

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

Problem Set 2

Due by 12:30pm on Thursday, October 3

Problem 1 (60 points)

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

$$\mathbf{diff} w = \text{the number of 1's in } w - 2(\text{the number of 0's in } w).$$

Thus

- $\mathbf{diff} \% = 0$;
- $\mathbf{diff} 0 = -2$;
- $\mathbf{diff} 1 = 1$;
- for all $x, y \in \{0,1\}^*$, $\mathbf{diff}(xy) = \mathbf{diff} x + \mathbf{diff} y$.

And, for all $w \in \{0,1\}^*$, $\mathbf{diff} w = 0$ iff w has twice as many 1's as 0's.

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

- (1) $\% \in X$;
- (2) $1 \in X$;
- (3) for all $x, y \in X$, $1x1y0 \in X$;
- (4) for all $x, y \in X$, $xy \in X$.

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

(a) Prove that $X \subseteq Y$. Hint: use induction on X . [20 points]

(b) Prove that $Y \subseteq X$, completing the proof that $X = Y$. Hint: use strong string induction. [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                (* 1 *)
| Rule3 of expl * expl (* 1x1y0 *)
| Rule4 of expl * expl (* xy *)
val printExplanation : expl -> unit
val test              : (str -> expl) -> str -> unit
```

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

- `Rule1` explains that $\% \in X$ because of rule (1) of X 's definition;
- `Rule2` explains that $1 \in X$ because of rule (2) of X 's definition;
- if $expl_1$ and $expl_2$ have type `expl`, then `Rule3(expl1, expl2)` explains that $1x_11x_20 \in X$ because of rule (3) of X 's definition, 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 $expl_2$;
- if $expl_1$ and $expl_2$ have type `expl`, then `Rule4(expl1, expl2)` explains that $x_1x_2 \in X$ because of rule (4) of X 's definition, 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.,

```
Rule4(Rule3(Rule3(Rule2, Rule1), Rule4(Rule2, Rule3(Rule2, Rule1))), Rule2)
```

explains why the string `1111011111001` is in X :

```
1111011111001 = 111101111100 @ 1 is in X, by rule (4)
  111101111100 = 1 @ 1110 @ 1 @ 1110 @ 0 is in X, by rule (3)
    1110 = 1 @ 1 @ 1 @ % @ 0 is in X, by rule (3)
      1 is in X, by rule (2)
      % is in X, by rule (1)
    1110 = 1 @ 1110 is in X, by rule (4)
      1 is in X, by rule (2)
      1110 = 1 @ 1 @ 1 @ % @ 0 is in X, by rule (3)
```

```
1 is in X, by rule (2)
% is in X, by rule (1)
1 is in X, by rule (2)
```

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; division into cases in the proof should correspond to the use of conditionals/pattern matching in the function definition; and the use of lemmas in your proof should correspond to the use of auxiliary functions in your function definition.

You can test your definition of `explain` using the function `test`. If w is not in Y , then `test explain w` explains why w is not in Y . Otherwise it calls `explain` on w . If the resulting explanation explains why another string is in X , `explain` notes that fact. Otherwise it calls `printExplanation` with the explanation. E.g., you can proceed as follows:

```
val doit = test explain;
doit(Str.fromString "%");
doit(Str.fromString "110110");
doit(Str.fromString "11111100");
```

and so on. You should do enough testing to give yourself reasonable confidence that your function definition is correct.

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.