Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Curry-Howard Isomorphism; Existential Types; Type Inference Section and Practice Problems

Week 9: Tue Mar 21–Fri Mar 24, 2023

---

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 $\forall \phi . \phi \implies \phi$, 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 $\tau_1 \wedge \tau_2 \implies \tau_1$, the corresponding type is $\tau_1 \times \tau_2 \to \tau_1$, and a term with that type is $\lambda x\!:\!\tau_1 \times \tau_2.\ \#1\ x$.

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

(a) $\forall \phi.\ \forall \psi.\ \phi \wedge \psi \implies \psi \vee \phi$

Answer: *The corresponding type is*

$$\forall X.\ \forall Y.\ X \times Y \to Y + X$$

A term with this type is

$$\Lambda X.\ \Lambda Y.\ \lambda x\!:\!X \times Y.\ \mathit{inl}_{Y+X}\ \#2\ x$$

(b) $\forall \phi.\ \forall \psi.\ \forall \chi.\ (\phi \wedge \psi \implies \chi) \implies (\phi \implies (\psi \implies \chi))$

Answer: *The corresponding type is*

$$\forall X.\ \forall Y.\ \forall Z.\ (X \times Y \to Z) \to (X \to (Y \to Z))$$

A term with this type is

$$\Lambda X.\ \Lambda Y.\ \Lambda Z. \lambda f\!:\!X \times Y \to Z.\ \lambda x\!:\!X.\ \lambda y\!:\!Y.\ f\ (x, y)$$

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

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

Answer: *The corresponding type is*

$$\exists X.\ \forall Y.\ Y \to X$$

A term with this type is

$$\mathit{pack}\ \{\textbf{\textit{int}}, \Lambda Y.\ \lambda y\!:\!Y.\ 42\}\ \mathit{as}\ \exists X.\ \forall Y.\ Y \to X$$

(d) $\forall \psi.\ \psi \implies (\forall \phi.\ \phi \implies \psi)$

Answer: *The corresponding type is*

$$\forall Y.\, Y \to (\forall X.\, X \to Y)$$

A term with this type is

$$\Lambda Y.\, \lambda a\!:\!Y.\, \Lambda X.\, \lambda x\!:\!X.\, a$$

(e) $\forall \psi.\, (\forall \phi.\, \phi \implies \psi) \implies \psi$

Answer: *A corresponding type is*

$$\forall Y.\, (\forall X.\, X \to Y) \to Y$$

A term with this type is

$$\Lambda Y.\, \lambda f\!:\!\forall X.\, X \to Y.\, f\ [\textbf{int}]\ 42$$

2 Existential types

(a) Write a term with type $\exists C.\, \{\, produce : \textbf{int} \to C, consume : C \to \textbf{bool}\,\}$. 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 ***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* ***int****) is equal to 42.*

$\textit{pack}\ \{\textbf{int}, \{\, produce = \lambda a\!:\!\textbf{int}.\, a,\ consume = \lambda a\!:\!\textbf{int}.\, a = 42\,\}\}$
$\textit{as}\ \exists C.\, \{\, produce : \textbf{int} \to C, consume : C \to \textbf{bool}\,\}$

(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* ***bool*** *as the witness type, and implement produce by comparing the integer argument to 42, and implement consume as the identity function.*

$\textit{pack}\ \{\textbf{bool}, \{\, produce = \lambda a\!:\!\textbf{int}.\, a = 42,\ consume = \lambda a\!:\!\textbf{bool}.\, a\,\}\}$
$\textit{as}\ \exists C.\, \{\, produce : \textbf{int} \to C, consume : C \to \textbf{bool}\,\}$

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

Answer: $\textit{unpack}\ \{D, r\} = v\ \textit{in}$
$\textit{let}\ d = r.produce\ 19\ \textit{in}$
$r.consume\ d$

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

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.

$$\frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \qquad \Gamma \vdash e_2\!:\!\tau_2 \triangleright C_2}{\Gamma \vdash (e_1, e_2)\!:\!\tau_1 \times \tau_2 \triangleright C_1 \cup C_2}$$

$$\frac{\Gamma \vdash e\!:\!\tau \triangleright C}{\Gamma \vdash \#1\ e\!:\!X \triangleright C \cup \{\tau \equiv X \times Y\}}\ X, Y \textit{ are fresh} \qquad \frac{\Gamma \vdash e\!:\!\tau \triangleright C}{\Gamma \vdash \#2\ e\!:\!Y \triangleright C \cup \{\tau \equiv X \times Y\}}\ X, Y \textit{ are fresh}$$

$$\frac{\Gamma \vdash e\!:\!\tau \triangleright C}{\Gamma \vdash \textit{inl}_{\tau_1+\tau_2}\ e\!:\!\tau_1 + \tau_2 \triangleright C \cup \{\tau \equiv \tau_1\}} \qquad \frac{\Gamma \vdash e\!:\!\tau \triangleright C}{\Gamma \vdash \textit{inr}_{\tau_1+\tau_2}\ e\!:\!\tau_1 + \tau_2 \triangleright C \cup \{\tau \equiv \tau_2\}}$$

$$\frac{\Gamma \vdash e_1\!:\!\tau_1 \triangleright C_1 \qquad \Gamma \vdash e_2\!:\!\tau_2 \triangleright C_2 \qquad \Gamma \vdash e_3\!:\!\tau_3 \triangleright C_3}{\Gamma \vdash \textit{case}\ e_1\ \textit{of}\ e_2 \mid e_3\!:\!Z \triangleright C_1 \cup C_2 \cup C_3 \cup \{\tau_1 \equiv X + Y, \tau_2 \equiv X \to Z, \tau_3 \equiv Y \to Z\}}\ X, Y, Z \textit{ are fresh}$$

(b) Determine a set of constraints C and type τ 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:

$$C = \{B \equiv X \times Y\ ,\ X \equiv \textit{int}\ ,\ B \equiv Z \times W\ ,\ A \equiv W \to U\ ,\ U \equiv \textit{int}\ ,\ A \equiv \textit{int} \to V\ ,\ V \equiv \textit{int}\}$$
$$\tau \equiv A \to B \to \textit{int}$$

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 $W \to U$ (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 **int** $\to V$ (where **int** is the type of expression 2 and V is a fresh type).*

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

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

Answer: *The result is a substitution equivalent to*

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