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.

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

Answer:

Note that in all of the rules below except for the rule for pairs \((e_1, e_2)\), the types in the premise and conclusion are connected only through constraints. The reason for this is the same as in the typing rule for function application, and for addition: we may not be able to derive that the premise has the appropriate type, e.g., for a projection \(#1 \ e\), we may not be able to derive that \(\Gamma \vdash e : \tau_1 \times \tau_2 \triangleright C\). We instead use constraints to ensure that the derived type is appropriate.

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

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

\[
\Gamma \vdash e_1 : \tau_1 \triangleright C_1 \quad \Gamma \vdash \text{inl}_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright \{\tau \equiv \tau_1\} \quad \frac{}{\Gamma \vdash \text{case} \ e_1 \text{ of } e_2 \mid e_3 : \tau \triangleright C}
\]

(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:

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

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

To see how we got these constraints, we will consider the subexpressions in turn (rather than trying to typeset a really 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 \( B \equiv X \times Y \). (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 \( B \equiv Z \times W \). (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 \( W \rightarrow U \) (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 \( \text{int} \rightarrow V \) (where \( \text{int} \) is the type of expression 2 and V is a fresh type).

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

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

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.)

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

Answer:

\[ \Lambda A. \lambda x : A \rightarrow \text{int}. \; 42 \; \text{has type} \]
\[ \forall A. (A \rightarrow \text{int}) \rightarrow \text{int} \]

\[ \lambda y : \forall X. X \rightarrow X. (y \ [\text{int}]) \; 17 \; \text{has type} \]
\[ (\forall X. X \rightarrow X) \rightarrow \text{int} \]
• \( \forall Y. \forall Z. (Y \to Z) \to Y \to Z \)

• \( \forall A. \forall B. \forall C. (A \to B \to C) \to B \to A \to C \)

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

### 3 Records and Subtyping

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

(i) Write an expression with type

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

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

**Answer:** The following expression has the appropriate type.

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

(ii) Assuming that the variable \text{y} is bound to the expression you wrote for part (i) above, write an expression that increments the contents of the cell twice.

**Answer:**

\[
\text{let } z = \text{y}.\text{inc} () \text{ in } \text{y}.\text{inc} ()
\]
(b) The following expression is well-typed (with type \textbf{int}). Show its typing derivation. (Note: you will need to use the subsumption rule.)

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

\textbf{Answer:}

For brevity, let \(e_1 \equiv \lambda x: \{\text{dogs : } \textbf{int}, \text{cats : } \textbf{int}\}. x.\text{dogs} + x.\text{cats}\) and let \(e_2 \equiv \{\text{dogs }= 2, \text{cats }= 7, \text{mice }= 19\}\).

The derivation has the following form.

\[
\begin{array}{c}
\vdash e_1: \{\text{dogs : } \textbf{int}, \text{cats : } \textbf{int}\} \rightarrow \textbf{int} \\
\vdash e_2: \{\text{dogs : } \textbf{int}, \text{cats : } \textbf{int}\}
\end{array}
\]

\[
\vdash e_1 \ e_2: \textbf{int}
\]

The derivation of \(e_1\) is straight forward:
The derivation of $e_2$ requires the use of subsumption, since we need to show that $e_2 \equiv \{\text{dogs} = 2, \text{cats} = 7, \text{mice} = 19\}$ has type $\{\text{dogs} : \text{int}, \text{cats} : \text{int}\}$.

\[
\begin{array}{c}
\vdash 2 : \text{int} \quad \vdash 7 : \text{int} \quad \vdash 19 : \text{int} \\
\frac{}{\vdash \{\text{dogs} = 2, \text{cats} = 7, \text{mice} = 19\} : \{\text{dogs} : \text{int}, \text{cats} : \text{int}, \text{mice} : \text{int}\}}
\end{array}
\]

\[
\frac{}{\vdash \{\text{dogs} : \text{int}, \text{cats} : \text{int}, \text{mice} : \text{int}\} \leq \{\text{dogs} : \text{int}, \text{cats} : \text{int}\}}
\]

\[
\frac{}{\vdash \{\text{dogs} = 2, \text{cats} = 7, \text{mice} = 19\} : \{\text{dogs} : \text{int}, \text{cats} : \text{int}\}}
\]