq -1$. We have that $x\mathtt{1}v$ is a prefix of $x\mathtt{1}y = w$ and $\mathbf{diff}(x\mathtt{1}v) = \mathbf{diff}\, x + 1 + \mathbf{diff}\, v = 0 + 1 + \mathbf{diff}\, v = 1 + \mathbf{diff}\, v \leq 1 + -1 = 0$, so that $x\mathtt{1}v$ is a prefix of w and $\mathbf{diff}(x\mathtt{1}v) \leq 0$. But x is the longest prefix of w with a diff that is ≤ 0 and $x\mathtt{1}v$ is strictly longer than x—contradiction. Thus $\mathbf{diff}\, v \geq 0$. $\square$

Now, we show that $Y \subseteq X$. Since $Y \subseteq \{\mathtt{0},\mathtt{1}\}^*$, it will suffice to show that, for all $w \in \{\mathtt{0},\mathtt{1}\}^*$,

$$\text{if } w \in Y, \text{ then } w \in X.$$

We proceed by strong string induction. Suppose $w \in \{\mathtt{0},\mathtt{1}\}^*$, and assume the inductive hypothesis: for all $x \in \{\mathtt{0},\mathtt{1}\}^*$, if x is a proper substring of w, then

$$\text{if } x \in Y, \text{ then } x \in X.$$

We must show that

$$\text{if } w \in Y, \text{ then } w \in X.$$

Suppose $w \in Y$. We must show that $w \in X$. There are two cases to consider.

- Suppose $w = \%$. Then $w = \% \in X$, by part (1) of the definition of X.
- Suppose $w = as$, for some $a \in \{\mathtt{0},\mathtt{1}\}$ and $s \in \{\mathtt{0},\mathtt{1}\}^*$.

  Suppose, toward a contradiction, that $a = \mathtt{0}$, so that $w = as = \mathtt{0}s$. Because $w \in Y$ and $\mathtt{0}$ is a prefix of w, we have that $-2 = \mathbf{diff}\, \mathtt{0} \geq 0$—contradiction. Thus $a = \mathtt{1}$, so that $w = as = \mathtt{1}s$.

  There are two sub-cases to consider.

  - Suppose $s \in Y$. By part (2) of the definition of X, we have that $\mathtt{1} \in X$. And, because s is a proper substring of w, the inductive hypothesis tells us that $s \in X$. Thus, by part (4) of the definition of X, we have that $w = \mathtt{1}s \in X$.

- Suppose $s \notin Y$. Because $s \in \{0,1\}^*$, there is a prefix of s with a negative diff. Let z be the shortest prefix of s such that $\textbf{diff}\, z \leq -1$, and let $t \in \{0,1\}^*$ be such that $s = zt$. Since $\textbf{diff}\, z \leq -1$, we have that $z \neq \%$, so that $z = ub$ for some $u \in \{0,1\}^*$ and $b \in \{0,1\}$. Thus $s = zt = ubt$ and $w = 1s = 1ubt$. Because u is a shorter prefix of s than z, it follows that $\textbf{diff}\, u \geq 0$.

  Suppose, toward a contradiction, that $b = 1$. Since $\textbf{diff}\, u{+}1 = \textbf{diff}\, u{+}\textbf{diff}\, b = \textbf{diff}(ub) = \textbf{diff}\, z \leq -1$, we have $\textbf{diff}\, u \leq -2$, contradicting the fact that $\textbf{diff}\, u \geq 0$. Thus $b = 0$, so that $z = ub = u0$, $s = zt = u0t$ and $w = 1s = 1u0t$.

  Since $\textbf{diff}\, u + -2 = \textbf{diff}(u0) = \textbf{diff}\, z \leq -1$, we have that $\textbf{diff}\, u \leq 1$. But $\textbf{diff}\, u \geq 0$, so that $\textbf{diff}\, u \in \{0,1\}$.

  Suppose, toward a contradiction, that $\textbf{diff}\, u = 0$. Since $w \in Y$ and $1u0$ is a prefix of w, it follows that $-1 = 1 + 0 + -2 = \textbf{diff}(1u0) \geq 0$—contradiction. Thus $\textbf{diff}\, u = 1$.

  To see that $u \in Y$, suppose v is a prefix of u. We must show that $\textbf{diff}\, v \geq 0$. Because u is a shorter prefix of s than z, it follows that v is a shorter prefix of s than z. Thus, by the definition of z, we have that $\textbf{diff}\, v \geq 0$.

  To see that $t \in Y$, suppose v is a prefix of t. We must show that $\textbf{diff}\, v \geq 0$. Because $w = 1u0t$, it follows that $1u0v$ is a prefix of w. But $w \in Y$, and thus $\textbf{diff}\, v = 1 + 1 + -2 + \textbf{diff}\, v = \textbf{diff}(1u0v) \geq 0$.

  Summarizing, we have that $w = 1u0t$, $u, t \in Y$ and $\textbf{diff}\, u = 1$. Since $\textbf{diff}\, u \geq 1$ and $u \in Y$, Lemma PS2.1.1 tells us that $u = x1y$, for some $x, y \in Y$. Thus $w = 1x1y0t$. Since x, y and t are all proper substrings of w, and $x, y, t \in Y$, the inductive hypothesis tells us that $x, y, t \in X$. Since $x, y \in X$, we have that $1x1y0 \in X$, by part (3) of the definition of X. Thus $w = (1x1y0)t \in X$, by part (4) of the definition of X.

Problem 2

Here is `ps2-explain.sml`:

```
(* ps2-explain.sml *)

(* validStrSilent w (silently) tests whether w is in Y *)

val validStrSilent = validStr false

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

   if w is an str of zeros and ones, and there is a prefix x of w
   such that f(diff x), then shortestPrefix f w returns (x, y), where x is
   the shortest such prefix and y is such that x @ y = w *)

fun shortestPrefix f (w : str) : str * str =
      let fun short(bs, n, nil)             =
                if f n then (bs, nil) else raise Fail "shouldn't happen"
            | short(bs, n, c_cs as c :: cs) =
                if f n
```

```
              then (bs, c_cs)
              else short(bs @ [c], n + diffSym c, cs)
      in short(nil, 0, w) end

(* val shortestNegativePrefix : str -> str * str

   if w is an str of zeros and ones, and there is a prefix x of w
   such that diff x <= ~1, then shortestNegativePrefix w returns (x, y),
   where x is the shortest such prefix and y is such that x @ y = w *)

val shortestNegativePrefix = shortestPrefix(fn n => n <= ~1)

(* longestPrefix : (int -> bool) -> str -> str * str

   if w is an str of zeros and ones, and there is a prefix x of w
   such that f(diff x), then longestPrefix f w returns (x, y), where x is
   the longest such prefix and y is such that x @ y = w *)

fun longestPrefix f (w : str) : str * str =
      let fun long(bs, n, lngstOpt, nil)             =
                if f n
                then (bs, nil)
                else (case lngstOpt of
                           NONE      => raise Fail "shouldn't happen"
                         | SOME long => long)
            | long(bs, n, lngstOpt, c_cs as c :: cs) =
                long(bs @ [c], n + diffSym c,
                     if f n then SOME(bs, c_cs) else lngstOpt,
                     cs)
      in long(nil, 0, NONE, w) end

(* val longestNonPositivePrefix : str -> str * str

   if w is an str of zeros and ones, then longestNonPositivePrefix w
   returns (x, y), where x is the longest prefix such that diff x <= 0
   and y is such that x @ y = w *)

val longestNonPositivePrefix = longestPrefix(fn n => n <= 0)

(* val splitPositiveValid : str -> str * str

   if w is an str of zeros and ones such that diff w >= 1 and w is in
   Y, then splitPositiveValid w returns a pair (x, y) such that w = x @
   [one] @ y and x, y are in Y *)

fun splitPositiveValid (w : str) : str * str =
      let val (x, z) = longestNonPositivePrefix w
      in (x, tl z) end
```

```
(* val explain : str -> expl

   if w is in Y, then strExplained(explain w) = w *)

fun explain (w : str) =
      if null w
        then Rule1
      else if isZero(hd w)
        then raise Fail "shouldn't happen"
      else (* isOne(hd w) *)
           let val s = tl w  (* w = [one] @ s *)
           in if validStrSilent s  (* if s is in Y *)
              then Rule4(Rule2, explain s)
              else (* w is not in Y *)
                   let val (z, t) = shortestNegativePrefix s
                       (* s = z @ t *)
                       val u      = Str.allButLast z
                       (* z = u @ [zero], diff u = 1, u is in Y
                          w = [one] @ u @ [zero] @ t, t is in Y *)
                       val (x, y) = splitPositiveValid u
                       (* u = x @ [one] @ y, x is in Y, y is in Y, t is in Y
                          w = ([one] @ x @ [one] @ y @ [zero]) @ t *)
                   in Rule4(Rule3(explain x, explain y), explain t) end
           end
```

And here is how `explain` was tested:

```
- use "ps2-framework.sml";
[opening ps2-framework.sml]
exception Error
val zero = - : sym
val one = - : sym
val isZero = fn : sym -> bool
val isOne = fn : sym -> bool
val diffSym = fn : sym -> int
val diff = fn : str -> int
val validStr = fn : bool -> str -> bool
datatype expl = Rule1 | Rule2 | Rule3 of expl * expl | Rule4 of expl * expl
val strExplained = fn : expl -> str
val printExplanation = fn : expl -> unit
val test = fn : (str -> expl) -> str -> unit
val it = () : unit
- use "ps2-explain.sml";
[opening ps2-explain.sml]
val validStrSilent = fn : str -> bool
val shortestPrefix = fn : (int -> bool) -> str -> str * str
val shortestNegativePrefix = fn : str -> str * str
val longestPrefix = fn : (int -> bool) -> str -> str * str
```

```
val longestNonPositivePrefix = fn : str -> str * str
val splitPositiveValid = fn : str -> str * str
val explain = fn : str -> expl
val it = () : unit
- val doit = test explain;
val doit = fn : str -> unit
- doit(Str.fromString "%");
% is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "1");
1 = 1 @ % is in X, by rule (4)
  1 is in X, by rule (2)
  % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "11");
11 = 1 @ 1 is in X, by rule (4)
  1 is in X, by rule (2)
  1 = 1 @ % is in X, by rule (4)
    1 is in X, by rule (2)
    % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "110");
110 = 110 @ % is in X, by rule (4)
  110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
    % is in X, by rule (1)
    % is in X, by rule (1)
  % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "110110");
110110 = 110 @ 110 is in X, by rule (4)
  110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
    % is in X, by rule (1)
    % is in X, by rule (1)
  110 = 110 @ % is in X, by rule (4)
    110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
      % is in X, by rule (1)
      % is in X, by rule (1)
    % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "11101110");
11101110 = 1 @ 1101110 is in X, by rule (4)
  1 is in X, by rule (2)
  1101110 = 110 @ 1110 is in X, by rule (4)
    110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
      % is in X, by rule (1)
      % is in X, by rule (1)
    1110 = 1 @ 110 is in X, by rule (4)
      1 is in X, by rule (2)
```

```
      110 = 110 @ % is in X, by rule (4)
        110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
          % is in X, by rule (1)
          % is in X, by rule (1)
        % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "111011100");
111011100 = 111011100 @ % is in X, by rule (4)
  111011100 = 1 @ 110 @ 1 @ 110 @ 0 is in X, by rule (3)
    110 = 110 @ % is in X, by rule (4)
      110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
        % is in X, by rule (1)
        % is in X, by rule (1)
      % is in X, by rule (1)
    110 = 110 @ % is in X, by rule (4)
      110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
        % is in X, by rule (1)
        % is in X, by rule (1)
      % is in X, by rule (1)
  % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "111011111010");
111011111010 = 1 @ 11011111010 is in X, by rule (4)
  1 is in X, by rule (2)
  11011111010 = 110 @ 11111010 is in X, by rule (4)
    110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
      % is in X, by rule (1)
      % is in X, by rule (1)
    11111010 = 1 @ 1111010 is in X, by rule (4)
      1 is in X, by rule (2)
      1111010 = 1 @ 111010 is in X, by rule (4)
        1 is in X, by rule (2)
        111010 = 111010 @ % is in X, by rule (4)
          111010 = 1 @ 110 @ 1 @ % @ 0 is in X, by rule (3)
            110 = 110 @ % is in X, by rule (4)
              110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
                % is in X, by rule (1)
                % is in X, by rule (1)
              % is in X, by rule (1)
            % is in X, by rule (1)
          % is in X, by rule (1)
val it = () : unit
- doit(Str.fromString "1111011111001");
1111011111001 = 1 @ 111011111001 is in X, by rule (4)
  1 is in X, by rule (2)
  111011111001 = 1 @ 11011111001 is in X, by rule (4)
    1 is in X, by rule (2)
    11011111001 = 110 @ 11111001 is in X, by rule (4)
```

```
      110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
        % is in X, by rule (1)
        % is in X, by rule (1)
      11111001 = 1 @ 1111001 is in X, by rule (4)
        1 is in X, by rule (2)
        1111001 = 111100 @ 1 is in X, by rule (4)
          111100 = 1 @ % @ 1 @ 110 @ 0 is in X, by rule (3)
            % is in X, by rule (1)
            110 = 110 @ % is in X, by rule (4)
              110 = 1 @ % @ 1 @ % @ 0 is in X, by rule (3)
                % is in X, by rule (1)
                % is in X, by rule (1)
              % is in X, by rule (1)
          1 = 1 @ % is in X, by rule (4)
            1 is in X, by rule (2)
            % is in X, by rule (1)
val it = () : unit
```