Before starting, be sure that you understand the course policy on Academic Integrity. Download code07.tar from the course website, which contains Standard ML files for System F interpreter.

1. **Church Encodings in System F**

   Recall the encodings of booleans, pairs, and natural numbers in the (untyped) Lambda Calculus:

   ```
   true = λx. λy. x
   false = λx. λy. y
   if = λb. λt. λf. b t f
   and = λb1. λb2. b1 b2 b1
   or = λb1. λb2. b1 b1 b2
   not = λb. λx. λy. b y x
   mkpair = λx. λy. λz. z x y
   fst = λp. p (λx. λy. x)
   snd = λp. p (λx. λy. y)
   0 = λs. λz. z
   1 = λs. λz. s z
   2 = λs. λz. s (s z)
   3 = λs. λz. s (s (s z))
   isZero = λn. n (λx. false) true
   isEven = λn. n not true
   succ = λn. λs. λz. s (n s z)
   plus = λm. λn. m succ n
   times = λm. λn. m (plus n) 0
   ```

   Although these values are well-typed in the Simply-Typed Lambda Calculus, they are not particularly useful, because they must be given a single type. For example, we could give `if` the type `(int → int → int) → int → int → int`, but this `if` only “works” for conditionals that return an int.

   With System F, we can recover useful, well-typed Church encodings of booleans, pairs, and natural numbers.
(a) We can define the boolean type\(^1\) and operations as follows:

\[
\text{bool} = \forall \alpha. \alpha \to \alpha \to \alpha
\]

\[
\text{true} : \text{bool}
\]

\[
\text{true} = \Lambda \alpha. \lambda x : \alpha. \lambda y : \alpha. x
\]

\[
\text{false} : \text{bool}
\]

\[
\text{false} = \Lambda \alpha. \lambda x : \alpha. \lambda y : \alpha. y
\]

\[
\text{if} : \forall \beta. \text{bool} \to \beta \to \beta \to \beta
\]

\[
\text{if} = \Lambda \beta. \lambda b : \text{bool}. \lambda t : \beta. \lambda f : \beta. b \lfloor \beta \rfloor t f
\]

Show how to write the terms \textbf{and}, \textbf{or}, and \textbf{not} in System F.

\[
\text{and} : \text{bool} \to \text{bool} \to \text{bool}
\]

\[
\text{or} : \text{bool} \to \text{bool} \to \text{bool}
\]

\[
\text{not} : \text{bool} \to \text{bool}
\]

(b) We can define the pair type as follows:

\[
\tau_1 \times \tau_2 = \forall \gamma. (\tau_1 \to \tau_2 \to \gamma) \to \gamma
\]

Show how to write the terms \textbf{mkpair}, \textbf{fst}, and \textbf{snd} in System F.

\[
\text{mkpair} : \forall \alpha. \forall \beta. \alpha \to \beta \to \alpha \times \beta
\]

\[
\text{fst} : \forall \alpha. \forall \beta. \alpha \times \beta \to \alpha
\]

\[
\text{snd} : \forall \alpha. \forall \beta. \alpha \times \beta \to \beta
\]

(c) We can define the natural number (Church numeral) type and natural numbers as follows:

\[
\text{nat} = \forall \alpha. (\alpha \to \alpha) \to \alpha \to \alpha
\]

\[
0 : \text{nat}
\]

\[
0 = \Lambda \alpha. \lambda s : \alpha \to \alpha. \lambda z : \alpha. z
\]

\[
1 : \text{nat}
\]

\[
1 = \Lambda \alpha. \lambda s : \alpha \to \alpha. \lambda z : \alpha. s z
\]

\[
2 : \text{nat}
\]

\[
2 = \Lambda \alpha. \lambda s : \alpha \to \alpha. \lambda z : \alpha. s (s z)
\]

\[
3 : \text{nat}
\]

\[
3 = \Lambda \alpha. \lambda s : \alpha \to \alpha. \lambda z : \alpha. s (s (s z))
\]

Show how to write the terms \textbf{isZero}, \textbf{isEven}, \textbf{succ}, \textbf{plus}, and \textbf{times} in System F.

\[
\text{isZero} : \text{nat} \to \text{bool}
\]

\[
\text{isEven} : \text{nat} \to \text{bool}
\]

\[
\text{succ} : \text{nat} \to \text{nat}
\]

\[
\text{plus} : \text{nat} \to \text{nat} \to \text{nat}
\]

\[
\text{times} : \text{nat} \to \text{nat} \to \text{nat}
\]

---

\(^1\)Think of the definition of \text{bool} as a “macro”: writing \text{bool} means writing \forall \alpha. \alpha \to \alpha \to \alpha.
2. Implementing Type Substitution in System F

The code provided defines an abstract syntax and a scanner/parser for System F with integers, addition, multiplication, greater-than, integer-based conditionals (0 is false, other integers are true), a mutable heap, units, pairs, sums, and roll/unrolls. (It is based closely upon the code from Homework 05.)

Implement the `typ_substitute` function (in `ast.sml`), which is used to type check type application, rolls, and unrolls. Note that you must implement capture-avoiding substitution. Review the notes from Lecture 08 (Lambda Calculus (cont’d)) on defining substitution; your solution should closely follow the (correct) Attempt 4. Also examine the provided `typ_equal` function, which uses type substitution to compare types (with bound type variables) for equality.

The reference solution is only 17 lines long; ask for help if you find yourself attempting to write significantly more than this.
3. Parametricity (and lack thereof)

(a) Give 4 values \(v\) in System F (the language from the previous problem without reference and recursive types) such that:

- \(\cdot \vdash v : \forall \alpha. (\alpha \ast \alpha) \rightarrow (\alpha \ast \alpha)\)
- Each \(v\) is not equivalent to the other three (i.e., given the same arguments, it may return different results).

(b) Consider System F with references (the language from the previous problem without recursive types).

Unsurprisingly, if \(v\) is a closed value of type \(\forall \alpha. (\alpha \rightarrow \text{bool} \rightarrow \text{bool})\), then \(v \left[ \tau \right] x z\) and \(v \left[ \tau \right] y z\) always produce the same result in an environment where \(x\) and \(y\) are bound to values of type \(\tau\) and \(z\) is bound to a value of type \(\text{bool}\).

Surprisingly, there exists a closed value \(v\) of type \(\forall \alpha. (\text{ref} \alpha \rightarrow \text{ref boolean} \rightarrow \text{bool})\) such that in some environment \(v \left[ \tau \right] x z\) evaluates to \(\text{true}\) but \(v \left[ \tau \right] y z\) evaluates to \(\text{false}\). Write down one such \(v\) and explain how to call \(v\) (i.e., what \(x\), \(y\), and \(z\) should be bound to) to get this surprising behavior. (Note: although this value behaves differently when applied to different arguments, it should always behave the same when applied to the same arguments.)

Hint: You can solve this problem in SML (i.e., you do not need any System F features not found in SML). In fact, here is a template that you might follow:

```sml
(* Define 'v'. 'v' is a closed value, in the sense that it does not mention 'x', 'y', or 'z'. *
  * (Technically, it is not completely closed because it will mention the operations '!', ':=', *
  * and possibly boolean constants and operations, but the important part is that it is not *
  * defined in terms of the 'x', 'y', or 'z' values.) *
) val v : 'a ref -> bool ref -> bool = ...

(* Define 'x', 'y', and 'z' values. *)
val z = ...
val y = ...
val x = ...

; (* Evaluate 'v' with different arguments. *
   * The repeated evaluations are to demonstrate *
   * that the answer depends upon the values of *
   * the arguments, not only on side effects. *
   * In particular, 'v x z' always returns 'true' and 'v y z' always returns 'false'. *
) val ans1 = v x z
    (* val ans1 = true : bool *)
val ans2 = v y z
    (* val ans2 = false : bool *)
val ans3 = v y z
    (* val ans3 = false : bool *)
val ans4 = v x z
    (* val ans4 = true : bool *)
val ans5 = v x z
    (* val ans5 = true : bool *)
val ans6 = v y z
    (* val ans6 = false : bool *)

; (* Check answers. *)
val [true, false, false, true, true, false] = [ans1, ans2, ans3, ans4, ans5, ans6];
```

Note that the best solution ensures that after each evaluation of \(v\) applied to arguments, the heap is unchanged (although the heap may be changed during the evaluation of \(v\), such changes are “undone” by the end of the evaluation).

You are encouraged to submit your solution as SML code; it is likely to be much simpler than trying to express your solution in the less convenient System F with references language.
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, 1b, 1c, 3a, 3b, and debriefing in a file named `homework07.pdf`
   - problem 2 in the file named `ast.sml`

must be submitted to the Homework 7 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.

**Document History**

Original version