Context-Free Grammars

This exercise is concerned with context-free grammars (CFGs). Please read Section 7.4 in
the tutorial which explains how to model CFGs as inductive definitions. Our particular
example is about defining valid sequences of parentheses.

Two grammars

The most natural definition of valid sequences of parentheses is this:

\[ S \rightarrow \varepsilon \mid (' S ') \mid S S \]

where \( \varepsilon \) is the empty word.

A second, somewhat unusual grammar is the following one:

\[ T \rightarrow \varepsilon \mid T (' T ') \]

Model both grammars as inductive sets \( S \) and \( T \) and prove \( S = T \).

The alphabet:

\textbf{datatype} \hspace{1em} \texttt{alpha = A \mid B}

Standard grammar:

\textbf{inductive} \hspace{1em} \text{set} \hspace{1em} \texttt{S :: "alpha list set" where}

\texttt{S1: "[] : S"} \hspace{1em} | \hspace{1em}
\texttt{S2: "w : S \Rightarrow A#w@[B] : S"} \hspace{1em} | \hspace{1em}
\texttt{S3: "v : S \Rightarrow w : S \Rightarrow v @ w : S"}

\texttt{declare \hspace{1em} S1 [iff] S2[intro!,simp]}

Nonstandard grammar:

\textbf{inductive} \hspace{1em} \text{set} \hspace{1em} \texttt{T :: "alpha list set" where}

\texttt{T1: "[] : T"} \hspace{1em} | \hspace{1em}
\texttt{T23: "v : T \Rightarrow w : T \Rightarrow v \circ A \# w \circ [B] : T"}
A recursive function

Instead of a grammar, we can also define valid sequences of parentheses via a test function: traverse the word from left to right while counting how many closing parentheses are still needed. If the counter is 0 at the end, the sequence is valid.

Define this recursive function and prove that a word is in $S$ iff it is accepted by your function. The $\Rightarrow$ direction is easy, the other direction more complicated.

fun balanced :: "alpha list ⇒ nat ⇒ bool" where
  "balanced [] 0 = True"
| "balanced (A#w) n = balanced w (Suc n)"
Correctness of the recognizer w.r.t. $S$:

```text
lemma [simp]: "balanced $w$ $n$ $\Rightarrow$ balanced ($w@[B]$) $n"
  apply (induct $w$ $n$ rule: balanced.induct)
  apply simp_all
done

lemma [simp]: "[balanced $v$ $n$; balanced $w$ 0] $\Rightarrow$ balanced ($v @ w$) $n"
  apply (induct $v$ $n$ rule: balanced.induct)
  apply simp_all
done

lemma "$w : S$ $\Rightarrow$ balanced $w$ 0"
  apply (erule S.induct)
  apply simp_all
done
```

Completeness of the recognizer w.r.t. $S$:

```text
lemma [iff]: "[A,B] : S"
  using S2[where $w = "[]"]$ by simp

lemma AB: assumes u: "u $\in$ S" shows "$\forall v . u = v@[w] $\Rightarrow$ v $@$ A $@$ B $@$ w $\in$ S"
using u
proof
  case S1 thus ?case by simp
next
  case (S2 u)
  have uS: "u $\in$ S" and
    IH: "$\forall v . u = v $@$ w $\Rightarrow$ v $@$ A $@$ B $@$ w $\in$ S" and
    asm: "A $@$ u $@$ [B] = v $@$ w" by fact+
  show "v $@$ A $@$ B $@$ w $\in$ S"
    proof (cases $v$)
      case Nil
      hence "$w = A $@$ u $@$ [B]" using asm by simp
      hence "$w $\in$ S" using uS by simp
      hence "[A,B] $@$ w $\in$ S" by(blast intro:S3)
      thus ?thesis using Nil by simp
    next
      case (Cons x $v'$)
      show ?thesis
    proof (cases $w$ rule:rev_cases)
```

3
case Nil
from uS have "(A # u @ [B]) @ [A,B] ∈ S" by(blast intro:S3)
thus ?thesis using Nil Cons asm by auto
next
case (snoc w' y)
hence u: "u = v' @ w'" and [simp]: "x = A & y = B"
using Cons asm by auto
from u have "v' @ A # B # w' ∈ S" by(rule IH)
hence "A # (v' @ A # B # w') @ [B] ∈ S" by(rule S.S2)
thus ?thesis using Cons snoc by auto
qed
qed
next
case (S3 v' w')
have v'S: "v' ∈ S" and w'S: "w' ∈ S"
and IHv: "\(\forall w. v' = v @ w \rightarrow v @ A # B # w \in S\)"
and IHw: "\(\forall w. w' = v @ w \rightarrow v @ A # B # w \in S\)"
and asm: "v' @ w' = v @ w" by fact+
then obtain r where "v' = v @ r \& r @ w' = w \lor v' @ r = v \land w' = r @ w" 
(is "?A \lor ?B")
by (auto simp:append_eq_append_conv2)
thus "v @ A # B # w ∈ S"
proof
assume A: ?A
hence "v @ A # B # r ∈ S" using IHv by blast
hence "(v @ A # B # r) @ w' ∈ S" using w'S by(rule S.S3)
thus ?thesis using A by auto
next
assume B: ?B
hence "r @ A # B # w ∈ S" using IHw by blast
with v'S have "v' @ (r @ A # B # w) ∈ S" by(rule S.S3)
thus ?thesis using B by auto
qed
qed

The same lemma for friends of the apply style:

lemma "u ∈ S \rightarrow ALL v w. u = v@w \rightarrow v @ A # B # w ∈ S"
apply(erule S.induct)
  apply simp
  apply(rename_tac u)
  apply(clarsimp simp:Cons_eq_append_conv)
  apply(rule conjI)
  apply(clarsimp)
apply (subgoal_tac "[A,B] @ (A # u @ [B]) : S")
  apply (simp)
  apply (blast intro:S3)
apply (clar simp simp:append_eq_append_conv2 Cons_eq_append_conv)
apply (rename_tac w w1 w2)
apply (erule disjE)
  apply clar simp
  apply (subgoal_tac "A # (w1 @ A # B # w2) @ [B] : S")
    apply simp
  apply (blast intro:S3)
apply clar simp
apply (erule disjE)
  apply clar simp
  apply (subgoal_tac "A # (u @ [A,B]) @ [B] : S")
    apply (simp)
  apply (blast intro:S3)
apply clar simp
apply (subgoal_tac "(A # u @ [B]) @ [A,B] : S")
  apply (simp)
  apply (blast intro:S3)
apply (clar simp simp:append_eq_append_conv2)
apply (rename_tac u v w x y)
apply (erule disjE)
  apply clar simp
  apply (subgoal_tac "(w @ A # B # y) @ v : S")
    apply (simp)
  apply (blast intro:S3)
apply clar simp
apply (blast intro:S3)
done

lemma "balanced w n n A @ w : S"
  apply (induct w n rule: balanced.induct)
  apply simp_all
    apply (simp add: replicate_app_Cons_same)
  apply (simp add: AB replicate_app_Cons_same[symmetric])
done