1 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 $\forall X. X \rightarrow X$, and a term with that type is $\lambda x : X. x$. Another example: for the logical formula $\tau_1 \land \tau_2 \Rightarrow \tau_1$, the corresponding type is $\tau_1 \times \tau_2 \rightarrow \tau_1$, and a term with that type is $\lambda x : \tau_1 \times \tau_2. \#1 x$

You may assume that the lambda calculus you are using for terms includes integers, functions, products, sums, universal types and existential types.

(a) $\forall \phi. \forall \psi. \phi \land \psi \Rightarrow \psi \lor \phi$

**Answer:** The corresponding type is $\forall X. \forall Y. X \times Y \rightarrow Y + X$

A term with this type is $\Lambda X. \Lambda Y. \lambda x : X \times Y. \text{inl}_X \#2 x$

(b) $\forall \phi. \forall \psi. \forall \chi. (\phi \land \psi \Rightarrow \chi) \Rightarrow (\phi \Rightarrow (\psi \Rightarrow \chi))$

**Answer:** The corresponding type is $\forall X. \forall Y. \forall Z. (X \times Y \rightarrow Z) \rightarrow (X \rightarrow (Y \rightarrow Z))$

A term with this type is $\Lambda X. \Lambda Y. \Lambda Z. \lambda f : X \times Y \rightarrow Z. \lambda x : X. \lambda y : Y. f (x, y)$

Note that this term curries the function, as we saw in class.

(c) $\exists \phi. \forall \psi. \psi \Rightarrow \phi$

**Answer:** The corresponding type is $\exists X. \forall Y. Y \rightarrow X$

A term with this type is $\text{pack} \{ \text{int}, \Lambda Y. \lambda y : Y. 42 \} \ as \ \exists X. \forall Y. Y \rightarrow X$

(d) $\forall \psi. \psi \Rightarrow (\forall \phi. \phi \Rightarrow \psi)$
Answer: The corresponding type is
\[ \forall Y. Y \rightarrow (\forall X. X \rightarrow Y) \]
A term with this type is
\[ \Lambda Y. \lambda a : Y. \lambda x : X. a \]

(e) \[ \forall \psi. (\forall \phi. \phi \Rightarrow \psi) \Rightarrow \psi \]

Answer: A corresponding type is
\[ \forall Y. (\forall X. X \rightarrow Y) \rightarrow Y \]
A term with this type is
\[ \Lambda Y. \lambda f : \forall X. X \rightarrow Y. f [\text{int}] 42 \]

2 Existential types

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

Answer:
In the following solution, we use \text{int} as the witness type, and implement produce using the identity function, and implement consume by testing whether the value of type \( C \) (i.e., of witness type \text{int}) is equal to 42.

\begin{verbatim}
pack \{ \text{int}, \{ \text{produce} = \lambda a : \text{int}. a, \text{consume} = \lambda a : \text{int}. a = 42 \} \} as \exists C. \{ \text{produce} : \text{int} \rightarrow C, \text{consume} : C \rightarrow \text{bool} \}
\end{verbatim}

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

Answer: Here’s another solution where instead we use \text{bool} as the witness type, and implement produce by comparing the integer argument to 42, and implement consume as the identity function.

\begin{verbatim}
pack \{ \text{bool}, \{ \text{produce} = \lambda a : \text{int}. a = 42, \text{consume} = \lambda a : \text{bool}. a \} \} as \exists C. \{ \text{produce} : \text{int} \rightarrow C, \text{consume} : C \rightarrow \text{bool} \}
\end{verbatim}

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

Answer: \begin{verbatim}
unpack \{ D, r \} = v in
let d = r.\text{produce} 19 in
r.\text{consume} d
\end{verbatim}
3 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.

- $(e_1, e_2)$
- $\#1 e$
- $\#2 e$
- inl$\tau_1 + \tau_2 e$
- inr$\tau_1 + \tau_2 e$
- case $e_1$ of $e_2 | e_3$

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.

\[
\begin{align*}
\Gamma \vdash e : \tau_1 \triangleright C & \quad \Gamma \vdash e : \tau_2 \triangleright C \\
\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \triangleright C \cup C
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : \tau \triangleright C & \quad \Gamma \vdash e : \tau \triangleright C \\
\Gamma \vdash \#1 e : \tau \triangleright \tau \equiv \tau_1 \times \tau_2 & \quad \Gamma \vdash \#2 e : \tau \triangleright \tau \equiv \tau_1 \times \tau_2
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : \tau \triangleright C & \quad \Gamma \vdash e : \tau \triangleright C \\
\Gamma \vdash inl_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{ \tau \equiv \tau_1 \} & \quad \Gamma \vdash inr_{\tau_1 + \tau_2} e : \tau_1 + \tau_2 \triangleright C \cup \{ \tau \equiv \tau_2 \}
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e_1 : \tau_1 \triangleright C & \quad \Gamma \vdash e_2 : \tau_2 \triangleright C \\
\Gamma \vdash e_3 : \tau_3 \triangleright C_3 & \quad \Gamma \vdash e_1 : \tau_1 \triangleright C \cup \{ \tau \equiv \tau_1 \} \\
\Gamma \vdash e_2 : \tau_2 \triangleright C \cup \{ \tau \equiv \tau_2 \} & \quad \Gamma \vdash e_3 : \tau_3 \triangleright C_3 \cup \{ \tau \equiv \tau_3 \}
\end{align*}
\]

\[
\Gamma \vdash case e_1 of e_2 | e_3 : \tau \triangleright C \quad X, Y, Z \text{ are fresh}
\]

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

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

and give the derivation for it.

Answer:

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

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

To see how we got these constraints, we will consider the subexpressions in turn (rather than trying to typeset a really 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 \((x \ 2)\) must all unify with \(\text{int}\).

(c) Recall the unification algorithm from Lecture 16. What is the result of \(\text{unify}(C)\) for the set of constraints \(C\) from Question 3(b) above?

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

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