Scribe: Christopher Capobianco and Walter Chang Lecturer: Andrew Myers

1 Curry-Howard Isomorphism (a.k.a. formulas-as-types)

1.1 History

In 1968, Mathematician William Howard, building on work by Haskel Curry, identified a one-to-one relationship between propositional formulas and logical proofs to types and programs respectively. More genreally, it was noticed that logical ideas have computational significance. This idea became known, rather naturally, as the Curry-Howard Isomorphism.

1.2 Logical Formulas

Here we define the form of the logical expressions we will use in our exploration of the Curry-Howard Isomorphism

$$\phi \quad ::= \quad T \mid F \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \neg\, \phi \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \mid x$$

1.3 Proof Rules

Now that we have our logical formulas, we need a way to prove any logical assertion. And so we define proof rules.

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

$$\frac{\phi_1}{\phi_1 \vee \phi_2} \qquad \frac{\phi_2}{\phi_1 \vee \phi_2}$$

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

2 Constructive Logic (a.k.a. Intuitionistic)

In Constructive Logic one needs to prove a logical formula is true by proving it is true, not by proving the negation is false (proof by contradiction). While the latter might be perfectly acceptable in classical logic, that method cannot be used in the Constructive logic system. So the following classical logic proof rules are not found in Constructive logic:

$$\frac{\neg\,\neg\,\phi}{\phi} \qquad \frac{}{\phi \vee \neg\,\phi} \qquad \frac{\phi_1 \vee \phi_2 \quad \neg\,\phi}{\phi_2}$$

For an $\Rightarrow$, we assume the hypothesis to be true and need to prove the consequence.

3 Natural Deduction

In 1934 Gerhard Getzen introduced Natural deduction. This proof method uses a list of assumptions to come upon a conclusion: $\phi_1, ..., \phi_n \vdash \phi$. That is we assert that ϕ is true based upon the assumption that $\phi_1, ..., \phi_n$ are all true. We call the set of assumptions, $\phi_1, ..., \phi_n$, Γ.

We need to stick Γ into the logical proof rules we defined above, which would result in:

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

$$\frac{\Gamma \vdash \phi_1}{\Gamma \vdash \phi_1 \vee \phi_2} \qquad \frac{\Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \vee \phi_2}$$

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

We now provide a proof to illustrate the application of the proof rules; here is the proof of transitivity of implications:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{... \vdash X \Rightarrow Y \wedge Y \Rightarrow Z}}{... \vdash Y \Rightarrow Z} \quad \dfrac{\dfrac{\dfrac{}{... \vdash X \Rightarrow Y \wedge Y \Rightarrow Z}}{... \vdash X \Rightarrow Y} \quad \dfrac{}{... \vdash X}}{... \vdash Y}}{X, Y, Z;\ (X \Rightarrow Y) \wedge (Y \Rightarrow Z), X \vdash Z}}{X, Y, Z;\ (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \vdash (X \Rightarrow Z)}}{\dfrac{X, Y, Z;\ \emptyset \vdash (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)}{\forall X, Y, Z\ .\ (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)}}$$

The proof tree for logical formulas, $\Gamma \vdash \phi$, looks like that for typing, $\Gamma \vdash e : \tau$. If we look closer to the each of the logical proof rules, we can find a corresponding one in the typing proof rules, e.g.:

$$\frac{\Gamma \vdash \phi_1 \quad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \wedge \phi_2} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 * \tau_2}$$

$$\frac{\Gamma \vdash \phi_1 \wedge \phi_2}{\Gamma \vdash \phi_1} \qquad \frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash left\ e : \tau_1}$$

$$\frac{\Gamma \vdash \phi_1}{\Gamma \vdash \phi_1 \vee \phi_2} \qquad \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash inl\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash \phi_1 \vee \Gamma \vdash \phi_2 \quad \Gamma \vdash \phi_1 \Rightarrow \phi_3 \quad \Gamma \vdash \phi_1 \Rightarrow \phi_3}{\Gamma \vdash \phi_3} \qquad \frac{\Gamma \vdash e_0 : \tau_1 + \tau_2 \quad \Gamma \vdash \lambda x.\, e_1 : \tau_1 \rightarrow \tau_3 \quad \Gamma \vdash \lambda x.\, e_2 : \tau_2 \rightarrow \tau_3}{\Gamma \vdash case\ e_0\ of\ x.e_1 \ \mid\ x.e_2 : \tau_3}$$

4 The Curry-Howard Isomorphism

We notice the following isomorphism between logical rules and typing rules.

Logical Rule	Typing Rule
$\wedge$	$*$
$\vee$	$+$
$\Rightarrow$	$\rightarrow$
$\forall$	$\forall$
T	1(B)
F	0 (empty domain)
$\phi_1 \Leftrightarrow \phi_2$	$\tau_1 \equiv \tau_2$
$\neg\phi$	$\tau \rightarrow 0$

Under this isomorphism, since $\vdash \phi \Leftrightarrow \ \vdash e : \tau$, if a statement is logically derivable then there is a program and a value of type τ. We say that τ is inhabited if you can construct a value of that type (it is possible to construct types for which no values exist; these types are uninhabited)

Since the proof is encoded in the structure of the term itself, it turns out that this is very useful for proof carrying code.

5 Logical Tautologies

Several well-known logical tautologies have interesting interpretations under this isomorphism. For example,

$(A \wedge B \Rightarrow C) \Leftrightarrow (A \Rightarrow B \Rightarrow C)$ becomes

$A * B \rightarrow C \equiv A \rightarrow B \rightarrow C$ which is a statement about currying and uncurrying.

Also, deMorgan's law, $A \wedge B \Leftrightarrow \neg(\neg A \vee \neg B)$ suggests that there is a way to encode sums using products and vice versa.

Lastly, double negation translates to $\tau \equiv (\tau \rightarrow 0) \rightarrow 0$ is continuation passing under the isomorphism:

$$\begin{aligned}
\mathcal{D}[\![1]\!] &= 1 \\
\mathcal{D}[\![\tau \rightarrow \tau']\!] &= \tau \rightarrow (\tau' \rightarrow 0) \rightarrow 0 \\
\mathcal{D}[\![\Gamma \vdash e : \tau]\!] &= ([\![\tau]\!] \rightarrow 0) \rightarrow 0 \\
\mathcal{D}[\![\Gamma, x : \tau \vdash x : \tau]\!] &= \lambda k : [\![\tau]\!] \rightarrow 0.\ k\ x \\
\mathcal{D}[\![\Gamma \vdash e_0\ e_1 : \tau']\!] &= \lambda k . [\![\tau']\!] \rightarrow 0.\ \mathcal{D}[\![\Gamma \vdash e_0 : \tau \rightarrow \tau']\!](\lambda f : [\![\tau \rightarrow \tau']\!].\ \mathcal{D}[\![\Gamma \vdash e_1 : \tau]\!](\lambda v : [\![\tau]\!].\ f\ v\ k)) : ([\![\tau']\!] \rightarrow 0) \\
\mathcal{D}[\![\Gamma \vdash \lambda x : \tau\ e : \tau \rightarrow \tau']\!] &= \lambda k : [\![tau \rightarrow \tau']\!] \rightarrow 0.\ k\ (\lambda v : [\![\tau]\!].\ \lambda k' : [\![\tau']\!] \rightarrow 0.\ \mathcal{D}[\![\Gamma \vdash e : \tau']\!] k')
\end{aligned}$$