

Comp 411 Notes - Fri, Feb 11 2005

LAMBDA EXTENSION

Scribes: Seth James Nielson and Mathias Ricken

Today we revisited the *Curry-Howard Isomorphism*, which relates logic with propositions to type systems. In any logical or type derivation it is important to restrict that derivation to a single constructor. Each operator should have at least one rule for introducing the operator (*Introduction* rule) and at least one rule for eliminating it (oddly enough an *Elimination* rule).

1 Review

What is the Curry-Howard isomorphism? It is the bijective mapping between types and propositions. The logical derivations that establish the truth of a proposition translate into the type productions that prove that a type is inhabited, and vice versa.

What does it mean for a type to be inhabited? A type is inhabited if there exists a term with that type.

Below is a simple illustration of the related syntax in the Curry-Howard isomorphism:

Propositional Logic			Type Systems		
$P ::=$	A	atomics	$t ::=$	b	primitives
	$P \Rightarrow P$	implications		$t \rightarrow t$	functions
	$P \wedge P$	conjunctions		$t \times t$	pairs
	$P \vee P$	disjunctions		$t + t$	sums

2 Introduction and Elimination Rules

Every term should be introducible and eliminatable from an expression. These rules are described below.

- *Introduction* rules - Under what hypotheses can we prove a proposition of a certain form?
- *Elimination* rules - What other propositions can we prove if our hypotheses prove a proposition of a certain form?

For example, here are the introduction and elimination rules for the implication operator \Rightarrow :

Introduction:

$$\frac{H, P_1 \vdash P_2}{H \vdash P_1 \Rightarrow P_2}$$

If we can prove P_2 under the hypotheses H and P_1 , then we can prove the term $P_1 \Rightarrow P_2$ under the hypotheses H . We are using our hypotheses to *introduce* a term of the form $P \Rightarrow P$.

Elimination:

$$\frac{H \vdash P_1 \Rightarrow P_2 \quad H \vdash P_1}{H \vdash P_2}$$

If we can prove both $P_1 \Rightarrow P_2$ and P_1 under the hypotheses H , then we can prove P_2 . We are using our hypothesis and a term of the form $P \Rightarrow P$ to prove something else that does not contain $P \Rightarrow P$ anymore. We are *eliminating* $P \Rightarrow P$.

The rules for implications in propositional logic correspond to the rules we have seen for function types in the lambda calculus.

2.1 Rules for and

Here are the introduction and elimination rules for **and**:

Introduction:

$$\frac{H \vdash P_1 \quad H \vdash P_2}{H \vdash P_1 \wedge P_2}$$

Elimination (2 rules):

$$\frac{H \vdash P_1 \wedge P_2}{H \vdash P_1} \quad \frac{H \vdash P_1 \wedge P_2}{H \vdash P_2}$$

These rules correspond to the rules for pairs in the extended lambda calculus.

2.2 Rules for or

And the introduction and elimination rules for **or**:

Introduction (2 rules):

$$\frac{H \vdash P_1}{H \vdash P_1 \vee P_2} \quad \frac{H \vdash P_2}{H \vdash P_1 \vee P_2}$$

Elimination:

$$\frac{H \vdash P_1 \vee P_2 \quad H, P_1 \vdash P_3 \quad H, P_2 \vdash P_3}{H \vdash P_3}$$

NOTE: The elimination rule listed above reads *If P1 or P2 and if P1 implies P3 and if P2 implies P3 then (for sure) P3*. This is a subtle rule.

These rules correspond to the rules for sum types in the extended lambda calculus, which are discussed below.

NOTE: In any given derivation, there should generally only be one constructor at a time.

Below is an example of a bad introduction rule (because it introduces two constructors):

$$\frac{H, P_1 \vdash P_3 \quad H, P_2 \vdash P_3}{H \vdash P_1 \vee P_2 \Rightarrow P_3}$$

2.3 Combination of Introduction and Elimination Rules

You can combine introduction and elimination rules:

$$E \frac{I \frac{H, P_1 \vdash P_2}{H \vdash P_1 \Rightarrow P_2} \quad H \vdash P_1}{H \vdash P_2}$$

Note that the consequent of the introduction rule matches one of the antecedents of the elimination rule.

Similarly, you can combine elimination and introduction:

$$I \frac{E \frac{H, P_1 \vdash P_1 \Rightarrow P_2 \quad H, P_1 \vdash P_1}{H, P_1 \vdash P_2}}{H \vdash P_1 \Rightarrow P_2}$$

Again, the consequent of the elimination rule matches the antecedent of the introduction rule. The second antecedent of the elimination rule, $H, P_1 \vdash P_1$ is always true by the variable rule.

2.4 Rules for Sum Types

Here are typing rules for sum types, the corresponding lambda calculus term for disjunctions in logic.

Introduction (2 rules):

$$\frac{\Gamma \vdash e : t_1}{\Gamma \vdash \mathbf{Left} \ e : t_1 + t_2} \quad \frac{\Gamma \vdash e : t_2}{\Gamma \vdash \mathbf{Right} \ e : t_1 + t_2}$$

We create elements of the sum type $t_1 + t_2$ by tagging a term e with either **Left** or **Right**, respectively, depending on whether the type of e is t_1 or t_2 . (The terms **Left** and **Right** are called **inl** and **inr** in the book.)

Elimination:

$$\frac{\Gamma \vdash e_0 : t_1 + t_2 \quad \Gamma, x : t_1 \vdash e_1 : t_3 \quad \Gamma, x : t_2 \vdash e_2 : t_3}{\Gamma \vdash \left(\begin{array}{l} \mathbf{match} \ e_0 \ \mathbf{with} \\ \quad \mathbf{Left} \ x \rightarrow e_1 \\ \quad \mathbf{Right} \ x \rightarrow e_2 \end{array} \right) : t_3}$$

Note that both branches must have the same type t_3 in the **match** statement.

The rules for sum types correspond to the rules for disjunctions in propositional logic.