The following deduction provides the evidence:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \quad \dfrac{}{w : A}\,w}{u\,w : B \wedge C}\supset E}{\mathbf{fst}\,(u\,w) : B}\wedge E_L}{\lambda w.\,\mathbf{fst}\,(u\,w) : A \supset B}\supset I^w \quad \dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \quad \dfrac{}{v : A}\,v}{u\,v : B \wedge C}\supset E}{\mathbf{snd}\,(u\,v) : C}\wedge E_R}{\lambda v.\,\mathbf{snd}\,(u\,v) : A \supset C}\supset I^v}{\langle(\lambda w.\,\mathbf{fst}\,(u\,w)), (\lambda v.\,\mathbf{snd}\,(u\,v))\rangle : (A \supset B) \wedge (A \supset C)}\wedge I}{\lambda u.\,\langle(\lambda w.\,\mathbf{fst}\,(u\,w)), (\lambda v.\,\mathbf{snd}\,(u\,v))\rangle : (A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))}\supset I^u$$

Programs in constructive propositional logic are somewhat uninteresting in that they do not manipulate basic data types such as natural numbers, integers, lists, trees, etc. We introduce such data types in Section 1.5, following the same method we have used in the development of logic.

To close this section we recall the guiding principles behind the assignment of proof terms to deductions.

1. For every deduction of A *true* there is a proof term M and deduction of $M : A$.

2. For every deduction of $M : A$ there is a deduction of A *true*

3. The correspondence between proof terms M and deductions of A *true* is a bijection.

We will prove these in Section 1.4.

1.2 Reduction

In the preceding section, we have introduced the assignment of proof terms to natural deductions. If proofs are programs then we need to explain how proofs are to be executed, and which results may be returned by a computation.

We explain the operational interpretation of proofs in two steps. In the first step we introduce a judgment of *reduction* $M \Longrightarrow M'$, read "M *reduces to* M'". A computation then proceeds by a sequence of reductions $M \Longrightarrow M_1 \Longrightarrow M_2 \ldots$, according to a fixed strategy, until we reach a value which is the result of the computation. In this section we cover reduction; we return to reduction strategies in Section **??**.

As in the development of propositional logic, we discuss each of the connectives separately, taking care to make sure the explanations are independent. This means we can consider various sublanguages and we can later extend our logic or programming language without invalidating the results from this section. Furthermore, it greatly simplifies the analysis of properties of the reduction rules.

In general, we think of the proof terms corresponding to the introduction rules as the *constructors* and the proof terms corresponding to the elimination rules as the *destructors*.

Conjunction. The constructor forms a pair, while the destructors are the left and right projections. The reduction rules prescribe the actions of the projections.

$$\begin{array}{rcl} \mathbf{fst}\,\langle M, N\rangle & \Longrightarrow & M \\ \mathbf{snd}\,\langle M, N\rangle & \Longrightarrow & N \end{array}$$

Truth. The constructor just forms the unit element, $\langle\,\rangle$. Since there is no destructor, there is no reduction rule.

Implication. The constructor forms a function by λ-abstraction, while the destructor applies the function to an argument. In general, the application of a function to an argument is computed by *substitution*. As a simple example from mathematics, consider the following equivalent definitions

$$f(x) = x^2 + x - 1 \qquad f = \lambda x.\, x^2 + x - 1$$

and the computation

$$f(3) = (\lambda x.\, x^2 + x - 1)(3) = [3/x](x^2 + x - 1) = 3^2 + 3 - 1 = 11$$

In the second step, we substitute 3 for occurrences of x in $x^2 + x - 1$, the *body of the λ-expression*. We write $[3/x](x^2 + x - 1) = 3^2 + 3 - 1$.

In general, the notation for the substitution of N for occurrences of u in M is $[N/u]M$. We therefore write the reduction rule as

$$(\lambda u{:}A.\, M)\, N \quad \Longrightarrow \quad [N/u]M$$

We have to be somewhat careful so that substitution behaves correctly. In particular, no variable in N should be bound in M in order to avoid conflict. We can always achieve this by renaming bound variables—an operation which clearly does not change the meaning of a proof term.

Disjunction. The constructors inject into a sum types; the destructor distinguishes cases. We need to use substitution again.

$$\begin{array}{rcl} \mathbf{case}\,\mathbf{inl}^B\, M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow & [M/u]N \\ \mathbf{case}\,\mathbf{inr}^A\, M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow & [M/w]O \end{array}$$

Falsehood. Since there is no constructor for the empty type there is no reduction rule for falsehood.

This concludes the definition of the reduction judgment. In the next section we will prove some of its properties.

As an example we consider a simple program for the composition of two functions. It takes a pair of two functions, one from A to B and one from B to C and returns their composition which maps A directly to C.

$$\mathsf{comp} \quad : \quad ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$$

We transform the following implicit definition into our notation step-by-step:

$$\begin{array}{rcl} \mathsf{comp}\,\langle f, g\rangle\,(w) & = & g(f(w)) \\ \mathsf{comp}\,\langle f, g\rangle & = & \lambda w.\ g(f(w)) \\ \mathsf{comp}\,u & = & \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)(w)) \\ \mathsf{comp} & = & \lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) \end{array}$$

The final definition represents a correct proof term, as witnessed by the following deduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{snd}\,u : B \supset C}\wedge E_R \qquad \dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{fst}\,u : A \supset B}\wedge E_L \quad \dfrac{}{w : A}\,w}{(\mathbf{fst}\,u)\,w : B}\supset E}{(\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : C}\supset E}{\lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : A \supset C}\supset I^w}{(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w)) : ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)}\supset I^u$$

We now verify that the composition of two identity functions reduces again to the identity function. First, we verify the typing of this application.

$$(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w))\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle \quad : \quad A \supset A$$

Now we show a possible sequence of reduction steps. This is by no means uniquely determined.

$$\begin{array}{rl} & (\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w))\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle \\ \Longrightarrow & \lambda w.\ (\mathbf{snd}\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle)\,((\mathbf{fst}\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle)\,w) \\ \Longrightarrow & \lambda w.\ (\lambda y.\ y)\,((\mathbf{fst}\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle)\,w) \\ \Longrightarrow & \lambda w.\ (\lambda y.\ y)\,((\lambda x.\ x)\,w) \\ \Longrightarrow & \lambda w.\ (\lambda y.\ y)\,w \\ \Longrightarrow & \lambda w.\ w \end{array}$$

We see that we may need to apply reduction steps to subterms in order to reduce a proof term to a form in which it can no longer be reduced. We postpone a more detailed discussion of this until we discuss the operational semantics in full.

1.3 Summary of Proof Terms

Judgments.

$M : A$	M is a proof term for proposition A
$M \Longrightarrow M'$	M reduces to M'

Proof Term Assignment.

Constructors	Destructors
$\dfrac{M : A \qquad N : B}{\langle M, N\rangle : A \wedge B}\,\wedge I$	$\dfrac{M : A \wedge B}{\mathbf{fst}\, M : A}\,\wedge E_L$ $\dfrac{M : A \wedge B}{\mathbf{snd}\, M : B}\,\wedge E_R$
$\dfrac{}{\langle\rangle : \top}\,\top I$	no destructor for $\top$
$\dfrac{\begin{array}{c}\overline{u : A}\,u \\ \vdots \\ M : B\end{array}}{\lambda u{:}A.\ M : A \supset B}\,\supset I^u$	$\dfrac{M : A \supset B \qquad N : A}{M\ N : B}\,\supset E$
$\dfrac{M : A}{\mathbf{inl}^B\, M : A \vee B}\,\vee I_L$ $\dfrac{N : B}{\mathbf{inr}^A\, N : A \vee B}\,\vee I_R$	$\dfrac{M : A \vee B \qquad \begin{array}{c}\overline{u : A}\,u \\ \vdots \\ N : C\end{array} \qquad \begin{array}{c}\overline{w : B}\,w \\ \vdots \\ O : C\end{array}}{\mathbf{case}\, M\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\,\vee E^{u,w}$
no constructor for $\bot$	$\dfrac{M : \bot}{\mathbf{abort}^C\, M : C}\,\bot E$

Reductions.

$$\begin{array}{rcl}
\mathbf{fst}\,\langle M, N\rangle & \Longrightarrow & M \\
\mathbf{snd}\,\langle M, N\rangle & \Longrightarrow & N \\
\text{no reduction for } \langle\,\rangle & & \\
(\lambda u{:}A.\ M)\,N & \Longrightarrow & [N/u]M \\
\mathbf{case}\,\mathbf{inl}^B\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow & [M/u]N \\
\mathbf{case}\,\mathbf{inr}^A\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow & [M/w]O \\
\text{no reduction for } \mathbf{abort} & &
\end{array}$$

Concrete Syntax. The concrete syntax for proof terms used in the mechanical proof checker has some minor differences to the form we presented above.

u	`u`	Variable
$\langle M, N\rangle$	`(M,N)`	Pair
$\mathbf{fst}\,M$	`fst M`	First projection
$\mathbf{snd}\,M$	`snd M`	Second projection
$\langle\,\rangle$	`()`	Unit element
$\lambda u{:}A.\ M$	`fn u => M`	Abstraction
$M\,N$	`M N`	Application
$\mathbf{inl}^B\,M$	`inl M`	Left injection
$\mathbf{inr}^A\,N$	`inr N`	Right injection
$\mathbf{case}\,M$	`case M`	Case analysis
$\quad\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N$	`  of inl u => N`	
$\quad\mid \mathbf{inr}\,w \Rightarrow O$	`   | inr w => O`	
	`end`	
$\mathbf{abort}^C\,M$	`abort M`	Abort

Pairs and unit element are delimited by parentheses '`(`' and '`)`' instead of angle brackets $\langle$ and $\rangle$. The `case` constructs requires an `end` token to mark the end of the a sequence of cases.

Type annotations are generally omitted, but a whole term can explicitly be given a type. The proof checker (which here is also a type checker) infers the missing information. Occasionally, an explicit type ascription `M : A` is necessary as a hint to the type checker.

For rules of operator precedence, the reader is refered to the on-line documentation of the proof checking software available with the course material. Generally, parentheses can be used to disambiguate or override the standard rules.

As an example, we show the proof term implementing function composition.

```
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.*