Before starting, be sure that you understand the course policy on Academic Integrity. Download `code06.tar` from the course website, which contains Standard ML files for a lambda-calculus interpreter.

1. Functions and Subtyping

Recall the operational semantics and type system for the Simply-Typed Lambda Calculus with constants, records, and subtyping:

\[
e : \tau \quad \tau \quad \tau \quad \Gamma \quad \vdash \\
\begin{align*}
e &::= e \mid x \mid \lambda x : \tau. e \mid e_1 e_2 \mid \{l_1=e_1; \ldots; l_n=e_n\} \mid e.l \\
v &::= e \mid \lambda x : \tau. e \mid \{l_1=v_1; \ldots; l_n=v_n\} \\
\tau &::= \text{int} \mid \tau \rightarrow \tau \mid \{l_1:\tau_1; \ldots; l_n:\tau_n\} \\
\Gamma &::= \cdot \mid \Gamma, x : \tau
\end{align*}
\]

\[
\vdash e \rightarrow_{cbv} e'
\]

\[
\begin{array}{llll}
\text{E-Apply} & \Gamma(x) = \tau & \vdash & \lambda x : \tau. e_1 \rightarrow_{cbv} e_2[v \mapsto x] \\
\text{E-Apply1} & e_f \rightarrow_{cbv} e'_f & \vdash & e_f e_a \rightarrow_{cbv} e'_f e_a \\
\text{E-Apply2} & e_i \rightarrow_{cbv} e'_i & \vdash & v_f e_a \rightarrow_{cbv} v'_f e'_a
\end{array}
\]

\[
\begin{array}{lll}
\text{T-Var} & \Gamma \vdash c : \text{int} & \vdash x : \tau \\
\text{T-Lam} & \Gamma, x : \tau, e_1 : \tau_1 \vdash e_b : \tau_2 & \vdash \lambda x : \tau. e_2 : \tau \\
\text{T-Apply} & \Gamma \vdash e_f : \tau_a \rightarrow \tau_r & \vdash e_a : \tau_a & \vdash \tau_r \\
\text{T-Select} & \Gamma \vdash e : \{l_1 : \tau_1; \ldots; l_n : \tau_n\} & \vdash e.l_i : \tau_i
\end{array}
\]

\[
\vdash \tau' \leq \tau
\]

\[
\begin{array}{lll}
\text{S-Refl} & \tau_1 \leq \tau_2 & \tau_2 \leq \tau_3 \\
\text{S-Trans} & \tau_1 \leq \tau_3 \\
\text{S-Arrow} & \tau_a \leq \tau'_a & \tau'_r \leq \tau_r & \tau_a \rightarrow \tau'_a \leq \tau_r \\
\text{S-Width} & \{l_1:\tau_1; \ldots; l_n:\tau_n; l:\tau\} \leq \{l_1:\tau_1; \ldots; l_n:\tau_n\} \\
\text{S-Perm} & \{l_1:\tau_1; \ldots; l_k:\tau_{i+1}; \ldots; l_n:\tau_n\} \leq \{l_1:\tau_1; \ldots; l_k:\tau_{i+1}; l_{i+1}:\tau_i; \ldots; l_n:\tau_n\} \\
\text{S-Depth} & \tau'_i \leq \tau_i & \{l_1:\tau_1; \ldots; l_i:\tau_i; \ldots; l_n:\tau_n\} \leq \{l_1:\tau_1; \ldots; l_i:\tau_i; \ldots; l_n:\tau_n\}
\end{array}
\]

Note that functions have an type-annotated argument \((\lambda x : \tau_a. e_b)\) and the \text{T-Lam} rule requires the body expression to be well-typed with the argument having its annotated type.
(a) Consider the following alternative S-ARROW rule for function subtyping, which is covariant in both the argument type and the result type:

\[
\frac{\tau_a' \leq \tau_a \quad \tau_r' \leq \tau_r}{\tau_a' \rightarrow \tau_r' \leq \tau_a \rightarrow \tau_r}
\]

Show that this rule is unsound.

To show that a rule is unsound:

i. give an example program

ii. show that the program is well-typed (by giving the typing derivation) with the rule

iii. show that the program gets stuck (by giving the evaluation steps)

(Hint: 1(a) is (probably) easier than 1(b).)

(b) Consider the following alternative S-ARROW rule for function subtyping, which is contravariant in both the argument type and the result type:

\[
\frac{\tau_a \leq \tau_a' \quad \tau_r \leq \tau_r'}{\tau_a \rightarrow \tau_r \leq \tau_a' \rightarrow \tau_r'}
\]

Show that this rule is unsound.

(Hint: 1(b) is (probably) harder than 1(a).)
2. References and Subtyping

We now extend the language from Problem 1 with a mutable heap (as in Homework 5):

```plaintext
\begin{align*}
  e & ::= c \mid x \mid \lambda x : \tau. e \mid e \mid \{ l_1 = e_1; \ldots ; l_n = e_n \} \mid e. l \mid \text{ref} e \mid ! e \mid e := e \mid a \\
  v & ::= c \mid \lambda x : \tau. e \mid \{ l_1 = v_1; \ldots ; l_n = v_n \} \mid a \\
  H & ::= \cdot \mid H, a \rightarrow v \\
  \tau & ::= \text{int} \mid \tau \rightarrow \tau \mid \{ l_1 : \tau_1; \ldots ; l_n : \tau_n \} \mid \text{ref} \tau \\
  \Gamma & ::= \cdot \mid \Gamma, x : \tau \mid \Gamma, a : \tau
\end{align*}
```

\[
H; e \rightarrow_{cbv} H'; e'
\]

\[
\begin{align*}
\text{E-Apply} & \quad \frac{H; \lambda x : \tau. e \rightarrow_{cbv} H'; e'}{H; e e_a \rightarrow_{cbv} H'; e' e_a} \\
\text{E-Select} & \quad \frac{H; \{ l_1 = v_1; \ldots ; l_n = v_n \} \mid l \rightarrow_{cbv} H; v}{H; e \rightarrow_{cbv} H'; e'} \\
\text{E-Alloc} & \quad \frac{a \notin \text{Dom}(H)}{H; \text{ref} v_a \rightarrow_{cbv} H, a \rightarrow v_a : a} \\
\text{E-Get} & \quad \frac{H ; (a) = v}{H : ! a \rightarrow_{cbv} H; v} \\
\text{E-Set} & \quad \frac{H[a] \leftarrow v = H'}{H; a := v \rightarrow_{cbv} H'; v} \\
\text{T-Record} & \quad \frac{\Gamma \vdash e_r : \tau}{H ; \{ l_1 = e_1; \ldots ; l_n = e_n \} \rightarrow_{cbv} H'; e'_r} \\
\text{T-Select} & \quad \frac{\Gamma \vdash e_r : \{ l_1 : \tau_1; \ldots ; l_n : \tau_n \}}{H ; e_r \rightarrow_{cbv} H'; e'_r} \\
\text{T-Lam} & \quad \frac{\Gamma, x : \tau_a \vdash e_b : \tau_r}{H ; \lambda x : \tau_a. e_b \rightarrow_{cbv} H; e_r} \\
\text{T-Var} & \quad \frac{\Gamma \vdash x : \tau}{\Gamma \vdash x : \tau} \\
\text{T-Select1} & \quad \frac{H ; e_r : \{ l_1 = e_1; \ldots ; l_n = e_n \} \rightarrow_{cbv} H'; e'_r}{H; e_r := e_a \rightarrow_{cbv} H'; e'_r := e_a} \\
\text{T-Apply} & \quad \frac{\Gamma \vdash e_f : \tau_a \rightarrow \tau_r \quad \Gamma \vdash e_a : \tau_a}{\Gamma \vdash e_f e_a : \tau_r} \\
\text{T-Subsumption} & \quad \frac{\Gamma \vdash e : \tau' \quad \tau' \leq \tau}{\Gamma \vdash e : \tau}
\end{align*}
\]

\[
\begin{align*}
\tau' & \leq \tau \\
\text{S-Ref} & \quad \frac{\tau \leq \tau_2 \quad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3} \\
\text{S-Perm} & \quad \frac{\{ l_1 : \tau_1; \ldots ; l_n : \tau_n \} \leq \{ l_1 : \tau_1; \ldots ; l_n : \tau_n \}}{} \\
\text{S-Select} & \quad \frac{\tau_2 \leq \tau \quad \tau \leq \tau'}{\text{ref} \tau' \leq \text{ref} \tau} \\
\text{S-Depth} & \quad \frac{\tau_i' \leq \tau_i}{\{ l_1 : \tau_1; \ldots ; l_i : \tau_i; \ldots ; l_n : \tau_n \} \leq \{ l_1 : \tau_1; \ldots ; l_i : \tau_i; \ldots ; l_n : \tau_n \}}
\end{align*}
\]

3
(a) Consider the following alternative S-Ref rule for reference subtyping, which is *covariant* in the contained type:

\[
\frac{\tau' \leq \tau}{\ref \tau' \leq \ref \tau}
\]

Show that this rule is *unsound*.  
*(Hint: 2(a) is (probably) harder than 2(b).)*

(b) Consider the following alternative S-Ref rule for reference subtyping, which is *contravariant* in the contained type:

\[
\frac{\tau \leq \tau'}{\ref \tau' \leq \ref \tau}
\]

Show that this rule is *unsound*.  
*(Hint: 2(b) is (probably) easier than 2(a).)*

(c) Consider the given *sound* S-Ref rule for reference subtyping, which is *invariant* in the contained type:

\[
\frac{\tau' \leq \tau \quad \tau \leq \tau'}{\ref \tau' \leq \ref \tau}
\]

Show that this rule is *not admissible*.  
To show that a rule is not admissible:

i. give an example program

ii. show that the program is well-typed (by giving the typing derivation) with the rule

iii. argue that the program is not well-typed without the rule

Note that the language already has reflexive subtyping; therefore, we can already derive \( \tau \leq \tau \) for all \( \tau \).  
*(Hint: Find a pair of types \( \tau_1 \) and \( \tau_2 \) such that \( \tau_1 \leq \tau_2 \) and \( \tau_2 \leq \tau_1 \), but \( \tau_1 \neq \tau_2 \).)*

*(Hint: The simplest solution makes essential use of the type-annotation in \( \lambda x : \tau. \ e \).)*
3. Implementing Subtyping

The code provided defines an abstract syntax and a scanner/parser for the simply-typed lambda calculus with integers, addition, multiplication, greater-than, integer-based conditionals (0 is false, other integers are true), tuples, a heap, subtyping, and type aliases. (It is based closely upon the code from Homework 05.) Some important notes:

- A program begins with zero or more “type aliases” of the form `type s = τ;`, where `s` is an identifier and `τ` is a type. A type alias makes `s` a type name that can be used in other types. With respect for subtyping, both `s ≤ τ` and `τ ≤ s`. (The provided type checker ensures that a program’s type aliases have no cyclic references and that each type alias defines a different type name.)

- The type checker does not allow implicit subsumption. Instead, the syntax `(e : τ)` indicates an explicit subsumption: if `e` has the type `τ'` and `τ' ≤ τ`, then `(e : τ)` has the type `τ`. (If `τ'` is not a subtype of `τ`, then `(e : τ)` should not type check.)

- Tuple types are written `τ_1 ∗ τ_2 ∗ · · · ∗ τ_n` and tuple expressions are written `e_1 ∗ e_2 ∗ · · · ∗ e_n`. (There is no syntax for tuple types or tuple expressions with fewer than two components, although the interpreter and typechecker support such tuple types and tuple expressions.)

- Tuple selections are written `e.i`, where `i` is an integer and the fields are numbered left-to-right starting with 1.

Implement the `subtype` function to support the following:

- The integer type is a subtype of the integer type.
- Function types have their usual contravariant argument and covariant result subtyping.
- Reference types are invariant (as described in Problem 2c).
- Tuple types have width and depth subtyping (but no permutation subtyping).
- A type name is a subtype of what it aliases and vice-versa

Use pattern matching on pairs of types. Also, using functions from the `ListPair` structure can make your solution more concise. The reference solution is only 15 lines long; ask for help if you find yourself attempting to write significantly more than this.
4. Debriefing

- How many hours did you spend on this assignment?
- Would you rate it as easy, moderate, or difficult?
- How deeply do you feel you understand the material it covers (0% – 100%)?
- If you have any other comments about the assignment, then please include them with your submission or send email to mtf@cs.rit.edu.

Submission

All components of the assignment:

- problems 1a.(i) (program), 1a.(ii) (typing derivation), 1a.(iii) (stuck evaluation), 1b.(i), 1b.(ii), 1b.(iii), 2a.(i), 2a.(ii), 2a.(iii), 2b.(i), 2b.(ii), 2b.(iii), problems 2c.(i) (program), 2c.(ii) (typing derivation), 2c.(iii) (argument), and debriefing in a file named homework06.pdf
- problem 3 in the file named lcsub.sml

must be submitted to the Homework 6 Assignment on MyCourses by the due date.

For handwritten components, please use either a document scanner or a mobile scanning app like Adobe Scan or Microsoft Lens.