```
term comp : (A => B) & (B => C) => (A => C) =
fn u => fn x => (snd u) ((fst u) x);
```

We also allow annotated deductions, where each line is annotated with a proof term. This is a direct transcription of deduction for judgments of the form $M : A$. As an example, we show the proof that $A \vee B \supset B \vee A$, first in the pure form.

```
proof orcomm : A | B => B | A =
begin
[ A | B;
  [ A;
    B | A];
  [ B;
    B | A];
  B | A ];
A | B => B | A
end;
```

Now we systematically annotate each line and obtain

```
annotated proof orcomm : A | B => B | A =
begin
[ u : A | B;
  [ v : A;
    inr v : B | A];
  [ w : B;
    inl w : B | A];
  case u
    of inl v => inr v
     | inr w => inl w
   end : B | A ];
fn u => case u
          of inl v => inr v
           | inr w => inl w
         end : A | B => B | A
end;
```

1.4 Properties of Proof Terms

In this section we analyze and verify various properties of proof terms. Rather than concentrate on reasoning within the logical calculi we introduced, we now want to reason about them. The techniques are very similar—they echo the ones we have introduced so far in natural deduction. This should not be surprising. After all, natural deduction was introduced to model mathematical reasoning, and we now engage in some mathematical reasoning about proof terms, propositions, and deductions. We refer to this as *meta-logical reasoning.*

First, we need some more formal definitions for certain operations on proof terms, to be used in our meta-logical analysis. One rather intuitive property of is that variable names should not matter. For example, the identity function at type A can be written as $\lambda u{:}A.\ u$ or $\lambda w{:}A.\ w$ or $\lambda u'{:}A.\ u'$, etc. They all denote the same function and the same proof. We therefore identify terms which differ only in the names of variables (here called u) bound in $\lambda u{:}A.\ M$, $\mathbf{inl}\, u \Rightarrow M$ or $\mathbf{inr}\, u \Rightarrow O$. But there are pitfalls with this convention: variables have to be renamed *consistently* so that every variable refers to the same binder before and after the renaming. For example (omitting type labels for brevity):

$$\begin{array}{rcl}
\lambda u.\ u & = & \lambda w.\ w \\
\lambda u.\ \lambda w.\ u & = & \lambda u'.\ \lambda w.\ u' \\
\lambda u.\ \lambda w.\ u & \neq & \lambda u.\ \lambda w.\ w \\
\lambda u.\ \lambda w.\ u & \neq & \lambda w.\ \lambda w.\ w \\
\lambda u.\ \lambda w.\ w & = & \lambda w.\ \lambda w.\ w
\end{array}$$

The convention to identify terms which differ only in the naming of their bound variables goes back to the first papers on the λ-calculus by Church and Rosser [CR36], is called the "*variable name convention*" and is pervasive in the literature on programming languages and λ-calculi. The term λ-calculus typically refers to a pure calculus of functions formed with λ-abstraction. Our proof term calculus is called a *typed λ-calculus* because of the presence of propositions (which an be viewed as types).

Following the variable name convention, we may silently rename when convenient. A particular instance where this is helpful is substitution. Consider

$$[u/w](\lambda u.\ w\, u)$$

that is, we substitute u for w in $\lambda u.\ w\, u$. Note that u is a variable visible on the outside, but also bound by λu. By the variable name convention we have

$$[u/w](\lambda u.\ w\, u) = [u/w](\lambda u'.\ w\, u') = \lambda u'.\ u\, u'$$

which is correct. But we cannot substitute without renaming, since

$$[u/w](\lambda u.\ w\, u) \neq \lambda u.\ u\, u$$

In fact, the right hand side below is invalid, while the left-hand side makes perfect sense. We say that u is *captured* by the binder λu. If we assume a hypothesis $u{:}\top \supset A$ then

$$[u/w](\lambda u{:}\top.\ w\, u) : A$$

but

$$\lambda u{:}\top.\ u\, u$$

is not well-typed since the first occurrence of u would have to be of type $\top \supset A$ but instead has type $\top$.

So when we carry out substitution $[M/u]N$ we need to make sure that no variable in M is *captured* by a binder in N, leading to an incorrect result.

Fortunately we can always achieve that by renaming some bound variables in N if necessary. We could now write down a formal definition of substitution, based on the cases for the term we are substituting into. However, we hope that the notion is sufficiently clear that this is not necessary.

Instead we revisit the substitution principle for hypothetical judgments. It states that if we have a hypothetical proof of C *true* from A *true* and we have a proof of A *true*, we can substitute the proof of A *true* for uses of the hypothesis A *true* and obtain a (non-hypothetical) proof of A *true*. In order to state this more precisely in the presence of several hypotheses, we recall that

$$\begin{array}{c} A_1\ \mathit{true} \ldots A_n\ \mathit{true} \\ \vdots \\ C\ \mathit{true} \end{array}$$

can be written as

$$\underbrace{A_1\ \mathit{true}, \ldots, A_n\ \mathit{true}}_{\Delta} \vdash C\ \mathit{true}$$

Generally we abbreviate several hypotheses by Δ. We then have the following properties, evident from the very definition of hypothetical judgments and hypothetical proofs

Weakening: If $\Delta \vdash C\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

Substitution: If $\Delta, A\ \mathit{true}, \Delta' \vdash C\ \mathit{true}$ and $\Delta \vdash A\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

As indicated above, weakening is realized by adjoining unused hypotheses, substitutions is realized by substitution of proofs for hypotheses.

For the proof term judgment, $M : A$, we use the same notation and write

$$\begin{array}{c} u_1{:}A_1\ \ldots\ u_n{:}A_n \\ \vdots \\ N : C \end{array}$$

as

$$\underbrace{u_1{:}A_1, \ldots, u_n{:}A_n}_{\Gamma} \vdash N : C$$

We use Γ to refer to collections of hypotheses $u_i{:}A_i$. In the deduction of $N : C$, each u_i stands for an unknown proof term for A_i, simply assumed to exist. If we actually find a proof $M_i{:}A_i$ we can eliminate this assumption, again by substitution. However, this time, the substitution has to perform two operations: we have to substitute M_i for u_i (the unknown proof term variable), and the deduction of $M_i : A_i$ for uses of the hypothesis $u_i{:}A_i$. More precisely, we have the following two properties:

Weakening: If $\Gamma \vdash N : C$ then $\Gamma, \Gamma' \vdash N : C$.

Substitution: If $\Gamma, u{:}A, \Gamma' \vdash N : C$ and $\Gamma \vdash M : A$ then $\Gamma, \Gamma' \vdash [M/u]N : C$.

Now we are in a position to state and prove our second meta-theorem, that is, a theorem about the logic under consideration. The theorem is called *subject reduction* because is concerns the *subject* M of the judgment $M : A$. It states that reduction preserves the type of an object. We make the hypotheses explicit as we have done in the explanations above.

Theorem 1.1 (Subject Reduction)
If $\Gamma \vdash M : A$ *and* $M \Longrightarrow M'$ *then* $\Gamma \vdash M' : A$.

Proof: We consider each case in the definition of $M \Longrightarrow M'$ in turn and show that the property holds. This is simply an instance of *proof by cases*.

Case: $\mathbf{fst}\,\langle M_1, M_2\rangle \Longrightarrow M_1$. By assumption we also know that

$$\Gamma \vdash \mathbf{fst}\,\langle M_1, M_2\rangle : A.$$

We need to show that $\Gamma \vdash M_1 : A$.

Now we inspect all inference rules for the judgment $M : A$ and we see that there is only one way how the judgment above could have been inferred: by $\wedge E_L$ from

$$\Gamma \vdash \langle M_1, M_2\rangle : A \wedge A_2$$

for some A_2. This step is called *inversion*, since we infer the premises from the conclusion of the rule. But we have to be extremely careful to inspect all possibilities for derivations so that we do not forget any cases.

Next, we apply inversion again: the judgment above could only have been inferred by $\wedge I$ from the two premises

$$\Gamma \vdash M_1 : A$$

and

$$\Gamma \vdash M_2 : A_2$$

But the first of these is what we had to prove in this case and we are done.

Case: $\mathbf{snd}\,\langle M_1, M_2\rangle \Longrightarrow M_2$. This is symmetric to the previous case. We write it an abbreviated form.

$\Gamma \vdash \mathbf{snd}\,\langle M_1, M_2\rangle : A$	Assumption
$\Gamma \vdash \langle M_1, M_2\rangle : A_1 \wedge A$ for some A_1	By inversion
$\Gamma \vdash M_1 : A_1$ and	
$\Gamma \vdash M_2 : A$	By inversion

Here the last judgment is what we were trying to prove.

Case: There is no reduction for $\top$ since there is no elimination rule and hence no destructor.

Case: $(\lambda u{:}A_1.\, M_2)\, M_1 \Longrightarrow [M_1/u]M_2$. By assumption we also know that

$$\Gamma \vdash (\lambda u{:}A_1.\, M_2)\, M_1 : A.$$

We need to show that $\Gamma \vdash [M_1/u]M_2 : A$.

Since there is only one inference rule for function application, namely implication elimination ($\supset E$), we can apply inversion and find that

$$\Gamma \vdash (\lambda u{:}A_1.\, M_2) : A_1' \supset A$$

and

$$\Gamma \vdash M_1 : A_1'$$

for some A_1'. Now we repeat inversion on the first of these and conclude that

$$\Gamma, u{:}A_1 \vdash M_2 : A$$

and, moreover, that $A_1 = A_1'$. Hence

$$\Gamma \vdash M_1 : A_1$$

Now we can apply the substitution property to these to judgments to conclude

$$\Gamma \vdash [M_1/u]M_2 : A$$

which is what we needed to show.

Case: $(\mathbf{case}\, \mathbf{inl}^C\, M_1\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow [M_1/u]N$. By assumption we also know that

$$\Gamma \vdash (\mathbf{case}\, \mathbf{inl}^C\, M_1\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) : A$$

Again we apply inversion and obtain three judgments

$$\begin{array}{l} \Gamma \vdash \mathbf{inl}^C\, M_1 : B' \vee C' \\ \Gamma, u{:}B' \vdash N : A \\ \Gamma, w{:}C' \vdash O : A \end{array}$$

for some B' and C'.

Again by inversion on the first of these, we find

$$\Gamma \vdash M_1 : B'$$

and also $C' = C$. Hence we can apply the substitution property to get

$$\Gamma \vdash [M_1/u]N : A$$

which is what we needed to show.

Case: $(\mathbf{case}\, \mathbf{inr}^B\, M_1\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow [M_1/u]N$. This is symmetric to the previous case and left as an exercise.

Case: There is no introduction rule for $\bot$ and hence no reduction rule.

□

The important techniques introduced in the proof above are *proof by cases* and *inversion*. In a proof by cases we simply consider all possibilities for why a judgment could be evident and show the property we want to establish in each case. Inversion is very similar: from the shape of the judgment we see it could have been inferred only in one possible way, so we know the premises of this rule must also be evident. We see that these are just two slightly different forms of the same kind of reasoning.

If we look back at our early example computation, we saw that the reduction step does not always take place at the top level, but that the redex may be embedded in the term. In order to allow this, we need to introduce some additional ways to establish that $M \Longrightarrow M'$ when the actual reduction takes place *inside* M. This is accomplished by so-called *congruence rules.*

Conjunction. As usual, conjunction is the simplest.

$$\frac{M \Longrightarrow M'}{\langle M, N\rangle \Longrightarrow \langle M', N\rangle} \qquad \frac{N \Longrightarrow N'}{\langle M, N\rangle \Longrightarrow \langle M, N'\rangle}$$

$$\frac{M \Longrightarrow M'}{\mathbf{fst}\, M \Longrightarrow \mathbf{fst}\, M'} \qquad \frac{M \Longrightarrow M'}{\mathbf{snd}\, M \Longrightarrow \mathbf{snd}\, M'}$$

Note that there is one rule for each subterm for each construct in the language of proof terms, just in case the reduction might take place in that subterm.

Truth. There are no rules for truth, since $\langle\,\rangle$ has no subterms and therefore permits no reduction inside.

Implication. This is similar to conjunction.

$$\frac{M \Longrightarrow M'}{M\, N \Longrightarrow M'\, N} \qquad \frac{N \Longrightarrow N'}{M\, N \Longrightarrow M\, N'}$$

$$\frac{M \Longrightarrow M'}{(\lambda u{:}A.\ M) \Longrightarrow (\lambda u{:}A.\ M')}$$

Disjunction. This requires no new ideas, just more cases.

$$\frac{M \Longrightarrow M'}{\mathbf{inl}^B\, M \Longrightarrow \mathbf{inl}^B\, M'} \qquad \frac{N \Longrightarrow N'}{\mathbf{inr}^A\, N \Longrightarrow \mathbf{inr}^A\, N'}$$

$$\frac{M \Longrightarrow M'}{(\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow (\mathbf{case}\, M'\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O)}$$

$$\frac{N \Longrightarrow N'}{(\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow (\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N' \mid \mathbf{inr}\, w \Rightarrow O)}$$

$$\frac{O \Longrightarrow O'}{(\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow (\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O')}$$

Falsehood. Finally, there *is* a congruence rule for falsehood, since the proof term constructor has a subterm.

$$\frac{M \Longrightarrow M'}{\mathbf{abort}^C\, M \Longrightarrow \mathbf{abort}^C\, M'}$$

We now extend the theorem to the general case of reduction on subterms. A proof by cases is now no longer sufficient, since the congruence rules have premises, for which we would have to analyze cases again, and again, etc.

Instead we use a technique called *structural induction* on proofs. In structural induction we analyse each inference rule, assuming the desired property for the premises, proving that they hold for the conclusion. If that is the case for all inference rules, the conclusion of each deduction must have the property.

Theorem 1.2 (Subterm Subject Reduction)
If $\Gamma \vdash M : A$ *and* $M \Longrightarrow M'$ *then* $\Gamma \vdash M' : A$ *where* $M \Longrightarrow M'$ *refers to the congruent interpretation of reduction.*

Proof: The cases where the reduction takes place at the top level of the term M, the cases in the proof of Theorem 1.1 still apply. The new cases are all very similar, and we only show one.

Case: The derivation of $M \Longrightarrow M'$ has the form

$$\frac{M_1 \Longrightarrow M_1'}{\langle M_1, M_2 \rangle \Longrightarrow \langle M_1', M_2 \rangle}$$

We also know that $\Gamma \vdash \langle M_1, M_2 \rangle : A$. We need to show that

$$\Gamma \vdash \langle M_1', M_2 \rangle : A$$

By inversion,

$$\Gamma \vdash M_1 : A_1$$

and

$$\Gamma \vdash M_2 : A_2$$

and $A = A_1 \wedge A_2$.

Since we are proving the theorem by structural induction and we have a deduction of $\Gamma \vdash M_1 : A_1$ we can now apply the induction hypothesis to $M_1 \Longrightarrow M_1'$. This yields

$$\Gamma \vdash M_1' : A_1$$

and we can construct the deduction

$$\dfrac{\Gamma \vdash M_1' : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1', M_2 \rangle : A_1 \wedge A_2} \wedge I$$

which is what we needed to show since $A = A_1 \wedge A_2$.

Cases: All other cases are similar and left as an exercise.

□

The importance of the technique of structural induction cannot be overemphasized in this domain. We will see it time and again, so the reader should make sure the understand each step in the proof above.

1.5 Primitive Recursion

In the preceding sections we have developed an interpretation of propositions as types. This interpretation yields function types (from implication), product types (from conjunction), unit type (from truth), sum types (from disjunction) and the empty type (from falsehood). What is missing for a reasonable programming language are basic data types such as natural numbers, integers, lists, trees, etc. There are several approaches to incorporating such types into our framework. One is to add a general definition mechanism for *recursive types* or *inductive types*. We return to this option later. Another one is to specify each type in a way which is analogous to the definitions of the logical connectives via introduction and elimination rules. This is the option we pursue in this section. A third way is to use the constructs we already have to define data. This was Church's original approach culminating in the so-called *Church numerals*. We will not discuss this idea in these notes.

After spending some time to illustrate the interpretation of propositions as types, we now introduce types as a first-class notion. This is not strictly necessary, but it avoids the question what, for example, **nat** (the type of natural numbers) means as a proposition. Accordingly, we have a new judgment τ *type* meaning "τ *is a type*". To understand the meaning of a type means to understand what elements it has. We therefore need a second judgment $t \in \tau$ (read: