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

Parametric polymorphism, records, subtyping, Curry-howard Isomorphism, Existential types
Section and Practice Problems

Monday March 30, 2015

---

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.)

- $\Lambda A.\, \lambda x\!:\!A \rightarrow \textsf{int}.\, 42$
- $\lambda y\!:\!\forall X.\, X \rightarrow X.\, (y\ [\textsf{int}])\ 17$
- $\Lambda Y.\, \Lambda Z.\, \lambda f\!:\!Y \rightarrow Z.\, \lambda a\!:\!Y.\, f\ a$
- $\Lambda A.\, \Lambda B.\, \Lambda C.\, \lambda f\!:\!A \rightarrow B \rightarrow C.\, \lambda b\!:\!B.\, \lambda a\!:\!A.\, f\ a\ b$

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

- $\forall X.\, X \rightarrow (X \rightarrow X)$
- $(\forall C.\, \forall D.\, C \rightarrow D) \rightarrow (\forall E.\, \textsf{int} \rightarrow E)$
- $\forall X.\, X \rightarrow (\forall Y.\, Y \rightarrow X)$

2 Records and Subtyping

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

$$\{\ \mathit{cell} : \textsf{int ref}, \mathit{inc} : \textsf{unit} \rightarrow \textsf{int}\ \}$$

such that invoking the function in the field *inc* will increment the contents of the reference in the field *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 **int**). Show its typing derivation. (Note: you will need to use the subsumption rule.)

$$(\lambda x\!:\!\{\mathit{dogs} : \textsf{int}, \mathit{cats} : \textsf{int}\}.\, x.\mathit{dogs} + x.\mathit{cats})\ \{\mathit{dogs} = 2, \mathit{cats} = 7, \mathit{mice} = 19\}$$

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 $\forall\phi.\phi \implies \phi$, the corresponding type is $\forall X.\, X \rightarrow X$, 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 $\tau_1 \times \tau_2 \rightarrow \tau_1$, 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) $\forall\phi.\, \forall\psi.\, \phi \wedge \psi \implies \psi \vee \phi$

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

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

(d) $\tau \implies (\forall\phi.\, \phi \implies \tau)$

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

4 Existential types

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

(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 $\exists C.\, \{\, \mathit{produce} : \textbf{int} \to C, \mathit{consume} : C \to \textbf{bool} \,\}$, 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.