CS611 Lecture 29 — Types as propositions — 8 November 2004

Scribe: Mike George, K. Vikram — Lecturer: Andrew Myers

The System F or the $\lambda^{\rightarrow}$ type system that we had been looking at so far has a deep connection to propositional logic (in particular first order propositional logic). The key feature of these systems is that they are *constructive* (also known as *intuitionistic*) in nature. In other words, such systems need a *witness* for any assertion that is proved. The basic philosophy behind intuitionism is that the existence of an object is equivalent to the possibility of construction. This contrasts with the classical approach to mathematics, which states that the existence of an entity can be proved by refuting its non-existence. As a result, the rules of classical logic which involve negation are not part of constructive systems. For instance, rules such as

$$\frac{}{\phi \vee \neg\phi} \quad \text{or} \quad \frac{\neg\neg\phi}{\phi}$$

have no place in constructive logic.

To show this correspondence between type systems and logic, we first describe the logic system that corresponds to a traditional type system. The syntax of this logic is as follows:

$$\begin{aligned}\phi ::= T \ \mid\ F \ \mid\ \phi_1 \Rightarrow \phi_2 \ \mid\ \phi_1 \wedge \phi_2 \ \mid\ \phi_1 \vee \phi_2 \\ \mid\ \forall X \in S.\phi \ \mid\ \exists X \in S.\phi \ \mid\ X\end{aligned}$$

For the definition of implication ($\Rightarrow$), we include *assumptions* in our judgments. For instance, if we wanted to prove ϕ, we have our judgment of the following form

$$\phi_1, \phi_2, ..., \phi_n \vdash \phi$$

This notation means that assuming a set of "assertions", $\phi_1, \phi_2, ..., \phi_n$ we prove that proposition ϕ holds. With this notation, we have the following rules that define $\Rightarrow$

$$\frac{\Gamma, \phi_1 \vdash \phi_2}{\Gamma \vdash \phi_1 \Rightarrow \phi_2} \qquad \text{(introduction rule)}$$

$$\frac{\Gamma \vdash \phi_1 \Rightarrow \phi_2 \quad \Gamma \vdash \phi_1}{\Gamma \vdash \phi_2} \qquad \text{(elimination rule or \textit{modus ponens})}$$

The axioms in this system are

$$\frac{}{\Gamma, \phi \vdash \phi} \quad \text{and} \quad \frac{}{T}$$

The rules for conjunction ($\wedge$) and disjunction ($\vee$) are

Conjunction ☞

$$\frac{\Gamma \vdash \phi_1 \quad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \wedge \phi_2} \qquad \frac{\Gamma \vdash \phi_1 \wedge \phi_2}{\Gamma \vdash \phi_1} \quad \frac{\Gamma \vdash \phi_1 \wedge \phi_2}{\Gamma \vdash \phi_2}$$

(introduction rule) (elimination rules)

Disjunction ☞

$$\frac{\Gamma \vdash \phi_1}{\Gamma \vdash \phi_1 \vee \phi_2} \quad \frac{\Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \vee \phi_2} \qquad \frac{\Gamma \vdash \phi_1 \vee \phi_2 \quad \Gamma \vdash \phi_1 \Rightarrow \phi_3 \quad \Gamma \vdash \phi_2 \Rightarrow \phi_3}{\phi_3}$$

(introduction rules) (elimination rule)

The reader might have noticed that there is no rule which can be used for a 'proof by contradiction' in this logic. The elimination rule for disjunction involves a negation in classical logic, whereas in the system under consideration we have avoided the use of negation.

The rules for the universal quantifier are

$$\frac{\Gamma, X \in S \vdash \phi}{\Gamma \vdash \forall X \in S.\phi} \qquad \frac{\Gamma \vdash \forall X \in S.\phi \quad \Gamma \vdash A \in S}{\Gamma \vdash \phi\{A/X\}}$$

Example: In this proof system let us now try to prove $\forall X, Y, Z.(X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)$. The proof of this would be

$$\frac{\frac{\frac{\frac{\frac{}{\Delta; \Gamma \vdash (X \Rightarrow Y) \wedge (Y \Rightarrow Z)}}{\Delta; \Gamma \vdash Y \Rightarrow Z} \quad \frac{\frac{\frac{}{\Delta; \Gamma \vdash (X \Rightarrow Y) \wedge (Y \Rightarrow Z)}}{\Delta; \Gamma \vdash X \Rightarrow Y} \quad \frac{}{\Delta; \Gamma \vdash X}}{\Delta; \Gamma \vdash Y}}{X, Y, Z \in \mathit{prop}; (X \Rightarrow Y) \wedge (Y \Rightarrow Z), X \vdash Z}}{X, Y, Z \in \mathit{prop}; (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \vdash X \Rightarrow Z}}{X, Y, Z \in \mathit{prop} \vdash (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)}$$

where $\Delta \equiv X, Y, Z \in \mathit{prop}$ and $\Gamma \equiv (X \Rightarrow Y) \wedge (Y \Rightarrow Z), X$. Δ is usually just written as a list of names indicating that these names stand for arbitrary propositions and Γ is a list of propositions.

Let us now see the correspondence to type systems. We shall list here the various rules in a type system and show the corresponding rule in the propositional logic described above.

Type System Rule	Corresponding Propositional Logic Rule
$\frac{}{\Gamma, x : \tau \vdash x : \tau}$	$\frac{}{\Gamma, \phi \vdash \phi}$
$\frac{}{\mathit{unit} : 1} \quad \frac{}{b : B}$	$\frac{}{T}$
$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2}$	$\frac{\Gamma, \phi_1 \vdash \phi_2}{\Gamma \vdash \phi_1 \Rightarrow \phi_2}$
$\frac{\Gamma \vdash e_0 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e_0\ e_1 : \tau_2}$	$\frac{\Gamma \vdash \phi_1 \Rightarrow \phi_2 \quad \Gamma \vdash \phi_1}{\Gamma \vdash \phi_2}$
$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2}$	$\frac{\Gamma \vdash \phi_1 \quad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \wedge \phi_2}$
$\frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash (\mathsf{left}\ e) : \tau_1}$	$\frac{\Gamma \vdash \phi_1 \wedge \phi_2}{\Gamma \vdash \phi_1}$
$\frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash (\mathsf{right}\ e) : \tau_2}$	$\frac{\Gamma \vdash \phi_1 \wedge \phi_2}{\Gamma \vdash \phi_2}$
$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash (\mathsf{inl}\ e) : \tau_1 + \tau_2}$	$\frac{\Gamma \vdash \phi_1}{\Gamma \vdash \phi_1 \vee \phi_2}$
$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash (\mathsf{inr}\ e) : \tau_1 + \tau_2}$	$\frac{\Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \vee \phi_2}$
$\frac{\Gamma \vdash e_1 : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_3 \quad \Gamma \vdash e_2 : \tau_2 \rightarrow \tau_3}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ e_1 \vert e_2 : \tau_3}$	$\frac{\Gamma \vdash \phi_1 \vee \phi_2 \quad \Gamma \vdash \phi_1 \Rightarrow \phi_3 \quad \Gamma \vdash \phi_2 \Rightarrow \phi_3}{\phi_3}$
$\frac{\Delta, X; \Gamma \vdash e : \sigma}{\Delta; \Gamma \vdash \Lambda X.e : \forall X.\sigma}$	$\frac{\Gamma, X \in S \vdash \phi}{\Gamma \vdash \forall X \in S.\phi}$
$\frac{\Delta; \Gamma \vdash e : \forall X.\sigma}{\Delta; \Gamma \vdash e[\tau] : \sigma\{\tau/X\}}$	$\frac{\Gamma \vdash \forall X \in S.\phi \quad \Gamma \vdash A \in S}{\Gamma \vdash \phi\{A/X\}}$

The above correspondence is obtained by essentially replacing a ϕ by a τ.

Given that we have this exact correspondence between typing rules in our type system and inference rules in constructive logic, we also have a correspondence between well-typed terms and proofs. In particular, we can view well-typed programs as proofs of theorems, and vice versa. For example, we can construct the following program:

$$\Lambda X.\Lambda Y.\Lambda Z.\ \lambda p\!:\!(X \to Y) * (Y \to Z).\ \lambda q\!:\!X.(\mathit{right}\ p)\ ((\mathit{left}\ p)\ q)$$

which corresponds to the logical statement

$$\forall X, Y, Z.(X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)$$

This correspondence between the terms (or their types) and proofs of propositions is also known as the *Curry-Howard Correspondence.*

Under this translation, the logical interpretation of the type τ is that the type τ is inhabited, i.e. there are some computations that result in type τ. Thus, the program can be thought of as method for producing the computation that results in type τ, which in turns proves that τ is inhabited. The fact that the rules match up exactly means that if we have a well-typed program, and we think of it as a proof, then each step in that proof is valid logically.

Here is a table which shows how to translate concepts between type theory and logic:

Types	**Propositions**
Well-formed terms	Proofs
Soundness and strong normalization	Soundness
type constructors $\to$, $*$, and $+$ respectively	logical connectives $\Rightarrow$, $\wedge$, and $\vee$ respectively
Abstracting over types using Λ	Abstracting over propositions using $\forall$
Base types, such as **1** and **int**, which we assume are inhabited	*true*
We can introduce a type **0** which is not inhabited, i.e. which no computation can return.	*false*
A pair of functions, one of type $\tau_1 \to \tau_2$ and the other of type $\tau_2 \to \tau_1$ (possibly but not necessarily an isomorphism of domains).	A logical equivalence $\phi_1 \equiv \phi_2$.

Given this translation of concepts, we can begin to take ideas from logic and pull them back to create ideas in type systems, which we will see are actually familiar concepts that we have seen before. For example, we can pull back the following logical identity:

$$A \wedge B \Rightarrow C \equiv A \Rightarrow (B \Rightarrow C)$$

and we see that there should be functions between the types $\tau_1 * \tau_2 \to \tau_3$ and $\tau_1 \to \tau_2 \to \tau_3$. But we've already seen these functions: the curry operator takes something of the form $\tau_1 * \tau_2 \to \tau_3$ and produces something of the type $\tau_1 \to \tau_2 \to \tau_3$, and the uncurry operator goes in the other direction. Note that the logical equivalence doesn't imply that the two corresponding functions are inverses as they are in this case, but in practice they often are.

Another example of something we might add to our logic and then pull back to our type system is existential quantifiers. If we pull the concept of existential quantifiers back into a type system, we get universal types, which will be discussed further in lecture 34.

A third example of the correspondence between types and logic is the connection between continuation and logical negation. τ-Continuations (i.e. a continuation that expects a value of type τ) can be thought of as computations that don't return, which in turn can be thought of as functions of the type $\tau \to \mathbf{0}$. On the other hand, for any proposition ϕ, $\neg\phi$ is logically equivalent to $\phi \Rightarrow \mathit{false}$, so using the correspondence above, we see that $\neg\phi$ corresponds to a τ continuation.

Now, constructive logic does not permit us to reduce $\neg\neg\phi$ to ϕ, but there is a sound translation from logical expressions to logical expressions given by replacing every term ϕ by $\neg\neg\phi$. What happens if we pull this translation back to our type system? We want to replace a type τ by $(\tau \to \mathbf{0}) \to \mathbf{0}$, so we might try something like:

$$\begin{aligned} [\![\Gamma \vdash n : \mathbb{Z}]\!] &= \lambda k : \mathbb{Z} \to \mathbf{0}. k\ n \\ [\![\Gamma \vdash x : \tau]\!] &= \lambda k : \tau \to \mathbf{0}. k\ x \end{aligned}$$

To figure out the translations of applications and lambdas, lets first look at the types. We'll take the type of a lambda term $\tau_1 \to \tau_2$ and think of it as the proposition $\phi_1 \Rightarrow \phi_2$. Applying our double negation translation, we have $\neg\neg(\phi_1 \Rightarrow \neg\neg\phi_2)$, which as a type is written

$$((\tau_1 \to (\tau_2 \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}$$

Let's try to construct a term of this type, given a lambda term of type $\tau_1 \to \tau_2$:

$$\begin{aligned} [\![\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \to \tau_2]\!] &= \lambda k : (\tau_1 \to (\tau_2 \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}. \\ &\qquad k\ (\lambda x : \tau_1.\lambda k' : \tau_2 \to \mathbf{0}. \underbrace{[\![\Gamma, x : \tau_1 \vdash e : \tau_2]\!]}_{(\tau_2 \to \mathbf{0}) \to \mathbf{0}}\ k') \end{aligned}$$

Finally, we can translate applications. Given an expression $\Gamma \vdash e_0\ e_1 : \tau$, we know that $\Gamma \vdash e_0 : \tau' \to \tau$ and $\Gamma \vdash e_1 : \tau'$, so we know that $[\![\Gamma \vdash e_0 : \tau' \to \tau]\!]$ has type $((\tau' \to (\tau \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}$, and $[\![\Gamma \vdash e_1 : \tau']\!]$ has type $(\tau' \to \mathbf{0}) \to \mathbf{0}$. Thus, let's construct a translation that has the right type (namely $\neg\neg\phi$ or $(\tau \to \mathbf{0}) \to \mathbf{0}$):

$$[\![\Gamma \vdash e_0\ e_1 : \tau]\!] = \lambda k : \tau \to \mathbf{0}.\ \underbrace{[\![\Gamma \vdash e_0 : \tau' \to \tau]\!]}_{((\tau' \to (\tau \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}}\ (\lambda f : \tau' \to (\tau \to \mathbf{0}) \to \mathbf{0}. \underbrace{[\![\Gamma \vdash e_1 : \tau']\!]}_{(\tau' \to \mathbf{0}) \to \mathbf{0}}(\lambda v : \tau'.f\ v\ k))$$

A quick check shows that this expression type checks, and has the desired type.

All of these expressions have the right type, but are they really an accurate translation of our original terms? And even if it is, why is it interesting? The answer is that if we ignore all of the typing annotations, then this is exactly our CPS translation! This shows that the process of CPS translation is completely analogous to adding some double negations into the logical statements that mirror our type system.