In this exercise, you are going to add lists to the call-by-value simply-typed lambda calculus (with booleans). A list is defined recursively as either the value nil or the value (cons \( v_1 \) \( v_2 \)), where \( v_2 \) is itself a list. For example, the list consisting of the booleans true, false, and true (in that order) would be represented by the following list value:

\[
(\text{cons true (cons false (cons true nil Bool))})
\]

The above list value has type Bool List. The value nil is annotated with the element type expected for this occurrence of the empty list. This annotation is necessary so that the type system doesn’t have to “guess” the correct type of an empty list. The new primitives head and tail access the first and second components of a cons cell, respectively.

The new cases of the language’s syntax are as follows:

\[
e \ ::= \ . . . \noalign{\smallskip} \text{wrong: } T \noalign{\smallskip} \text{nil}[T] \noalign{\smallskip} \text{cons } e_1 \ e_2 \noalign{\smallskip} \text{head } e \noalign{\smallskip} \text{tail } e
\]

\[
T \ ::= \ . . . \noalign{\smallskip} T \ \text{List}
\]

The expression denoted wrong: \( T \) is a new value that is used to represent errors that will not be prevented by the static type system and can therefore not be considered stuck. In particular, the type system will not ensure that head and tail are always invoked on a non-empty list. The expression wrong is annotated with a type so that the type system does not need to “guess” it (see the typing rule T-Wrong below).

## 1 Values

Show the new cases of the grammar for values. I’ve provided one of them.

I would accept this grammar:

\[
v \ ::= \ . . . \noalign{\smallskip} \text{wrong: } T \noalign{\smallskip} \text{nil}[T] \noalign{\smallskip} \text{cons } v_1 \ v_2
\]

I would also accept this one:

\[
v \ ::= \ . . . \noalign{\smallskip} \text{wrong: } T \noalign{\smallskip} l
\]

\[
l \ ::= \ \text{nil}[T] \noalign{\smallskip} \text{cons } v \ l
\]
2 Operational Semantics

Show the new rules of the operational semantics. I’ve provided the rules for head; you should add the other rules.

\[
\frac{e_1 \rightarrow e'_1}{\text{cons } e_1 e_2 \rightarrow \text{cons } e'_1 e_2} \quad (\text{E-Cons1})
\]

\[
\frac{e_2 \rightarrow e'_2}{\text{cons } v_1 e_2 \rightarrow \text{cons } v_1 e'_2} \quad (\text{E-Cons2})
\]

\[
\frac{e \rightarrow e'}{\text{head } e \rightarrow \text{head } e'} \quad (\text{E-Head})
\]

\[
\frac{\text{head } \text{nil}[T] \rightarrow \text{wrong} : T}{\text{(E-HeadNil)}}
\]

\[
\frac{\text{head } (\text{cons } v_1 v_2) \rightarrow v_1}{\text{(E-HeadCons)}}
\]

\[
\frac{e \rightarrow e'}{\text{tail } e \rightarrow \text{tail } e'} \quad (\text{E-Tail})
\]

\[
\frac{\text{tail } \text{nil}[T] \rightarrow \text{wrong} : T \text{ List}}{\text{(E-TailNil)}}
\]

\[
\frac{\text{tail } (\text{cons } v_1 v_2) \rightarrow v_2}{\text{(E-TailCons)}}
\]

Technically, we also need the following rules, to handle reductions involving the “wrong” value:

\[
\frac{\text{head } (\text{wrong} : T \text{ List}) \rightarrow \text{wrong} : T}{\text{(E-HeadWrong)}}
\]

\[
\frac{\text{tail } (\text{wrong} : T \text{ List}) \rightarrow \text{wrong} : T \text{ List}}{\text{(E-TailWrong)}}
\]

In addition, we should also add rules to handle the appearance of “wrong” in other parts of the syntax. For example, we should add an E-AppWrong rule to handle the case when the function value in a function call ends up being the “wrong” value. This is annoying, because it means we also have to revisit the cases of the proof involving function applications, to make sure this new rule is safe. A similar thing would be required for the if expression. I ignore these extra rules here for simplicity.

3 Typing Rules

Show the new typing rules. I’ve provided the rules for wrong and head; you should add the rules for nil, cons and tail.

\[
\frac{\Gamma \vdash (\text{wrong} : T) : T}{\text{(T-Wrong)}}
\]

\[
\frac{\Gamma \vdash \text{nil}[T] : T \text{ List}}{\text{(T-Nil)}}
\]
\[
\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T \text{ List}}{\Gamma \vdash \text{cons } e_1 \ e_2 : T \text{ List}} \quad \text{(T-Cons)}
\]

\[
\frac{\Gamma \vdash e : T \text{ List}}{\Gamma \vdash \text{head } e : T} \quad \text{(T-Head)}
\]

\[
\frac{\Gamma \vdash e : T \text{ List}}{\Gamma \vdash \text{tail } e : T \text{ List}} \quad \text{(T-Tail)}
\]

## 4 Type Soundness

a. State (no proof necessary) any new parts of the Canonical Forms lemma that are necessary in the proofs below.

**Lemma** (Canonical Forms): If \( \Gamma \vdash v : T \text{ List} \), then \( v \) has the form \( \text{nil}[T] \), (\( \text{cons } v_1 \ v_2 \)) , or (\( \text{wrong: } T \text{ List} \)).

b. Add the relevant cases to the proof of the Progress theorem. I’ve provided the cases for wrong and head; you should add the cases for nil, cons and tail.

**Theorem** (Progress): If \( \vdash e : T \), then either \( e \) is a value or there exists \( e' \) such that \( e \rightarrow e' \).

- Case T-Wrong: Then \( e = \text{wrong: } T \), so \( e \) is a value.
- Case T-Nil: Then \( e = \text{nil}[T'] \), so \( e \) is a value.
- Case T-Cons: Then \( e = \text{cons } e_1 \ e_2 \) and \( T = T' \text{ List} \) and \( \vdash e_1 : T' \) and \( \Gamma \vdash e_2 : T' \text{ List} \). By the inductive hypothesis, we have that either \( e_1 \) is a value or there exists \( e'_1 \) such that \( e_1 \rightarrow e'_1 \). Similarly, either \( e_2 \) is a value or there exists \( e'_2 \) such that \( e_2 \rightarrow e'_2 \). We perform a case analysis on these possibilities:
  - Case there exists \( e'_1 \) such that \( e_1 \rightarrow e'_1 \): Then by E-Cons1 we have \( \text{cons } e_1 \ e_2 \rightarrow \text{cons } e'_1 \ e_2 \).
  - Case \( e_1 \) is a value \( v_1 \): There are two sub-cases.
    * Case there exists \( e'_2 \) such that \( e_2 \rightarrow e'_2 \): Then by E-Cons2 we have \( \text{cons } v_1 \ e_2 \rightarrow \text{cons } v_1 \ e'_2 \).
    * Case \( e_2 \) is a value \( v_2' \): Then \( e = \text{cons } v_1 \ v_2' \), so \( e \) is a value.
- Case T-Head: Then \( e = \text{(head } e_1 \text{) and } \vdash e_1 : T \text{ List} \). By the inductive hypothesis, \( e_1 \) is either a value or there exists \( e'_1 \) such that \( e_1 \rightarrow e'_1 \). In the latter case, by E-Head we have \( \text{(head } e_1 \text{) } \rightarrow \text{(head } e'_1 \text{) } \). In the former case, by the Canonical Forms Lemma (assuming you wrote it properly) we have that \( e_1 \) is either \( \text{nil}[T'] \), (\( \text{cons } v_1 \ v_2 \)), or (\( \text{wrong: } T \text{ List} \)). If \( e_1 \) is \( \text{nil}[T] \), then by E-HeadNil we have \( e \rightarrow \text{wrong: } T \). If \( e_1 \) has the form (\( \text{cons } v_1 \ v_2 \)), then by E-HeadCons we have \( e \rightarrow v_1 \). If \( e_1 \) has the form (\( \text{wrong: } T \text{ List} \)), then by E-HeadWrong we have \( e \rightarrow \text{wrong: } T \).
- Case T-Tail: Then \( e = \text{(tail } e_1 \text{) and } T = T' \text{ List} \) and \( \vdash e_1 : T' \text{ List} \). By the inductive hypothesis, \( e_1 \) is either a value or there exists \( e'_1 \) such that \( e_1 \rightarrow e'_1 \). In the latter case, by E-Tail we have \( \text{(tail } e_1 \text{) } \rightarrow \text{(tail } e'_1 \text{) } \). In the former case, by the Canonical Forms Lemma we have that \( e_1 \) is either \( \text{nil}[T'] \), (\( \text{cons } v_1 \ v_2 \)), or (\( \text{wrong: } T' \text{ List} \)). If \( e_1 \) is \( \text{nil}[T'] \), then by E-TailNil we have \( e \rightarrow \text{wrong: } T' \text{ List} \). If \( e_1 \) has the form (\( \text{cons } v_1 \ v_2 \)), then by E-TailCons we have \( e \rightarrow v_2 \). If \( e_1 \) has the form (\( \text{wrong: } T' \text{ List} \)), then by E-HeadWrong we have \( e \rightarrow \text{wrong: } T' \text{ List} \).

c. Add the relevant cases to the proof of the Type Preservation theorem. I’ve provided the cases for wrong and head; you should add the cases for nil, cons and tail.

**Theorem** (Type Preservation): If \( \Gamma \vdash e : T \) and \( e \rightarrow e' \), then \( \Gamma \vdash e' : T \).
Case T-Wrong: Then $e = \text{wrong}; T$. By inspection, there is no $e'$ such that wrong: $T \rightarrow e'$ (assuming you didn’t add such a rule), so this case is satisfied trivially.

Case T-Nil: Then $e = \text{nil}(T')$. By inspection, there is no $e'$ such that $\text{nil}(T') \rightarrow e'$, so this case is satisfied trivially.

Case T-Cons: Then $e = \text{cons} \ e_1 \ e_2$ and $T = T'$ List and $\Gamma \vdash e_1 : T'$ and $\Gamma \vdash e_2 : T'$ List. We’re given that $e \rightarrow e'$. Case analysis of the last rule used in the derivation of this reduction step:

- Case E-Cons1: Then $e' = \text{cons} \ e_1' \ e_2$ and $e_1 \rightarrow e_1'$. By the inductive hypothesis we have that $\Gamma \vdash e_1' : T'$. Therefore, by T-Cons we have $\Gamma \vdash \text{cons} \ e_1' \ e_2 : T$.
- Case E-Cons2: Then $e' = \text{cons} \ e_1' \ e_2$ and $e_2 \rightarrow e_2'$. By the inductive hypothesis we have that $\Gamma \vdash e_2' : T'$ List. Therefore, by T-Cons we have $\Gamma \vdash \text{cons} \ e_1 \ e_2' : T$.

Case T-Head: Then $e = \text{(head} \ e_1\text{)}$ and $\Gamma \vdash e_1 : T$ List. We’re given that $e \rightarrow e'$. Case analysis of the last rule used in the derivation of this reduction step:

- Case E-Head: Then $e' = \text{head} e_1'$ and $e_1 \rightarrow e_1'$. By the inductive hypothesis we have that $\Gamma \vdash e_1' : T$ List. Therefore, by T-Head we have $\Gamma \vdash e' : T$.
- Case E-HeadNil: Then $e_1 = \text{nil}(T)$ and $e' = \text{wrong} ; T'$. Since $\Gamma \vdash e_1 : T$ List, by T-Nil (assuming you wrote it properly) we have that $T' = T$. Then by T-Wrong we have $\Gamma \vdash (\text{wrong} ; T) : T$.
- Case E-HeadCons: Then $e_1 = (\text{cons} \ v_1 \ v_2)$ and $e' = v_1$. Since $\Gamma \vdash e_1 : T$ List, by T-Cons (assuming you wrote it properly) we have that $\Gamma \vdash v_1 : T$.
- Case E-HeadWrong: Then $e_1 = \text{wrong} ; T'$ List and $e' = \text{wrong} ; T'$. Since $\Gamma \vdash e_1 : T$ List, by T-Wrong we have $T' = T$, so again by T-Wrong we have $\Gamma \vdash (\text{wrong} ; T) : T$.

Case T-Tail: Then $e = (\text{tail} \ e_1\text{)}$ and $\Gamma \vdash e_1 : T'$ List. We’re given that $e \rightarrow e'$. Case analysis of the last rule used in the derivation of this reduction step:

- Case E-Tail: Then $e' = \text{tail} e_1'$ and $e_1 \rightarrow e_1'$. By the inductive hypothesis we have that $\Gamma \vdash e_1' : T'$ List. Therefore, by T-Tail we have $\Gamma \vdash e' : T$.
- Case E-TailNil: Then $e_1 = \text{nil}(T'\prime)$ and $e' = \text{wrong} ; T'$. Since $\Gamma \vdash e_1 : T'$ List, by T-Nil we have that $T'' = T'$. Then by T-Wrong we have $\Gamma \vdash (\text{wrong} ; T') : T'$ List.
- Case E-TailCons: Then $e_1 = (\text{cons} \ v_1 \ v_2)$ and $e' = v_2$. Since $\Gamma \vdash e_1 : T'$ List, by T-Cons we have that $\Gamma \vdash v_2 : T'$ List.
- Case E-TailWrong: Then $e_1 = \text{wrong} ; T''$ List and $e' = \text{wrong} ; T''$. Since $\Gamma \vdash e_1 : T'$ List, by T-Wrong we have that $T'' = T'$. Then by T-Wrong we have $\Gamma \vdash (\text{wrong} ; T') : T'$ List.