1 Type Inference

(a) Recall the constraint-based typing judgment \( \Gamma \vdash e : \tau \triangleright C \). Give inference rules for products and sums. That is, for the following expressions.

- \((e_1, e_2)\)
- \(#1 e\)
- \(#2 e\)
- \(\text{inl}_{\tau_1 + \tau_2} e\)
- \(\text{inr}_{\tau_1 + \tau_2} e\)
- \(\text{case } e_1 \text{ of } e_2 \mid e_3\)

\[
\frac{\Gamma \vdash e_1 : \tau_1 \triangleright C_1 \quad \Gamma \vdash e_2 : \tau_2 \triangleright C_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \triangleright C_1 \cup C_2}
\]

\[
\frac{\Gamma \vdash e : \tau \triangleright C \quad X, Y \text{ are fresh}}{\Gamma \vdash \#1 e : X \triangleright C \cup \{\tau = X \times Y\}}
\]

\[
\frac{\Gamma \vdash e : \tau \triangleright C \quad X, Y \text{ are fresh}}{\Gamma \vdash \#2 e : Y \triangleright C \cup \{\tau = X \times Y\}}
\]

\[
\frac{\Gamma \vdash e : \tau \triangleright C \quad \Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{\tau = \tau_1\}}{\Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{\tau = \tau_1\}}
\]

\[
\frac{\Gamma \vdash e : \tau \triangleright C \quad \Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{\tau = \tau_2\}}{\Gamma \vdash \text{inr}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{\tau = \tau_2\}}
\]

\[
\frac{\Gamma \vdash e_1 : \tau_1 \triangleright C_1 \quad \Gamma \vdash e_2 : \tau_2 \triangleright C_2 \quad \Gamma \vdash e_3 : \tau_3 \triangleright C_3 \quad X, Y, Z \text{ are fresh}}{\Gamma \vdash \text{case } e_1 \text{ of } e_2 \mid e_3 : Z \triangleright C_1 \cup C_2 \cup C_3 \cup \{\tau_1 = X + Y, \tau_2 = X \rightarrow Z, \tau_3 = Y \rightarrow Z\}}
\]

(b) Determine a set of constraints \( C \) and type \( \tau \) such that

\[
\vdash \lambda x : A. \lambda y : B. (\#1 y) + (x \ (#2 y)) + (x \ 2) : \tau \triangleright C
\]

and give the derivation for it.

**Answer:**

\[
C = \{ B = X \times Y \ , \ X = \text{int} \ , \ B = Z \times W \ , \ A = W \rightarrow U \ , \ U = \text{int} \ , \ A = \text{int} \rightarrow V \ , \ V = \text{int} \}
\]

\[
\tau = A \rightarrow B \rightarrow \text{int}
\]

*To see how we got these constraints, we will consider the subexpressions in turn (rather than trying to typeset a really really big derivation).*
The expression \( \#1 \ y \) requires us to add a constraint that the type of \( y \) (i.e., \( B \)) is equal to a product type for some fresh variables \( X \) and \( Y \), thus constraint \( B = X \times Y \). (And expression \( \#1 \ y \) has type \( X \).)

The expression \( (\#2 \ y) \) similarly requires us to add a constraint that the type of \( y \) (i.e., \( B \)) is equal to a product type for some fresh variables \( Z \) and \( W \), thus constraint \( B = Z \times W \). (And expression \( \#2 \ y \) has type \( W \).)

The expression \( x (\#2 \ y) \) requires us to add a constraint that unifies the type of \( x \) (i.e., \( A \)) with a function type \( W \to U \) (where \( W \) is the type of \( \#2 \ y \) and \( U \) is a fresh type variable).

The expression \( x \ 2 \) requires us to add a constraint that unifies the type of \( x \) (i.e., \( A \)) with a function type \( \text{int} \to V \) (where \( \text{int} \) is the type of expression \( 2 \) and \( V \) is a fresh type).

The addition operations leads us to add constraints \( X = \text{int}, U = \text{int}, \) and \( V = \text{int} \) (i.e., the types of expressions \( (\#1 \ y), (x (\#2 \ y)) \) and \( (x \ 2) \) must all unify with \( \text{int} \).)

(c) Recall the unification algorithm from Lecture 14. What is the result of \( \text{unify}(C) \) for the set of constraints \( C \) from Question 1(b) above?

**Answer:** The result is a substitution equivalent to

\[
[A \mapsto \text{int} \to \text{int}, B \mapsto \text{int} \times \text{int}, X \mapsto \text{int}, Y \mapsto \text{int}, Z \mapsto \text{int}, W \mapsto \text{int}, U \mapsto \text{int}, V \mapsto \text{int}]
\]

### 2 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.)

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

**Answer:**

- \( \Lambda A. \lambda x : A \to \text{int} \ . \ 42 \) has type \( \forall A. (A \to \text{int}) \to \text{int} \)
- \( \lambda y : \forall X. \ X \to X. (y \ [\text{int}]) \ . \ 17 \) has type \( (\forall X. \ X \to X) \to \text{int} \)
- \( \Lambda Y. \Lambda Z. \lambda f : Y \to Z. \lambda a : Y. \ f \ a \) has type \( \forall Y. \forall Z. (Y \to Z) \to Y \to Z \)
- \( \Lambda A. \Lambda B. \Lambda C. \lambda f : A \to B \to C. \lambda b : B. \lambda a : A. \ f \ a \ b \) has type \( \forall A. \forall B. \forall C. (A \to B \to C) \to B \to A \to C \)
(b) For each of the following types, write an expression with that type.

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

Answer:

- $\forall X. X \to (X \to X)$ is the type of $\lambda x : X. \lambda y : X. y$
- $(\forall C. \forall D. C \to D) \to (\forall E. \text{int} \to E)$ is the type of $\lambda f : \forall C. \forall D. C \to D. \lambda x : \text{int}. \lambda y : \text{int}. f [\text{int}] [E] x$
- $\forall X. X \to (\forall Y. Y \to X)$ is the type of $\lambda x : X. \lambda y : Y. x$

3 Records and Subtyping

(a) Assume that we have a language with references and records.

(i) Write an expression with type

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

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

Answer: The following expression has the appropriate type.

$$\text{let } x = \text{ref} \, 14 \text{ in}$$
$$\{ \text{cell = x, inc = } \lambda u : \text{unit}. x := (\! x + 1) \}$$

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

Answer:

$$\text{let } z = y.\text{inc} () \text{ in } y.\text{inc} ()$$

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

$$\{ \lambda x : \{ \text{dogs : int, cats : int} \}. x.\text{dogs} + x.\text{cats} \} \{ \text{dogs = 2, cats = 7, mice = 19} \}$$
Answer:

For brevity, let $e_1 \equiv \lambda x : \{\text{dogs : int, cats : int}\}. x.\text{dogs} + x.\text{cats}$ and let $e_2 \equiv \{\text{dogs = 2, cats = 7, mice = 19}\}$. The derivation has the following form.

\[
\frac{T-\text{APP}}{\vdash e_1 : \{\text{dogs : int, cats : int}\} \rightarrow \text{int}} \quad \frac{T-\text{APP}}{\vdash e_2 : \{\text{dogs : int, cats : int}\}} \quad \vdash e_1 e_2 : \text{int}
\]

The derivation of $e_1$ is straight forward:

The derivation of $e_2$ requires the use of subsumption, since we need to show that $e_2 \equiv \{\text{dogs = 2, cats = 7, mice = 19}\}$ has type $\{\text{dogs : int, cats : int}\}$. 
\[
\begin{array}{c}
\vdash 2 : \text{int} & \vdash 7 : \text{int} & \vdash 19 : \text{int} \\
\vdash \{\text{dogs} = 2, \text{cats} = 7, \text{mice} = 19\} : \{\text{int}, \text{cats} : \text{int}, \text{mice} : \text{int}\} & \{\text{int}, \text{cats} : \text{int}, \text{mice} : \text{int}\} \leq \{\text{dogs} : \text{int}, \text{cats} : \text{int}\} \\
\vdash \{\text{dogs} = 2, \text{cats} = 7, \text{mice} = 19\} : \{\text{int}, \text{cats} : \text{int}\}
\end{array}
\]