1 Proofs Are Programs

As discussed previously in lecture, there is a tight correspondence between the structure of a derivation for a constructive proof and a term in some particular programming language. This leads to the slogans “proofs are programs” and “propositions are types”. The (Curry-Howard-Lambek) correspondence can be fleshed out for the logic we’re studying (intuitionistic propositional logic) by the following table

<table>
<thead>
<tr>
<th>Propositions</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \land B$</td>
<td>$A \ast B$</td>
</tr>
<tr>
<td>$A \lor B$</td>
<td>$A + B$</td>
</tr>
<tr>
<td>$A \supset B$</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$\top$</td>
<td>1 (unit)</td>
</tr>
<tr>
<td>$\bot$</td>
<td>0 (void)</td>
</tr>
</tbody>
</table>

Based on this we can produce a version of our rules from the previous recitation that annotate each proposition step in the derivation with the program that it constructs. Those rules are:

- $\frac{M : A \quad N : B}{\langle M, N \rangle : A \land B}$ ($\land$I)
- $\frac{M : A \land B}{\text{fst } M : A}$ ($\land$E$_1$)
- $\frac{M : A \land B}{\text{snd } M : B}$ ($\land$E$_2$)
- $\frac{M : A}{\text{inl } M : A \lor B}$ ($\lor$I$_1$)
- $\frac{N : B}{\text{inr } N : A \lor B}$ ($\lor$I$_2$)
- $\frac{M : A \lor B \quad N : C \quad O : C}{\text{case } M \text{ of inl } u \Rightarrow N \mid \text{inr } w \Rightarrow O} \quad \lor E^{u,w}$
- $\frac{u : A}{\frac{M : B}{\text{fn } u \Rightarrow M : A \supset B}} \quad \supset I^u$
- $\frac{M : A \supset B \quad N : A}{M \circ N : B} \quad \supset E$
- $\frac{\langle \rangle : \top}{\top I}$
- $\frac{M : \bot}{\text{abort } M : A} \quad \bot E$

2 Translation

We now turn to the question of translating proofs to programs and back again. In these notes, we present both for the sake of accessibility.

**Task 1.** $(A \supset B \supset C) \supset (B \supset A \supset C)$

---

1 Of course, what makes this correspondence so remarkable is that it extends far beyond this one logic. It is quite robust and extends to almost any well-behaved logic. It also maps between logic and functional programming and lattices which are just closed cartesian categories.
Solution 1: Proof:

\[
\begin{array}{c}
\text{true} A \supset B \supset C \\
\hline
\text{true} A \\
\hline
\text{true} B \supset C \\
\hline
\text{true} B \\
\hline
\text{true} C \\
\hline
\text{true} A \supset C \\
\hline
\text{true} B \supset A \supset C \\
\hline
\end{array}
\]

Program:

\[
\text{fn } f = \text{fn } b = \text{fn } a = (f a) b
\]

Task 2. \((A \supset B) \lor (A \supset C) \supset A \supset (B \lor C)\)

Solution 2: Proof:

Let \(X\) be:

\[
\begin{array}{c}
\text{true} A \supset B \\
\hline
\text{true} A \\
\hline
\text{true} B \\
\hline
\text{true} B \lor C \\
\hline
\end{array}
\]

Let \(Y\) be:

\[
\begin{array}{c}
\text{true} A \supset C \\
\hline
\text{true} A \\
\hline
\text{true} C \\
\hline
\text{true} B \lor C \\
\hline
\end{array}
\]

The overall proof is:

\[
\begin{array}{c}
\text{true}(A \supset B) \lor (A \supset C) \\
\hline
\text{true} A \\
\hline
\text{true} B \lor C \\
\hline
\end{array}
\]

Program:

\[
\text{fn } u \Rightarrow \text{fn } v \Rightarrow \text{case } u \text{ of } \text{inl } f \Rightarrow \text{inl } (f v) \mid \text{inr } g \Rightarrow \text{inr } (g v)
\]

3 Inventing proof terms

Task 3. Let’s consider a new connective \(\land\). We’ll give the intro and elim rules and try to come up with constructors, destructors and reduction rules that make sense.

\[
\begin{array}{c}
\text{true} B \quad u \\
\hline
\text{true} A \\
\hline
\text{true} A \land B \quad \land I_1
\end{array}
\]

\[
\begin{array}{c}
\text{true} B \\
\hline
\text{true} A \land B \quad \land I_2
\end{array}
\]

\[
\begin{array}{c}
\text{true} A \\
\hline
\text{true} \bot \quad u \\
\hline
\text{true} A \land B \quad \land I_1
\end{array}
\]

\[
\begin{array}{c}
\text{true} B \\
\hline
\text{true} A \land B \quad \land I_2
\end{array}
\]
Solution 3: Let’s come up with constructors that make sense for \( \land \)

\[
\begin{array}{c}
\frac{}{trueA} \\
\frac{}{trueB} \\
\frac{}{trueC}
\end{array} \quad \frac{}{falseA} \quad \frac{}{falseB} \quad \frac{}{falseC}
\]

\[
\frac{trueA \land falseB}{falseC} \quad \frac{falseA \land trueB}{falseC}
\]

And the destructor...

\[
\begin{array}{c}
\frac{u : A}{falseA} \\
\frac{}{falseB} \\
\frac{}{falseC}
\end{array} \quad \frac{v : falseB}{falseC} \\
\frac{w : falseA}{falseC} \quad \frac{x : B}{falseC}
\]

Now we still need to define a reduction rule for \( \land \). Reduction rules are applied when the destructor is applied to a constructor.

\[
\begin{align*}
\text{case } \text{lt}(u, v) & \Rightarrow M \mid \text{rt}(w, x) \Rightarrow N : C \\
\text{case } \text{rt}(u, M, N) & \Rightarrow A \land B
\end{align*}
\]

4 Reductions

Let’s try reducing a term until we can no longer apply reduction rules.

Task 4.

\[
\begin{align*}
fn a & \Rightarrow fn b \Rightarrow (fn f \Rightarrow fn p \Rightarrow ((fn (fn u \Rightarrow a, fn u \Rightarrow b)) (fn p))) \langle fn u \Rightarrow a, fn u \Rightarrow b \rangle \langle b, a \rangle
\end{align*}
\]

Solution 4:

\[
\begin{align*}
fn a & \Rightarrow fn b \Rightarrow (fn p \Rightarrow ((fn (fn u \Rightarrow a, fn u \Rightarrow b)) (fn p))) \langle fn u \Rightarrow a, fn u \Rightarrow b \rangle \langle b, a \rangle
\end{align*}
\]

Notice at this point we have a few options on how to proceed. It’s actually the case that there is a term that we will reach no matter which order we apply reduction rules. It’s generally known as the Church Rosser theorem that if a term finishes reducing in two ways, then they arrive at the same place. With our system we’ll always react a “normal” form, so we can apply rules in such a way that save us the trouble of writing a lot.

\[
\begin{align*}
fn a & \Rightarrow fn b \Rightarrow (fn p \Rightarrow ((fn (fn u \Rightarrow a, fn u \Rightarrow b)) (fn p))) \langle fn u \Rightarrow a, fn u \Rightarrow b \rangle \langle b, a \rangle
\end{align*}
\]
\[ \text{fn } a \Rightarrow \text{fn } b \Rightarrow (\text{fn } p \Rightarrow (\text{fn } u \Rightarrow a) (\text{fst } p), (\text{fn } u \Rightarrow b) (\text{snd } p))) \langle b, a \rangle \]

\[ \text{fn } a \Rightarrow \text{fn } b \Rightarrow (\text{fn } p \Rightarrow \langle a, (\text{fn } u \Rightarrow b) (\text{snd } p) \rangle) \langle b, a \rangle \]

\[ \text{fn } a \Rightarrow \text{fn } b \Rightarrow (\text{fn } p \Rightarrow \langle a, b \rangle) \langle b, a \rangle \]

\[ \text{fn } a \Rightarrow \text{fn } b \Rightarrow \langle a, b \rangle \]
1 Harmony

Proof-theoretic harmony is a necessary, but not sufficient, condition for the well-behavedness of a logic; harmony ensures that the connectives are locally well-behaved, and is closely related to the critical cases of cut and identity elimination which we may discuss later on. Therefore, when designing or extending a logic, checking harmony is a first step.

From the verificationist standpoint, a connective is harmonious if its elimination rules are neither too strong nor too weak in relation to its introduction rules. The first condition is called local soundness and the second condition is called local completeness. The content of the soundness condition is a method to reduce or simplify proofs, and the content of completeness is a method to expand any arbitrary proof into a canonical proof (i.e. one that ends in an introduction rule).

1.1 Conjunction

Local soundness for conjunction is witnessed by the following two reduction rules:

Local completeness is witnessed by the following expansion rule:

When regarded as generating relations on programs rather than proofs, the reduction and expansion rules can be recast into another familiar format:

1.2 Disjunction

Local soundness:
Local completeness:

\[
\begin{array}{c}
\frac{D}{A \lor B \text{ true}} \\
M \rightarrow E
\end{array}
\] 

1.3 Implication

Local soundness:

\[
\begin{array}{c}
\frac{A \text{ true}}{D} & \frac{B \text{ true}}{E} \\
A \supset B \text{ true} & \rightarrow
\end{array}
\]

Local completeness:

\[
\begin{array}{c}
\frac{D}{A \supset B \text{ true}} & \frac{A \text{ true}}{E} \\
M \rightarrow E & \text{case } M \text{ of } \text{inl } u \Rightarrow \text{inl } u \mid \text{inr } v \Rightarrow \text{inr } v
\end{array}
\]

1.4 Experiment: Alternative Implication

What if we replaced the \(\supset E\) rule with the following elimination rule:

\[
\begin{array}{c}
\frac{B \text{ true}}{u} & \frac{A \supset B \text{ true}}{A \text{ true}} & \frac{C \text{ true}}{E}
\end{array}
\]

The program/proof term assignment is as follows:

\[
\begin{array}{c}
\frac{u : B}{L : A \supset B \quad M : A \quad N : C} & \frac{u}{\text{let } u = LM \quad \text{in } N : C \rightarrow E}
\end{array}
\]

Task 1. Can we show local soundness and completeness for this version of the implication connective?

Solution 1:
Task 2. Last week in recitation we made up introduction and elimination rules to go with a new connective $\&$. They are listed again below. Show local soundness and completeness of this connective’s rules.

Solution 2: Local soundness:

"Proof search" is not a mere matter of practice: it is praxis. The dialectic of proof search is to discover ways to pare down the state space of a logic, and then synthesize this into a new logic which is exactly as expressive as the old one. This new restricted logic not only has better search complexity, but also exposes critical semantic content which tends to have been obscured in the original logic.

Perhaps the most famous example of this process is Andreoli’s focalization; in recent lectures, we have begun to study a simpler instance of this process, namely the decomposition of truth into verification and use. The passage to verifications constitutes a collation of upward and downward deductions respectively.
2 Verifications and Uses

“Verifications” are proofs that proceed upwards from conclusions to premises; this is also known as backward inference or refinement-style proof. On the other hand, “uses” are proofs that proceed from premise to conclusion, also known as forward inference. The judgment \( A \uparrow \) stands for verifications of \( A \), and the judgment \( A \downarrow \) stands for uses of \( A \).

The rules for verifications and uses of the conjunction connective are as follows:

\[
\begin{align*}
A \uparrow & \quad B \uparrow & \quad \wedge I \\
\hline
A \wedge B \uparrow & \\
\hline
A \wedge B \downarrow & \quad \wedge E_1 \\
A \downarrow & \quad B \downarrow & \quad \wedge E_2
\end{align*}
\]

On this basis, you may think that verifications correspond to introduction forms and uses correspond to elimination forms. This is not correct, as can be seen from the case of disjunction:

\[
\begin{align*}
A \uparrow & \quad \lor_1 \\
\hline
A \lor B \uparrow & \\
\hline
B \uparrow & \quad \lor_2 \\
\hline
A \lor B \downarrow & \quad C \uparrow & \quad C \uparrow & \quad \lor E^{u,v}
\end{align*}
\]

Will the elimination rule for implication result have a verification or a use in its conclusion?

\[
\begin{align*}
A \downarrow & \quad A \uparrow & \quad B \uparrow & \quad \supset I^u \\
\hline
A \supset B \uparrow \\
\hline
A \supset B \downarrow & \quad A \uparrow & \quad B \downarrow & \quad \supset E
\end{align*}
\]

One dimension along which connectives vary is polarity: some connectives are positive, and some are negative. We cannot yet make this distinction precise, but some students have already begun to observe it. Later on, we may see that negative connectives have elimination forms as uses, but positive connectives have elimination forms as verifications.

The calculus of verifications and uses has one extra rule which was not visible in the original logic:

\[
\begin{align*}
A \downarrow & \quad \uparrow
\end{align*}
\]

Would it be reasonable to add the inverse of the above rule, which concludes \( A \downarrow \) from \( A \uparrow \)? What would be the consequences of this?