1 Parametric polymorphism

(a) For each of the following System F expressions, is the expression well-typed, and if so, what type does it have? (If you are unsure, try to construct a typing derivation. Make sure you understand the typing rules.)

- \( \Lambda A. \lambda x : A \to \text{int}. 42 \)
- \( \lambda y : \forall X. X \to X. (y \text{ [int]})) 17 \)
- \( \Lambda Y. \Lambda Z. \lambda f : Y \to Z. \lambda a : Y. f a \)
- \( \Lambda A. \Lambda B. \Lambda C. \lambda f : A \to B \to C. \lambda b : B. \lambda a : A. f a b \)

(b) For each of the following types, write an expression with that type.

- \( \forall X. X \to (X \to X) \)
- \( (\forall C. \forall D. C \to D) \to (\forall E. \text{int} \to E) \)
- \( \forall X. X \to (\forall Y. Y \to X) \)

2 Records and Subtyping

(a) Assume that we have a language with references and records. Write an expression with type

\[
\{ \text{cell} : \text{int ref}, \text{inc} : \text{unit} \to \text{int} \}
\]

such that invoking the function in the field \text{inc} will increment the contents of the reference in the field \text{cell}.

Assuming that the variable \( y \) is bound to your expression, write an expression that increments the contents of the cell twice.

(b) The following expression is well-typed (with type \text{int}). Show its typing derivation. (Note: you will need to use the subsumption rule.)

\[
(\lambda x : \{ \text{dogs} : \text{int}, \text{cats} : \text{int} \}. x. \text{dogs} + x. \text{cats}) \{ \text{dogs} = 2, \text{cats} = 7, \text{mice} = 19 \}
\]

3 Curry-Howard Isomorphism

The following logical formulas are tautologies, i.e., they are true. For each tautology, state the corresponding type, and come up with a term that has the corresponding type.

For example, for the logical formula \( \forall \phi. \phi \Rightarrow \phi \), the corresponding type is \( \forall X. X \to X \), and a term with that type is \( \Lambda X. \lambda x : X. x \). Another example: for the logical formula \( \tau_1 \land \tau_2 \Rightarrow \tau_1 \), the corresponding type is \( \tau_1 \times \tau_2 \to \tau_1 \), and a term with that type is \( \lambda x : \tau_1 \times \tau_2 \to \tau_1 \). You may assume that the lambda calculus you are using for terms includes integers, functions, products, sums, universal types and existential types.

(a) \( \forall \phi. \forall \psi. \phi \land \psi \Rightarrow \psi \lor \phi \)
4 Existential types

(a) Write a term with type \( \exists C. \{ \text{produce} : \text{int} \to C, \text{consume} : C \to \text{bool} \} \). Moreover, ensure that calling the function \( \text{produce} \) will produce a value of type \( C \) such that passing the value as an argument to \( \text{consume} \) will return true if and only if the argument to \( \text{produce} \) was 42. (Assume that you have an integer comparison operator in the language.)

(b) Do the same as in part (a) above, but now use a different witness type.

(c) Assuming you have a value \( v \) of type \( \exists C. \{ \text{produce} : \text{int} \to C, \text{consume} : C \to \text{bool} \} \), use \( v \) to "produce" and "consume" a value (i.e., make sure you know how to use the \texttt{unpack} \( \{X, x\} = e_1 \text{ in } e_2 \) expression.)