Examples for the Curry-Howard Isomorphism (appendix for Lecture Slides 04)

Assaf Kfoury

March 2026

We present several, relatively simple, examples to illustrate the Curry-Howard Isomorphism. Each example involves a formal proof written in natural-deduction style, such that, in each line of the formal proof, an appropriate proof term (a term of an extended λ-calculus) is inserted.

Our version of natural deduction is in ***Lecture Slides 02***, and again in ***Lecture Slides 04*** ; our version is at a small variance from that in the book [LCHI, Chapters 2 and 4] , though equivalent to it.

For simplicity, we write proof terms without any type annotations, *i.e.*, these are terms in a simply-typable λ-calculus in Curry style.

1 A First Example

We start with a very simple example, setting the notational conventions for all the examples in the rest of this handout.

The wff $(q \to p) \to (r \to q) \to (r \to p)$ is a tautology of *propositional logic* – in fact, it is a tautology of *implicational propositional logic*, which is certified by the following natural-deduction formal proof; this one is organized as a linear sequence of wff's numbered $1,\ 2,\ 3,\ \ldots$.

1.	$(q \to p)$	assumption
2.	$(r \to q)$	assumption
3.	r	assumption
4.	q	→E 2, 3
5.	p	→E 1, 4
6.	$(r \to p)$	→I 3–5
7.	$(r \to q) \to (r \to p)$	→I 2–6
8.	$(q \to p) \to (r \to q) \to (r \to p)$	→I 1–7

Below is the same natural-deduction proof, annotated with proof terms (the highlighted parts). We view the expression $(q \to p) \to (r \to q) \to (r \to p)$ and all its subexpressions as both *propositional wff's* and *types*. We can thus read a *wff* as a *type* of the corresponding proof term:

1.	$\boldsymbol{x}$ **:** $(q \to p)$	assumption
2.	$\boldsymbol{y}$ **:** $(r \to q)$	assumption
3.	$\boldsymbol{z}$ **:** r	assumption
4.	$\boldsymbol{y}\,\boldsymbol{z}$ **:** q	→E 2, 3
5.	$\boldsymbol{x}\,(\boldsymbol{y}\,\boldsymbol{z})$ **:** p	→E 1, 4
6.	$\boldsymbol{\lambda z.\, x}\,(\boldsymbol{y}\,\boldsymbol{z})$ **:** $(r \to p)$	→I 3–5
7.	$\boldsymbol{\lambda y.\lambda z.\, x}\,(\boldsymbol{y}\,\boldsymbol{z})$ **:** $(r \to q) \to (r \to p)$	→I 2–6
8.	$\boldsymbol{\lambda x.\lambda y.\lambda z.\, x}\,(\boldsymbol{y}\,\boldsymbol{z})$ **:** $(q \to p) \to (r \to q) \to (r \to p)$	→I 1–7

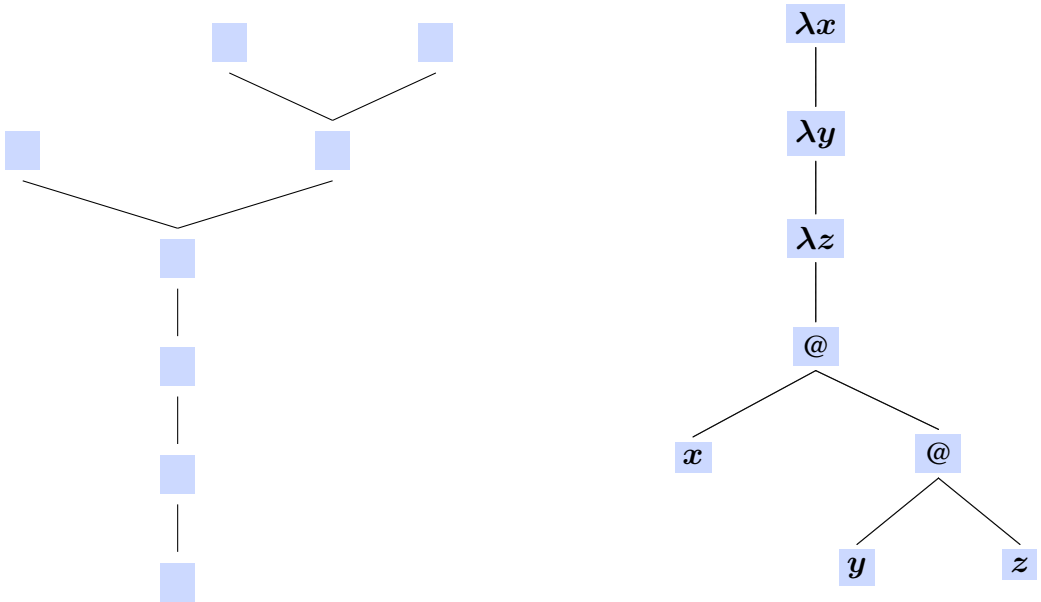
After inserting the proof terms, the line numbers along the leftmost column become redundant. Proof terms can be used instead of line numbers. For example, on line 5, the justification is written as "→E 1, 4" which means line 5 is obtained by applying the wff $(q \to p)$ on line 1 to the wff q on line 4; hence, instead of "→E 1, 4" we can write "→E $\boldsymbol{x}$, $(\boldsymbol{y}\,\boldsymbol{z})$" to mean the same thing or, even more simply, "→E" since the application $\boldsymbol{x}\,(\boldsymbol{y}\,\boldsymbol{z})$ already appears on line 5 in the middle column.

Natural deductions in the form of linear numbered sequences, such as the one above, are convenient for referencing and explaining particular sub-terms and sub-wff's in the proof, as well as for saving space on the page. But they are less convenient in exhibiting other aspects, such as the relationship between the shape of wff's and the structure of formal proofs.

To this end, we repeat the preceding natural deduction, but now organized in the form of a *proof tree* rather than as a linear numbered sequence. This will make clear a connection with the *abstract syntax tree* (AST) of the proof term $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$ and the shape of the wff whose validity it proves. Let Γ be the environment $\{x : (q \to p), y : (r \to q), z : r\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \boldsymbol{x:}\ (q \to p) \qquad \dfrac{\Gamma \vdash \boldsymbol{y:}\ (r \to q) \qquad \Gamma \vdash \boldsymbol{z:}\ r}{\Gamma \vdash \boldsymbol{y\,z:}\ r}}{\Gamma \vdash \boldsymbol{x\,(y\,z):}\ p}}{\Gamma - \{z : r\} \vdash \boldsymbol{\lambda z.\, x\,(y\,z):}\ (r \to p)}}{\Gamma - \{y : (r \to q), z : r\} \vdash \boldsymbol{\lambda y.\lambda z.\, x\,(y\,z):}\ (r \to q) \to (r \to p)}}{\varnothing \vdash \boldsymbol{\lambda x.\lambda y.\lambda z.\, x\,(y\,z):}\ (q \to p) \to (r \to q) \to (r \to p)}$$

Note how the preceding *proof tree* (schematically on the left) matches the AST (upside down!) of the proof term $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$:



The structure of the formal proof is therefore *syntax-directed*, *i.e.*, directed by the syntactic structure of the corresponding proof term, which is here $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$.

Working in reverse, if we start from the final λ-term $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$, it is *simply typable* provided we can attach a *simple type* (or, from logic's perspective, a *propositional wff*) to each node of the AST according to the typing rules of the ***simply-typed λ-calculus***.

From a programming viewpoint, the propositional wff $(q \to p) \to (r \to q) \to (r \to p)$ considered in this section is the type of *function composition*. It can be read to say:

> Given input a of type r, function g of type $(r \to q)$, and function f of type $(q \to p)$, applying g to a before applying f to $g(a)$ returns a value $f(g(a))$ of type p.

The composition of f (applied second) and g (applied first) is also written $f \circ g$, and $(f \circ g)\,a$ means the same thing as $f(g(a))$. In SML or OCaml, we can write the following code to implement *function composition*:

```
(* definition of function composition *)

let compo f g = (fun x -> f (g x) ) ;;

(* apply compo to two arguments, then apply the result to "a" *)

compo (fun x -> x ^ "c") (fun x -> x ^ "b") "a" ;;
```

In the last line above, `compo` is given two arguments: Each one is a function which takes a `string` as input and appends another `char` to it, before returning it as output. When `compo` is given these two arguments, and its output is applied to the argument `"a"`, the final result is `"abc"`. The implementation of `compo` is just a sugared version of the proof term $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$. A similar code can be written in any of the following: Haskell, Python, Scheme, and many others.

Exercise 1. A famous closed λ-term is the so-called **S** combinator which is $\mathbf{S} \triangleq \lambda x.\lambda y.\lambda z.\, x\,z\,(y\,z)$. It is 'famous' because it appears in many theoretical studies and applications, along with two other famous combinators: $\mathbf{I} \triangleq \lambda x.\, x$ and $\mathbf{K} \triangleq \lambda x.\lambda y.\, x$. [1]

In this exercise, we ask you to carry out the reverse process from the one presented in this section. Specifically, carry out the following in sequence:

(a) Write a *proof tree* (or *typing derivation*) for the combinator **S**, which happens to be simply-typable. (For a reference, see the *proof tree* for the λ-term $\lambda x.\lambda y.\lambda z.\, x\,(y\,z)$ earlier in this section, which shows that the λ-term is simply-typable.)

(b) From the *proof tree* in part (a), write the corresponding *natural-deduction proof* annotated with proof terms, in the form of a linear numbered sequence.

(c) From the (annotated) *natural-deduction proof* in part (b), write an (unannotated) *natural-deduction proof* for the propositional wff $(p \to q \to r) \to (p \to q) \to p \to r$, again in the form of a linear numbered sequence but without proof terms.

[1] **I**, **K**, and **S**, and their properties are first considered in [LCHI, Example 1.3.6, page 11].

Exercise 2. Consider the λ-term $M \triangleq \big(\lambda x.\lambda y.\lambda z.\, x\,(y\,z)\big)(\lambda v.v)(\lambda w.w)$.

(a) Write an annotated *proof tree* for M whose final proposition (*i.e.*, type) is $(p \to p)$.
Hint: Part of the requested *proof tree* reproduces the *proof tree* earlier in this section where the variables q and r are replaced by p.

(b) Apply the rule (**$\to$Cut**) defined on slide 20 of ***Lecture Slides 04*** to the *proof tree* in (a), in order to obtain a *proof tree in normal form* for the proposition / type $(p \to p)$.

2 Several Formal Proofs for the Same Propositional WFF

For the same propositional wff φ, there are in general many proofs certifying that φ is a tautology. The annotation of proofs with proof terms allows us to uniquely identify each proof.

This is illustrated with simple examples. Consider the propositional tautology $p \to (p \to p)$, or $p \to p \to p$ by the right-associativity of '$\to$'. This is already discussed in [LCHI, Sect 4.3]; we comment on it a little differently here. Two distinct natural-deduction formal proofs for it:

1.	p	assumption	
2.	p	assumption	
3.	p	repeat 1	
4.	$p \to p$	$\to$I 2–3	
5.	$p \to (p \to p)$	$\to$I 1–4	

1.	p	assumption	
2.	p	assumption	
3.	p	repeat 2	
4.	$p \to p$	$\to$I 2–3	
5.	$p \to (p \to p)$	$\to$I 1–4	

The difference between the two proofs is the justification on line 3: On the left, it says "repeat 1"; on the right, it says "repeat 2". Inserting proof terms makes it easier to notice the difference:

1.	$\boldsymbol{x}:p$	assumption	
2.	$\boldsymbol{y}:p$	assumption	
3.	$\boldsymbol{x}:p$	repeat 1	
4.	$\boldsymbol{\lambda y.x}:p \to p$	$\to$I 2–3	
5.	$\boldsymbol{\lambda x.\lambda y.x}:p \to (p \to p)$	$\to$I 1–4	

1.	$\boldsymbol{x}:p$	assumption	
2.	$\boldsymbol{y}:p$	assumption	
3.	$\boldsymbol{y}:p$	repeat 2	
4.	$\boldsymbol{\lambda y.y}:p \to p$	$\to$I 2–3	
5.	$\boldsymbol{\lambda x.\lambda y.y}:p \to (p \to p)$	$\to$I 1–4	

The final proof terms, $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$, make clear the difference between the two proofs. In the form of *proof trees*, the two preceding natural deductions look as follows:

$$\dfrac{\dfrac{\boldsymbol{x}:p,\ \boldsymbol{y}:p \vdash \boldsymbol{x}:p}{\boldsymbol{x}:p \vdash \boldsymbol{\lambda y.x}:p \to p}}{\vdash \boldsymbol{\lambda x.\lambda y.x}:p \to p \to p}$$

$$\dfrac{\dfrac{\boldsymbol{x}:p,\ \boldsymbol{y}:p \vdash \boldsymbol{y}:p}{\boldsymbol{x}:p \vdash \boldsymbol{\lambda y.y}:p \to p}}{\vdash \boldsymbol{\lambda x.\lambda y.y}:p \to p \to p}$$

Here is another simple example, also discussed in [LCHI, Sect 4.3]. Consider a natural-deduction proof of propositional atom q from three premises $\{p \to p \to q,\ p,\ p\}$ in two different ways:

1.	$p \to p \to q$	premise	
2.	p	premise	
3.	p	premise	
4.	$p \to q$	$\to$E 1 , 2	
5.	q	$\to$E 2 , 4	

1.	$p \to p \to q$	premise	
2.	p	premise	
3.	p	premise	
4.	$p \to q$	$\to$E 1 , 2	
5.	q	$\to$E 3 , 4	

Below are the two preceding proofs annotated with proof terms:

1.	$\boldsymbol{f}:\ p \to p \to q$	premise	1.	$\boldsymbol{f}:\ p \to p \to q$	premise
2.	$\boldsymbol{x}:\ p$	premise	2.	$\boldsymbol{x}:\ p$	premise
3.	$\boldsymbol{y}:\ p$	premise	3.	$\boldsymbol{y}:\ p$	premise
4.	$\boldsymbol{f}\,\boldsymbol{x}:\ p \to q$	$\to$E 1 , 2	4.	$\boldsymbol{f}\,\boldsymbol{x}:\ p \to q$	$\to$E 1 , 2
5.	$\boldsymbol{f}\,\boldsymbol{x}\,\boldsymbol{x}:\ q$	$\to$E 2 , 4	5.	$\boldsymbol{f}\,\boldsymbol{x}\,\boldsymbol{y}:\ q$	$\to$E 3 , 4

Again here, the proof terms closely describe the structure of the formal proofs.

We leave it to you to draw the *proof trees* and their relation to the corresponding *abstract syntax trees* for the examples in this section, just as we did at the end of the preceding Section 1.

Is it a '*correspondence*' or an '*isomorphism*'? Some authors use the word '*correspondence*' rather than '*isomorphism*', and sometimes (which is a little sloppy) the two words interchangeably. There is a reason for this, which is also discussed in the book [LCHI, Sect 4.3, pp 81-83] .

The preceding examples show that typed λ-terms are annotated proofs and, in this section, some proofs can be annotated in more than one way. Typed λ-terms contain more information than proofs; that is, they convey more information than simply whether or not there is a proof for a proposition or, put differently, whether or not a type is inhabited. In the limited sense of whether a proof exists for a given proposition, the *Curry-Howard Isomorphism* is more a '*correspondence*' than an '*isomorphism*'.

An *isomorphism* is more than just a *correspondence*. The former word implies a *bijective correspondence* between two domains which is also congruent with the operations that are respectively applied on the two domains. In this sense, there is indeed an *isomorphism* between typed λ-terms and proofs, as we keep track in the proofs of the way they are carried out and the way in which proofs are manipulated; in particular, a typed λ-term describes the specific way in which a proof is carried out, and not just that there is a proof.

3 Simple Examples with the Logical Connectives $\{\rightarrow, \wedge\}$

In the presence of '$\wedge$', proof terms have to be written in an extended λ-calculus with *pair* $\langle \_, \_ \rangle$ and *projections* $(\pi_1 \_)$ and $(\pi_2 \_)$. Here is a little natural-deduction proof involving '$\wedge$':

1.	$p \wedge q$	assumption
2.	p	$\wedge E_1$ 1
3.	q	$\wedge E_2$ 1
4.	$q \wedge p$	$\wedge I$ 3 , 2
5.	$(p \wedge q) \rightarrow (q \wedge p)$	$\rightarrow I$ 1–4

Inserting proof terms, the preceding natural deduction looks as follows:

1.	$\boldsymbol{x}$ **:** $p \wedge q$	assumption
2.	$\boldsymbol{\pi_1\, x}$ **:** p	$\wedge E_1$ 1
3.	$\boldsymbol{\pi_2\, x}$ **:** q	$\wedge E_2$ 1
4.	$\boldsymbol{\langle \pi_2\, x,\ \pi_1\, x \rangle}$ **:** $q \wedge p$	$\wedge I$ 3 , 2
5.	$\boldsymbol{\lambda x.\langle \pi_2\, x,\ \pi_1\, x \rangle}$ **:** $(p \wedge q) \rightarrow (q \wedge p)$	$\rightarrow I$ 1–4

The proof term $\boldsymbol{\lambda x.\langle \pi_2\, x,\ \pi_1\, x \rangle}$ reflects exactly the structure of the natural deduction that proves the wff $(p \wedge q) \rightarrow (q \wedge p)$.

The *propositional wff* $(p \wedge q) \rightarrow (q \wedge p)$, now seen as as program *type*, is of course the type of the familiar '`flip`' function in functional programming: It takes a pair of arguments and returns the pair after permuting its two arguments. If you are familiar with Haskell, you may write:

```
flip = \ x -> (snd x,fst x)
```

and if you ask for the type of '`flip`', you get back:

```
flip :: (a,b) -> (b,a)
```

Instead of π_1 and π_2, Haskell uses the operators '`fst`' and '`snd`'; and instead of the notation $(a \wedge b)$, it writes (a, b). If you are familiar with SML or OCaml, you may write the following code:

```
let flip = fun x -> (snd x,fst x) ;;
```

which is a little friendlier to read than the Haskell code. The type assigned by OCaml to '`flip`' is:

```
flip : 'a * 'b -> 'b * 'a
```

instead of '`flip :: (a,b) -> (b,a)`' and each is a little syntactic sugar for the other.

Below is another natural-deduction proof. This one establishes the validity of the propositional wff $(p \to q \wedge r) \to (p \to q) \wedge (p \to r)$:

1.	$p \to q \wedge r$	assumption
2.	p	assumption
3.	$q \wedge r$	$\to$E 1 , 2
4.	q	$\wedge$E$_1$ 3
5.	$p \to q$	$\to$I 2–4
6.	p	assumption
7.	$q \wedge r$	$\to$E 1 , 6
8.	r	$\wedge$E$_2$ 7
9.	$p \to r$	$\to$I 6–8
10.	$(p \to q) \wedge (p \to r)$	$\wedge$I 5 , 9
11.	$(p \to q \wedge r) \to (p \to q) \wedge (p \to r)$	$\to$I 1–10

Annotating each line in the preceding natural deduction with an appropriate proof term, we get:

1.	$\boldsymbol{x}$ **:** $p \to q \wedge r$	assumption
2.	$\boldsymbol{y}$ **:** p	assumption
3.	$\boldsymbol{x\,y}$ **:** $q \wedge r$	$\to$E 1 , 2
4.	$\boldsymbol{\pi_1(x\,y)}$ **:** q	$\wedge$E$_1$ 3
5.	$\boldsymbol{\lambda y.\pi_1(x\,y)}$ **:** $p \to q$	$\to$I 2–4
6.	$\boldsymbol{z}$ **:** p	assumption
7.	$\boldsymbol{x\,z}$ **:** $q \wedge r$	$\to$E 1 , 6
8.	$\boldsymbol{\pi_2(x\,z)}$ **:** r	$\wedge$E$_2$ 7
9.	$\boldsymbol{\lambda z.\pi_2(x\,z)}$ **:** $p \to r$	$\to$I 6–8
10.	$\boldsymbol{\langle \lambda y.\pi_1(x\,y), \lambda z.\pi_2(x\,z) \rangle}$ **:** $(p \to q) \wedge (p \to r)$	$\wedge$I 5 , 9
11.	$\boldsymbol{\lambda x.\langle \lambda y.\pi_1(x\,y), \lambda z.\pi_2(x\,z) \rangle}$ **:** $(p \to q \wedge r) \to (p \to q) \wedge (p \to r)$	$\to$I 1–10

Exercise 3. There are four parts $\{(a), (b), (c), (d)\}$. Divide each part into three subparts:

(i) write the natural-deduction proof for the given propositional wff,

(ii) annotate the natural deduction from subpart (i) with appropriate proof terms, and

(iii) translate the final proof term in subpart (ii) into the code of any of the following programming languages: Haskell, SML, OCaml, Scheme, Python, or any other that you know.

Do (i), (ii), and (iii), in the style of the first example in this section, namely, $(p \wedge q) \to (q \wedge p)$. Here are the four parts:

(a) $p \to q \to (p \wedge q)$

(b) $(p \wedge q) \to p \to q$

(c) $((p \wedge q) \to r) \to p \to q \to r$

(d) $(p \to q \to r) \to (p \wedge q) \to r$

The type in (c) is that of the so-called *currying function*, and the type in (d) is that of the so-called *uncurrying function*.

4 Simple Examples with the Logical Connectives $\{\rightarrow, \wedge, \vee, \bot\}$

Is $((p \vee q) \rightarrow r) \rightarrow p \rightarrow r$ a tautology of IPC? It takes a little thinking to agree that it is. The following natural-deduction proof confirms it:

1.	$(p \vee q) \rightarrow r$	assumption
2.	p	assumption
3.	$p \vee q$	$\vee\text{I}_1$ 2
4.	r	$\rightarrow$E 1 , 3
5.	$p \rightarrow r$	$\rightarrow$I 2–4
6.	$((p \vee q) \rightarrow r) \rightarrow (p \rightarrow r)$	$\rightarrow$I 1–5

Below is the same natural deduction, now annotated with proof terms:

1.	$\boldsymbol{x}$ **:** $(p \vee q) \rightarrow r$	assumption
2.	$\boldsymbol{y}$ **:** p	assumption
3.	$\mathbf{in_1}\ \boldsymbol{y}$ **:** $p \vee q$	$\vee\text{I}_1$ 2
4.	$\boldsymbol{x}\ (\mathbf{in_1}\ \boldsymbol{y})$ **:** r	$\rightarrow$E 1 , 3
5.	$\boldsymbol{\lambda y.}\ \boldsymbol{x}\ (\mathbf{in_1}\ \boldsymbol{y})$ **:** $p \rightarrow r$	$\rightarrow$I 2–4
6.	$\boldsymbol{\lambda x.\lambda y.}\ \boldsymbol{x}\ (\mathbf{in_1}\ \boldsymbol{y})$ **:** $((p \vee q) \rightarrow r) \rightarrow (p \rightarrow r)$	$\rightarrow$I 1–5

Here again, the final proof term $\boldsymbol{\lambda x.\lambda y.}\ \boldsymbol{x}\ (\mathbf{in_1}\ \boldsymbol{y})$ says exactly how the natural deduction is written in order to prove the wff $((p \vee q) \rightarrow r) \rightarrow p \rightarrow r$.

For another simple example, consider the propositional wff $(p \rightarrow \neg\neg p)$; it is an intuitionistic tautology, as certified by the following natural deduction:

1.	p	assumption
2.	$\neg p$	assumption
3.	$\bot$	$\neg$E 1,2
4.	$\neg\neg p$	$\neg$I 2–3
5.	$p \rightarrow \neg\neg p$	$\rightarrow$I 1–4

Below is the same as the preceding natural deduction, but now annotated with proof terms. Keep in mind that $\neg\varphi$ is an abbreviation for $(\varphi \rightarrow \bot)$ for any wff φ, which explains the use of rule $(\neg\text{E})$ on line 3 and rule $(\neg\text{I})$ on line 4:

1.	$\boldsymbol{x}$ **:** p	assumption
2.	$\boldsymbol{y}$ **:** $\neg p$	assumption
3.	$\boldsymbol{y}\,\boldsymbol{x}$ **:** $\bot$	$\neg$E 1,2
4.	$\boldsymbol{\lambda y.y\,x}$ **:** $\neg\neg p$	$\neg$I 2–3
5.	$\boldsymbol{\lambda x.\lambda y.y\,x}$ **:** $p \to \neg\neg p$	$\to$I 1–4

In contrast to the preceding wff, the converse implication $(\neg\neg p \to p)$ is not an intuitionistic tautology, but of course, it is a classical tautology.

Exercise 4. Write a natural deduction for the propositional wff $(\neg\neg p \to p)$ three times:

1. Use the rules of ICP in addition to a single use of $(\neg\neg\text{E})$.
2. Use the rules of ICP in addition to a single use of (PBC).
3. Use the rules of ICP in addition to a single use of (LEM).

For a more complicated example, involving all of the logical connectives in $\{\to, \wedge, \vee, \bot\}$, consider the third of ***de Morgan's Laws***. A natural-deduction proof for it was given in an earlier handout which is here reproduced verbatim, but now annotated with appropriate proof terms:

1.	$\boldsymbol{x}$ **:** $\neg(p \vee q)$	assume
2.	$\boldsymbol{y}$ **:** p	assume
3.	$\mathbf{in_1}\,\boldsymbol{y}$ **:** $p \vee q$	$\vee\text{I}_1$ 2
4.	$\boldsymbol{x}\,(\mathbf{in_1}\,\boldsymbol{y})$ **:** $\bot$	$\neg$E 1 , 3
5.	$\boldsymbol{\lambda y.}\,\boldsymbol{x}\,(\mathbf{in_1}\,\boldsymbol{y})$ **:** $\neg p$	$\neg$I 2–4
6.	$\boldsymbol{z}$ **:** q	assume
7.	$\mathbf{in_2}\,\boldsymbol{z}$ **:** $p \vee q$	$\vee\text{I}_2$ 6
8.	$\boldsymbol{x}\,(\mathbf{in_2}\,\boldsymbol{z})$ **:** $\bot$	$\neg$E 1 , 7
9.	$\boldsymbol{\lambda z.}\,\boldsymbol{x}\,(\mathbf{in_2}\,\boldsymbol{z})$ **:** $\neg q$	$\neg$I 6–8
10.	$\langle\boldsymbol{\lambda y.}\,\boldsymbol{x}\,(\mathbf{in_1}\,\boldsymbol{y}),\ \boldsymbol{\lambda z.}\,\boldsymbol{x}\,(\mathbf{in_2}\,\boldsymbol{z})\rangle$ **:** $\neg p \wedge \neg q$	$\wedge$I 5 , 9
11.	$\boldsymbol{\lambda x.}\,\langle\boldsymbol{\lambda y.}\,\boldsymbol{x}\,(\mathbf{in_1}\,\boldsymbol{y}),\ \boldsymbol{\lambda z.}\,\boldsymbol{x}\,(\mathbf{in_2}\,\boldsymbol{z})\rangle$ **:** $\neg(p \vee q) \to (\neg p \wedge \neg q)$	$\to$I 1–10

The final proof term $\boldsymbol{\lambda x.}\,\langle\boldsymbol{\lambda y.}\,\boldsymbol{x}\,(\mathbf{in_1}\,\boldsymbol{y}),\ \boldsymbol{\lambda z.}\,\boldsymbol{x}\,(\mathbf{in_2}\,\boldsymbol{z})\rangle$ mimics exactly the structure of the natural-deduction proof.

Exercise 5. Annotate the other natural-deduction proofs for the remaining ***de Morgan's Laws*** with appropriate proof terms; these were carried out in the earlier handout. Use the example of the third ***de Morgan's Law*** above as a guide.

Hint 1: You will be able to do this for the second and the fourth ***de Morgan's Laws***, but not for the first. The first does not hold intuitionistically.

Hint 2: The two natural deductions for the fourth ***de Morgan's Law*** in the earlier handout are not intuitionistically legal; they both use rule $(\neg\neg\mathrm{E})$ of classical logic for which there is no proof term. So, you first have to find another natural deduction for it which does not use $(\neg\neg\mathrm{E})$ and is intuitionistically legal.