CS611 Lecture 36 **The Curry-Howard Isomorphism** 26 November 2003

Scribe: Jonathan Moon Lecturer: Radu Rugina

1 Universal Types

In classical λ-calculus, we presented combinators such as $true = \lambda x.\, \lambda y.\, x$. To use similar combinators in the typed λ-calculus, however, we need to define a version of each combinator for every type. For example,

$$\begin{aligned} true_{\,\mathsf{int}} &= \lambda x : \mathsf{int}.\, \lambda y : \mathsf{int}.\, x \quad : \mathsf{int} \to \mathsf{int} \to \mathsf{int} \\ true_{\,\mathsf{bool}} &= \lambda x : \mathsf{bool}.\, \lambda y : \mathsf{bool}.\, x \quad : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} \end{aligned}$$

Another alternative, which would allow code reuse, is to use a generic combinator universally quantified over all possible types. In that case, the above combinators are instantiations of this generic combinator:

$$true_\Lambda = \Lambda T.\lambda x : T.\ \lambda y : T.\ x \quad : \forall T.T \to T \to T$$

An example of universal types in real languages are templates in C++. Generic types will probably be soon present in Java, too.

We now extend the simply typed λ-calculus to include universal types. The resulting calculus is referred to as the Polymorphic λ-Calculus, or as "System F". Its syntax is as follows:

$$\begin{aligned} e &::= x \mid \lambda x : \tau.e \mid e_0 e_1 \mid \Lambda T.e \mid e[\tau] \\ \tau &::= \tau_1 \to \tau_2 \mid \forall T.\tau \mid T \end{aligned}$$

In the syntax of types, T are type variables which get quantified in universal types $\forall T.\tau$. Note that there are two additional expressions and both have type annotations. The expression $\Lambda T.e$ is an expression having universal type. The expression $e[\tau]$ takes an expression with universal type and instantiates that type to τ.

1.1 Operational Semantics

The semantic rules that describe the execution in the new calculus are:

$$\frac{e \to e'}{e[\tau] \to e'[\tau]} \qquad \frac{}{(\Lambda T.e)[\tau] \to e[T \mapsto \tau]}$$

When evaluating $e[\tau]$, we first reduce expression e as much as possible with the first rule, stopping when we can apply the second. These rules show that expressions with universal types are not being further reduced, so they are treated as values:

$$v ::= \lambda x : \tau.e \mid \Lambda T.e$$

1.2 Typing Rules

We next want to write typing rules to enforce that the evaluation of expressions under the above semantics doesn't get stuck. To express typing the rules, we first modify the definition of typing environments Γ. An environment now contains both mappings from variables to their types, as well as type variables T:

$$\Gamma ::= \phi \mid \Gamma \cup \{x : \tau\} \mid \Gamma \cup \{T\}$$

With this definition, the typing rules for the new constructs are as follows. The first rule introduces universal quantification and the second instantiates the universal type to a particular τ.

$$\frac{\Gamma \cup \{T\} \vdash e : \tau}{\Gamma \vdash \Lambda T.e \ :\ \forall T.\tau} \qquad \frac{\Gamma \vdash e : \forall T.\tau}{\Gamma \vdash e[\tau'] : \tau[T \mapsto \tau']}$$

We can use these rules to prove that the expression $true_\Lambda$ presented above is well typed. We leave the typing derivation for this expression as an exercise. More interesting, one can find appropriate type annotations to make the combinator $self = \lambda x.xx$ well-typed (without the use of recursive types!). Doing so is left as an exercise.

2 Curry-Howard Isomorphism

In the late 60's, researchers have observed that there is a one-to-one relationship between types and logical formulae, and between type derivations and logical proofs. This connection is known as the Curry-Howard Isomorphism.

2.1 Constructive Logic

The branch of logic involved in this isomorphism is known as Constructive Logic (also known as Intuitionistic Logic). The syntax for this logic is the same as the standard second order predicate logic, but the way logic proofs is different.

$$\varphi ::= T \mid F \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid x \mid \forall x.\varphi$$

The intuition behind Constructive logic is that proofs in this system are built by showing that a formula holds, rather by showing that the negation of that formula doesn't hold. Therefore, proofs by contradiction do not hold in this system.

Logic proofs are build using a proof system consisting of a set of rules. This proof system is referred to as *natural deduction*. The judgments in this system are of the form $\Gamma \vdash \varphi$, meaning that we can prove formula φ, given a set of assumptions Γ, which represents a set of formulae. The rules in this proof system are as follows:

$$\frac{}{\Gamma \cup \{\varphi\} \vdash \varphi} \qquad \text{(axiom)}$$

$$\frac{\Gamma \vdash \varphi_1 \quad \Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \wedge \varphi_2} \qquad (\wedge \text{ introduction})$$

$$\frac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_1} \quad \frac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_2} \qquad (\wedge \text{ elimination})$$

$$\frac{\Gamma \vdash \varphi_1}{\Gamma \vdash \varphi_1 \vee \varphi_2} \quad \frac{\Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \vee \varphi_2} \qquad (\vee \text{ introduction})$$

$$\frac{\Gamma \vdash \varphi_1 \vee \varphi_2 \quad \Gamma \vdash \varphi_1 \Rightarrow \varphi_3 \quad \Gamma \vdash \varphi_2 \Rightarrow \varphi_3}{\Gamma \vdash \varphi_3} \qquad (\vee \text{ elimination})$$

$$\frac{\Gamma \cup \{\varphi_1\} \vdash \varphi_2}{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2} \qquad (\Rightarrow \text{ introduction})$$

$$\frac{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2 \quad \Gamma \vdash \varphi_1}{\Gamma \vdash \varphi_2} \qquad (\Rightarrow \text{ elimination})$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x.\varphi} \quad (x \notin FV(\Gamma)) \qquad (\forall \text{ introduction})$$

$$\frac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[x \mapsto \varphi']} \qquad (\forall \text{ elimination})$$

There are two additional rules for constant truth values:

$$\frac{}{\Gamma \vdash T} \quad (T \text{ introduction}) \qquad \frac{\Gamma \vdash F}{\Gamma \vdash \varphi} \quad (F \text{ elimination})$$

The key point in Constructive Logic is the following. If we denote the negation of a formula $\neg\varphi$ to be, by definition, a shorthand for the formula $\varphi \Rightarrow F$, then the above system doesn't allow us to prove the following facts, which hold in classic logic:

$$\frac{\neg\neg\varphi}{\varphi} \qquad \frac{}{\varphi \vee \neg\varphi}$$

2.2 Types and the Isomorphism

Let's now consider the syntax for types:

$$\tau ::= \mathsf{unit} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid T \mid \forall T.\tau$$

If we compare this syntax with the syntax for formulas, we already see similarities. These correspondences become clear by observing the relationship between typing rules $\Gamma \vdash e : \tau$ and logic rules $\Gamma \vdash \varphi$:

Typing rule	**Proof rule**
$\dfrac{\Gamma(x) = \tau}{\Gamma \vdash x.\tau}$	$\dfrac{}{\Gamma \cup \{\varphi\} \vdash \varphi}$
$\dfrac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$	$\dfrac{\Gamma \vdash \varphi_1 \quad \Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \wedge \varphi_2}$
$\dfrac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}\ e : \tau_1} \quad \dfrac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd}\ e : \tau_2}$	$\dfrac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_1} \quad \dfrac{\Gamma \vdash \varphi_1 \wedge \varphi_2}{\Gamma \vdash \varphi_2}$
$\dfrac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) : \tau_1 + \tau_2} \quad \dfrac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) : \tau_1 + \tau_2}$	$\dfrac{\Gamma \vdash \varphi_1}{\Gamma \vdash \varphi_1 \vee \varphi_2} \quad \dfrac{\Gamma \vdash \varphi_2}{\Gamma \vdash \varphi_1 \vee \varphi_2}$
$\dfrac{\Gamma \vdash e_0 : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \to \tau_3 \quad \Gamma \vdash e_2 : \tau_2 \to \tau_3}{\Gamma \vdash \mathsf{case}\ e_0\ \mathsf{of}\ (e_1 \mid e_2) : \tau_3}$	$\dfrac{\Gamma \vdash \varphi_1 \vee \varphi_2 \quad \Gamma \vdash \varphi_1 \Rightarrow \varphi_3 \quad \Gamma \vdash \varphi_2 \Rightarrow \varphi_3}{\Gamma \vdash \varphi_3}$
$\dfrac{\Gamma \cup \{x : \tau_1\} \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \to \tau_2}$	$\dfrac{\Gamma \cup \{\varphi_1\} \vdash \varphi_2}{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2}$
$\dfrac{\Gamma \vdash e : \tau_1 \to \tau_2 \quad \Gamma \vdash e' : \tau_1}{\Gamma \vdash e\ e' : \tau_2}$	$\dfrac{\Gamma \vdash \varphi_1 \Rightarrow \varphi_2 \quad \Gamma \vdash \varphi_1}{\Gamma \vdash \varphi_2}$
$\dfrac{\Gamma \cup \{T\} \vdash e.\tau}{\Gamma \vdash \Lambda T.e : \forall T.\tau}$	$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x.\varphi} \quad (x \notin FV(\Gamma))$
$\dfrac{\Gamma \vdash e : \forall T.\tau}{\Gamma \vdash e[\tau'] : \tau[T \to \tau']}$	$\dfrac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[x \mapsto \varphi']}$

2.3 Summary

The following table summarizes the one-to-one correspondences between the type domain and the logic domain:

types	$\longleftrightarrow$	formulae
typing rules	$\longleftrightarrow$	natural deductive rules
typing derivations	$\longleftrightarrow$	logical proofs
$\to, \times, +, \forall$	$\longleftrightarrow$	$\Rightarrow, \wedge, \vee, \forall$
unit (or 1)	$\longleftrightarrow$	T
0	$\longleftrightarrow$	F
$\tau \to 0$	$\longleftrightarrow$	$\neg\varphi$
$\tau_1 \cong \tau_2$	$\longleftrightarrow$	$\varphi_1 \iff \varphi_2$

2.4 Examples

Finally we consider several examples of logical formulae that have interesting interpretations under this isomorphism.

1. The logic equivalence $(A \wedge B) \Rightarrow C \iff A \Rightarrow B \Rightarrow C$ corresponds to the type isomorphism

$$(\tau_A \times \tau_B) \to \tau_C \cong \tau_A \to \tau_B \to \tau_C$$

   which expresses the fact that the curried and the uncurried versions of a function are isomorphic.

2. The logical formula $(A \Rightarrow B) \ \wedge \ (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$ corresponds to the type of the expression

$$\lambda p \in (\tau_A \to \tau_B) \times (\tau_B \to \tau_C).\lambda x \in A.(\mathsf{snd}\ p)((\mathsf{fst}\ p)x)$$

   which we recognize as the *compose* function, discussed in the Domain Constructions lectures.

3. $(A \Rightarrow B) \iff (A \wedge (B \Rightarrow F)) \Rightarrow F$ corresponds to the type isomorphism:

$$\tau_A \to \tau_B \cong (\tau_A \times (\tau_B \to 0)) \to 0$$

   which expresses the fact that a function is isomorphic to a continuation which takes a pair as argument: the original parameter of the function, and a continuation which is invoked with the result of the function.