Exercise 1 : Subtyping and Mutations (10 points)

In this exercise, we study how to extend the system $STLC_{<:}$ plus Top ($\top$) and Bottom ($\bot$) (Appendix A) with mutable references. The syntax and semantics of mutable references are the same as presented during the lectures.

Alice suggests to extend the system with the following subtyping rule in addition to the conventional typing and evaluation rules:

$$\frac{T_1 <: T_2}{\text{Ref } T_1 <: \text{Ref } T_2} \qquad \text{(S-ALICE)}$$

Bob finds that Alice's subtyping rule is unsound. Instead, he proposes the following rule:

$$\frac{T_2 <: T_1}{\text{Ref } T_1 <: \text{Ref } T_2} \qquad \text{(S-BOB)}$$

Problem 1. Find a program that type checks with the subtyping rule S-Alice, but gets stuck at runtime. Write down both the initial term and the term that gets stuck.

Problem 2. Find a program that type checks with the subtyping rule S-Bob, but gets stuck at runtime. Write down both the initial term and the term that gets stuck.

You may assume natural numbers and Boolean values in the system, as well as basic operations on them. You may also assume sequence and let-bindings to make it easier to write programs.

```
Solution to problem 1.

let x = ref 5
let f = \x: Ref[Top]. \y: Top. x := y
in f x false; !x + 3

The term that gets stuck: false + 3


Solution to problem 2.

let x = ref false
let f = \x: Ref[Bot]. !x
in (f x) + 3

false + 3
```

Exercise 2 : Type reconstruction for lists (10 points)

In this exercise, we consider the simply-typed lambda calculus (Appendix B) with booleans and natural numbers (Appendix C) but with no other extensions (in particular, there's no subtyping or `Bot` type). We extend this calculus with primitives for lists and operations on lists with operational semantics provided in Appendix D:

t	::=	...	**Terms**
	|	`nil`	Empty list
	|	`cons t t`	List constructor
	|	`head t`	Head of a list
	|	`tail t`	Tail of a list
	|	`isnil t`	Test for empty list

v	::=	...	**Values**
	|	`nil`	Empty list
	|	`cons v v`	List constructor

T	::=	...	**Types**
	|	`List T`	Type of a list with elements of type `T`

Now, your task is to extend the type system of the original calculus with rules for type reconstruction that accommodate additional syntactic forms, without adding new terms or types to the calculus. In order to fulfill the assignment, do one of the following for the new terms:

- Specify additional cases for the type reconstruction algorithm TP introduced at the lecture of Week 9 of the course.
- *Or* provide additional constraint-based typing rules for the type reconstruction algorithm explained in Chapter 22 of "Types and Programming Languages".

A refresher: `cons`, `head` and `tail` work like in all functional languages. `cons` prepends an element in its first argument to a list in its second argument. `head` cuts the 1st element from a list and returns it. `tail` cuts the 1st element from a list and returns the remaining list. Examples: `head (cons x xs) == x`, `tail (cons x xs) == xs` for all `x` and `xs`.

Solution:

$$\frac{\texttt{T1}\ \textit{fresh}}{\Gamma \vdash \texttt{nil} : \texttt{List T1} \mid \emptyset}$$

$$\frac{\Gamma \vdash t_1 : \texttt{T1} \mid \texttt{C1} \qquad \Gamma \vdash t_2 : \texttt{T2} \mid \texttt{C2}}{\Gamma \vdash \texttt{cons}\ t_1\ t_2 : \texttt{T2} \mid \texttt{C1} \cup \texttt{C2} \cup \{\texttt{List T1} = \texttt{T2}\}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{T1} \mid \texttt{C1} \qquad \texttt{T2}\ \textit{fresh}}{\Gamma \vdash \texttt{head}\ t_1 : \texttt{T2} \mid \texttt{C1} \cup \{\texttt{T1} = \texttt{List T2}\}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{T1} \mid \texttt{C1} \qquad \texttt{T2}\ \textit{fresh}}{\Gamma \vdash \texttt{tail}\ t_1 : \texttt{T1} \mid \texttt{C1} \cup \{\texttt{T1} = \texttt{List T2}\}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{T1} \mid \texttt{C1} \qquad \texttt{T2}\ \textit{fresh}}{\Gamma \vdash \texttt{isnil}\ t_1 : \texttt{Bool} \mid \texttt{C1} \cup \{\texttt{T1} = \texttt{List T2}\}}$$

Exercise 3 : Transitivity of Procedural $F_{<:}$ Subtyping (10 points)

In this problem, we study procedural subtyping in System $F_{<:}$. System $F_{<:}$ is an extension of System F with subtyping of types and bounds on type variables. The types in System $F_{<:}$ are defined as follows:

$$T ::= \mathit{Top} \mid X \mid T \to T \mid \forall X <: T.T$$

One approach to formulating subtyping in $F_{<:}$ is procedural subtyping. Like algorithmic subtyping, procedural subtyping rules naturally translate to an implementation; the distinction is that procedural subtyping is not guaranteed to terminate. The subtyping rules are given as follows:

$$\Gamma \vdash T <: \mathit{Top} \tag{S-Top}$$

$$\Gamma \vdash X <: X \tag{S-TVar-Refl}$$

$$\frac{X <: T \in \Gamma \qquad \Gamma \vdash T <: U}{\Gamma \vdash X <: U} \tag{S-TVar-Trans}$$

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \tag{S-Fun}$$

$$\frac{\Gamma, X <: U \vdash S <: T}{\Gamma \vdash \forall X <: U.S \;<:\; \forall X <: U.T} \tag{S-All}$$

For this problem, we may assume the typing environment Γ to be just a list of type bounds:

$$\Gamma ::= \emptyset \mid \Gamma, X <: T$$

The typing environment Γ is used in the rule S-TVar-Trans, and it is augmented in the rule S-All. For simplicity, in the rule S-All, we require the upper bound of both universal types to be the same type U.

Prove the following theorem in the subtyping system.

Theorem 1 (Transitivity). *If $\Gamma \vdash S <: U$ and $\Gamma \vdash U <: T$, then $\Gamma \vdash S <: T$.*

Proof. Induction on the type derivation $\Gamma \vdash S <: U$. There are five cases.

- S-Top. We have $U = \mathit{Top}$. Invert $\Gamma \vdash \mathit{Top} <: T$, we know $T = \mathit{Top}$. Apply S-Top.
- S-Tvar-Refl. We have $S = U = X$, trivial.
- S-Tvar-Trans. We have $S = X$, $X <: Q \in \Gamma$ and $Q <: U$. By IH, we have $Q <: T$. Now apply S-Tvar-Trans.
- S-Fun. We have $S = S_1 \to S_2$, $U = U_1 \to U_2$, $U_1 <: S_1$ and $S_2 <: U_2$.

  $\Gamma \vdash U_1 \to U_2 <: T$ may only have been derived using S-Top or S-Fun. In the former case, we have $T = \mathit{Top}$ and we apply S-Top. In the latter case we have $T = T_1 \to T_2$ and $T_1 <: U_1, U_2 <: T_2$. By IH, we have $T_1 <: S_1$ and $S_2 <: T_2$. By S-Fun, we have $\Gamma \vdash S <: T$.

- S-ALL. We have $U = \forall X <: Q.U_1$, $S = \forall X <: Q.S_1$ and $\Gamma, X <: Q \vdash S_1 <: U_1$.

  $\Gamma \vdash \forall X <: Q.U_1 < T$, may only have been derived using S-TOP or S-ALL. In the former case $T = Top$ and we apply S-TOP. In the latter case we have $T = \forall X.T_1$ and $\Gamma, X <: Q \vdash U_1 <: T_1$. By IH, we have $S_1 <: T_1$. We can conclude by S-ALL.

$\square$

Appendix A: **STLC with references and sequencing**

Syntax:

$$
\begin{array}{lcll}
t & ::= & & \textbf{terms:} \\
 & | & x & \text{variable} \\
 & | & \lambda x\colon T.\ t & \text{abstraction} \\
 & | & t\ t & \text{application} \\
 & | & \texttt{unit} & \text{unit} \\
 & | & t; t & \text{sequence} \\
 & | & \texttt{ref}\ \ t & \text{reference creation} \\
 & | & !t & \text{dereference} \\
 & | & t := t & \text{assignment} \\
 & | & l & \text{store location} \\
v & ::= & & \textbf{values:} \\
 & | & \lambda x\colon T.\ t & \text{abstraction value} \\
 & | & \texttt{unit} & \text{unit value} \\
 & | & l & \text{store location} \\
T & ::= & & \textbf{types:} \\
 & | & T \to T & \text{function type} \\
 & | & \texttt{Unit} & \text{unit type} \\
 & | & \texttt{Ref}\ \ T & \text{reference type} \\
\mu & ::= & & \textbf{stores:} \\
 & | & \emptyset & \text{empty store} \\
 & | & \mu, l \mapsto v & \text{location binding}
\end{array}
$$

Evaluation rules:

$$\frac{t_1 \mid \mu \longrightarrow t_1' \ \mid \mu'}{t_1\ t_2 \mid \mu \longrightarrow t_1'\ t_2 \mid \mu'} \qquad \text{(E-App1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1\ t_2 \mid \mu \longrightarrow v_1\ t_2' \mid \mu'} \qquad \text{(E-App2)}$$

$$(\lambda x : T_{11}.t_{12})\ v_2 \mid \mu \longrightarrow [x \mapsto v_2]t_{12} \mid \mu \qquad \text{(E-AppAbs)}$$

$$\frac{l \notin dom(\mu)}{\texttt{ref}\ v_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto v_1)} \qquad \text{(E-RefV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\texttt{ref}\ t_1 \mid \mu \longrightarrow \texttt{ref}\ t_1' \mid \mu'} \qquad \text{(E-Ref)}$$

$$\frac{\mu(l) = v}{!l \mid \mu \longrightarrow v \mid \mu} \qquad \text{(E-DerefLoc)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \qquad \text{(E-Deref)}$$

$$l := v_2 \mid \mu \longrightarrow \texttt{unit} \mid [l \mapsto v]\mu \qquad \text{(E-Assign)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1 := t_2 \mid \mu \longrightarrow t_1' := t_2 \mid \mu'} \tag{E-Assign1}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1 := t_2 \mid \mu \longrightarrow v_1 := t_2' \mid \mu'} \tag{E-Assign2}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1; t_2 \mid \mu \longrightarrow t_1'; t_2 \mid \mu'} \tag{E-Seq}$$

$$\texttt{unit}; t_2 \mid \mu \longrightarrow t_2 \mid \mu \tag{E-SeqNext}$$

Typing rules:

$$\frac{x : T \in \Gamma}{\Gamma \mid \Sigma \vdash x : T} \tag{T-Var}$$

$$\frac{\Gamma, x : T_1 \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash \lambda x{:}\, T_1 . t_2 : T_1 \to T_2} \tag{T-Abs}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1\ t_2 : T_{12}} \tag{T-App}$$

$$\Gamma \mid \Sigma \vdash \texttt{unit} :\ \texttt{Unit} \tag{T-Unit}$$

$$\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \texttt{Ref}\ T_1} \tag{T-Loc}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1}{\Gamma \mid \Sigma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ T_1} \tag{T-Ref}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11}}{\Gamma \mid \Sigma \vdash !t_1 : T_{11}} \tag{T-Deref}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \texttt{Unit}} \tag{T-Assign}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Unit} \qquad \Gamma \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash t_1; t_2 : T_2} \tag{T-Seq}$$

Appendix B: **STLC with lists**

t	::=		**Terms**
	|	...	(*STLC terms*)
	|	`nil`	Empty list
	|	`cons t t`	List constructor
	|	`head t`	Head of a list
	|	`tail t`	Tail of a list
	|	`isnil t`	Test for empty list

v	::=		**Values**
	|	...	(*STLC values*)
	|	`nil`	Empty list
	|	`cons v v`	List constructor

T	::=		**Types**
	|	...	(*STLC types*)
	|	`List T`	Type of a list with elements of type `T`

Evaluation rules (omitted STLC rules):

$$\frac{t_1 \longrightarrow t_1'}{\texttt{cons}\ t_1\ t_2 \longrightarrow \texttt{cons}\ t_1'\ t_2} \qquad \text{(E-Cons1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\texttt{cons}\ v_1\ t_2 \longrightarrow \texttt{cons}\ v_1\ t_2'} \qquad \text{(E-Cons2)}$$

$$\texttt{isnil}\ (\texttt{nil}) \longrightarrow \texttt{true} \qquad \text{(E-IsNilNil)}$$

$$\texttt{isnil}\ (\texttt{cons}\ v_1\ v_2) \longrightarrow \texttt{false} \qquad \text{(E-IsNilCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{isnil}\ t_1 \longrightarrow \texttt{isnil}\ t_1'} \qquad \text{(E-IsNil)}$$

$$\texttt{head}\ (\texttt{cons}\ v_1\ v_2) \longrightarrow v_1 \qquad \text{(E-HeadCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{head}\ t_1 \longrightarrow \texttt{head}\ t_1'} \qquad \text{(E-Head)}$$

$$\texttt{tail}\ (\texttt{cons}\ v_1\ v_2) \longrightarrow v_2 \qquad \text{(E-TailCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{tail}\ t_1 \longrightarrow \texttt{tail}\ t_1'} \qquad \text{(E-Tail)}$$

Typing rules (omitted STLC rules):

Typing rules for this calculus constitute the problem statement of exercise 2.