CS 198:205 Fall 2016

Homework 6

DUE: **SUNDAY, November 6th, 2016** at 11:00am

A. Suppose you are given a set E. Here is a definition of SEQ_{Rosen}(E) over E:

1. *nil* is in SEQ_{Rosen}(E);
2. if $e \in$ E and $z \in SEQ_{Rosen}$(E) then $[z, e] \in SEQ_{Rosen}$(E)
3. nothing else is in SEQ_{Rosen}(E).

This definition is almost identical to $SEQ(E)$ given in class, except that the recursion case reverses the order of the elements, adding V-elements to the right, not to the left. (It was inspired by the book's definition of strings.) So, for E = $\{a, b, c\}$, the following would be three elements of SEQ_{Rosen}(E):
nil , $[nil, a]$, $[[nil, a], b]$.

Please read the handout on sequences posted on the lecture notes web page. You are now going to repeat some of the things in the lecture, but with SEQ_{Rosen}(E) instead of $SEQ(E)$. You **must** work with the definition of sequences, not strings.

(i) define concatenation CONCAT, and length LENGTH over SEQ_{Rosen}(E).

(ii) prove the equivalent of Theorem 1 for your definitions: *"For any E and any $w \in SEQ_{Rosen}(E)$, $y \in SEQ_{Rosen}(E)$, it is the case that* LENGTH(CONCAT(w, y)) = LENGTH(w) + LENGTH(y)*"*.

(iii) Do the equivalent of Exercise 1 in the handout, concerning CountBs, for SEQ_{Rosen}(E). (Check the updated lecture notes for Sequences, to see Example 0 (in red), which gives you a hint on how to define CountBs.)

B*. [Much more challenging - the grading will not be detailed]

This question looks at yet another definition of what might be called strings. For this, you might first want to look at my notes on strings, which I have moved to sakai Resources.

Suppose you are given the following inductive definition of $\Sigma^{@}$, based alphabet Σ:

- BASIS 1: λ is in $\Sigma^{@}$;
- BASIS 2: σ is in $\Sigma^{@}$ for every element σ of Σ;
- RECURSION: $(x.y)$ is in $\Sigma^{@}$ whenever x and y are both in $\Sigma^{@}$;
- nothing else is in $\Sigma^{@}$.

So, for $\Sigma_3 = \{a, b, c\}$, the following would be some of the elements of $\Sigma_3^{@}$: λ , a , $(a.\lambda)$, $(a.b)$, $(\lambda.\lambda)$, $(((a.\lambda).a).b)$, $(a.((\lambda.a).b)$. Note that the parentheses are needed because $a.b.c$ is ambiguous, since it could be $((a.b).c)$ or $(a.(b.c))$. And we can no longer drop λ to abbreviate things.

The *intuitive notion of equality* we want for $\Sigma^{@}$ values v and w is that v=$(((a.\lambda).a).b)$ and w=$(a.((\lambda.a).b)$ are equal because the actual letters are the same in left-to-right order ("aab" in this case). Clearly, in $\Sigma^{@}$ one needs to work harder to get the notion of equality to correspond to our intuition Give a recursive definition of the function $equals(v, w)$, which returns a boolean value indicating if v and w are equal in the above sense. (For example, all of the following $(b.\lambda)$, $(\lambda.b)$, $(b.(\lambda.\lambda)$, $((\lambda.b).\lambda)$, $((\lambda.\lambda).b)$, ... should be *equals* to b).