Exercise 1 : Curry-Howard Isomorphism (8 points)

Give proofs of the following propositional formula using the Curry-Howard isomorphism between constructive logic and typed λ-calculus with products and sums (see Appendix A for details).

1. $(A \wedge B) \Rightarrow C \Rightarrow ((C \wedge A) \wedge B)$
2. $(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow (A \vee B) \Rightarrow C$
3. $(A \vee B \Rightarrow C) \Rightarrow ((A \Rightarrow C) \wedge (B \Rightarrow C))$
4. $((A \Rightarrow B \vee C) \wedge (B \Rightarrow D) \wedge (C \Rightarrow D)) \Rightarrow (A \Rightarrow D)$

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`.

Exercise 3 : Subtyping for products (10 points)

The subtyping rule for products can be stated as:

$$\frac{S_1 \ <: \ T_1 \qquad S_2 \ <: \ T_2}{S_1 \times S_2 \ <: \ T_1 \times T_2} \qquad \text{(S-PROD)}$$

In the course you were presented with the inversion lemma for subtyping with function types i.e., S-Arrow. Your task for this exercise is to write a proof for the following theorem for STLC with products and subtyping (see Appendices E and F).

Theorem 1. *If $S_1 \times S_2 <: T$, then either $T = \texttt{Top}$ or else $T = T_1 \times T_2$, with $S_1 <: T_1$ and $S_2 <: T_2$.*

Hint: proof the theorem by induction on the last used subtyping rule. State any lemmas that you use (without proof).

Appendix A: **Curry-Howard Isomorphism**

The *Curry-Howard isomorphism* or *Curry-Howard correspondence* establishes a connection between type systems and logical calculi based on an observation that the ways we build types are structurally similar to the ways we build formulae.

According to the Curry-Howard isomorphism, proofs can be represented as programs and formulae they prove can be represented as types of those programs. Here is a (non-comprehensive) list of some examples of how concepts from constructive logic correspond to concepts from the simply typed lambda calculus.

Constructive logic	Simply typed lambda calculus
Formula	Type
$A \Rightarrow B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
Proof of a formula	Term that inhabits a type

Appendix B: **The simply-typed lambda calculus**

$$
\begin{array}{llll}
t & ::= & & \textbf{terms:} \\
 & | & x & \text{variable} \\
 & | & \lambda x\colon T.\, t & \text{abstraction} \\
 & | & t\ t & \text{application} \\
 \\
v & ::= & & \textbf{values:} \\
 & | & \lambda x\colon T.\, t & \text{abstraction-value} \\
 \\
T & ::= & & \textbf{types:} \\
 & | & T \to T & \text{type of functions}
\end{array}
$$

Evaluation rules:

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

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

$$(\lambda\ x\colon T_1.\ t_1)\ v_2 \longrightarrow [x \to v_2]\ t_1 \qquad \text{(E-AppAbs)}$$

Typing rules:

$$\frac{x\colon T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma,\ x\colon T_1 \vdash t_2\ :\ T_2}{\Gamma \vdash (\lambda\, x\colon T_1.\ t_2) : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2} \qquad \text{(T-App)}$$

Appendix C: **Booleans, natural numbers and unit**

t	::=		**terms :**	v	::=		**values :**
	|	`true`	*constant true*		|	`true`	*true value*
	|	`false`	*constant false*		|	`false`	*false value*
	|	`if` t `then` t `else` t	*condition*		|	`unit`	*unit value*
	|	`unit`	*constant unit*				
	|	`0`	*constant zero*		|	nv	*numeric value*
	|	`succ` t	*successor*	nv	::=		**numeric values :**
	|	`pred` t	*predecessor*		|	`0`	*zero value*
	|	`iszero` t	*zero test*		|	`succ` nv	*successor value*

Evaluation rules

(E-PredZero) $\texttt{pred 0} \longrightarrow \texttt{0}$

(E-PredSucc) $\texttt{pred}\ (\texttt{succ}\ nv_1) \longrightarrow nv_1$

(E-Succ) $$\frac{t_1 \longrightarrow t_1'}{\texttt{succ}\ t_1 \longrightarrow \texttt{succ}\ t_1'}$$

(E-Pred) $$\frac{t_1 \longrightarrow t_1'}{\texttt{pred}\ t_1 \longrightarrow \texttt{succ}\ t_1'}$$

(E-IsZeroZero) $\texttt{iszero}\ 0 \longrightarrow \texttt{true}$

(E-IsZeroPred) $\texttt{iszero}\ (\texttt{succ}\ nv_1) \longrightarrow \texttt{false}$

(E-IsZero) $$\frac{t_1 \longrightarrow t_1'}{\texttt{iszero}\ t_1 \longrightarrow \texttt{iszero}\ t_1'}$$

(E-If) $$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3}$$

(E-IfTrue) $\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2$

(E-IfFalse) $\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3$

Typing rules

(T-True) $\texttt{true} : \texttt{Bool}$

(T-False) $\texttt{false} : \texttt{Bool}$

(T-If) $$\frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$$

(T-Zero) $\texttt{0} : \texttt{Nat}$

(T-Succ) $$\frac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$$

(T-Pred) $$\frac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$$

(T-IsZero) $$\frac{t_1 : \texttt{Nat}}{\texttt{iszero}\ t_1 : \texttt{Bool}}$$

(T-Unit) $\texttt{unit} : \texttt{Unit}$

Appendix D: **STLC with lists**

$$
\begin{array}{lllr}
t & ::= & & \textbf{Terms} \\
 & | & ... & \text{(STLC terms)} \\
 & | & \texttt{nil} & \text{Empty list} \\
 & | & \texttt{cons}\ t\ t & \text{List constructor} \\
 & | & \texttt{head}\ t & \text{Head of a list} \\
 & | & \texttt{tail}\ t & \text{Tail of a list} \\
 & | & \texttt{isnil}\ t & \text{Test for empty list}
\end{array}
$$

$$
\begin{array}{lllr}
v & ::= & & \textbf{Values} \\
 & | & ... & \text{(STLC values)} \\
 & | & \texttt{nil} & \text{Empty list} \\
 & | & \texttt{cons}\ v\ v & \text{List constructor}
\end{array}
$$

$$
\begin{array}{lllr}
T & ::= & & \textbf{Types} \\
 & | & ... & \text{(STLC types)} \\
 & | & \texttt{List}\ T & \text{Type of a list with elements of type } T
\end{array}
$$

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.

Appendix E: **Subtyping extension to STLC**

(S-Refl) $S \;<:\; S$ (S-Trans) $\dfrac{S \;<:\; U \qquad U \;<:\; T}{S \;<:\; T}$

(S-Top) $S \;<:\; \texttt{Top}$ (S-Arrow) $\dfrac{T_1 \;<:\; S_1 \qquad S_2 \;<:\; T_2}{S_1 \;\rightarrow\; S_2 \;<:\; T_1 \;\rightarrow\; T_2}$

Appendix F: **Product extension to STLC**

$$\begin{array}{rcll}
t & ::= & \ldots & \textbf{terms:} \\
 & | & \{t, t\} & \text{pair} \\
 & | & t.1 & \text{first projection} \\
 & | & t.2 & \text{second projection} \\
 \\
v & ::= & \ldots & \textbf{values:} \\
 & | & \{v, v\} & \text{pair value} \\
 \\
T & ::= & \ldots & \textbf{types:} \\
 & | & T_1 \times T_2 & \text{product type}
\end{array}$$

Typing rules:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \tag{T-Pair}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.1 : T_1} \tag{T-Proj1}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.2 : T_2} \tag{T-Proj2}$$

New evaluation rules:

$$\{v_1, v_2\}.1 \longrightarrow v_1 \tag{E-PairBeta1}$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \tag{E-PairBeta2}$$

$$\frac{t \longrightarrow t'}{t.1 \longrightarrow t'.1} \tag{E-Proj1}$$

$$\frac{t \longrightarrow t'}{t.2 \longrightarrow t'.2} \tag{E-Proj2}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \tag{E-Pair1}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \tag{E-Pair2}$$