You can turn in handwritten solutions to this assignment. To keep your graders happy, please write clearly, leave lots of whitespace, and use standard-sized (8.5 by 11in) paper! You may be penalized upto 20 points if you do not follow these instructions. Handwritten solutions should be submitted at Lindley Hall 301G by 5pm on the due date.

If you choose to typeset your solutions, you may use LaTeX or Word. If you use LaTeX, there is a template available for your use at the course website. (Remember to look in b522.sty for macros you can use.) A pdf file containing the solutions can be submitted online by midnight on the due date.

This homework is worth 100 points and you have about two weeks to complete it. It is a long and challenging assignment, so start early!

1. **Recursive types** (15 pts.)

   Consider mutually recursive type definitions like the following:

   ```
   type Node = Edge list
   type Edge = Node * Node
   ```

   Eliminate the mutual recursion by giving recursive (μ) types for Node and Edge, and show that the unfoldings of your Node and Edge types satisfy their respective equations. You may assume that list and $\times$ (which is the notation we've been using for the $*$ type) are built-in type constructors.

2. **Subtyping** (20 pts.)

   For each of the following questions, answer Yes or No. If the answer is Yes, show the subtyping derivation. If the answer is No, give either a *term* that demonstrates how type safety breaks if we allow the two types in the subtype relation, or a *short explanation* of why type safety is preserved even if we allow the two types in the subtype relation.

   (a) (5 pts) Is $\{\mathsf{x} : \mathsf{Top} \to \mathsf{Ref\ Top}\}$ a subtype of $\{\mathsf{x} : \mathsf{Top} \to \mathsf{Top}\}$?

   (b) (5 pts) Is $\{\mathsf{x} : \mathsf{Top} \to \mathsf{Ref\ Top}\}$ a subtype of $\{\mathsf{x} : \mathsf{Ref\ Top} \to \mathsf{Ref}\ \{\mathsf{y} : \mathsf{Top}\}\}$?

   (c) (5 pts) Is $\{\mathsf{x} : \mathsf{Ref}\ \{\mathsf{y} : \mathsf{Top}\}\}$ a subtype of $\{\mathsf{x} : \mathsf{Ref\ Top}\}$?

   (d) (5 pts) Is $\{\mathsf{x} : \mathsf{Top}\}$ a subtype of $\{\mathsf{x} : \{\,\}\}$?

3. **Encoding sum and product types in the polymorphic lambda calculus** (40 pts.)

   Sum types and product types, as seen in $\lambda^{\to +\times}$, are important features in programming languages. In this problem, we will show that by using some insights from the Curry-Howard isomorphism, sum types in $\lambda^{\to +\times}$ can be encoded using product types and universal types in the polymorphic lambda calculus. Similarly, we can encode $\lambda^{\to +\times}$ product types using sum types and universal types in the polymorphic lambda calculus.

   Our source language will consist of the simply-typed λ-calculus extended with sum and product types. (Here B ranges over base types, such as **Bool**, **Int**, **Unit**, etc., while b ranges over constants of base type, such as **true**, **false**, integers n, **null**, etc.)

   $$\begin{array}{llll} \textit{Types} & T & ::= & B \mid T_1 \to T_2 \mid T_1 + T_2 \mid T_1 \times T_2 \\ \textit{Terms} & e & ::= & b \mid x \mid \lambda x{:}T.\, e \mid e_1\ e_2 \mid \mathbf{inl}_{T_1+T_2}\, e \mid \mathbf{inr}_{T_1+T_2}\, e \mid \\ & & & \quad \mathbf{case}\ e_0\ \mathbf{of\ inl}\, x \Rightarrow e_1 \mid \mathbf{inr}\, y \Rightarrow e_2 \mid (e_1, e_2) \mid \mathbf{fst}\ e \mid \mathbf{snd}\ e \end{array}$$

   The typing rules for this language are the same as defined in class.

The target sum language $\mathbf{F}^+$ is the polymorphic lambda calculus extended with only sum types.

$$\begin{array}{llll} \textit{Types} & T & ::= & B \mid \alpha \mid T_1 \to T_2 \mid \forall\alpha.T \mid T_1 + T_2 \\ \textit{Terms} & e & ::= & b \mid x \mid \lambda x{:}T.\, e \mid e_1\ e_2 \mid \Lambda\alpha.\, e \mid e\,[T] \mid \mathbf{inl}_{T_1+T_2}\, e \mid \mathbf{inr}_{T_1+T_2}\, e \mid \\ & & & \mathbf{case}\ e_0\ \mathbf{of}\ \mathbf{inl}\, x \Rightarrow e_1 \mid \mathbf{inr}\, y \Rightarrow e_2 \end{array}$$

The target product language $\mathbf{F}^\times$ is the polymorphic lambda calculus extended with only product types.

$$\begin{array}{llll} \textit{Types} & T & ::= & B \mid \alpha \mid T_1 \to T_2 \mid \forall\alpha.T \mid T_1 \times T_2 \\ \textit{Terms} & e & ::= & b \mid x \mid \lambda x{:}T.\, e \mid e_1\ e_2 \mid \Lambda\alpha.\, e \mid e\,[T] \mid (e_1,\, e_2) \mid \mathbf{fst}\ e \mid \mathbf{snd}\ e \end{array}$$

The target languages have the same typing rules as the source language (provided that we add the context Δ to each judgment that appears in each of those typing rules), extended with two rules for supporting polymorphism:

$$\frac{\Delta, \alpha; \Gamma \vdash e : T}{\Delta; \Gamma \vdash \Lambda\alpha.\, e : \forall\alpha.T}\ \text{(T-TAbs)} \qquad \frac{\Delta; \Gamma \vdash e : \forall\alpha.T \qquad \Delta \vdash T_1}{\Delta; \Gamma \vdash e\,[T_1] : T\{T_1/\alpha\}}\ \text{(T-TApp)}$$

Note that the rule for type abstraction requires that the new type variable α be fresh, to prevent the capture of type variables appearing in Γ.

Both target languages have the usual rules for well-formed type expressions:

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha}\ \text{(WF-TVar)} \qquad \frac{}{\Delta \vdash B}\ \text{(WF-Grnd)}$$

$$\frac{\Delta \vdash T_1 \qquad \Delta \vdash T_2}{\Delta \vdash T_1 \to T_2}\ \text{(WF-Fun)} \qquad \frac{\Delta, \alpha \vdash T}{\Delta \vdash \forall\alpha.T}\ \text{(WF-All)}$$

$$\frac{\Delta \vdash T_1 \qquad \Delta \vdash T_2}{\Delta \vdash T_1 + T_2}\ \text{(WF-Sum)—}\mathbf{F}^+\ \text{only} \qquad \frac{\Delta \vdash T_1 \qquad \Delta \vdash T_2}{\Delta \vdash T_1 \times T_2}\ \text{(WF-Prod)—}\mathbf{F}^\times\ \text{only}$$

Your goal in this problem is to provide type translations from the source language to each of the two target languages, and to show that these translations work. Each typed translation will convert typing judgments in the source language into typing judgments in the target language. This translation will be done in such a way that a term translation will automatically have a target language type derivation if the original (source) term had a type derivation in the source language.

The Curry-Howard isomorphism will assist you in constructing these translations. We know that the type $T_1 + T_2$ corresponds to the formula $T_1 \vee T_2$ and the type $T_1 \times T_2$ corresponds to the formula $T_1 \wedge T_2$. From De Morgan's rules in classical logic, we know that $T_1 \vee T_2 \equiv \neg(\neg T_1 \wedge \neg T_2)$, and that $T_1 \wedge T_2 \equiv \neg(\neg T_1 \vee \neg T_2)$.

(a) (5 pts) As mentioned in class, types whose formulae contain negation can be generated by using continuations, but our target language has no continuations.

i. Give a formula that is equivalent to $A \wedge B$ but contains only logical operators for which there are corresponding types in $\mathbf{F}^+$.

ii. Give a formula that is equivalent to $A \vee B$ but contains only logical operators for which there are corresponding types in $\mathbf{F}^\times$.

Hint: Use universal quantification to express a formula equivalent to $\neg A$. When you apply this to De Morgan's rules, be careful about the scope of your quantifiers! If you get stuck later on in this problem, you should revisit your answer to this part.

(b) (5 pts) We translate a source language type T into an $\mathbf{F}^+$ type $\mathcal{T}^+[\![T]\!]$ and into an $\mathbf{F}^\times$ type $\mathcal{T}^\times[\![T]\!]$.

i. What logically equivalent $\mathbf{F}^+$ type should $\mathcal{T}^+[\![\cdot]\!]$ map $T_1 \times T_2$ to? And, the easier question: what logically equivalent $\mathbf{F}^+$ type should $\mathcal{T}^+[\![\cdot]\!]$ map B, $T_1 \to T_2$, and $T_1 + T_2$ to?

ii. What logically equivalent $\mathbf{F}^\times$ type should $\mathcal{T}^\times[\![\cdot]\!]$ map $T_1 + T_2$ to? And, the easier question: what logically equivalent $\mathbf{F}^\times$ type should $\mathcal{T}^\times[\![\cdot]\!]$ map B, $T_1 \to T_2$, and $T_1 \times T_2$ to?

(c) (10 pts) In this part, you will define the important parts of a type-preserving translation function $\mathcal{E}^+[\![\cdot]\!]$ which, when applied to a source language typing judgment, produces a well-typed $\mathbf{F}^+$ term. It will be useful to have a semantic function $\mathcal{G}^+[\![\cdot]\!]$ that simply maps all the types of variables in Γ into the target language:

$$\begin{array}{lcl} \mathcal{G}^+[\![\emptyset]\!] & = & \emptyset \\ \mathcal{G}^+[\![\Gamma, x{:}T]\!] & = & \mathcal{G}^+[\![\Gamma]\!], x{:}\mathcal{T}^+[\![T]\!] \end{array}$$

Then $\mathcal{E}^+[\![\cdot]\!]$ is to be defined in such a way that if $\Gamma \vdash e : T$ in the source language, then in the target language, we should have:

$$\emptyset;\ \mathcal{G}^+[\![\Gamma]\!] \vdash \mathcal{E}^+[\![\Gamma \vdash e : T]\!] : \mathcal{T}^+[\![T]\!]$$

Define $\mathcal{E}^+[\![\cdot]\!]$ on the typing judgments for (e_1, e_2) and **fst** e. You will need to introduce new variables; give any side conditions needed to control the selection of variable names.

(Optionally, you can define $\mathcal{E}^+[\![\cdot]\!]$ on the typing judgment for **snd** e as well, but you can skip that because it should be symmetric to the definition for **fst** e.)

(d) (10 pts) Similarly, you can define $\mathcal{E}^\times[\![\cdot]\!]$ (and $\mathcal{G}^\times[\![\cdot]\!]$). The translation function $\mathcal{E}^\times[\![\cdot]\!]$, when applied to a source language typing judgment, produces a well-typed $\mathbf{F}^\times$ term. The function $\mathcal{G}^\times[\![\cdot]\!]$ that maps all the types of variables in Γ into the target language $\mathbf{F}^\times$ can be defined as follows:

$$\begin{array}{lcl} \mathcal{G}^\times[\![\emptyset]\!] & = & \emptyset \\ \mathcal{G}^\times[\![\Gamma, x{:}T]\!] & = & \mathcal{G}^\times[\![\Gamma]\!], x{:}\mathcal{T}^\times[\![T]\!] \end{array}$$

Then $\mathcal{E}^\times[\![\cdot]\!]$ is to be defined in such a way that if $\Gamma \vdash e : T$ in the source language, then in the target language, we should have:

$$\emptyset;\ \mathcal{G}^\times[\![\Gamma]\!] \vdash \mathcal{E}^\times[\![\Gamma \vdash e : T]\!] : \mathcal{T}^\times[\![T]\!]$$

Define $\mathcal{E}^\times[\![\cdot]\!]$ on the typing judgments for $\mathbf{inl}_{T_1+T_2}\, e$ and **case** e_0 **of inl** $x \Rightarrow e_1 \mid$ **inr** $y \Rightarrow e_2$. If you introduce new variables, give any side conditions needed to control the selection of variable names.

(Optionally, again, you can define $\mathcal{E}^\times[\![\cdot]\!]$ on the typing judgment for $\mathbf{inr}_{T_1+T_2}\, e$ as well, but you can skip that because it should be symmetric to the definition for $\mathbf{inl}_{T_1+T_2}\, e$.)

(e) (10 pts) Show that the expressions that are your translations in 4(c) and 4(d) are well-formed. For each translated expression, give a type derivation in which the tops of the proof tree are either axioms or are judgments that you are assured of having because the source-language expression is well-formed.

(To make your type derivations fit on the page, you can label a subtree and then show the derivation for that subtree separately. You can also define abbreviations for lengthy expressions or types that appear repeatedly. Of course, don't overdo this! You want the end result to still be readable without having to repeatedly refer to a key.)

4. **Strong normalization** (25 pts.)

(a) (20 pts) Show that all expressions in the language $\lambda^{\to +}$ are strongly normalizing by extending the proof of strong normalization for $\lambda^\to$.

(b) (5 pts) Where does the proof fail if we add recursive types to the language?