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

Problem Set 2

Due by 12:30pm on Thursday, October 4

You must submit your problem set solution as a hard copy, either: at the* beginning *of class; or, no later than 12:05pm, via the CS Department drop box labeled "CS 591 S2". In addition, see the instructions in Problem 2 for emailing the Forlan code for Problem 2 to me, no later than 12:30pm.

Problem 1 (60 points)

Define a function $\mathbf{diff} \in \{\mathsf{0},\mathsf{1}\}^* \to \mathbb{Z}$ by: for all $w \in \{\mathsf{0},\mathsf{1}\}^*$,

$$\mathbf{diff}\, w = \text{the number of } \mathsf{1}\text{'s in } w - \text{the number of } \mathsf{0}\text{'s in } w.$$

Thus:

- $\mathbf{diff}\, \% = 0$;
- $\mathbf{diff}\, \mathsf{1} = 1$;
- $\mathbf{diff}\, \mathsf{0} = -1$; and
- for all $x, y \in \{\mathsf{0},\mathsf{1}\}^*$, $\mathbf{diff}\,(xy) = \mathbf{diff}\, x + \mathbf{diff}\, y$.

And, $\mathbf{diff}\, w = 0$ iff w has an equal number of $\mathsf{0}$'s and $\mathsf{1}$'s.

Let X be the least subset of $\{\mathsf{0},\mathsf{1}\}^*$ such that:

(1) $\% \in X$;

(2) for all $x, y \in X$, $x\mathsf{0}y\mathsf{1} \in X$;

Let $Y = \{\, w \in \{\mathsf{0},\mathsf{1}\}^* \mid \mathbf{diff}\, w = 0 \text{ and, for all prefixes } v \text{ of } w, \mathbf{diff}\, v \leq 0 \,\}$.

(a) Use induction on X to prove that $X \subseteq Y$. [20 points]

(b) Use strong string induction to prove that $Y \subseteq X$. Your proof should be "constructive" in the sense that an algorithm for explaining why elements of Y are in X can be extracted from it. [40 points]

Problem 2 (40 points)

The context for this problem is Problem 1 and the Forlan/SML file `ps2-framework.sml` (see the course website). Among the definitions in this file are the following datatype and functions:

```
datatype expl =
           Rule1                  (* % *)
         | Rule2 of expl * expl   (* x0y1 *)
val printExplanation : expl -> unit
val test             : (str -> expl) -> str -> unit
```

A value of type `expl` explains why a string is in X. The two constructors correspond to the two rules of X's definition.

- `Rule1` explains that $\% \in X$ because of rule (1) of X's definition.
- If $expl_1$ and $expl_2$ have type `expl`, then `Rule2`$(expl_1, expl_2)$ explains that $x_1\mathtt{0}x_2\mathtt{1} \in X$ because of rule (2), where x_1 is the string whose membership in X is explained by $expl_1$, and x_2 is the string whose membership in X is explained by $expl_2$.

E.g.,

```
Rule2(Rule2(Rule1, Rule2(Rule2(Rule1, Rule1), Rule1)), Rule2(Rule1, Rule1))
```

explains why the string 0010110011 is in X:

```
0010110011 = 001011 @ 0 @ 01 @ 1 is in X, by rule (2)
  001011 = % @ 0 @ 0101 @ 1 is in X, by rule (2)
    % is in X, by rule (1)
    0101 = 01 @ 0 @ % @ 1 is in X, by rule (2)
      01 = % @ 0 @ % @ 1 is in X, by rule (2)
        % is in X, by rule (1)
        % is in X, by rule (1)
      % is in X, by rule (1)
  01 = % @ 0 @ % @ 1 is in X, by rule (2)
    % is in X, by rule (1)
    % is in X, by rule (1)
```

The function `printExplanation` turns elements of `expl` into such human-readable explanations.

Your job is to define a function

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

that, when given an element w of Y, returns a value of type `expl` that explains why w is in X. (When called with a w that is not in Y, it doesn't matter what your function returns, or even whether it returns.)

As closely as possible, make the structure of your function definition match the structure of the proof you gave in Problem 1(b). In particular: induction in the proof should correspond to recursion in your function definition; and division into cases in the proof should correspond to the use of conditionals/pattern matching in the function definition.

You can test your definition of `explain` using the function `test`. If w is in Y, then `test explain` w evaluates `printExplanation(explain` w); otherwise, it explains why w is not in Y. E.g., you can proceed as follows:

```
val doit = test explain;
doit(Str.fromString "%");
doit(Str.fromString "01");
doit(Str.fromString "00011011");
```

and so on.

Your solution should reside in a file called `ps2-explain.sml`. This file should not include—either textually or via a call to `use`—the contents of `ps2-framework.sml`. Instead, you should load (using `use`) `ps2-framework.sml` once at the beginning of a Forlan session.

Please email your `ps2-explain.sml` to me (`stough@bu.edu`) as a plain text attachment, in addition to including a listing of it as part of your solution to the problem set. The subject line of your message should include the text "[591S2:PS2]". Also include a Forlan transcript, showing how you tested your `explain` function.