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

Curry-Howard isomorphism; Existential types; Sub-structural type systems Section and Practice Problems

Mar 27-Mar 30, 2018

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

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

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

2 Existential types

(a) Write a term with type $\exists C.\ \{\ \mathit{produce} : \textsf{int} \to C, \mathit{consume} : C \to \textsf{bool}\ \}$. 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 $\exists C.\ \{\ \mathit{produce} : \textsf{int} \to C, \mathit{consume} : C \to \textsf{bool}\ \}$, use v to "produce" and "consume" a value (i.e., make sure you know how to use the $\textsf{unpack}\ \{X, x\} = e_1\ \textsf{in}\ e_2$ expression.

3 Sub-structural type systems

Recall the linear lambda calculus from Lecture 17.

(a) Suppose we extended the language with a negation operation $\textsf{not}\ e$. Intuitively, e needs to have boolean type, and the expression evaluates e to a value v and then evaluates to the negation of the boolean represented by v. The result of the negation is unrestricted (i.e., it is not linear). Write a typing rule for the negation operator.

(b) Suppose we extended the language with a conjunction operator, $e_1\ \textsf{and}\ e_2$. Intuitively, e_1 and e_2 both need to have boolean type, and the expression evaluates to the conjunction of the booleans. The boolean value that is the result of the conjunction is unrestricted (i.e., it is not linear). Write a typing rule for the conjunction operator.

(c) Let context $\Gamma = x\!:\!\mathsf{lin}\ (\mathsf{lin}\ \textbf{bool} \times \mathsf{un}\ \textbf{bool})$ and expression $e \equiv \mathsf{split}\, x\, \mathsf{as}\, y, z\, \mathsf{in}\, \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \mathsf{not}\ z$.

Is e well-typed for context Γ? That is, does $\Gamma \vdash e\!:\!\mathsf{un}\ \textbf{bool}$ hold? If so, give the derivation.

(d) Consider the execution of the following (well-typed) expression, starting from the empty store. What does the final store look like? (In particular, recall that locations containing linear values are removed once the linear value is used.)

$$(\mathsf{lin}\ \lambda x\!:\!\mathsf{lin}\ (\mathsf{un}\ \textbf{bool} \times \mathsf{un}\ \textbf{bool}).\ \mathsf{split}\, x\, \mathsf{as}\, y, z\, \mathsf{in}\, y)\ (\mathsf{lin}\ (\mathsf{un}\ \mathsf{false}, \mathsf{un}\ \mathsf{true}))$$