COMP0009 Logic

Lecture notes · 2026–27

Part 1: Type Theory

Chapter 3

Natural Deduction

Alexander V. Gheorghiu

Department of Computer Science, University College London

This chapter is issued on its own from the complete lecture notes for COMP0009 Logic. Its section, theorem, exercise and equation numbers are those of the complete notes, so that the two can be read side by side; references to other chapters and parts point into the complete notes.

Contents

3. Natural Deduction 1
3.1. Derivations in NJ 1
3.2. Equivalence with the Hilbert system 10
3.3. Folding up proofs 14
3.4. Removing detours 20
3.5. The simply typed λ-calculus 23
3.6. Normalisation 31
3.7. Programs as proofs 36
3.8. Dropping the types 40
3.9. Fitch proofs *(optional)* 43
History 47
Exercises 53

3. Natural Deduction

The bus to college is unreliable, and the first lecture tomorrow is at nine. Lying awake, you think it through:

1. The bus is cancelled. — supposed
2. If the bus is cancelled, I walk. — known
3. So I walk. — from 2 and 1
4. If I walk, I miss the start of the lecture. — known
5. So I miss the start of the lecture. — from 4 and 3
6. So *if* the bus is cancelled, I miss the start of the lecture. — from 1–5; the supposition paid off

Look at line 1, and then at line 6. You haven't found out whether the bus will be cancelled, and you can't until the morning. You have found out what follows if it is, and line 6 says so without depending on line 1 at all. It is a rule, and you act on it: you set the alarm twenty minutes earlier.

That's how people reason when they don't know: suppose something, follow it to something you can check or act on, and keep the *if* when you're done with the supposition. A doctor supposes an illness and looks for the symptom it would cause; an engineer supposes a bug and looks for the line it would have left in the log. A Hilbert proof has no line for "suppose", so this most natural form of reasoning has no place in it at all.

Gerhard Gentzen made the same observation in 1934:[1]

> The formalization of logical deduction, especially as it has been developed by Frege, Russell, and Hilbert, is rather far removed from the forms of deduction used in practice in mathematical proofs. Considerable formal advantages are achieved in return. In contrast, I intended first to set up a formal system which comes as close as possible to actual reasoning. The result was a "calculus of natural deduction".

That calculus is the subject of this chapter.

3.1. **Derivations in NJ.** What is reasoning made of? Steps, each licensed by a logical rule. Line 3 of the argument about the bus takes one: from an *if*, and the thing it is conditional on, conclude the rest. We write a rule as a bar, as in Chapter 1: the premisses above it, the conclusion below, and the rule's name on the right. Write c for "the bus is cancelled", w for "I walk" and s for "I miss the start of the lecture". Line 3 is the first bar below; the second is the step from two facts to both of them at once.

$$\frac{c \to w \qquad c}{w}\ {\to}\mathrm{E} \qquad\qquad \frac{c \qquad w}{c \wedge w}\ {\wedge}\mathrm{I}$$

The general rules put φ and ψ for the particular claims.

Every connective gets two kinds of rule: one says how to prove a statement with that connective on the outside, and one says how to use such a statement once you have it:

- to prove $\varphi \wedge \psi$, prove both halves; to use it, take whichever half you need;

[1] Gentzen (1934, p. 68), in Szabo's translation. Gentzen's story is told at the end of the chapter, in the History.

Introduction (to prove it)	**Elimination** (to use it)
$\dfrac{\varphi \qquad \psi}{\varphi \wedge \psi}\,\wedge\mathrm{I}$	$\dfrac{\varphi \wedge \psi}{\varphi}\,\wedge\mathrm{E}_1 \qquad \dfrac{\varphi \wedge \psi}{\psi}\,\wedge\mathrm{E}_2$
$\dfrac{\begin{array}{c}[\varphi]\\ \vdots\\ \psi\end{array}}{\varphi \to \psi}\,\to\mathrm{I}$	$\dfrac{\varphi \to \psi \qquad \varphi}{\psi}\,\to\mathrm{E}$
$\dfrac{\varphi}{\varphi \vee \psi}\,\vee\mathrm{I}_1 \qquad \dfrac{\psi}{\varphi \vee \psi}\,\vee\mathrm{I}_2$	$\dfrac{\varphi \vee \psi \qquad \begin{array}{c}[\varphi]\\ \vdots\\ \chi\end{array} \qquad \begin{array}{c}[\psi]\\ \vdots\\ \chi\end{array}}{\chi}\,\vee\mathrm{E}$
—	$\dfrac{\bot}{\chi}\,\bot\mathrm{E}$

FIGURE 3.1. The rules of NJ.

- to prove $\varphi \vee \psi$, prove one side; to use it, handle both cases;
- to prove $\varphi \to \psi$, suppose φ, reach ψ, and discharge the supposition; to use it, apply modus ponens;
- nothing proves $\bot$; to use it, conclude anything at all.

So $\bot$ has rules of the second kind only. The first kind are called *introduction* rules and the second *elimination* rules, and the eliminations are, in a sense the end of the chapter makes more precise, consequences of the introductions: once you know how a statement is established, you know what may be got out of it.[2] Figure 3.1 draws all of them, in two columns. These rules, with no axioms at all, make up the system called NJ. The N is for *natürlich*, natural; the J comes in Part 2.

Read one at a time, the rules of Figure 3.1 say these things in symbols. ($\to$E) is modus ponens, with the implication on the left, and ($\vee$E) is proof by cases. ($\bot$E) says that from an absurdity anything follows; no rule introduces $\bot$, since there is no honest way to establish an absurdity.

The one subtlety is the square brackets. A formula in brackets above a column of dots is a supposition, and the rule *discharges* it: what was supposed above the bar is no longer assumed below it. To see what that means, take a proof you already know, that the square of an even number is even. Consider an arbitrary number n, and *suppose* that it is even. Then $n = 2k$ for some k, so $n^2 = 4k^2 = 2(2k^2)$, and n^2 is even. So *if* n is even, then n^2 is even.

[2] "The introductions represent, as it were, the 'definitions' of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions" (Gentzen, 1934, p. 80). Lorenzen named this the *inversion principle* in 1955.

Now ask two questions of that last step.[3] Is the supposition still needed? No: it has moved into the statement, which now holds of every n, odd ones included. Is it wrong to keep it? No, but "n^2 is even, provided n is even" is the weaker claim the proof had reached one line earlier, before the supposition was paid off. A bar marked (→I) is that last step, and the brackets are the *provided* crossed out.

How are the rules chained into proofs? Not as a list, the way a Hilbert proof is. To prove $\varphi \wedge \psi$ you prove φ and ψ independently, each from what it needs, and a supposition made in one case of a proof by cases must not leak into the other. A list hides which lines rest on which; a tree shows it. So a proof here is a tree of formulas, held together by the rules: the conclusion of one bar becomes a premiss of the next, and the tree is read downwards:

- the given facts and the suppositions sit at its leaves, at the top, written once each;
- the thing proved is the single formula at the root;
- the shape of the tree is the shape of the argument.

Here are lines 1 to 5 of the argument about the bus, with the supposition still in force:

$$\dfrac{w \to s \qquad \dfrac{c \to w \qquad c}{w}\,{\to}\mathrm{E}}{s}\,{\to}\mathrm{E}$$

Two bars, both (→E). Three leaves: c, which was supposed, and $c \to w$ and $w \to s$, which were known. Nothing in the tree says which of the three was supposed and which was known, and the next step needs it to.

Line 6 pays off the supposition, and it is drawn as one more bar under the tree:

$$\dfrac{\dfrac{w \to s \qquad \dfrac{c \to w \qquad [c]^1}{w}\,{\to}\mathrm{E}}{s}\,{\to}\mathrm{E}}{c \to s}\,{\to}\mathrm{I},1$$

The new bar is (→I): from a proof of s that supposes c, conclude $c \to s$ and stop supposing. The square brackets round the leaf c are a receipt. They say: *this was supposed, and the bar marked 1 has paid for it.* A leaf in brackets is *discharged*.

The two things known are *open*: they are not in brackets, no bar has paid for them, and the conclusion $c \to s$ rests on them. So the tree establishes $c \to s$ from $\{c \to w,\ w \to s\}$ and from nothing else. That's exactly the rule the alarm clock was set by. Which leaves are open and which are discharged is what a tree's conclusion rests on, and the brackets are where the tree keeps count.

Each rule joins trees already built into a bigger one. Draw each given tree as a triangle, its leaves along the top and its conclusion at the point, and the rules become pictures of trees being joined (Figure 3.2):

- (→E) and (∧I) set two trees side by side and draw one conclusion beneath them;

[3] They are van Dalen's, who asks them of Euclid's proof that the base angles of an isosceles triangle are equal (van Dalen, 2013, p. 30).

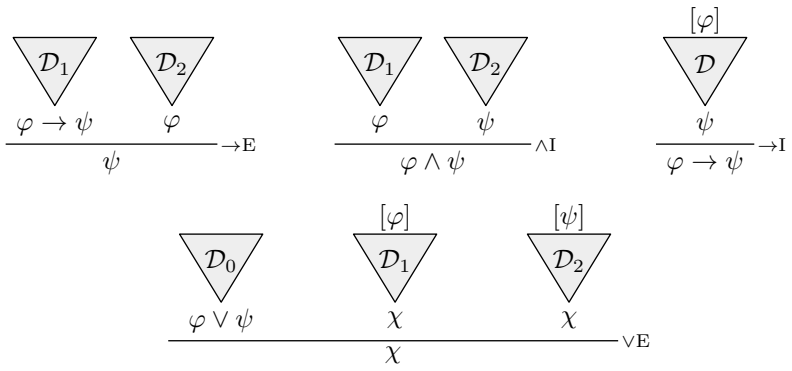


FIGURE 3.2. The rules as ways of combining trees.

- ($\to$I) takes a single tree, with or without φ among its leaves, and discharges every open φ in it; in the figure, a formula in brackets on a triangle's top edge is one the new bar discharges;
- ($\vee$E) is the largest picture: a tree for $\varphi \vee \psi$, a tree from φ to χ and a tree from ψ to χ combine into one tree for χ, discharging φ in the second and ψ in the third.

Definition 3.1 (derivations)**.** A *derivation* is a finite tree of formulas, drawn with its root at the bottom, in which some leaves are marked as *discharged*. Its *conclusion* is the formula at its root. The derivations are defined inductively:

1. (*hypothesis*) for every formula φ, the one-node tree φ is a derivation, with conclusion φ, and its leaf is not discharged;
2. (*rule*) suppose $\mathcal{D}_1, \ldots, \mathcal{D}_k$ are derivations whose conclusions are, in order, the premisses of an instance of a rule in Figure 3.1 (for ($\to$I) and ($\vee$E), the formulas under the dots). Then the tree that sets them side by side above a bar labelled with the rule, with the rule's conclusion beneath it, is a derivation with that conclusion (Figure 3.2). In it,
   - an ($\to$I) bar concluding $\varphi \to \psi$ marks as discharged every leaf φ above it that is not discharged already,
   - an ($\vee$E) bar on $\varphi \vee \psi$ marks as discharged, in the same way, the leaves φ in its second premiss and the leaves ψ in its third,
   - no other bar discharges anything;
3. nothing else is a derivation.

In words: a derivation is a tree built by the rules, and its brackets say which leaves a bar below has paid for. The definition is inductive, like that of the combinator terms, so it comes with a principle of proof. To show that every derivation has a property, it is enough to show it for the one-node trees, and to show that each rule, applied to derivations with the property, builds one with it. This is *rule induction*.

Negation needs no rules of its own. As in Chapter 2, $\neg\varphi$ abbreviates $\varphi \to \bot$, so the rules for $\to$ cover it: to prove $\neg\varphi$, suppose φ and reach $\bot$; to use it, put it beside φ and get $\bot$.

The numbers on the brackets and bars aren't part of the definition, which says exactly which leaves each bar discharges and needs no labels. They appear in practice because a drawn tree may contain several discharging bars, and the reader has to see which receipt belongs to which.[4]

Example 3.2 (Baker: a double charge)**.** Baker works on the payments service of an online shop, and at ten past ten a support ticket lands in her queue: a customer has been charged twice for one order. She doesn't know why. She has two things to go on. The runbook says that when an attempt to charge a card times out, the app retries it, and that a retry goes to the card network as a fresh charge. The order record says the customer paid at nine fifty-two. So she reasons:

1. The first attempt timed out. — supposed
2. If an attempt times out, the app retries it. — runbook
3. So the app retried. — from 2 and 1
4. If the app retries, the card is charged a second time. — runbook
5. So the card was charged twice. — from 4 and 3
6. So *if* the first attempt timed out, the card was charged twice. — from 1–5; the supposition paid off

She files line 6 in the runbook as a rule and goes to look for a timeout in the gateway's log.

With t for "the first attempt timed out", r for "the app retried" and d for "the card was charged twice", her reasoning is this tree:

$$\dfrac{\dfrac{r \to d \qquad \dfrac{t \to r \qquad [t]^1}{r}\,{\to}\mathrm{E}}{d}\,{\to}\mathrm{E}}{t \to d}\,{\to}\mathrm{I},1$$

It is a derivation. Each leaf is a one-node tree, by the first clause of Definition 3.1. The two ($\to$E) bars and the ($\to$I) bar come by the second, and the ($\to$I) bar marks the leaf t as discharged.

What, then, does a derivation establish? Its conclusion, from the leaves it has not discharged. Those leaves get a name.

Definition 3.3 (open hypotheses)**.** The *open hypotheses* of a derivation $\mathcal{D}$ are the formulas at its leaves that are not marked as discharged. We write $\mathrm{OH}(\mathcal{D})$ for the set of them.

[4]The trees are Gentzen's, root at the bottom, as he drew them and as no computer scientist draws any other tree. The square brackets round a discharged leaf are Prawitz's, from the book (1965) that made natural deduction a subject again thirty years after Gentzen; Gentzen himself marked a discharged formula only with its number.

Since derivations are defined inductively, the same set can be computed by recursion on the last rule:

$$\mathrm{OH}(\mathcal{D}) \coloneqq \begin{cases} \{\varphi\} & \text{a leaf } \varphi, \\ \mathrm{OH}(\mathcal{D}') \setminus \{\varphi\} & (\to\mathrm{I}), \text{ discharging } \varphi, \\ \mathrm{OH}(\mathcal{D}_1) \cup \mathrm{OH}(\mathcal{D}_2) & (\to\mathrm{E}),\ (\wedge\mathrm{I}), \\ \mathrm{OH}(\mathcal{D}') & (\wedge\mathrm{E}),\ (\vee\mathrm{I}),\ (\bot\mathrm{E}), \\ \mathrm{OH}(\mathcal{D}_0) \cup (\mathrm{OH}(\mathcal{D}_1) \setminus \{\varphi\}) & (\vee\mathrm{E}), \text{ discharging } \varphi,\ \psi, \\ \quad \cup\, (\mathrm{OH}(\mathcal{D}_2) \setminus \{\psi\}) & \end{cases}$$

where $\mathcal{D}'$ is the one subderivation of a one-premiss rule, $\mathcal{D}_1$, $\mathcal{D}_2$ the two of a two-premiss rule, and $\mathcal{D}_0$, $\mathcal{D}_1$, $\mathcal{D}_2$ those of $(\vee\mathrm{E})$, left to right. Only two lines take something *away*: the $(\to\mathrm{I})$ and $(\vee\mathrm{E})$ lines, for the two rules that discharge.

Definition 3.4 (consequence)**.** We write $\Gamma \vdash_{\mathrm{NJ}} \varphi$, and say that φ is *derivable from* Γ *in NJ*, or an NJ-consequence of Γ, when there is a derivation $\mathcal{D}$ whose conclusion is φ and whose open hypotheses are among Γ, $\mathrm{OH}(\mathcal{D}) \subseteq \Gamma$. Such a derivation *witnesses* $\Gamma \vdash_{\mathrm{NJ}} \varphi$.

The tree of the bus and the lecture, with its discharge, witnesses $\{c \to w,\ w \to s\} \vdash_{\mathrm{NJ}} c \to s$, and the same tree without its last bar witnesses $\{c,\ c \to w,\ w \to s\} \vdash_{\mathrm{NJ}} s$. The definition asks for a subset, not the whole of Γ: a derivation may use fewer hypotheses than Γ offers, but never more. The one-node tree p has $\mathrm{OH} = \{p\}$, so it witnesses $\{p,\ q\} \vdash_{\mathrm{NJ}} p$ as well as $\{p\} \vdash_{\mathrm{NJ}} p$; the q is on offer and never used.

Example 3.5 (Baker, cont'd: the rule she filed)**.** Compute the open hypotheses of Baker's tree in Example 3.2, from the leaves down. The three leaves give $\{r \to d\}$, $\{t \to r\}$ and $\{t\}$, by the first line of Definition 3.3. The upper $(\to\mathrm{E})$ bar takes the union of the last two, $\{t \to r,\ t\}$, and the lower one adds $r \to d$. The $(\to\mathrm{I})$ bar takes t away, leaving $\mathrm{OH} = \{t \to r,\ r \to d\}$.

So the tree witnesses $\{t \to r,\ r \to d\} \vdash_{\mathrm{NJ}} t \to d$: the rule rests on the runbook and on nothing Baker supposed. Without its last bar, the tree witnesses $\{t,\ t \to r,\ r \to d\} \vdash_{\mathrm{NJ}} d$.

Example 3.6 ($\vdash_{\mathrm{NJ}} p \to p$)**.** The smallest proof there is, $p \to p$. In the Hilbert system it takes five lines (Lemma 2.7), the first of which nobody would write unprompted:

	no hypotheses	
1	$(p \to (p \to p) \to p) \to (p \to (p \to p)) \to p \to p$	(II) $[p,\ p \to p,\ p]$
2	$p \to (p \to p) \to p$	(I) $[p,\ p \to p]$
3	$(p \to (p \to p)) \to p \to p$	MP 1, 2
4	$p \to (p \to p)$	(I) $[p,\ p]$
5	$p \to p$	MP 3, 4

In NJ it is one leaf and one bar:

$$\frac{[p]^1}{p \to p}\ {\to}\mathrm{I}, 1$$

The leaf has OH $= \{p\}$ by the first line of Definition 3.3, and the bar takes p away by its ($\to$I) line, leaving OH $= \varnothing$. So the tree witnesses $\vdash_{\mathrm{NJ}} p \to p$: it depends on nothing at all. That's what a theorem is.

Example 3.7 (commutativity of $\wedge$)**.** Swap the two halves of a conjunction. In the Hilbert system, from the hypothesis $p \wedge q$, it takes eight lines:

	$p \wedge q$	
1	$p \wedge q$	hypothesis
2	$p \wedge q \to q$	(snd) $[p,\ q]$
3	q	MP 2, 1
4	$p \wedge q \to p$	(fst) $[p,\ q]$
5	p	MP 4, 1
6	$q \to p \to q \wedge p$	(pair) $[q,\ p]$
7	$p \to q \wedge p$	MP 6, 3
8	$q \wedge p$	MP 7, 5

In NJ, three bars, one for each step of the argument:

$$\dfrac{\dfrac{p \wedge q}{q}\,\wedge\mathrm{E}_2 \qquad \dfrac{p \wedge q}{p}\,\wedge\mathrm{E}_1}{q \wedge p}\,\wedge\mathrm{I}$$

Both leaves are $p \wedge q$, so OH $= \{p \wedge q\}$, and the tree witnesses $\{p \wedge q\} \vdash_{\mathrm{NJ}} q \wedge p$. Now put one more bar under it, supposing $p \wedge q$ and discharging it:

$$\dfrac{\dfrac{\dfrac{[p \wedge q]^1}{q}\,\wedge\mathrm{E}_2 \qquad \dfrac{[p \wedge q]^1}{p}\,\wedge\mathrm{E}_1}{q \wedge p}\,\wedge\mathrm{I}}{p \wedge q \to q \wedge p}\,{\to}\mathrm{I}, 1$$

The supposition appears at two leaves, because it was used twice. The ($\to$I) line of Definition 3.3 removes both at once: OH $= \varnothing$, and the tree witnesses $\vdash_{\mathrm{NJ}} p \wedge q \to q \wedge p$. The Hilbert system gets there from the eight lines only through the Deduction Theorem, which roughly triples them.

Example 3.8 (Baker, cont'd: proof by cases)**.** At half past ten the app's log says that the gateway answered the second attempt, and the entry is truncated: the answer was a decline or an approval, and the log doesn't say which. The runbook says that an attempt the gateway has answered, either way, is written to the ledger. With n for "the gateway declined", o for "the gateway approved", and l for "the attempt is in the ledger":

$$\dfrac{n \vee o \qquad \dfrac{n \to l \qquad [n]^1}{l}\,{\to}\mathrm{E} \qquad \dfrac{o \to l \qquad [o]^2}{l}\,{\to}\mathrm{E}}{l}\,\vee\mathrm{E}, 1, 2$$

This is proof by cases, ($\vee$E), and here the bracket notation does something no other rule does: one bar discharges a *different* formula in each branch. OH $= \{n \vee o,\ n \to l,\ o \to l\}$: three things the log and the runbook said. The tree witnesses $\{n \vee o,\ n \to l,\ o \to l\} \vdash_{\mathrm{NJ}} l$, and Baker goes to the ledger without ever learning which it was.

Each supposition is in force only in its own branch, under its own number, and the disjunction in the first premiss stays open.

The ($\to$I) line of Definition 3.3 removes φ from a set, and three things follow from reading it literally.

- *Multiple discharge.* Every open leaf labelled φ goes at once. A supposition used three times sits at three leaves, and one bar discharges all three.
- *Vacuous discharge.* When φ was never there, $\mathrm{OH}(\mathcal{D}) \setminus \{\varphi\}$ is just $\mathrm{OH}(\mathcal{D})$: the bar discharges nothing, and the tree is still a derivation.
- *Which bar takes a leaf.* A leaf is discharged by the first bar below it, on its path to the root, that discharges its formula from the part of the tree the leaf stands in: an ($\to$I) bar whose antecedent it is, or an ($\vee$E) bar whose second premiss it stands in and whose left side it is, or whose third premiss and right side. A bar further down finds that leaf already discharged, though it may still take other copies of the same formula.

An ($\vee$E) bar does the same in each branch separately: φ leaves the second premiss and ψ the third, and nothing in the first premiss is discharged.[5]

A discharge cannot be forgotten. The brackets record what Definition 3.3 computes: leave them off a leaf that a bar below it takes, and the result is not a different derivation with more open hypotheses but the same derivation drawn wrongly.

Example 3.9 (which bar takes the leaf?)**.** Suppose p and discharge it twice:

$$\dfrac{\dfrac{[p]^{1}}{p \to p}\,{\to}\mathrm{I},1}{p \to p \to p}\,{\to}\mathrm{I},2$$

Both bars discharge p, and there is one leaf p to share between them. The ($\to$I) line of Definition 3.3 settles it. The bar marked 1 removes every open p from OH, so when the bar marked 2 comes, no open p is left: it discharges p vacuously, and the leaf carries the number 1. The tree witnesses $\vdash_{\mathrm{NJ}} p \to p \to p$. A bar can be vacuous even though its formula stands at a leaf above it, because an earlier bar took that leaf.

Example 3.10 (vacuous discharge)**.** A tree that looks as if it ought to be wrong:

$$\dfrac{\dfrac{\dfrac{p \wedge q}{q}\,{\wedge}\mathrm{E}_2 \qquad \dfrac{p \wedge q}{p}\,{\wedge}\mathrm{E}_1}{q \wedge p}\,{\wedge}\mathrm{I}}{p \to q \wedge p}\,{\to}\mathrm{I},1$$

The last bar concludes $p \to q \wedge p$ by ($\to$I). The rule asks only that the subtree above it conclude $q \wedge p$, which it does, and it discharges p, of which there is no leaf. So this *is* a derivation, by a vacuous discharge, and its OH is still $\{p \wedge q\}$: it witnesses $\{p \wedge q\} \vdash_{\mathrm{NJ}} p \to q \wedge p$, which is right: to prove "if p then $q \wedge p$" it is enough to have $q \wedge p$.

[5] Some books let a rule discharge a chosen subset of the open copies rather than all of them (van Dalen, 2013, p. 33, for one), or forbid discharging a formula that does not occur. Neither convention changes which consequences can be witnessed.

A tree that is *not* a derivation has a bar whose premisses do not have the shape its rule requires: a bar labelled →E under q and p, say, or ∧I with one premiss. The definition refuses those by having no rule for them, and the clerk refuses them by failing to match a shape.

Example 3.11 (discharging in one branch)**.** Two small trees. In the first, both cases suppose the same formula:

$$\dfrac{\dfrac{[p \vee p]^1 \qquad [p]^2 \qquad [p]^3}{p}\ \vee\mathrm{E},2,3}{p \vee p \to p}\ \to\mathrm{I},1$$

Each branch is a single leaf: the one-node tree p is a tree from p to p, and once the bar discharges it, it leaves nothing open. The two suppositions are both p, but they are discharged separately, one per branch, under the numbers 2 and 3, and the tree witnesses $\vdash_{\mathrm{NJ}} p \vee p \to p$.

In the second, the same letter has two fates:

$$\dfrac{p \vee q \qquad [p]^1 \qquad p}{p}\ \vee\mathrm{E},1,2$$

The second premiss is the leaf p, discharged. The third premiss has to be a tree from q to p, and the one-node tree p is one: it uses no q, so discharge 2 is vacuous, and its leaf p is not discharged, because the (∨E) line removes p only from the second premiss. So $\mathrm{OH} = \{p \vee q\} \cup (\{p\} \setminus \{p\}) \cup (\{p\} \setminus \{q\}) = \{p \vee q,\ p\}$, and the tree witnesses $\{p \vee q,\ p\} \vdash_{\mathrm{NJ}} p$. That claim has a one-leaf derivation too; the point is the bookkeeping.

So one (∨E) bar can discharge properly in one branch and vacuously in the other, and a formula discharged in one branch may stand open in the other: brackets on one leaf p say nothing about another.

Every tree can be checked in the same four steps, and none needs any understanding of what the letters mean.

1. Every bar has premisses of the shape its rule requires.
2. Every bracket carries a number, and the bar with that number is the first bar below the leaf, on its path to the root, that discharges its formula from the part of the tree the leaf stands in.
3. An (→I) bar discharges every open copy of its formula above it, and every such copy carries its number; it is vacuous only if there is none. An (∨E) bar discharges the left side in its second premiss and the right side in its third, and nothing in its first.
4. The leaves left without brackets are the open hypotheses. The tree witnesses $\Gamma \vdash_{\mathrm{NJ}} \varphi$ for every Γ that contains them.

That's the clerk's job again, done on a tree instead of a list.

One last tree, to read aloud rather than check. It proves $p \to \neg\neg p$:

$$\dfrac{\dfrac{\dfrac{[\neg p]^1 \qquad [p]^2}{\bot}\ \to\mathrm{E}}{\neg\neg p}\ \to\mathrm{I},1}{p \to \neg\neg p}\ \to\mathrm{I},2$$

Suppose p. Suppose as well $\neg p$, which is $p \to \bot$; by modus ponens the two give $\bot$. So the second supposition can't stand: discharge it, concluding

$\neg p \to \bot$, which is $\neg\neg p$. Now discharge the first: *if* p, then $\neg\neg p$. Nothing is left open, so the tree witnesses $\vdash_{\mathrm{NJ}} p \to \neg\neg p$.

The converse, $\neg\neg p \to p$, has no derivation at all, though nothing in the rules looks as if it should forbid one. Exercise 3.8 proves this with the tools of §3.6. Yet if p is read as true or false, $\neg\neg p \to p$ is true either way. There is no conflict here: $\vdash_{\mathrm{NJ}}$ says what the rules derive, not what is true, and these are different questions. How they are related is the business of Part 2.

Remark (two trees that look alike).

$$\dfrac{\begin{array}{c}[\varphi]\\ \vdots\\ \bot\end{array}}{\neg\varphi}\,{\to}\mathrm{I} \qquad \text{and} \qquad \dfrac{\begin{array}{c}[\neg\varphi]\\ \vdots\\ \bot\end{array}}{\varphi}\,?$$

The first is ($\to$I), since $\neg\varphi$ is $\varphi \to \bot$: suppose φ, reach an absurdity, conclude that φ fails. The second is proof by contradiction as you were taught it: suppose the opposite, reach an absurdity, conclude the thing itself. It looks like the first with the $\neg$ moved, and it is not a rule of this chapter. Read with ($\to$I), its upper part proves $\neg\neg\varphi$, not φ, and nothing here gets from $\neg\neg\varphi$ to φ (Exercise 3.8). Part 2 says what changes when you allow it.[6]

3.2. **Equivalence with the Hilbert system.** NJ has no axioms and nine rules; the Hilbert system HJ has nine schemas and one rule. Do they prove the same things? They do, and the way to see it is to read each axiom of HJ as a rule.

Take the schema (pair), $\varphi \to (\psi \to \varphi \wedge \psi)$: given φ, and then ψ, you may have $\varphi \wedge \psi$. An axiom is a one-line proof, so $\Gamma \vdash_{\mathrm{HJ}} \varphi \to (\psi \to \varphi \wedge \psi)$ for every Γ at all. The Deduction Theorem (Theorem 2.12) says that

$$\Gamma, \varphi \vdash_{\mathrm{HJ}} \chi \quad \text{exactly when} \quad \Gamma \vdash_{\mathrm{HJ}} \varphi \to \chi,$$

where Γ, φ is short for $\Gamma \cup \{\varphi\}$. Apply it to the axiom, with $\psi \to \varphi \wedge \psi$ for χ, and then once more:

$$\begin{aligned} & \Gamma \vdash_{\mathrm{HJ}} \varphi \to (\psi \to \varphi \wedge \psi),\\ \text{so}\quad & \Gamma, \varphi \vdash_{\mathrm{HJ}} \psi \to \varphi \wedge \psi,\\ \text{so}\quad & \Gamma, \varphi, \psi \vdash_{\mathrm{HJ}} \varphi \wedge \psi. \end{aligned}$$

The arrows have come out of the formula and gone above the line, among the hypotheses.

Now suppose both halves have been proved, $\Gamma \vdash_{\mathrm{HJ}} \varphi$ and $\Gamma \vdash_{\mathrm{HJ}} \psi$. By transitivity (Proposition 2.9) a hypothesis you can prove may be replaced by its proof. Splice each proof in for its hypothesis, and what is left is

$$\text{if } \Gamma \vdash_{\mathrm{HJ}} \varphi \text{ and } \Gamma \vdash_{\mathrm{HJ}} \psi, \text{ then } \Gamma \vdash_{\mathrm{HJ}} \varphi \wedge \psi.$$

The axiom has become a fact with two conditions: a *rule*, in the sense in which modus ponens is a rule, something you do rather than something you are told. Every schema can be read this way, and here is the whole list. It starts with the three facts that are not readings of axioms: reflexivity, modus ponens and the Deduction Theorem itself.

[6] Van Dalen calls this "a particular point which tends to confuse novices" (van Dalen, 2013, p. 33); his system has the second tree as a rule, and ours does not.

Proposition 3.12 (derived rules)**.** *For all sets of formulas* Γ *and all formulas* φ*,* ψ*,* χ*, in the Hilbert system:*

1. *if* $\varphi \in \Gamma$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi$*;* (reflexivity)
2. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi$ *and* $\Gamma \vdash_{\mathrm{HJ}} \varphi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \psi$*;* (modus ponens)
3. *if* $\Gamma, \varphi \vdash_{\mathrm{HJ}} \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi$*;* (the Deduction Theorem)
4. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi$ *and* $\Gamma \vdash_{\mathrm{HJ}} \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi \wedge \psi$*;* (pair)
5. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi \wedge \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi$*;* (fst)
6. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi \wedge \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \psi$*;* (snd)
7. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi \vee \psi$*;* (left)
8. *if* $\Gamma \vdash_{\mathrm{HJ}} \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi \vee \psi$*;* (right)
9. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi \vee \psi$*, and* $\Gamma, \varphi \vdash_{\mathrm{HJ}} \chi$*, and* $\Gamma, \psi \vdash_{\mathrm{HJ}} \chi$*, then* $\Gamma \vdash_{\mathrm{HJ}} \chi$*;* (either)
10. *if* $\Gamma \vdash_{\mathrm{HJ}} \bot$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi$*;* (absurd)
11. *if* $\Gamma \vdash_{\mathrm{HJ}} \varphi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \psi \to \varphi$*;* (I)
    if $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi \to \chi$ *and* $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi$ *then* $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \chi$*.* (II)

Proof. Each item says that from Hilbert proofs of some formulas we can build a Hilbert proof of another. We give the building instructions, item by item.

- *Item 1.* A hypothesis is a one-line proof of itself.
- *Item 2.* Write the proof of $\varphi \to \psi$, write the proof of φ below it, and add the line ψ, by modus ponens on their two last lines.
- *Item 3.* This is the Deduction Theorem (Theorem 2.12), read from left to right.
- *Items 4–8 and 10.* These are built in the same way: write the given proof or proofs, then the axiom with the same name, then one modus ponens for each given proof. Item 5, for instance, follows the proof of $\varphi \wedge \psi$ with two lines:

$\varphi \wedge \psi \to \varphi$	(fst)
φ	modus ponens.

  Item 4 uses (pair), $\varphi \to \psi \to \varphi \wedge \psi$, and needs two modus ponens, one for each of its two given proofs. Items 6, 7, 8 and 10 use (snd), (left), (right) and (absurd), with one each.
- *Item 9.* First turn the two branches into implications. By the Deduction Theorem, $\Gamma, \varphi \vdash_{\mathrm{HJ}} \chi$ gives $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \chi$, and $\Gamma, \psi \vdash_{\mathrm{HJ}} \chi$ gives $\Gamma \vdash_{\mathrm{HJ}} \psi \to \chi$. Write these two proofs and the proof of $\varphi \vee \psi$, then the axiom
  $$(\varphi \to \chi) \to (\psi \to \chi) \to \varphi \vee \psi \to \chi \qquad \text{(either)}$$
  and use modus ponens three times, once for each of the three proofs.
- *Item 11.* For the first half, write the proof of φ, then the axiom (I), $\varphi \to \psi \to \varphi$, and use modus ponens once. For the second, write the two given proofs, then the axiom (II), $(\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$, and use modus ponens twice.

□

Read for what they do to formulas, items 2 to 10 are the rules of NJ:

- item 2 is (→E), and item 3, the Deduction Theorem, is (→I);

- item 4 is ($\wedge$I), and items 5 and 6 are ($\wedge$E);
- items 7 and 8 are ($\vee$I), and item 9 is ($\vee$E);
- item 10 is ($\bot$E).

Item 11 is the exception. It says that the axioms (I) and (II), read as rules, follow from modus ponens and the Deduction Theorem.

So in the Hilbert system the rules of NJ are theorems *about* proofs, each paid for with an axiom and, for the two that discharge, with the Deduction Theorem's construction. In NJ they *are* the proofs. The proposition is half of the equivalence; the other half is to draw each axiom as a tree.

Lemma 3.13 (the axioms as trees)**.** *Every instance of each of the nine schemas of HJ has a derivation in NJ with no open hypotheses.*

Proof. Every schema is an implication, and the recipe is the same each time: suppose the antecedents, reach the consequent with the rules, and discharge the suppositions in reverse order.

For (I), suppose φ and suppose ψ. The conclusion φ is already there, since it was supposed. Discharge ψ, then φ:

$$\dfrac{\dfrac{[\varphi]^1}{\psi \to \varphi}\,{\to}\mathrm{I},2}{\varphi \to \psi \to \varphi}\,{\to}\mathrm{I},1$$

The bar marked 2 discharges ψ, and there is no leaf ψ anywhere in the tree. The definition permits this, since $\{\varphi\} \setminus \{\psi\}$ is just $\{\varphi\}$. To prove "if ψ then φ" it is enough to prove φ, and a supposition you never used can still be paid off. In the instance where ψ is φ, the bar marked 2 takes the leaf, since it is the first bar below the leaf that discharges its formula, and the bar marked 1 is the vacuous one; either way $\mathrm{OH} = \varnothing$.

For (II) there are three suppositions and three discharges, and this is the first tree in which the numbers earn their keep:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi \to \psi \to \chi]^1 \qquad [\varphi]^3}{\psi \to \chi}\,{\to}\mathrm{E} \qquad \dfrac{[\varphi \to \psi]^2 \qquad [\varphi]^3}{\psi}\,{\to}\mathrm{E}}{\chi}\,{\to}\mathrm{E}}{\varphi \to \chi}\,{\to}\mathrm{I},3}{(\varphi \to \psi) \to \varphi \to \chi}\,{\to}\mathrm{I},2}{(\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi}\,{\to}\mathrm{I},1$$

The supposition φ is used twice and discharged once, by the bar marked 3.

The other schemas follow the same recipe. Here are (pair), (snd) and (right):

$$\dfrac{\dfrac{\dfrac{[\varphi]^1 \qquad [\psi]^2}{\varphi \wedge \psi}\,\wedge\mathrm{I}}{\psi \to \varphi \wedge \psi}\,{\to}\mathrm{I},2}{\varphi \to \psi \to \varphi \wedge \psi}\,{\to}\mathrm{I},1 \qquad \dfrac{\dfrac{[\varphi \wedge \psi]^1}{\psi}\,\wedge\mathrm{E}_2}{\varphi \wedge \psi \to \psi}\,{\to}\mathrm{I},1 \qquad \dfrac{\dfrac{[\psi]^1}{\varphi \vee \psi}\,\vee\mathrm{I}_2}{\psi \to \varphi \vee \psi}\,{\to}\mathrm{I},1$$

The trees for (fst) and (left) are their mirror images, and (absurd) and (either) are Exercise 3.2.

One caution about the numbers. They assume that φ, ψ and χ name different formulas. In an instance where two of them coincide, a leaf may belong to an earlier bar than its number says, since a leaf goes to the first

bar below it that discharges its formula, but the open hypotheses are $\varnothing$ either way. □

Theorem 3.14 (equivalence with the Hilbert system)**.** *For every set of formulas* Γ *and every formula* φ *built from letters and* $\bot$ *with* $\to$, $\wedge$ *and* $\vee$,

$$\Gamma \vdash_{\mathrm{NJ}} \varphi \qquad \textit{if and only if} \qquad \Gamma \vdash_{\mathrm{HJ}} \varphi.$$

In words: the two systems prove the same things from the same hypotheses.

Proof. Each direction turns one kind of proof into the other. From right to left we go down a Hilbert proof one line at a time and build a tree for each line, as the proof of the Deduction Theorem built a new proof for each line. From left to right we take a tree apart at its last bar.

Right to left. Let $\varphi_1, \ldots, \varphi_n$ be a Hilbert proof of φ from Γ, so that φ_n is φ. For each k from 1 to n, write $P(k)$ for the claim $\Gamma \vdash_{\mathrm{NJ}} \varphi_k$. We prove $P(k)$ for every k by induction on k, and the theorem is $P(n)$. So let $k \le n$, and suppose, as the induction hypothesis, that $P(i)$ holds for every $i < k$: a tree with open hypotheses in Γ has already been built for each line above line k. (When $k = 1$ there are no lines above, and line 1 is a hypothesis or an axiom; the first two cases below use no earlier lines.) Line k is justified in one of three ways.

- *A hypothesis.* Then $\varphi_k \in \Gamma$. The one-node tree φ_k has $\mathrm{OH} = \{\varphi_k\}$, which is a subset of Γ.
- *An axiom.* By Lemma 3.13, φ_k has a derivation with no open hypotheses, and $\varnothing$ is a subset of Γ.
- *Modus ponens.* Then there are earlier lines i and j such that φ_i is the formula $\varphi_j \to \varphi_k$. By the induction hypothesis both have trees with open hypotheses in Γ. Put the two trees side by side and draw one ($\to$E) bar under them. The new tree concludes φ_k, and its open hypotheses are those of the two trees together, still in Γ. In the bar notation of Chapter 1:
$$\frac{\Gamma \vdash_{\mathrm{NJ}} \varphi_j \to \varphi_k \qquad \Gamma \vdash_{\mathrm{NJ}} \varphi_j}{\Gamma \vdash_{\mathrm{NJ}} \varphi_k}$$

Left to right. Here we count bars. For each n, write $Q(n)$ for the claim: whenever a derivation with n bars has conclusion ψ and open hypotheses Δ, then $\Delta \vdash_{\mathrm{HJ}} \psi$. We prove $Q(n)$ for every n by induction on n. This is enough: if $\Gamma \vdash_{\mathrm{NJ}} \varphi$, some derivation of φ has open hypotheses $\Delta \subseteq \Gamma$, so $\Delta \vdash_{\mathrm{HJ}} \varphi$, and monotonicity (Proposition 2.9) gives $\Gamma \vdash_{\mathrm{HJ}} \varphi$.

Base case, $n = 0$. A derivation with no bars is a single node ψ, with $\Delta = \{\psi\}$, and $\{\psi\} \vdash_{\mathrm{HJ}} \psi$ because a hypothesis is a one-line proof.

Inductive step. Let $n > 0$, and suppose $Q(m)$ for every $m < n$. Take a derivation with n bars and look at its last bar. Each derivation standing above that bar has fewer than n bars, so the induction hypothesis applies to it. There is one case for each rule.

- *($\to$E).* The derivations above the bar conclude $\varphi \to \psi$ and φ, with open hypotheses Δ_1 and Δ_2, and the whole tree has $\Delta_1 \cup \Delta_2$. By the induction hypothesis $\Delta_1 \vdash_{\mathrm{HJ}} \varphi \to \psi$ and $\Delta_2 \vdash_{\mathrm{HJ}} \varphi$. Monotonicity puts

both on the common set $\Delta_1 \cup \Delta_2$, and modus ponens finishes:

$$\frac{\Delta_1 \vdash_{\mathrm{HJ}} \varphi \to \psi \qquad \Delta_2 \vdash_{\mathrm{HJ}} \varphi}{\Delta_1 \cup \Delta_2 \vdash_{\mathrm{HJ}} \psi}$$

- *($\wedge I$), ($\wedge E$), ($\vee I$), ($\bot E$).* The same, with the item of Proposition 3.12 for the rule in place of modus ponens. For ($\wedge$I), for instance, $\Delta_1 \vdash_{\mathrm{HJ}} \varphi$ and $\Delta_2 \vdash_{\mathrm{HJ}} \psi$ give $\Delta_1 \cup \Delta_2 \vdash_{\mathrm{HJ}} \varphi \wedge \psi$ by monotonicity and item 4.
- *($\to I$).* The derivation above the bar concludes ψ with open hypotheses Δ, and the whole tree concludes $\varphi \to \psi$ with open hypotheses $\Delta \setminus \{\varphi\}$. The set Δ may or may not contain φ. Monotonicity puts φ in, whether or not it was there, and the Deduction Theorem takes it out again:

	$\Delta \vdash_{\mathrm{HJ}} \psi$	induction hypothesis,
so	$(\Delta \setminus \{\varphi\}) \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi$	monotonicity,
so	$\Delta \setminus \{\varphi\} \vdash_{\mathrm{HJ}} \varphi \to \psi$	the Deduction Theorem.

- *($\vee E$).* The derivations above the bar conclude $\varphi \vee \psi$, χ and χ, with open hypotheses Δ_0, Δ_1 and Δ_2. The whole tree concludes χ, with open hypotheses

$$\Delta = \Delta_0 \cup (\Delta_1 \setminus \{\varphi\}) \cup (\Delta_2 \setminus \{\psi\}).$$

By the induction hypothesis and monotonicity, $\Delta \vdash_{\mathrm{HJ}} \varphi \vee \psi$, $\Delta, \varphi \vdash_{\mathrm{HJ}} \chi$ and $\Delta, \psi \vdash_{\mathrm{HJ}} \chi$. Proof by cases, item 9 of Proposition 3.12, gives $\Delta \vdash_{\mathrm{HJ}} \chi$.

In every case the hypotheses on the left of $\vdash_{\mathrm{HJ}}$ are exactly the open hypotheses that Definition 3.3 computes: the brackets on a tree keep track of the turnstile. □

From here on we write $\vdash_{\mathrm{NJ}}$, and $\vdash_{\mathrm{HJ}}$ would always do as well. Nothing proved about HJ has to be proved again, and nothing in this chapter will ever cite an axiom.

3.3. **Folding up proofs.** We now want to analyse proofs: to compare them, transform them, and ask when two of them are really the same. Trees are hard to hold in the head, and harder to compute with, so we fold them up into a notation, a single expression that records how the tree was built, rule by rule. Each derivation will be *represented* by a term.

A one-node derivation of A is a hypothesis: give it a name, say x, and write $x : A$, read "x is a proof of A". The name represents the derivation.

Every other derivation is built from smaller ones by a rule, so it is enough to say how each rule builds the term for the new tree from the terms for the trees it joins. Draw a derivation as a triangle, as in Figure 3.2, with its name inside and its conclusion at the point. We write $M : A$ when the term M represents a derivation of A, and an equation with $M : A$ on the left and a triangle on the right says that M represents the derivation drawn in the triangle. If $\mathcal{D}_1$ is a derivation of $A \to B$ represented by M, and $\mathcal{D}_2$ a derivation of A represented by N, then ($\to$E) joins them into a derivation of B, and it is represented by $M\,N$, the first applied to the second:

$$M : A \to B \;=\; \begin{array}{c} \mathcal{D}_1 \\ \hline A \to B \end{array} \qquad N : A \;=\; \begin{array}{c} \mathcal{D}_2 \\ \hline A \end{array}$$

$$M\,N : B \;=\; \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \hline A \to B & A \end{array}}{B}\,{\to}\mathrm{E}$$

A proof of $A \to B$ behaves like a function: give it a proof of A, and it gives back a proof of B.

The question is how to represent discharge, which ($\to$I) needs. That rule takes a derivation $\mathcal{D}$ of B, represented by M, and discharges the hypotheses A in it. Suppose those hypotheses are all named x. The new derivation, $\mathcal{D}$ extended by a bar that discharges them, is represented by $\lambda x.\, M$:

$$M : B \;=\; \begin{array}{c} x : A \\ \mathcal{D} \\ \hline B \end{array}$$

$$\lambda x.\, M : A \to B \;=\; \dfrac{\begin{array}{c} [x : A] \\ \mathcal{D} \\ \hline B \end{array}}{A \to B}\,{\to}\mathrm{I}$$

The λ *binds* the name x. Below the bar the supposition has been paid off, and outside the λ the name x means nothing, just as a function's parameter means nothing outside its body. So $\lambda x.\, M$ is the function that takes a proof x of A and returns the proof M of B that $\mathcal{D}$ builds from it. The notation is older than this use of it, and the History at the end of the chapter says where it came from.

The term records the rule used at each bar, and the rules come in two kinds. An introduction rule builds a proof of a connective, and the term it gives is a *constructor*; an elimination rule takes such a proof apart to use it, and the term it gives is a *destructor*. So λ is the constructor for $\to$, and application is its destructor.

The other connectives follow the same pattern. ($\wedge$I) joins a derivation $\mathcal{D}_1$ of A, represented by M, and a derivation $\mathcal{D}_2$ of B, represented by N, into a derivation of $A \wedge B$, represented by the pair (M, N), so pairing is the constructor for $\wedge$; ($\wedge$E) takes a half back out, with the destructors *fst* and *snd*:

$$M : A \;=\; \begin{array}{c}\mathcal{D}_1\\ A\end{array} \qquad N : B \;=\; \begin{array}{c}\mathcal{D}_2\\ B\end{array}$$

$$(M, N) : A \wedge B \;=\; \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ A & B\end{array}}{A \wedge B}\,\wedge\mathrm{I}$$

$$M : A \wedge B \;=\; \begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}$$

$$\mathit{fst}\, M : A \;=\; \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{A}\,\wedge\mathrm{E}_1 \qquad \mathit{snd}\, M : B \;=\; \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{B}\,\wedge\mathrm{E}_2$$

We apologise for $M\,N$ and (M, N), which look alike and mean different things: juxtaposition is application, and brackets with a comma make a pair. Both notations are standard, and we keep them.

A derivation that ends in ($\vee$I) is represented by the term of its premiss, with a tag that says which side was proved; the tags *Left* and *Right* are the constructors for $\vee$:

$$M : A \;=\; \begin{array}{c}\mathcal{D}\\ A\end{array} \qquad M : B \;=\; \begin{array}{c}\mathcal{D}\\ B\end{array}$$

$$\mathit{Left}\, M : A \vee B \;=\; \dfrac{\begin{array}{c}\mathcal{D}\\ A\end{array}}{A \vee B}\,\vee\mathrm{I}_1 \qquad \mathit{Right}\, M : A \vee B \;=\; \dfrac{\begin{array}{c}\mathcal{D}\\ B\end{array}}{A \vee B}\,\vee\mathrm{I}_2$$

How is a proof by cases folded? As a pattern match:

```
case M:
    Left x   =>  P
    Right y  =>  Q
```

When M turns out to be *Left* of some value, call that value x and run P; if it turns out to be *Right* of some value, call that value y and run Q. So the **case** on *Left* 5 runs P with 5 for x, and never looks at Q.

In the triangles, the two branches are derivations $\mathcal{D}_1$ and $\mathcal{D}_2$ of the same formula C, represented by P and Q, and the suppositions they discharge are named x and y. Written with the branches between braces and a bar between them, the pattern match is the destructor for $\vee$, and it represents the whole ($\vee$E) derivation:

$$M : A \vee B \;=\; \begin{array}{c}\mathcal{D}_0\\ A \vee B\end{array} \qquad P : C \;=\; \begin{array}{c}x : A\\ \mathcal{D}_1\\ C\end{array} \qquad Q : C \;=\; \begin{array}{c}y : B\\ \mathcal{D}_2\\ C\end{array}$$

$$\textbf{case}\ M\ \textbf{of}\ \{\, \mathit{Left}\, x \Rightarrow P \mid \mathit{Right}\, y \Rightarrow Q \,\} : C \;=\; \dfrac{\begin{array}{ccc}\mathcal{D}_0 & [x : A] & [y : B]\\ \vdots & \mathcal{D}_1 & \mathcal{D}_2\\ A \vee B & C & C\end{array}}{C}\ \vee\mathrm{E}$$

Each branch's name is in force only in its own branch, as each case's supposition is.

There is no introduction rule for $\bot$, so it has no constructor: nothing builds a $\bot$. Its destructor is *absurd*, since if you ever have a $\bot$ you may conclude anything, which is what `absurd :: Void -> a` says:

$$M : \bot \;=\; \begin{array}{c}\mathcal{D}\\ \bot\end{array} \qquad \mathit{absurd}\, M : C \;=\; \dfrac{\begin{array}{c}\mathcal{D}\\ \bot\end{array}}{C}\ \bot\mathrm{E}$$

In the language of constructors and destructors, the rules of NJ are precisely the rules of Figure 3.3. Write beside each formula the name of its proof, and collect the names in force at each point, with their formulas, into a list Γ, the *context*; then $\Gamma \vdash_\lambda M : A$ says that M is a proof of A from the named hypotheses in Γ. A tree whose every node is such a judgement is NJ in *sequent style*: the hypotheses available at each point are written at that point, rather than read off the leaves. A hypothesis becomes (var), which looks its name up in the context. Each introduction rule becomes the rule for its constructor and each elimination rule the rule for its destructor: (abs) is ($\to$I) and (app) is ($\to$E), and the rules below the dotted line, for pairs, tags, **case** and *absurd*, are those of the other connectives. This is the *simply typed λ-calculus*, and §3.4 says what it is good for.

Remark (product and sum types). Many books, and most writing about programming languages, write the type of pairs as $A \times B$, a *product type*, and the type of tagged values as $A + B$, a *sum type*, with rules named ($\times$I), (+E) and so on. These notes keep the logical signs throughout, since the types are the formulas. The History says where the other notation comes from.

Theorem 3.15 (Curry–Howard)**.** *Folding is a one-to-one correspondence between the derivations of NJ, with a name written on each hypothesis as in this section, and the typings by the rules of Figure 3.3.*

*(i) (*Erasure.*) Take a typing of $\Gamma \vdash_\lambda M : A$ and delete every term, keeping the names on the hypotheses:*

- *(var) becomes a hypothesis;*
- *(app) an $\to$E bar;*
- *(abs) an $\to$I bar discharging the hypotheses with the variable's name;*

$$\frac{}{\Gamma \vdash_\lambda x : A}\ \text{var} \quad (\text{if } x : A \text{ is in } \Gamma) \qquad \frac{\Gamma \vdash_\lambda M : A \to B \qquad \Gamma \vdash_\lambda N : A}{\Gamma \vdash_\lambda M\, N : B}\ \text{app}$$

$$\frac{\Gamma,\ x : A \vdash_\lambda M : B}{\Gamma \vdash_\lambda \lambda x.\, M : A \to B}\ \text{abs}$$

. . . . . .

$$\frac{\Gamma \vdash_\lambda M : A \qquad \Gamma \vdash_\lambda N : B}{\Gamma \vdash_\lambda (M, N) : A \wedge B}\ \wedge\text{I}$$

$$\frac{\Gamma \vdash_\lambda M : A \wedge B}{\Gamma \vdash_\lambda \mathit{fst}\, M : A}\ \wedge\text{E}_1 \qquad \frac{\Gamma \vdash_\lambda M : A \wedge B}{\Gamma \vdash_\lambda \mathit{snd}\, M : B}\ \wedge\text{E}_2$$

$$\frac{\Gamma \vdash_\lambda M : A}{\Gamma \vdash_\lambda \mathit{Left}\, M : A \vee B}\ \vee\text{I}_1 \qquad \frac{\Gamma \vdash_\lambda M : B}{\Gamma \vdash_\lambda \mathit{Right}\, M : A \vee B}\ \vee\text{I}_2$$

$$\frac{\Gamma \vdash_\lambda M : A \vee B \qquad \Gamma,\ x : A \vdash_\lambda P : C \qquad \Gamma,\ y : B \vdash_\lambda Q : C}{\Gamma \vdash_\lambda \mathbf{case}\ M\ \mathbf{of}\ \{\, \mathit{Left}\, x \Rightarrow P \mid \mathit{Right}\, y \Rightarrow Q \,\} \ :\ C}\ \vee\text{E}$$

$$\frac{\Gamma \vdash_\lambda M : \bot}{\Gamma \vdash_\lambda \mathit{absurd}\, M : C}\ \bot\text{E}$$

FIGURE 3.3. The typing rules of the simply typed λ-calculus.

- *each rule for $\wedge$, $\vee$ and $\bot$ the NJ rule of the same name.*

The result is a derivation of A whose open hypotheses are among the formulas declared in Γ, a set we write $|\Gamma|$.

*(ii) (*Decoration*, the folding of this section.) Take a derivation of A, with a name on each hypothesis, and write terms from the leaves down:*

- *a variable at each leaf;*
- *an application at each $\to$E;*
- *a λ on the discharged name at each $\to$I;*
- *the matching constructor or destructor at each other bar.*

The result is a typing of $\Gamma \vdash_\lambda M : A$ for some term M, with Γ declaring the open hypotheses.

Each undoes the other. In one line: $\Gamma \vdash_\lambda M : A$ for some term M if and only if $|\Gamma| \vdash_{\mathrm{NJ}} A$.

Proof. Put Figure 3.3 beside Figure 3.1. Each typing rule is the NJ rule of the same name with a term written beside every formula: the premisses, the conclusion, and what each rule takes out of the context or out of OH are otherwise identical. Both constructions use this match, one rule at a time.

For (i) we go by induction on the number of rules in the typing. Look at its last rule. The typings above it have fewer rules, so by the induction hypothesis each erases to a derivation of its formula with open hypotheses among those declared in its context. Draw the matching NJ bar under these

derivations. Where the typing rule removes $x : A$ from the context, as (abs) does, the NJ bar discharges the hypotheses named x, so the open hypotheses of the new derivation are again among those declared.

For (ii) we go by induction on the number of bars in the derivation, in the same way: decorate the derivations above the last bar, which have fewer bars, and then write the term that the matching typing rule prescribes.

Each construction undoes the other. Decorating a derivation and then erasing the terms gives back the derivation, and erasing a typing and then decorating it gives back the typing: each rule of one system matches exactly one rule of the other, so at every step there is only one way to go.

The names are what make the match exact. (abs) discharges one variable, while ($\to$I), as Definition 3.3 computes it, discharges every open hypothesis with its formula. Once the names are rubbed out, two variables of the same type can become a single discharge: $\lambda x.\, \lambda y.\, x$ at type $A \to A \to A$ erases to the tree that supposes A once and discharges it twice, and there the leaf belongs to the inner bar and the outer bar is vacuous. The last line of the theorem is not affected, since it says only that the open hypotheses are *among* $|\Gamma|$. □

The case that matters is the λ, and it is easy to undersell: "proofs resemble programs" says far less than the theorem does. The theorem says that a derivation in NJ and a typing of a program with parameters, pairs, unions and `case` are the same tree with a column added; and that the rule which puts a parameter in scope and takes it out again is, column removed, the rule for supposing and discharging. Who saw this, and when, is told in the History, and is why the theorem carries two names.[7]

Example 3.16 (Baker, cont'd: the morning's data)**.** Baker's morning had both. Her evidence that the first attempt had taken the money was an authorisation code and a capture id held together: a pair, which is what a settled charge *is* in the payments database, one record with two fields, either of which can be taken back out. The gateway's answer to the second attempt was a value with a tag on it, `Declined` with a reason or `Approved` with a code, and to use it she had to branch on the tag. A tagged union is how a compiler makes it impossible to treat an answer as approved without looking at the tag.

[7] Howard wrote his notes in 1969 and never submitted them; they went round as photocopies for eleven years and were printed in 1980 (Howard, 1980), in the same birthday volume for Curry in which Gandy printed Turing's lost proof of normalisation. De Bruijn, in Eindhoven, found the same thing on his own at about the same time, building the checker of Exercise 2.4, and neither knew of the other. De Bruijn's name is on nothing, which puts him in Schönfinkel's company. Howard's notes credit a talk of Tait's in 1967 for the part of the correspondence that the next sections are about: that running a program is Gentzen's normalisation. "Simply typed" is Church's phrase from 1940, and the "simple" is the *simple theory of types*, Ramsey's 1926 simplification of the *ramified* theory of *Principia*, in which a function's type had recorded not only what it took and returned but at what stage it had been defined. It has nothing to do with difficulty.

Her proof by cases (Example 3.8) folds to a **case**. Name the log's answer s, the two runbook rules f and g, and the two suppositions x and y:

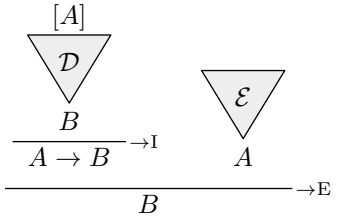
$$\dfrac{s : n \vee o \qquad \dfrac{f : n \to l \qquad [x : n]^1}{f\,x : l}\,{\to}\mathrm{E} \qquad \dfrac{g : o \to l \qquad [y : o]^2}{g\,y : l}\,{\to}\mathrm{E}}{\mathbf{case}\ s\ \mathbf{of}\ \{\, \mathit{Left}\,x \Rightarrow f\,x \mid \mathit{Right}\,y \Rightarrow g\,y \,\} : l}\,{\vee}\mathrm{E},1,2$$

Each branch ends in (→E), the rule (app) of Figure 3.3, so it folds to an application, the destructor for →. The (∨E) bar is the figure's (∨E): it folds to the **case**, the destructor for ∨, which binds x in its first branch and y in its second, as the bar discharged $[n]^1$ in one and $[o]^2$ in the other. In the notation of the figure, with Γ the context $s : n \vee o,\ f : n \to l,\ g : o \to l$, the tree shows

$$\Gamma \vdash_\lambda \mathbf{case}\ s\ \mathbf{of}\ \{\, \mathit{Left}\,x \Rightarrow f\,x \mid \mathit{Right}\,y \Rightarrow g\,y \,\} : l.$$

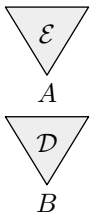
3.4. **Removing detours.** For now we restrict attention to implication, whose terms are variables, applications and λs. The other connectives, with their constructors and destructors, return in §3.5.

A proof takes a *detour* when it introduces a connective and at once eliminates it. The commonest is a lemma proved and used once. A derivation $\mathcal{D}$ of B from the supposition A is closed by (→I), giving $A \to B$, and at once applied by (→E) to a derivation $\mathcal{E}$ of A:



Nothing is wrong with this proof, but it goes the long way round. The formula $A \to B$ appears in the middle of the tree and nowhere else: it is neither the conclusion nor a hypothesis, but a lemma brought in for the occasion and dropped at once.

The detour is easy to straighten out. The derivation $\mathcal{D}$ shows how to get B from every proof of A, and $\mathcal{E}$ is one. So put a copy of $\mathcal{E}$ in place of each leaf A that the (→I) bar was to discharge, and drop both bars:



Why straighten proofs at all? A proof with no detours uses no concepts other than those contained in its final result. For → and ∧ this can be made exact: every formula in a proof without detours is a piece of its conclusion or of one of its open hypotheses. Now consider a proof of ⊥ from no hypotheses at all. If it had no detours, every formula in it would be a piece of ⊥; but ⊥ has no pieces, so there would be nothing to build the proof from. Hence if every provable formula has a proof without detours, nothing proves ⊥.

The theorem to aim for, then, is that every formula provable in NJ has a proof without detours. A direct proof for NJ was published only some thirty years after the system was invented, and the History at the end of the chapter says why. We take a simpler route, through the folding of §3.3. The first step is to say exactly what happened when the detour above was removed; §3.5 does the same for every connective.

Fold the detour up. Let M be the term of $\mathcal{D}$, with x the name of the hypotheses A that the ($\to$I) bar discharges, and let N be the term of $\mathcal{E}$. The detour is then $(\lambda x.\, M)\, N$, a function applied to an argument. The straightened proof is $\mathcal{D}$ with a copy of $\mathcal{E}$ at each leaf named x, so its term is M with N in place of each x, which we write $M[N/x]$. Removing the detour is therefore the step

$$(\lambda x.\, M)\, N \;\rightsquigarrow\; M[N/x].$$

As a program this is familiar: to call a function, run its body with the argument in place of the parameter. What remains is to define $M[N/x]$ precisely.

The first question is which occurrences of x are replaced. The straightened tree is a composition: the hypotheses A of $\mathcal{D}$ that were open in $\mathcal{D}$, and closed only by the ($\to$I) bar, have each been replaced by a proof. A leaf that some bar inside $\mathcal{D}$ discharges is not open in $\mathcal{D}$, and it stays where it is. On the term side, open hypotheses are *free variables*. A λ *binds* its parameter: the occurrences of x in $\lambda x.\, M$ are *bound*, and an occurrence of a variable that no λ binds is *free*.

Definition 3.17 (free variables)**.** The set $\mathrm{FV}(M)$ of *free variables* of a term M is defined by recursion on the shape of M:

$$\mathrm{FV}(M) \coloneqq \begin{cases} \{x\} & \text{if } M \text{ is a variable } x, \\ \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) & \text{if } M \text{ is } M_1\, M_2, \\ \mathrm{FV}(M_1) \setminus \{x\} & \text{if } M \text{ is } \lambda x.\, M_1. \end{cases}$$

A term with no free variables is *closed*.

Only the third line does anything: a λ removes its own parameter from the set. That line has the shape of the ($\to$I) line of Definition 3.3, $\mathrm{OH}(\mathcal{D}) \setminus \{A\}$, with a variable in place of a formula: a λ closes its parameter as the ($\to$I) bar closes its hypothesis.

Substitution replaces the free occurrences of x in M by N, and nothing else. So it must step around the λs in M, and that is where the care is needed.

Definition 3.18 (substitution)**.** For terms M and N and a variable x, the term $M[N/x]$, "M with N for x", is defined by recursion on the shape of M, one line per form, except that the variable and abstraction cases each split in two, according to whether the variable met is the very one being

replaced:

$$M[N/x] \coloneqq \begin{cases} N & \text{if } M \text{ is } x, \\ y & \text{if } M \text{ is a variable } y \neq x, \\ M_1[N/x]\ M_2[N/x] & \text{if } M \text{ is } M_1\, M_2, \\ \lambda x.\, M_1 & \text{if } M \text{ is } \lambda x.\, M_1, \\ \lambda y.\ M_1[N/x] & \text{if } M \text{ is } \lambda y.\, M_1,\ y \neq x,\ y \notin \mathrm{FV}(N). \end{cases}$$

The first three lines replace the variable when you meet it and otherwise pass the substitution down. The fourth stops at a λx, because every x inside it is bound and there is nothing to replace.

The side condition on the last line is the one subtlety. Pushing N under a λy when y is free in N would turn that free y into a bound one and change what N means, so the definition declines. The remedy is to rename the y of $\lambda y.\, M_1$ to a letter that occurs nowhere else, and then substitute. Rename first.[8] Chapter 4 shows the same capture on the logic side, where it has been known for a century.

Renaming is harmless. Two terms that differ only in the names of their bound variables, like $\lambda x.\ x$ and $\lambda y.\ y$, describe the same function, and we treat them as the same term. The renaming is called *α-conversion*, and terms that differ by it are *α-equivalent*. We never mark the step, since we take it only to keep variables apart.

The detour above sat at the root of its proof, but a detour can sit anywhere inside a proof, so the step must be allowed anywhere inside a term. That is the whole definition:

Definition 3.19 (β-reduction)**.** The one-step relation $M \rightsquigarrow N$, "M reduces to N", is given by the rule

$$\frac{}{(\lambda x.\, P)\, Q \ \rightsquigarrow\ P[Q/x]}\ \beta$$

together with a rule for each place a subterm can sit, which says that a step taken inside a term is a step of the term:

$$\frac{P \rightsquigarrow P'}{P\,Q \ \rightsquigarrow\ P'\,Q}\ \mathrm{app}_1 \qquad \frac{Q \rightsquigarrow Q'}{P\,Q \ \rightsquigarrow\ P\,Q'}\ \mathrm{app}_2 \qquad \frac{P \rightsquigarrow P'}{\lambda x.\, P \ \rightsquigarrow\ \lambda x.\, P'}\ \mathrm{abs}$$

The first rule is the step itself, called β, and $(\lambda x.\, P)\, Q$ is a *redex*: the term of a detour, with the term of the straightened proof on the right. To apply a function to an argument, substitute the argument for the parameter: that's what a function call does, and it's all a function call does. The other three rules say where a step may be taken, in the function, in the argument or under a λ, just as a detour may sit anywhere in a proof. This is the only

[8] Getting this wrong is a rite of passage, and the first Lisp got it wrong. A function passed as an argument looked up its free variables where it was called rather than where it was written, so a variable of the caller could capture one of the function's own. John McCarthy, its designer, later wrote: "I must confess that I regarded this difficulty as just a bug and expressed confidence that Steve Russell would soon fix it" (McCarthy, 1978). Lisps went on looking variables up that way for many years; Scheme, in 1975, was the first to do what the last line says by default.

reduction we use, so we call it β-reduction and write $\rightsquigarrow$ with no subscript.[9] Running means taking steps until there is none to take.

Example 3.20 (Baker, cont'd: the reply on the ticket)**.** Baker's morning has a detour of exactly this kind. At ten past ten she proved the rule $t \to d$ by supposing t and discharging (Example 3.5). At twenty to eleven, say, the gateway's log gives her t, a timeout at nine fifty-two, and she applies the rule to it:

$$\dfrac{\dfrac{\dfrac{r \to d \qquad \dfrac{t \to r \qquad [t]^1}{r}\,{\to}\mathrm{E}}{d}\,{\to}\mathrm{E}}{t \to d}\,{\to}\mathrm{I},1 \qquad t}{d}\,{\to}\mathrm{E}$$

An ($\to$I) met at once by ($\to$E): a detour, with the one-node tree t as $\mathcal{E}$.

The reply she writes on the ticket does not mention the rule. It puts the log's t in place of the supposition, and the two bars go:

$$\dfrac{r \to d \qquad \dfrac{t \to r \qquad t}{r}\,{\to}\mathrm{E}}{d}\,{\to}\mathrm{E}$$

Both trees witness $\{t,\ t \to r,\ r \to d\} \vdash_{\mathrm{NJ}} d$, but the second contains only facts about this charge, "no concepts other than those contained in its final result". In terms, with $f : t \to r$ and $g : r \to d$ for the two runbook rules, x for her supposition and e for the log's entry, the first tree is $(\lambda x.\, g\,(f\,x))\,e$ and the second is $g\,(f\,e)$, one β step later.

3.5. **The simply typed λ-calculus.** The detour of §3.4 was an ($\to$I) followed at once by ($\to$E). Every connective with an introduction rule has a detour of its own, and they can be described together. Call the premiss of an elimination rule that carries the connective being eliminated its *major* premiss, and the others *minor*: the major premiss is $A \to B$ in ($\to$E), $A \wedge B$ in ($\wedge\mathrm{E}_1$) and ($\wedge\mathrm{E}_2$), $A \vee B$ in ($\vee$E), and $\bot$ in ($\bot$E).

Definition 3.21 (maximal formula)**.** A *maximal formula* in a derivation is an occurrence of a formula that is the conclusion of an introduction rule and, at once, the major premiss of an elimination rule. A derivation with no maximal formula is *normal*.

A maximal formula is a detour: a formula built only to be taken apart. By the folding of §3.3 it is a constructor met at once by its destructor. The detour of §3.4 is a maximal formula $A \to B$, and the other connectives have theirs.

- A pair built by ($\wedge$I) and taken apart at once by ($\wedge\mathrm{E}_1$) keeps the proof of A and throws away the proof of B: $\mathit{fst}\,(M, N)$ becomes M, and with ($\wedge\mathrm{E}_2$), $\mathit{snd}\,(M, N)$ becomes N.
- A tag put on by ($\vee$I) and examined at once by ($\vee$E) chooses the branch it names, and the proof inside the tag goes in for that branch's supposition: a **case** on $\mathit{Left}\,R$ becomes $P[R/x]$, and on $\mathit{Right}\,R$ it becomes $Q[R/y]$.

[9] The letters are Church's: α for renaming, β for the step. There is a third, η, which says that $\lambda x.\, M\,x$ and M behave alike when x is not free in M; we have no need of it.

Since $\bot$ has no introduction rule, it is never a maximal formula.

What does a derivation with no maximal formula fold up to? Call the part of a destructor that folds its major premiss its *major part*: the M in $M\,N$, in $\mathit{fst}\,M$, $\mathit{snd}\,M$ and $\mathit{absurd}\,M$, and the M examined by $\mathbf{case}\ M\ \mathbf{of}\ \{\ldots\}$. A maximal formula folds to a destructor whose major part is a constructor. In a normal derivation that never happens, and its term has a shape of its own.

Definition 3.22 (normal and neutral terms)**.** The neutral and the normal terms are defined together. A term is *neutral* if it is one of:

- a variable;
- a destructor whose major part is neutral and whose other parts are normal, that is, $N\,M$, $\mathit{fst}\,N$, $\mathit{snd}\,N$, $\mathbf{case}\ N\ \mathbf{of}\ \{\ldots\}$ or $\mathit{absurd}\,N$, with N neutral and M and the branches normal.

A term is *normal* if it is one of:

- a constructor, that is, $\lambda x.\,M$, (M, N), $\mathit{Left}\,M$ or $\mathit{Right}\,M$, whose parts are normal;
- a neutral term.

A neutral term is a chain of destructors hanging off a variable, its *head*, and the head is free in it, since nothing in the chain binds it. On the proof side it is a hypothesis taken apart by eliminations, each working on what the one before produced. A normal term puts constructors on top of such chains: introductions near the root, building the conclusion, on eliminations that hang off hypotheses.

Lemma 3.23 (normal terms)**.** *Normal terms are the terms of normal derivations:*

(i) a term is normal exactly when no destructor in it has a constructor as its major part;

(ii) a derivation is normal exactly when the typed term it folds up to is normal.

Proof. We prove (i) in two halves and then deduce (ii).

Suppose first that M is normal. By the definition, the parts of a normal term are normal; applying this to the parts, and to their parts, every subterm of M is normal too. Now take a destructor in M. It is normal and it is not a constructor, so it is neutral, and by the definition its major part is neutral. A neutral term is not a constructor, so no destructor in M has a constructor as its major part.

For the converse, suppose that no destructor in M has a constructor as its major part. We show that M is normal, by induction on the size of M, the number of symbols in it. The parts of M are smaller, and no destructor in them has a constructor as its major part either, so by the induction hypothesis they are normal. Now look at the form of M.

- *A variable.* It is neutral, and so normal.
- *A constructor.* Its parts are normal, so it is normal.

- *A destructor.* Its major part is normal and, by the supposition, not a constructor, so it is neutral. Its other parts are normal. So the destructor is neutral, and therefore normal.

For (ii), fold the derivation up (Theorem 3.15). A constructor is the conclusion of an introduction rule, and a destructor is an elimination rule whose major premiss is its major part. So a maximal formula, the conclusion of an introduction that is at once the major premiss of an elimination, is exactly a destructor in the term whose major part is a constructor. By (i), the derivation has none exactly when its term is normal. □

So the proofs without detours that §3.4 asks for are exactly the derivations whose terms are normal, and what is wanted is a way to bring every term into that shape. A term that is not normal has a destructor meeting a constructor somewhere in it, and the list above says what each such meeting should become. Rewriting the meetings one at a time is reduction. To define it for the full calculus we first need free variables and substitution, as §3.4 had them for $\to$. Below, F stands for one of *fst*, *snd*, *Left*, *Right* and *absurd*, the forms with one part.

Definition 3.24 (free variables, all connectives)**.** On the full calculus, $\mathrm{FV}(M)$ is defined by recursion on the shape of M:

$$\mathrm{FV}(M) \coloneqq \begin{cases} \{x\} & \text{if } M \text{ is a variable } x, \\ \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) & \text{if } M \text{ is } M_1\, M_2 \text{ or } (M_1, M_2), \\ \mathrm{FV}(M_1) \setminus \{x\} & \text{if } M \text{ is } \lambda x.\, M_1, \\ \mathrm{FV}(M_1) & \text{if } M \text{ is } F\, M_1, \end{cases}$$

and, for a **case**,

$$\begin{aligned} &\mathrm{FV}(\mathbf{case}\ M_0\ \mathbf{of}\ \{\, \mathit{Left}\, y \Rightarrow M_1 \mid \mathit{Right}\, z \Rightarrow M_2 \,\}) \\ &\qquad \coloneqq \mathrm{FV}(M_0) \cup (\mathrm{FV}(M_1) \setminus \{y\}) \cup (\mathrm{FV}(M_2) \setminus \{z\}). \end{aligned}$$

The new lines pass through the parts, and the **case** line removes each branch's variable from that branch, as the λ line removes the parameter from the body.

Definition 3.25 (substitution, all connectives)**.** On the full calculus, $M[N/x]$ is defined by recursion on the shape of M:

$$M[N/x] \coloneqq \begin{cases} N & \text{if } M \text{ is } x, \\ y & \text{if } M \text{ is a variable } y \neq x, \\ M_1[N/x]\ M_2[N/x] & \text{if } M \text{ is } M_1\, M_2, \\ (M_1[N/x],\ M_2[N/x]) & \text{if } M \text{ is } (M_1, M_2), \\ F\,(M_1[N/x]) & \text{if } M \text{ is } F\, M_1, \\ \lambda x.\, M_1 & \text{if } M \text{ is } \lambda x.\, M_1, \\ \lambda y.\ M_1[N/x] & \text{if } M \text{ is } \lambda y.\, M_1,\ y \neq x,\ y \notin \mathrm{FV}(N), \end{cases}$$

and, for a **case** whose branch variables y and z are neither x nor free in N,

$$\begin{aligned} &(\mathbf{case}\ M_0\ \mathbf{of}\ \{\, \mathit{Left}\, y \Rightarrow M_1 \mid \mathit{Right}\, z \Rightarrow M_2 \,\})[N/x] \\ &\qquad \coloneqq \mathbf{case}\ M_0[N/x]\ \mathbf{of}\ \{\, \mathit{Left}\, y \Rightarrow M_1[N/x] \mid \mathit{Right}\, z \Rightarrow M_2[N/x] \,\}. \end{aligned}$$

$$\frac{}{(\lambda x.\, P)\, Q \;\rightsquigarrow\; P[Q/x]}\ \beta \qquad \frac{}{\mathit{fst}\,(P, Q) \;\rightsquigarrow\; P}\ \text{fst} \qquad \frac{}{\mathit{snd}\,(P, Q) \;\rightsquigarrow\; Q}\ \text{snd}$$

$$\frac{}{\mathbf{case}\ (\mathit{Left}\, R)\ \mathbf{of}\ \{\, \mathit{Left}\, x \Rightarrow P \mid \mathit{Right}\, y \Rightarrow Q \,\} \;\rightsquigarrow\; P[R/x]}\ \mathit{Left}$$

$$\frac{}{\mathbf{case}\ (\mathit{Right}\, R)\ \mathbf{of}\ \{\, \mathit{Left}\, x \Rightarrow P \mid \mathit{Right}\, y \Rightarrow Q \,\} \;\rightsquigarrow\; Q[R/y]}\ \mathit{Right}$$

FIGURE 3.4. The steps of the simply typed λ-calculus.

A branch variable that is x or is free in N is renamed first, as the parameter of a λ is; renaming a branch variable that is x leaves nothing to replace in that branch, just as the sixth line leaves $\lambda x.\, M_1$ alone.

Definition 3.26 (β-reduction, all connectives)**.** On the full calculus the one-step relation $\rightsquigarrow$ is given by the rules of Figure 3.4, one for each constructor met by its destructor, together with a rule for each place a subterm can sit, which says that a step taken inside a part is a step of the whole: (app_1), (app_2) and (abs) of Definition 3.19, and one such rule for each part of each other form. There is no rule for *absurd*, because $\bot$ has no constructor for it to meet. A term in which no step can be taken is in *normal form*.

Terms are written with as few brackets as the combinators of Chapter 1, by the same two conventions:

- application groups to the left, so $M\, N\, P$ is $(M\, N)\, P$;
- a λ reaches as far to the right as it can, so $\lambda x.\ M\, N$ is $\lambda x.\, (M\, N)$, not $(\lambda x.\, M)\, N$.

Remark 3.27 (One input at a time)*.* A term is a machine with a slot: put an argument in, and out comes the result, which may be another machine, waiting for the next argument. A λ makes a machine with one slot, so a function of two arguments is a function that returns a function: $\lambda x.\, \lambda y.\, M$ takes an x and becomes $\lambda y.\, M$ with that x in place, a machine waiting for a y.

There are now two things a term can be: *normal*, which by Lemma 3.23 means that it folds up a normal derivation, and *in normal form*, which means that no step can be taken in it. The left-hand side of each rule of Figure 3.4 is a destructor whose major part is a constructor, so a normal term is in normal form. For typed terms the converse holds as well, and that is what the rules were made for: a typed term that has stopped running folds up a proof with no detours.

Proposition 3.28 (normal forms)**.** *A typed term is in normal form exactly when it is normal.*

Proof. First, a normal term is in normal form. By Lemma 3.23, no destructor in it has a constructor as its major part. Every left-hand side in Figure 3.4 is such a destructor, so none occurs in the term, and no step can be taken in it.

Conversely, take a typed term in normal form, and a destructor in it. We show that its major part is not a constructor. The types decide what kind of constructor it could be, one destructor at a time.

- *An application.* Its major part has an arrow type. The only constructor with an arrow type is a λ, and a λ there would make a β redex.
- *A projection.* Its major part has a type $A \wedge B$. The only constructor of that type is a pair, which would make a redex.
- *A* **case***.* Its major part has a type $A \vee B$. The only constructors of that type are the tags, which would make a redex.
- *An absurd.* Its major part has type $\bot$, and no constructor has that type.

The term is in normal form, so it has no redex. So no destructor in it has a constructor as its major part, and by Lemma 3.23 the term is normal. □

Without the types the converse fails (Exercise 3.10).

So a typed term is in normal form exactly when the derivation it folds up is normal, and the question of §3.4, whether every proof can be straightened into one with no detours, is a question about running: does every typed term run to a normal form? That is *termination*, the subject of §3.6. Two questions about steps come first: does a step keep the type, and does the order of the steps matter?

Removing a detour restructures a proof without changing what it proves: the straightened derivation has the same conclusion, and its open hypotheses are among the old ones. On the terms, a step never changes the type. The property is called *subject reduction*, because the term is the subject of the judgement $M : A$. It held for the combinators (Theorem 1.29), and it holds here for the whole calculus.

Theorem 3.29 (subject reduction)**.** *If* $\Gamma \vdash_\lambda M : A$ *and* $M \rightsquigarrow N$*, then* $\Gamma \vdash_\lambda N : A$*.*

Proof sketch. The proof needs one fact about substitution first. Suppose $\Gamma, x : A \vdash_\lambda M : B$ and $\Gamma \vdash_\lambda N : A$. Then $\Gamma \vdash_\lambda M[N/x] : B$: putting a term of type A in place of a variable of type A keeps the type. To see why, take the typing of M and find each leaf where (var) gives $x : A$. Replace each such leaf by a copy of the typing of $N : A$. Every rule further down is still correctly applied, because a typing rule looks only at the types of its premisses, and those have not changed. The new tree types M with N in place of x, at the same type B. (On the proof side this is transitivity: a proof of A spliced in for a hypothesis A.)

With that fact in hand, we check each kind of step.

- *A β step.* The term $(\lambda x.\, M)\, N : B$ was typed by (app) from $\lambda x.\, M : A \to B$ and $N : A$, and the first of these by (abs) from $\Gamma, x : A \vdash_\lambda M : B$. The fact gives $M[N/x] : B$.
- *A projection.* The term $\mathit{fst}\,(M, N) : A$ was typed from $(M, N) : A \wedge B$, which was typed from $M : A$. So the result M has type A. The same holds for *snd*.
- *A* **case** *on a tag.* The term **case** $(\mathit{Left}\, M)$ **of** $\{\ldots\} : C$ was typed from $\mathit{Left}\, M : A \vee B$, so $M : A$, and from $\Gamma, x : A \vdash_\lambda P : C$ for its first branch.

The fact gives $P[M/x] : C$. A **case** on $\mathit{Right}\, M$ is the same with the second branch.

- *A step inside a larger term.* The subterm that steps keeps its type, by the cases above, so the lines of the typing that type it can be replaced by lines of the same type, and the rest of the typing is unchanged.

□

Example 3.30 (running typed terms)**.** Take $a : A$. Each line below names the rule of Figure 3.4 used, and the type is the same at every line, as subject reduction says.

(a) A pair made and taken apart, after a function call:

$$\begin{aligned} \mathit{fst}\,((\lambda x.\ (x,x))\,a) &\rightsquigarrow \mathit{fst}\,(a,a) && \beta,\ \text{inside } \mathit{fst} \\ &\rightsquigarrow a && \text{fst} \end{aligned}$$

all of type A. On the proof side there are two maximal formulas: the $A \to A \wedge A$ proved by discharging and used at once, and the $A \wedge A$ built by ($\wedge$I) and taken apart at once. Each step removes one, and what is left is the one-node derivation a.

(b) A function applied twice, with the identity as the function:

$$\begin{aligned} &(\lambda f.\, \lambda x.\ f\,(f\,x))\,(\lambda y.\ y)\,a \\ &\rightsquigarrow (\lambda x.\ (\lambda y.\ y)\,((\lambda y.\ y)\,x))\,a && \beta,\ \text{in the function} \\ &\rightsquigarrow (\lambda y.\ y)\,((\lambda y.\ y)\,a) && \beta \\ &\rightsquigarrow (\lambda y.\ y)\,a \rightsquigarrow a && \beta,\ \beta \end{aligned}$$

all of type A, with $\lambda f.\, \lambda x.\ f\,(f\,x) : (A \to A) \to A \to A$.

(c) A tag examined at once:

$$\mathbf{case}\ (\mathit{Left}\,a)\ \mathbf{of}\ \{\, \mathit{Left}\,x \Rightarrow (x,x) \mid \mathit{Right}\,y \Rightarrow (y,y) \,\} \rightsquigarrow (a,a) \qquad \mathit{Left}$$

both of type $A \wedge A$, with $\mathit{Left}\,a : A \vee A$.

Remark 3.31 (the order of the steps). A term may have several redexes, and then the order of the steps is a choice. Take $(\lambda x.\ f\,x\,x)\,((\lambda y.\ y)\,z)$, with $f : A \to A \to B$ and $z : A$. It has two: the whole term, and the argument $(\lambda y.\ y)\,z$ inside it. Take the whole term first, and the argument is copied before it is run, so it has to be run twice:

$$\begin{aligned} (\lambda x.\ f\,x\,x)\,((\lambda y.\ y)\,z) &\rightsquigarrow f\,((\lambda y.\ y)\,z)\,((\lambda y.\ y)\,z) \\ &\rightsquigarrow f\,z\,((\lambda y.\ y)\,z) \rightsquigarrow f\,z\,z. \end{aligned}$$

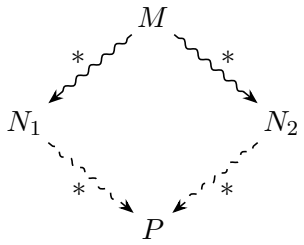
Take the argument first, and it is run once and then copied:

$$(\lambda x.\ f\,x\,x)\,((\lambda y.\ y)\,z) \rightsquigarrow (\lambda x.\ f\,x\,x)\,z \rightsquigarrow f\,z\,z.$$

Three steps one way, two the other, and the same answer. It is the choice every language makes, whether an argument is evaluated before the call or only where it is used.

Could two orders ever give different answers? For the combinators the Church–Rosser theorem said no, and we took it on loan (Theorem 1.6). Here we prove it. Write $M \rightsquigarrow^* N$ when M runs to N in finitely many steps,

perhaps none. Church–Rosser says that two runs from one term can always be brought back together:



Theorem 3.32 (Church–Rosser)**.** *If $M \rightsquigarrow^* N_1$ and $M \rightsquigarrow^* N_2$, then there is a term P such that $N_1 \rightsquigarrow^* P$ and $N_2 \rightsquigarrow^* P$.*

Proof. The proof has three stages. First we define a larger kind of step, which may contract several redexes at once. Second we show that two of these larger steps from one term can always be brought back together by one more large step on each side, closing a diamond. Third we fit such diamonds together, like tiles, to close every pair of runs.

Stage one: parallel steps. Say that M *reduces in parallel* to N, written $M \Rrightarrow N$, if N comes from M by contracting, all at once, some of the redexes already present in M: all of them, some of them or none. In $(\lambda x.\ f\,x\,x)\,((\lambda y.\ y)\,z)$, for instance, a parallel step may contract the inner redex, giving $(\lambda x.\ f\,x\,x)\,z$; or the outer one, giving $f\,((\lambda y.\ y)\,z)\,((\lambda y.\ y)\,z)$; or both, giving $f\,z\,z$; or neither. For $\rightarrow$ the rules are

$$\frac{}{x \Rrightarrow x} \qquad \frac{M \Rrightarrow M'}{\lambda x.\,M \Rrightarrow \lambda x.\,M'}$$

$$\frac{M \Rrightarrow M' \qquad N \Rrightarrow N'}{M\,N \Rrightarrow M'\,N'} \qquad \frac{M \Rrightarrow M' \qquad N \Rrightarrow N'}{(\lambda x.\,M)\,N \Rrightarrow M'[N'/x]}$$

The third rule leaves an application as it is and works inside its two parts. The fourth contracts a redex, after working inside its two parts. The other forms get one rule like the third each, passing $\Rrightarrow$ through their parts, and one like the fourth for each rule of Figure 3.4: $\mathit{fst}\,(M, N) \Rrightarrow M'$ when $M \Rrightarrow M'$, and so on. Three facts about parallel steps are needed.

(a) *Parallel steps and runs.* Every term reduces in parallel to itself, by contracting nothing. A single step is a parallel step that contracts one redex. And a parallel step is a run: if $M \Rrightarrow N$ then $M \rightsquigarrow^* N$. For the fourth rule, for instance, run M to M' and N to N', which is possible by induction, and then take one β step:

$$(\lambda x.\,M)\,N \ \rightsquigarrow^*\ (\lambda x.\,M')\,N' \ \rightsquigarrow\ M'[N'/x].$$

So a run is the same as a sequence of parallel steps.

(b) *Parallel steps and substitution.* If $M \Rrightarrow M'$ and $N \Rrightarrow N'$, then $M[N/x] \Rrightarrow M'[N'/x]$. In words, parallel steps taken inside a term and inside what is put into it can equally be taken after the substitution. This is proved by induction on the rules used for $M \Rrightarrow M'$; in the case of the fourth rule it also needs the fact that two substitutions can be done in either order, $P[Q/y][N/x]$ being $P[N/x][Q[N/x]/y]$ when y is neither x nor free in N.

(c) *Doing everything at once.* Let the *complete development* M^* of M be the result of contracting every redex of M at once:

$$M^* := \begin{cases} x & \text{if } M \text{ is } x, \\ \lambda x.\, M_1^* & \text{if } M \text{ is } \lambda x.\, M_1, \\ M_1^*[M_2^*/x] & \text{if } M \text{ is } (\lambda x.\, M_1)\, M_2, \\ M_1^*\, M_2^* & \text{if } M \text{ is } M_1\, M_2 \text{ and } M_1 \text{ is not a } \lambda, \end{cases}$$

and likewise for the other forms. The key fact is the *triangle*: if $M \Rrightarrow N$, then $N \Rrightarrow M^*$. Whichever redexes the parallel step to N contracted, the ones it left can still be contracted, and contracting them reaches M^*.

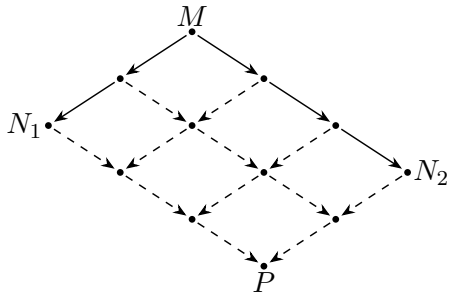
The triangle is proved by induction on the size of M. If M is a variable, N and M^* are that variable too. If M is a λ, or an application whose function is not a λ, the parallel step and the complete development both work inside the parts, and the induction hypothesis for the parts gives the result. The case that carries the idea is a redex $M = (\lambda x.\, M_1)\, M_2$. The parallel step worked inside the parts, $M_1 \Rrightarrow N_1$ and $M_2 \Rrightarrow N_2$, and by the induction hypothesis $N_1 \Rrightarrow M_1^*$ and $N_2 \Rrightarrow M_2^*$. Then there are two possibilities.

- *The step contracted the redex.* Then N is $N_1[N_2/x]$, and fact (b) gives $N \Rrightarrow M_1^*[M_2^*/x]$, which is M^*.
- *The step left the redex.* Then N is $(\lambda x.\, N_1)\, N_2$, still a redex, and the fourth rule contracts it now: $N \Rrightarrow M_1^*[M_2^*/x]$, which is M^*.

The other forms are handled in the same way.

Stage two: the diamond. Suppose $M \Rrightarrow N_1$ and $M \Rrightarrow N_2$. By the triangle, $N_1 \Rrightarrow M^*$ and $N_2 \Rrightarrow M^*$. So two parallel steps from one term are brought back together by one parallel step on each side, and the meeting point is M^*.

Stage three: tiling. Now let $M \rightsquigarrow^* N_1$ and $M \rightsquigarrow^* N_2$. By fact (a) these runs are sequences of parallel steps, say k steps to N_1 and l steps to N_2. Draw them as two edges of a grid, and fill the grid with diamonds, one square at a time, starting at M:



Each square's two upper edges are already drawn when we reach it, and they are two parallel steps from the square's top corner; stage two supplies the two lower edges. After $k \times l$ squares the grid is full. Its bottom corner P has $N_1 \Rrightarrow \cdots \Rrightarrow P$ along one side and $N_2 \Rrightarrow \cdots \Rrightarrow P$ along the other, and by fact (a) again $N_1 \rightsquigarrow^* P$ and $N_2 \rightsquigarrow^* P$. □

So a term has at most one normal form: the order of the steps can change whether it is reached, and what it costs, but not what it is.[10]

3.6. **Normalisation.** One theorem is left, and it is the one we were after. A typed term in normal form is the term of a proof without detours, and by Church–Rosser the normal form a term reaches does not depend on the order of the steps. What we do not yet know is whether every typed term reaches one. For the combinators it did: every typed program stopped. It does here too. Normalisation holds for the whole logic, but disjunction brings complications of its own, and a remark after the proof shows where, so we prove it for terms without $\vee$ only.[11]

Theorem 3.33 (normalisation)**.** *Every typed term runs to a normal form.*

Proof for terms without $\vee$*.* The idea is to attach to each term a measure that a well-chosen step always makes smaller. A measure that keeps getting smaller cannot do so for ever, so the run must stop.[12]

The measure. Give each type a *degree*, the height of its tree:

- a letter and $\bot$ have degree 1;
- $A \to B$ and $A \wedge B$ have degree one more than the larger of the degrees of A and B.

So $p \to q$ has degree 2, and $(p \to q) \to p$ has degree 3. The *degree of a redex* is the degree of the type of the constructor in it:

- the type $A \to B$ of $\lambda x.\, M$ in $(\lambda x.\, M)\, N$;
- the type $A \wedge B$ of (M, N) in $\mathit{fst}\,(M, N)$ or $\mathit{snd}\,(M, N)$.

By subject reduction a typed term stays typed as it runs, so these degrees stay defined. The measure of a term that is not in normal form is the pair (d, n), where d is the largest degree of a redex in it and n is the number of redexes of that degree.

The key observation. A step on a redex of degree d creates new redexes only of degree less than d. A new redex appears only where a constructor has come to stand under a destructor, and that can happen in two ways.

(a) In a β step $(\lambda x.\, M)\, N \rightsquigarrow M[N/x]$, an occurrence of x in M that had a destructor applied to it ($x\,P$, $\mathit{fst}\,x$ or $\mathit{snd}\,x$) now has N in its place. If N is a λ or a pair, that is a new redex. Its constructor N has the type A of x, and the degree of A is less than that of $A \to B$.

[10] The theorem, for this calculus, is from 1936 (Church and Rosser, 1936). The proof by parallel steps is Tait's and Martin-Löf's, from about 1971; the complete development M^* is Takahashi's (Takahashi, 1995).

[11] Two proofs handle $\vee$. Prawitz (1965) adds the commuting conversions described in Remark 3.36, which move a destructor into the branches of a **case**, and counts maximal segments where the proof below counts maximal formulas; the counting argument then goes through. Tait (1967) defines, by induction on types, which terms are *computable*, shows that every computable term normalises, and shows that every typed term is computable. His method covers every connective at once and shows more: every order of steps reaches the normal form, not just a well-chosen one. Girard et al. (1989, chapters 6 and 10) give it.

[12] The argument that follows is Prawitz's, for derivations (1965), and Girard et al. (1989, chapter 4) give it for $\to$ and $\wedge$ much as here. Turing had found it for terms in the early 1940s, in a note that was published only in 1980 (Gandy, 1980).

(b) The result of the step stands where the redex stood. If the result is a λ or a pair, and the redex had a destructor applied to it, as in $((\lambda x.\, M)\, N)\, P$ or $\mathit{fst}\,(\mathit{fst}\,(M, N))$, that is a new redex. Its constructor has the type B of the result in the β case, and A or B in a projection: again of smaller degree than $A \to B$ or $A \wedge B$.

Every other redex after the step was there before, though perhaps copied: a β step makes one copy of N, with all its redexes, for each occurrence of x in M.

The choice of step. Take a typed term with measure (d, n). Choose a redex R of degree d that contains no other redex of degree d. One exists: if a redex of degree d contains another, look at that one instead, and since the term is finite this stops. The choice matters because of copying. Had R contained a second redex of degree d in its argument, the step could have copied it, and the number of redexes of degree d could have gone up.

Take the step on R. Then:

- R itself is gone;
- by the key observation, no new redex of degree d or more has appeared;
- the copies made of the argument of R contain no redex of degree d, because R contained none.

So after the step there are two possibilities:

- no redex of degree d is left, and the measure is (d', n') with $d' < d$, or the term is in normal form;
- the largest degree is still d, and the measure is $(d, n - 1)$.

Why the run stops. Compare measures as words are ordered in a dictionary: first by d, and by n only when the ds are equal. Each chosen step makes the measure smaller in this order. When d goes down, n may go up, by any amount, but that does not matter, because d has gone down. Now d is a positive whole number, so it can go down only finitely many times; and while d stays the same, n goes down by one at each step, so that too can happen only finitely many times in a row. So after finitely many steps no redex is left, and the term is in normal form. □

It is an argument about the shapes of terms and nothing else: it counts redexes and watches the count go down.

Remark 3.34 (why disjunction fails). With $\vee$ the key observation fails. A step on a **case** puts a branch where the **case** stood, and the branch can have any type at all. Take $a : p$, and write I for $\lambda y.\, y$, of type $r \to r$. The term

$$\mathit{fst}\,(\mathbf{case}\ (\mathit{Left}\, a)\ \mathbf{of}\ \{\, \mathit{Left}\, x \Rightarrow (I, I) \mid \mathit{Right}\, z \Rightarrow (I, I)\,\})$$

has one redex, the **case** on a tag. Give $\vee$ a degree in the same way as $\wedge$. Then this redex has the degree of $p \vee q$, which is 2, and its only step gives $\mathit{fst}\,(I, I)$, a redex of degree 3, the degree of $(r \to r) \wedge (r \to r)$. The largest degree has gone up, and with a single redex there was no other step to choose. On the proof side, a *fst* meeting a **case** is what Remark 3.36 calls a maximal segment, and the measure does not see it.

With normalisation in hand, every proof can be straightened.

Corollary 3.35 (normal derivations)**.** *Every derivation can be rewritten, by removing maximal formulas, into a normal derivation of the same formula whose open hypotheses are among those of the original.*

Proof. Take a derivation of A. We rewrite it in four moves.

1. Fold it into a typed term, by Curry–Howard: $\Gamma \vdash_\lambda M : A$, where Γ declares its open hypotheses.
2. Run M to a normal form N, by normalisation.
3. By subject reduction, N has the same type in the same context: $\Gamma \vdash_\lambda N : A$.
4. Unfold this typing into a derivation of A. Its open hypotheses are among those that Γ declares, which are the open hypotheses of the original.

The term N is in normal form, so it is normal, and a normal term is the term of a normal derivation: the derivation it unfolds to is normal. □

By Church–Rosser, the normal derivation reached does not depend on the order in which the detours are removed.

Remark 3.36 (maximal segments)*.* A normal derivation can still go the long way round when $\vee$ is involved. With $s : A \vee A$, no step can be taken in

$$\mathit{fst}\,(\mathbf{case}\ s\ \mathbf{of}\ \{\ \mathit{Left}\,x \Rightarrow (x, x) \mid \mathit{Right}\,y \Rightarrow (y, y)\ \}),$$

because *fst* meets a **case** rather than a pair. Yet the formula $A \wedge A$ is built by ($\wedge$I) in each branch, carried down through the ($\vee$E) bar as its conclusion, and taken apart by ($\wedge\mathrm{E}_1$) below it: created by an introduction and eventually destroyed by an elimination, with no single occurrence of it maximal. A chain of occurrences of one formula of this kind, passed down through minor premisses of ($\vee$E) from an introduction to the major premiss of an elimination, is a *maximal segment*, and a maximal formula is a maximal segment of length one; a conclusion of ($\bot$E) taken apart by an elimination is a detour of the same sort.

Removing them takes further steps, the *commuting conversions*, which move the *fst* into the branches, as in

$$\begin{aligned}&\mathit{fst}\,(\mathbf{case}\ M\ \mathbf{of}\ \{\ \mathit{Left}\,x \Rightarrow P \mid \mathit{Right}\,y \Rightarrow Q\ \})\\ &\qquad\rightsquigarrow\ \mathbf{case}\ M\ \mathbf{of}\ \{\ \mathit{Left}\,x \Rightarrow \mathit{fst}\,P \mid \mathit{Right}\,y \Rightarrow \mathit{fst}\,Q\ \},\end{aligned}$$

and similar ones for the other destructors and for *absurd*. With them every derivation can be rewritten into one with no maximal segment, and every formula in such a derivation is a piece of its conclusion or of one of its open hypotheses, for all the connectives.[13]

We do without them: the proof of consistency (Theorem 3.38) goes through the shape of closed terms, and β is enough for that.

Now watch normalisation pay off, in the argument that made us want it. A normal proof never invents material: for $\to$ and $\wedge$, every formula in it is a piece of its conclusion or of one of its open hypotheses (with $\vee$ the same holds, after further rewriting steps). So to ask whether $\bot$ can be proved

[13]The conversions, and this form of the normalisation theorem, are Prawitz's (Prawitz, 1965).

from nothing is to ask what a normal proof of $\bot$ from no open hypotheses could look like. Folded up, such a proof is a closed normal term, and there are very few of those, since a neutral term has a free variable at its head.

Lemma 3.37 (shape of closed normal forms)**.** *A closed typed term in normal form is a constructor. Consequently there is no closed term in normal form whose type is a letter p, and none whose type is $\bot$.*

Proof. Take a closed typed term in normal form. It is normal, so it is a constructor or a neutral term. A neutral term has a free variable, its head; our term is closed, so it is not neutral, and it is therefore a constructor.

A constructor has type $A \to B$, $A \wedge B$ or $A \vee B$, according to the rule that introduces it, and none of these is a letter or $\bot$. □

Theorem 3.38 (consistency)**.** *There is no closed term of type $\bot$: $\nvdash_{\mathrm{NJ}} \bot$. More generally, for every letter p, $\nvdash_{\mathrm{NJ}} p$.*

Proof. Suppose, for a contradiction, that $\vdash_{\mathrm{NJ}} \bot$. Then:

$\vdash_\lambda M : \bot$ for some closed term M	Curry–Howard,
$M \rightsquigarrow^* N$, with N in normal form	normalisation,
$\vdash_\lambda N : \bot$	subject reduction, at each step,
N is closed	steps create no free variables.

So N is a closed term in normal form of type $\bot$, and the shape lemma says there is no such term. The argument for a letter p is the same, with p in place of $\bot$. □

Read what we have shown. The logic of this chapter does not prove everything. It does not prove $\bot$, and it does not prove any bare claim p: whatever it proves has an arrow, an *and* or an *or* on the outside. So it proves things about how claims relate, and never a bare claim out of nothing.

The argument used nothing about what the letters mean, no table, no notion of a formula being the case. It's an argument about the shapes of strings, and through programs it takes a page.

Ask whether arithmetic proves everything, rather than the bare connectives, and the question is much harder. It is the subject of Chapter 12, where the answer is of a different kind.

One more thing falls out of the shape lemma, and it's worth having now.

Corollary 3.39 (disjunction property)**.** *If $\vdash_{\mathrm{NJ}} A \vee B$, then $\vdash_{\mathrm{NJ}} A$ or $\vdash_{\mathrm{NJ}} B$.*

Proof. Suppose $\vdash_{\mathrm{NJ}} A \vee B$. By Curry–Howard there is a closed term of type $A \vee B$. By normalisation and subject reduction it runs to a closed term in normal form of the same type, and by the shape lemma that term is a constructor. The only constructors of type $A \vee B$ are the two tags, so it is one of:

- *Left* M with $\vdash_\lambda M : A$, and then $\vdash_{\mathrm{NJ}} A$;
- *Right* M with $\vdash_\lambda M : B$, and then $\vdash_{\mathrm{NJ}} B$.

□

In this logic you can't establish "A or B" without establishing one of them and saying which, because a proof of a disjunction is a value with a tag on

it. A hypothesis may say "A or B" without saying which; a theorem cannot. Hold on to this. Part 2 has a use for it.

Finding a tree. Normalisation has a practical side, and it's the one you'll use most. Every derivable formula has a normal derivation, and a normal derivation has the shape of a normal term: introductions near the root, building the conclusion, sitting on chains of eliminations that hang off hypotheses and take them apart. So when you look for a tree, you need only look for one of that shape, and the shape tells you where to start: at the bottom, with the goal, working *upwards*. Take the currying formula of Exercise 3.5, $(p \wedge q \to r) \to p \to q \to r$.

The goal is an implication, and there's one sensible way to prove an implication: suppose its antecedent and aim for its consequent. So the last bar is ($\to$I), and what's left above it is a smaller problem:

$$\dfrac{\begin{array}{c}[p \wedge q \to r]^1 \\ \vdots \\ p \to q \to r\end{array}}{(p \wedge q \to r) \to p \to q \to r}\ {\to}\mathrm{I},1$$

The new goal is an implication too, and so is the one after it: two more bars, two more suppositions.

$$\dfrac{\dfrac{\dfrac{\begin{array}{ccc}[p \wedge q \to r]^1 & [p]^2 & [q]^3 \\ & \vdots & \\ & r & \end{array}}{q \to r}\ {\to}\mathrm{I},3}{p \to q \to r}\ {\to}\mathrm{I},2}{(p \wedge q \to r) \to p \to q \to r}\ {\to}\mathrm{I},1$$

Now the goal is r, a letter, and a letter has no introduction rule: there's nothing left to build on the way up. So turn round and work *downwards*, from what you have. Of the three suppositions, only $p \wedge q \to r$ can produce an r, by ($\to$E), and only if you can supply $p \wedge q$; and ($\wedge$I) makes that from the other two:

$$\dfrac{p \wedge q \to r \qquad \dfrac{p \qquad q}{p \wedge q}\ \wedge\mathrm{I}}{r}\ {\to}\mathrm{E}$$

Put this in the gap and the tree is finished: three introductions at the bottom, an elimination and the ($\wedge$I) it needed at the top, and no detour anywhere. The method, in four lines:

1. While the goal is $\varphi \to \psi$, suppose φ and make ψ the goal. While it is $\varphi \wedge \psi$, make two goals of it. A goal $\neg\varphi$ is $\varphi \to \bot$: suppose φ and aim for $\bot$.
2. When the goal is a letter or $\bot$, work down from what you have, taking it apart with elimination rules until the goal turns up. A $\bot$ usually comes from ($\to$E) on a negation you have.
3. If you have a disjunction, split on it with ($\vee$E) early, and solve each case.

4. If you want a disjunction, use ($\vee$I) last, once you know which side you can prove.

The fourth line is the one people break, and the disjunction property says why. Try to prove $p \vee q \to q \vee p$. After supposing $p \vee q$, the goal is $q \vee p$, and it is tempting to reach for ($\vee\mathrm{I}_1$) at once and aim for q:

$$\dfrac{\begin{array}{c}[p \vee q]^1 \\ \vdots \\ q\end{array}}{q \vee p}\ \vee\mathrm{I}_1$$

and then you're stuck, because nothing you have gives q: it might be p that holds. A proof of a disjunction has to say which side, and under the supposition $p \vee q$ you can't say until you've split on it. Line 3 first: split on $p \vee q$, and in each case you know which side to choose.

Programmers know the method as *type-directed* programming: the type $A \to B$ says to write $\lambda x.$ first, a value of type $A \vee B$ in hand says to write a **case**, and what's left is filling the holes.

It's a guide, not an algorithm. A choice made early, which side of an $\vee$ or which hypothesis to apply, may have to be undone. When every attempt sticks, that's a hint, not a proof, that there's no tree; but the shape argument can turn the hint into a proof, and Exercise 3.8 does it for $\neg\neg p \to p$.[14]

3.7. **Programs as proofs.** With its steps, the simply typed λ-calculus is a programming language, a small one with its type checker built in: the rules of Figure 3.3 are the checker, and the steps of Figure 3.4 are the interpreter. Its programs are proofs, and each typed program proves the formula that is its type. Here are three, each a theorem given with the program that proves it.

Example 3.40 (swapping a pair)**.** The program $\lambda p.\ (\mathit{snd}\, p,\ \mathit{fst}\, p)$ has type $A \wedge B \to B \wedge A$, so it proves that formula. In the context $p : A \wedge B$:

$\mathit{snd}\, p : B$	($\wedge\mathrm{E}_2$)
$\mathit{fst}\, p : A$	($\wedge\mathrm{E}_1$)
$(\mathit{snd}\, p,\ \mathit{fst}\, p) : B \wedge A$	($\wedge$I)
$\lambda p.\ (\mathit{snd}\, p,\ \mathit{fst}\, p) : A \wedge B \to B \wedge A$	(abs), which discharges p.

This is the tree we drew when learning to discharge (Example 3.7), with A and B for its p and q, and the two bracketed leaves are the two occurrences of p in the body.

- A variable used twice is a hypothesis discharged at two leaves;
- a variable used once, at one;
- a parameter never used at all, $\lambda x.\, \lambda y.\ x$, which is K, is the receipt for a debt of nothing.

[14] The advice to take formulas apart from the bottom upwards and then build the required ones, and its link with normalisation, are van Dalen's (van Dalen, 2013, pp. 33, 36–37).

Example 3.41 (contraposition)**.** The program $\lambda f.\, \lambda k.\, \lambda a.\ k\,(f\,a)$ proves contraposition, $(A \to B) \to (\neg B \to \neg A)$. In the context $f : A \to B,\ k : B \to \bot,\ a : A$, two uses of (app) give $k\,(f\,a) : \bot$, and three uses of (abs) give the type claimed. Rub out the program before the last two uses of (abs), and what is left is modus tollens: from $A \to B$ and $\neg B$, conclude $\neg A$.

Example 3.42 (swapping a union)**.** The program $\lambda s.$ **case** s **of** $\{\, \mathit{Left}\, a \Rightarrow \mathit{Right}\, a \mid \mathit{Right}\, b \Rightarrow \mathit{Left}\, b \,\}$ has type $A \vee B \to B \vee A$, so it proves that formula, by cases. Rub out the program and what is left is an ($\vee$E) bar with an ($\vee$I) bar in each branch. The two branch variables a and b are its two discharged leaves A and B, each in scope only in its own branch.

Programs like these are ones you could have written without knowing what they proved:

- a function that swaps a pair;
- a function that composes a test with a function;
- a function that swaps the tags on a union.

The types were there all along, as Curry–Howard says they must be.

Now take the three rules (var), (app) and (abs), rub out the programs, and read what is left.

- A supposition in force may be written down.
- From $A \to B$ and A, conclude B.
- If B follows on the supposition A, conclude $A \to B$ and drop the supposition.

That is how you were taught to prove an implication, in the first mathematics lecture you attended. Every time the checker reads a definition `f x = ...`, that is what it checks.

Remark 3.43 (at work)*.* None of this is a curiosity of logic books. The language ML, of 1978, is the typed λ-calculus with the types worked out by the compiler, offered with the promise that "well-typed programs cannot go wrong" (Milner, 1978). OCaml, F# and Haskell descend from it, and every lambda or closure in Java, C#, Swift, Rust or TypeScript is checked by rule (abs). The Glasgow Haskell compiler translates Haskell into a small typed λ-calculus of its own and can type-check that again after each optimisation, so that a compiler bug shows up as a type error.

On the logic side, the simply typed calculus is the logic of the HOL family of proof assistants (HOL4, HOL Light, Isabelle/HOL), and in Coq (now Rocq), Lean and Agda a proof *is* a typed λ-term, in a richer calculus than this one, checked by a small kernel with rules like (var), (app) and (abs). Between them they have checked the seL4 operating-system kernel (Klein et al., 2009), the CompCert C compiler (Leroy, 2009), the Four Colour Theorem (Gonthier, 2008) and the Kepler conjecture (Hales et al., 2017). In each the trusted part is a type checker, and whoever wants the theorem does the hard part, finding the term.

Consider these two programs. The first takes two arguments and returns the first:

$$(\lambda x.\, \lambda y.\ x)\, M\, N \ \rightsquigarrow\ (\lambda y.\ M)\, N \ \rightsquigarrow\ M,$$

renaming y first if it is free in M. The second takes three, and applies the first to the third and to what the second makes of the third:

$$(\lambda x.\, \lambda y.\, \lambda z.\; x\, z\, (y\, z))\, M\, N\, P \;\rightsquigarrow\; (\lambda y.\, \lambda z.\; M\, z\, (y\, z))\, N\, P$$
$$\rightsquigarrow\; (\lambda z.\; M\, z\, (N\, z))\, P \;\rightsquigarrow\; M\, P\, (N\, P).$$

What are their types? Fold each up as in §3.3, with the discharged hypotheses named for the parameters. The first is

$$\dfrac{\dfrac{[x : A]^1}{\lambda y.\; x : B \to A}\,{\to}\mathrm{I},2}{\lambda x.\, \lambda y.\; x : A \to B \to A}\,{\to}\mathrm{I},1$$

where the bar marked 2 discharges nothing, since y is never used. The second begins

$$\dfrac{\dfrac{\dfrac{[x : A \to B \to C]^1 \qquad [z : A]^3}{x\, z : B \to C}\,{\to}\mathrm{E} \qquad \dfrac{[y : A \to B]^2 \qquad [z : A]^3}{y\, z : B}\,{\to}\mathrm{E}}{x\, z\, (y\, z) : C}\,{\to}\mathrm{E}}{\lambda z.\; x\, z\, (y\, z) : A \to C}\,{\to}\mathrm{I},3$$

where z is used twice and discharged once, and two more ($\to$I) bars, discharging y and then x, end it at

$$\lambda x.\, \lambda y.\, \lambda z.\; x\, z\, (y\, z) : (A \to B \to C) \to (A \to B) \to A \to C.$$

So the types are $A \to B \to A$ and $(A \to B \to C) \to (A \to B) \to A \to C$. Steps and types are both those of the combinators K and S, so we name the two terms after them:

$$\mathsf{K} \coloneqq \lambda x.\, \lambda y.\; x,$$
$$\mathsf{S} \coloneqq \lambda x.\, \lambda y.\, \lambda z.\; x\, z\, (y\, z).$$

Every combinator term is now a λ-term, and the two constants need no longer be taken as given. Exercise 3.4 checks the two typings again in the style of Figure 3.3.

Remark (looking back: the missing rule). Chapter 1's checker had no rule for a name with a parameter. Rule (abs) is that rule. With it the compiler no longer has to compile anything to decide whether to say yes: it reads the definition as written. Adopting it types nothing new, as the next theorem shows.

Theorem 3.44 (translation into combinators)**.** *Every term M built from variables by application and λ translates to a term M° built from K, S and variables by application, with no λ in it, such that:*

(i) M° has the same free variables as M and does the same thing to every argument: if M is $\lambda x.\, P$ then $M^\circ\, N \rightsquigarrow^* P^\circ[N/x]$ for every N;

(ii) M° has the types M has: if $\Gamma \vdash_\lambda M : A$ then $\Gamma \vdash_{\mathrm{KS}} M^\circ : A$.

Conversely every combinator term is a λ-term, by the definitions of K and S above, with the same types.

Proof. The plan is to remove the λs one at a time, innermost first. An innermost abstraction $\lambda x.\, P$ has no λ in its body P, so it is enough to know

how to remove a single λ from a body built by application alone. For such a body P, define $\langle x\rangle P$, "P abstracted over x", by recursion on P:

$$\langle x\rangle P := \begin{cases} \mathsf{K}\,P & \text{if } x \text{ does not occur in } P, \\ \mathsf{S}\,\mathsf{K}\,\mathsf{K} & \text{if } P \text{ is } x, \\ \mathsf{S}\,(\langle x\rangle P_1)\,(\langle x\rangle P_2) & \text{if } P \text{ is } P_1\,P_2 \text{ and } x \text{ occurs in it.} \end{cases}$$

The result contains no λ and no x. We check two things about it: it behaves as $\lambda x.\,P$ does, and it has the type $\lambda x.\,P$ has.

Behaviour. For every term N,

$$(\langle x\rangle P)\,N \;\rightsquigarrow^*\; P[N/x].$$

We prove this by induction on the size of P, one case for each line of the definition.

1. If x does not occur in P: $\mathsf{K}\,P\,N \rightsquigarrow P$, and $P[N/x]$ is P, since there is no x to replace.
2. If P is x: $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,N \rightsquigarrow \mathsf{K}\,N\,(\mathsf{K}\,N) \rightsquigarrow N$, and $x[N/x]$ is N.
3. If P is $P_1\,P_2$: the S step hands N to both halves,

   $$\mathsf{S}\,(\langle x\rangle P_1)\,(\langle x\rangle P_2)\,N \;\rightsquigarrow\; (\langle x\rangle P_1)\,N\,((\langle x\rangle P_2)\,N),$$

   and by the induction hypothesis, applied to each half, this runs to $P_1[N/x]\,P_2[N/x]$, which is $(P_1\,P_2)[N/x]$.

Type. If $\Gamma, x : A \vdash_{\mathrm{KS}} P : B$, then $\Gamma \vdash_{\mathrm{KS}} \langle x\rangle P : A \to B$, the type that $\lambda x.\,P$ has. The same induction proves it.

1. If x does not occur in P: then $P{:}B$ without using x, and $\mathsf{K}{:}B \to A \to B$, so $\mathsf{K}\,P : A \to B$.
2. If P is x: then B is A, and $\mathsf{S}\,\mathsf{K}\,\mathsf{K} : A \to A$, as Chapter 1 showed.
3. If P is $P_1\,P_2$, with $P_1 : D \to B$ and $P_2 : D$: by the induction hypothesis $\langle x\rangle P_1 : A \to D \to B$ and $\langle x\rangle P_2 : A \to D$. Then S, at the type $(A \to D \to B) \to (A \to D) \to A \to B$, gives $\mathsf{S}\,(\langle x\rangle P_1)\,(\langle x\rangle P_2) : A \to B$.

The whole term. Now take an arbitrary term M. Find an innermost abstraction $\lambda x.\,P$ in it, and replace it by $\langle x\rangle P$. Repeat until no λ is left, and call the result M°. Each replacement keeps the free variables, since $\langle x\rangle P$ has those of $\lambda x.\,P$; it keeps the behaviour on arguments, by the first check; and it keeps the type, by the second, since a typing looks only at the types of the parts. So M° has the properties (i) and (ii). □

Remark (looking back: the Deduction Theorem as a compiler). You've seen the three lines of $\langle x\rangle P$ before.

1. The first writes a K when the parameter is not used;
2. the second writes the identity for the parameter itself;
3. the third writes an S to push the parameter through an application.

Those are, case for case, the three cases of the proof of the Deduction Theorem in Chapter 2, with the programs left on the lines:

1. an axiom or hypothesis weakened by (I);
2. the supposition itself, proved by the five-line proof of $p \to p$;
3. a modus ponens distributed by (II).

That chapter said the proof was the compiler that turns your `\ta ->` into combinators. Here it is, three lines long.

Remark 3.45 (one theorem, twice). Rub the programs out of the translation and it is a theorem you have already seen. By Curry's correspondence (Theorem 2.13) a typed combinator term is a Hilbert proof, and by Curry–Howard a typed λ-term is a derivation in NJ. The translation turns a derivation into a Hilbert proof of the same formula, and its converse, the definitions of K and S, turns a Hilbert proof into a derivation: K and S are the trees drawn for the schemas (I) and (II) in §3.2, folded up. So with the programs rubbed out, the theorem says that NJ and HJ prove the same formulas, which is the equivalence theorem of that section, and its proof is the proof given there.

3.8. **Dropping the types.** The typing rules pick out the terms that come from trees. Drop them and keep the steps, and any term built from the forms is a program, whether or not it comes from a tree. Here is the whole language at last.

Definition 3.46 (λ-terms)**.** Fix a stock of variables $x, y, z, f, \ldots$. The terms are defined inductively:

- every variable x is a term;
- if M and N are terms then $M\,N$ is a term, the *application* of M to N;
- if x is a variable and M is a term then $\lambda x.\, M$ is a term, the *abstraction* of M over x, the function that takes x and returns M;
- if M, N, P and Q are terms and x and y are variables, then (M, N), $\mathit{fst}\,M$, $\mathit{snd}\,M$, $\mathit{Left}\,M$, $\mathit{Right}\,M$, $\mathit{absurd}\,M$ and $\mathbf{case}\ M\ \mathbf{of}\ \{\, \mathit{Left}\,x \Rightarrow P \mid \mathit{Right}\,y \Rightarrow Q \,\}$ are terms, the forms of the other connectives;
- nothing else is a term.

In short,

$$\begin{aligned} M, N, P, Q \ ::= {} & x \mid M\,N \mid \lambda x.\, M \\ & \mid (M, N) \mid \mathit{fst}\,M \mid \mathit{snd}\,M \mid \mathit{Left}\,M \mid \mathit{Right}\,M \mid \mathit{absurd}\,M \\ & \mid \mathbf{case}\ M\ \mathbf{of}\ \{\, \mathit{Left}\,x \Rightarrow P \mid \mathit{Right}\,y \Rightarrow Q \,\}. \end{aligned}$$

Application groups to the left, as it did for the combinators, and a λ reaches as far to the right as it can, so $\lambda x.\ M\,N$ is $\lambda x.\,(M\,N)$, not $(\lambda x.\, M)\,N$. We keep the ready-made integers and $+$.

You've written the third form many times. It is `\x -> M` in Haskell, `lambda x: M` in Python and `x => M` in JavaScript;[15] every `def f(x): return M` is one with a name attached. These terms, with the steps of §3.5 and no typing rules, are the *untyped λ-calculus*, the setting in which the λ was first studied.[16]

[15] Python's `lambda` came from Church by way of Lisp. John McCarthy, designing Lisp in 1958, needed a notation for functions passed as arguments, and "it seemed natural to use the λ-notation of Church", from Church's little book of 1941. Twenty years later he added: "I didn't understand the rest of his book, so I wasn't tempted to try to implement his more general mechanism for defining functions" (McCarthy, 1978). In 2005 Python's designer proposed taking `lambda` out of the language, because "most Python users are unfamiliar with Lisp or Scheme, so the name is confusing". It stayed.

[16] Church used it in 1936 to exhibit a problem no procedure can settle (Church, 1936b), and gave it types in 1940. Why λ? Whitehead and Russell had written the function with

Every step of §3.5 still makes sense in it. What fails is the theorem about where runs end.

First, the Monster: the function that applies its argument to itself, applied to itself. In λ notation,

$$\Omega \;=\; (\lambda x.\; x\,x)\,(\lambda x.\; x\,x) \;\rightsquigarrow\; (\lambda x.\; x\,x)\,(\lambda x.\; x\,x) \;\rightsquigarrow\; \cdots,$$

and each step gives back the term it started from. Nothing in the definition of λ-terms says what may be applied to what, and $x\,x$, which no typing accepts, is a term like any other. Self-application of this kind sank both of the logics these calculi were first built for, and in both the repair was the same: types.

Two runs of one term can still go apart, and the Church–Rosser theorem (Theorem 3.32) still brings them back together: its proof never looked at a type. So an untyped term, too, has at most one normal form.

Without types, though, the order can decide whether the end is reached at all. Put Ω where it is thrown away, as in $(\lambda x.\; y)\,\Omega$: the outer redex gives y in one step, while a run that keeps choosing the redex inside Ω never stops.

Fixing an order. Church–Rosser leaves the order free: every order that ends, ends at the same normal form. It does not say that every order ends, and $(\lambda x.\; y)\,\Omega$ is one that needn't. So a language picks an order, by deleting or restricting rules. *Lazy* evaluation, or call by name, keeps β and (app_1) and deletes (app_2) and (abs): an argument is run only where it is used.[17] *Eager* evaluation, or call by value, keeps (app_1) and (app_2) and lets β fire only when its argument is in normal form: an argument is run before the call. (Which of the two to try first is a further choice, and a language settles that too.) On $(\lambda x.\; y)\,\Omega$, lazy gives y in one step and eager runs Ω for ever. Neither runs under a λ, and real languages don't either, so a run can stop at a term with redexes still inside it, a *value*; a value need not be a normal form, and about values Church–Rosser says nothing.

The steps of Chapter 1 are β steps, as §3.7 showed for K and S. The step $\mathsf{K}\,M\,N \rightsquigarrow M$ is two β steps on $(\lambda x.\,\lambda y.\; x)\,M\,N$, and the step $\mathsf{S}\,M\,N\,P \rightsquigarrow M\,P\,(N\,P)$ is three.

Remark (looking back: combinators). Everything Chapter 1 did with combinators carries over. A definition $f\,x_1 \cdots x_n \coloneqq M$ is the term $\lambda x_1.\; \cdots \lambda x_n.\, M$ with the name left off, and unfolding it is one β step for each parameter; with the K and S steps above, all three kinds of step there are β. The Church–Rosser theorem there is the combinator version of the one here, and was proved after it. And Ω is the Monster, the program that runs for ever: combinator terms didn't say what may be applied to what either, and the Monster's story reads unchanged in the new grain.

---

body M as $\hat{x}M$, with a hat; the story told by Church's student Rosser is that a typesetter could not set the hat over the letter and moved it in front, where it became a capital lambda and then a small one (Rosser, 1984). Church himself, asked late in life, said the choice had been arbitrary. Barendregt tells both versions and believes neither entirely (Barendregt, 1997). The three forms are in Church (1940, §1), where he also proves that K and S are λ-terms.

[17] Haskell refines this to *call by need*, which runs a copied argument once and shares the result; the difference doesn't matter here.

The translation of Theorem 3.44 never looked at a type. Its three lines compile every untyped term built from variables by application and λ, the Monster included, into K and S, and the term compiled does the same thing to every argument.

Theorem 3.47 (Turing completeness)**.** *Every function a Turing machine computes is computed by a closed λ-term, and every function a closed λ-term computes is computed by a Turing machine.*

Proof. Chapter 1 proved this for combinator terms (Theorem 1.19), and the translation of Theorem 3.44 converts each way without changing what is computed. □

That's the order natural to these notes, which built the combinators first. History went the other way round, and the History at the end of the chapter tells it: one theorem, proved by three groups of people about three notations.

A term may not refer to itself, so recursion needs help. A recursive definition says that a function is one layer of work applied to the function itself, $f = F\,f$, which makes f a *fixed point* of F. Add to the combinators a constant *fix* with the single step $\mathit{fix}\,F \rightsquigarrow F\,(\mathit{fix}\,F)$, and $\mathit{fix}\,F$ is such a fixed point: it unfolds one layer of F when it is needed, and no sooner.

Example 3.48 (factorial)**.** Send something ordinary the whole way round: factorial, as you would write it on a whiteboard.

```
fact(n):
    if n = 0 then return 1
    else return n * fact(n - 1)
```

Four things in it are not in the combinator language: a parameter, a name that calls itself, a conditional, and arithmetic. Take them away one at a time. The parameter becomes a λ, which is what this section was for. Write *isZero*, *mult* and *pred* for the test, the product and the predecessor, and use a boolean as its own conditional, so that "if b then M else N" is $b\,M\,N$. Then

$$\mathit{fact} \;\coloneqq\; \lambda n.\; \mathit{isZero}\,n\;1\;(\mathit{mult}\,n\,(\mathit{fact}\,(\mathit{pred}\,n))).$$

Nothing has really happened yet, because *fact* still occurs on the right-hand side. So make the recursive call a parameter too:

$$F \;\coloneqq\; \lambda r.\,\lambda n.\; \mathit{isZero}\,n\;1\;(\mathit{mult}\,n\,(r\,(\mathit{pred}\,n))).$$

The term F mentions nothing but its own two parameters: give it the function to call and a number, and it does one layer of the work. What we want is a term *fact* with $\mathit{fact} = F\,\mathit{fact}$, and $\mathit{fix}\,F$ is one, so define $\mathit{fact} \coloneqq \mathit{fix}\,F$. One step shows how it runs:

$$\mathit{fix}\,F\,\overline{3} \;\rightsquigarrow\; F\,(\mathit{fix}\,F)\,\overline{3},$$

and on the right F's parameter r is $\mathit{fix}\,F$ again: the recursive call, ready to unfold once more when it is needed and no sooner.[18]

[18] In the untyped calculus *fix* need not be added: Curry's $\mathsf{Y} = \lambda f.\;(\lambda x.\;f\,(x\,x))\,(\lambda x.\;f\,(x\,x))$ satisfies $\mathsf{Y}\,F \rightsquigarrow^{*} F\,(\mathsf{Y}\,F)$ by two β steps, and Turing gave one of his own in 1937, Θ, which does the same in a form that reduces *to* $F\,(\Theta\,F)$ rather than merely to something equal to it (Turing, 1937b). Both are the Monster's self-application put to work, which is why neither has a type (§3.6).

The arithmetic was combinators already: the numerals, addition and multiplication were all built from K and S, and with more work so is the predecessor. We keep the three as names only to keep the term on the page.

One thing remains, the two λs, and the translation $\langle x\rangle P$ removes them. Write B for the body, so that $F = \lambda r.\, \lambda n.\, B$, and read B as the application $(\mathit{isZero}\, n\ \ 1)\,(\mathit{mult}\, n\,(r\,(\mathit{pred}\, n)))$. The line for an application splits it, the other two lines finish each half, and with I written for $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$,

$$\langle n\rangle B \;=\; \mathsf{S}\ \underbrace{(\mathsf{S}\,(\mathsf{S}\,(\mathsf{K}\,\mathit{isZero})\,\mathsf{I})\,(\mathsf{K}\,1))}_{\text{the test and the base case}}\ \underbrace{(\mathsf{S}\,(\mathsf{S}\,(\mathsf{K}\,\mathit{mult})\,\mathsf{I})\,(\mathsf{S}\,(\mathsf{K}\,r)\,(\mathsf{S}\,(\mathsf{K}\,\mathit{pred})\,\mathsf{I})))}_{\text{the recursive case}},$$

twenty-six symbols, with r still free. Abstract r out of that by the same three lines and every variable is gone:

$$F^\circ = \mathsf{S}\,(\mathsf{K}\,(\mathsf{S}\,A))\,(\mathsf{S}\,(\mathsf{K}\,(\mathsf{S}\,C))\,(\mathsf{S}\,(\mathsf{S}\,(\mathsf{K}\,\mathsf{S})\,(\mathsf{S}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{I}))\,(\mathsf{K}\,E))),$$

$$\text{where}\quad \begin{cases} A = \mathsf{S}\,(\mathsf{S}\,(\mathsf{K}\,\mathit{isZero})\,\mathsf{I})\,(\mathsf{K}\,1), \\ C = \mathsf{S}\,(\mathsf{K}\,\mathit{mult})\,\mathsf{I}, \\ E = \mathsf{S}\,(\mathsf{K}\,\mathit{pred})\,\mathsf{I}. \end{cases}$$

There is not a variable, a parameter, a conditional or a recursive call anywhere in it, only K, S, three arithmetic constants and juxtaposition. Yet $\mathit{fix}\, F^\circ$ applied to $\overline{4}$ runs to 24, by the K and S steps and the steps of the constants. A program you could have written in your first week sits on two constants in exactly this way, and so does every program you'll ever compile.

Remark 3.49 (general recursion). Every language you use has recursion. Model it by a constant *fix* with the type $(A \to A) \to A$ for every A and the step $\mathit{fix}\, f \rightsquigarrow f\,(\mathit{fix}\, f)$, as a recursive definition such as factorial needs. With it, $\mathit{fix}\,(\lambda x.\ x)$ has every type A there is, $\bot$ and every letter included, so the consistency theorem fails; and it runs for ever, so normalisation fails with it. The two theorems go together, as they came together.

Read as a logic, a language with general recursion proves everything, and a type in such a language promises only what the program does *if* it returns. The price of factorial is the consistency of the logic. Every language you use pays it, and pays it knowingly: the loop is worth more to a programmer than consistency is. A programmer who wants both must say in the type where the recursion is allowed to go, and that takes a logic with more to say than this one.

The same price is why the checker's *yes*, in the languages you use, means "safe" and never "correct". The next chapter is about the gap between the two.

3.9. **Fitch proofs** *(optional).* You've been drawing boxes all along, and they are a second notation for NJ. A proof can be written not as a tree but down the page, one line after another as a Hilbert proof is, with one addition:

- a supposition opens a box;
- every line inside the box is reasoned under it;
- the box closes at the line that discharges it, which stands outside.

Drawn with the box as a vertical bar and the supposition underlined at its top, a proof of this kind is called a *Fitch proof*.[19] The boxes in which the combinator typings were written are Fitch proofs with the programs on, each a single box because nothing there was ever discharged.

The two notations were invented separately, as the History at the end of the chapter tells, and Theorem 3.52 at the end of this section says that they agree.

A Fitch proof is a Hilbert proof that may open boxes inside its box, and it is an NJ derivation written down the page: a box is a subtree, and the nesting of boxes records what OH records.

Definition 3.50 (Fitch proof)**.** A *box* has a *head* above a horizontal line and, below the line, a finite sequence of *entries*, each of which is one of:

- a formula, called a *line*;
- another box, called a *subbox*.

The head of the outermost box is a set of formulas. The head of a subbox is a single formula, its *supposition*, and its last entry is a line. The lines are numbered in one sequence down the page, whichever box they stand in, and a subbox is cited by the numbers i–j of its first and last lines:

	Γ	
1	φ_1	*reason*
	$\vdots$	
	ψ	
i	φ_i	*reason*
	$\vdots$	
j	φ_j	*reason*
	$\vdots$	
n	φ_n	*reason*

A line or subbox is *in scope* at line k if:

- it comes before line k, and
- it is an entry of a box that contains line k: the box that line k stands in, or one around it.

So a subbox is in scope after it has closed, and the lines inside it are not.

Let Γ be a set of formulas, the *hypotheses*. A *Fitch proof of φ from Γ* is a box with head Γ whose last entry is the line φ, and in which every line φ_k, in a subbox or not, is justified in one of the following ways, where "earlier" means "in scope at line k":

1. (*hyp*) φ_k is a member of Γ, or the supposition of a subbox that contains line k;

[19] After Frederic Fitch, at Yale, whose *Symbolic Logic: An Introduction* (1952) made the notation standard. His bars are Jaśkowski's boxes with their right-hand sides rubbed out. Every proof assistant in use presents its proofs this way, with "suppose" as `intro` or `assume` and the box as indentation, and so does the scope rule of every compiler: a parameter is in scope inside the function and not outside it.

2. (→E) there are earlier lines φ_i and φ_j such that φ_i is the formula $\varphi_j \to \varphi_k$;
3. (→I) φ_k is $\psi \to \chi$, and there is an earlier subbox with supposition ψ whose last line is χ;
4. (∧I) φ_k is $\psi \wedge \chi$, and ψ and χ are earlier lines;
5. (∧E$_1$, ∧E$_2$) some earlier line is $\varphi_k \wedge \chi$, for some χ, or $\psi \wedge \varphi_k$, for some ψ;
6. (∨I$_1$, ∨I$_2$) φ_k is $\psi \vee \chi$, and ψ, or χ, is an earlier line;
7. (∨E) some earlier line is $\psi \vee \chi$, and there are two earlier subboxes, one with supposition ψ and one with supposition χ, each with last line φ_k;
8. (⊥E) $\bot$ is an earlier line.

Theorem 3.52 below shows that φ has a Fitch proof from Γ exactly when $\Gamma \vdash_{\mathrm{NJ}} \varphi$, so no new symbol is needed.

Put this beside Definition 2.4. The outer box is the same box, with the hypotheses above the line and the lines below it. (hyp) is the hypothesis clause, with the supposition of every box around a line counted among that line's hypotheses, and (→E) is modus ponens, word for word. The axiom clause has gone: the rules that open subboxes do its work.

Everything new comes with the subbox, and so does the one restriction. A line inside a subbox was reasoned under its supposition, so once the subbox has closed, nothing outside it may cite that line. Only the subbox as a whole may be cited, and only by (→I) and (∨E), the two rules that discharge.

Each line has a *context*: Γ together with the suppositions of the subboxes that contain it. A line φ with context Δ stands for the judgement $\Delta \vdash_{\mathrm{NJ}} \varphi$. Read that way, each clause is one of the typing rules of Figure 3.3 with the terms deleted:

- (hyp) is (var);
- (→E) is (app);
- (→I) is (abs), whose premiss $\Delta, \psi \vdash_{\mathrm{NJ}} \chi$ is the last line of a subbox with supposition ψ.

Example 3.51 (Baker, cont'd: her rule as a box)**.** Baker's rule once more, drawn earlier as a tree (Example 3.5) and now as a box:

	$t \to r,\ r \to d$	
	$\quad t$	
1	$\quad t$	(hyp)
2	$\quad t \to r$	(hyp)
3	$\quad r$	(→E) 2, 1
4	$\quad r \to d$	(hyp)
5	$\quad d$	(→E) 4, 3
6	$t \to d$	(→I) 1–5

It is her six lines of Example 3.2 with the justifications formalised. Lines 2 and 4 stand inside the subbox and cite hypotheses from the head of the

outer box, which is around them. Line 6 stands outside, and cites the subbox as a whole.

Trees and boxes are two notations for one system: they prove the same things from the same hypotheses, and they prove them by the same steps.

Theorem 3.52 (trees and boxes)**.** *For every set of formulas* Γ *and every formula* φ*, the following are equivalent:*

(i) $\Gamma \vdash_{\mathrm{NJ}} \varphi$*: some tree derivation has conclusion* φ *and all its open hypotheses in* Γ*;*

(ii) φ *has a Fitch proof from* Γ*.*

Moreover, the translations both ways go step by step:

- *each bar of a tree becomes a line justified by the same rule;*
- *each line of a Fitch proof not justified by (hyp) becomes a bar of the same rule.*

Proof. Each direction translates one notation into the other, bar for line.

Left to right. We show, by induction on the number of bars in a tree derivation $\mathcal{D}$ with conclusion φ, that φ has a Fitch proof from every set Γ that contains $\mathrm{OH}(\mathcal{D})$. One fact about Fitch proofs is used throughout. A Fitch proof from Γ can be copied, with its lines renumbered, to each place where every member of Γ is available, as a hypothesis or as the supposition of a box around that place. Each (hyp) line of the copy is still justified, and every other line cites only lines and subboxes of the copy, which are still in scope. Now look at the last bar of $\mathcal{D}$; the trees above it have fewer bars, so the induction hypothesis applies to them.

- *No bar.* $\mathcal{D}$ is the one-node tree φ. Then $\varphi \in \mathrm{OH}(\mathcal{D}) \subseteq \Gamma$, and the Fitch proof is the single line φ, by (hyp).
- *($\to$E), ($\wedge$I), ($\wedge$E), ($\vee$I), ($\bot$E).* The open hypotheses of each tree above the bar lie inside $\mathrm{OH}(\mathcal{D})$, so inside Γ. By the induction hypothesis each premiss has a Fitch proof from Γ. Copy these proofs, one after another, into a box with head Γ, and add the conclusion as a new line, justified by the rule from the last lines of the copies.
- *($\to$I).* The tree above the bar, $\mathcal{D}_1$, concludes ψ, and φ is $\chi \to \psi$. The bar removes χ from the open hypotheses, so $\mathrm{OH}(\mathcal{D}_1) \subseteq \Gamma \cup \{\chi\}$. By the induction hypothesis ψ has a Fitch proof from $\Gamma \cup \{\chi\}$. In a box with head Γ, open a subbox with supposition χ, copy that proof into it, close the subbox, and add the line $\chi \to \psi$, by ($\to$I).
- *($\vee$E).* The same idea: first a Fitch proof of the disjunction from Γ, then one subbox for each branch, with the branch's supposition, and finally the conclusion, by ($\vee$E).

Right to left. Take a Fitch proof from Γ, and write Δ_k for the context of line k: the head of the outer box together with the suppositions of the subboxes around line k. Going down the page, we build for each line k a tree with conclusion φ_k and open hypotheses inside Δ_k.

- *A (hyp) line.* It gets the one-node tree φ_k, and $\varphi_k \in \Delta_k$.
- *Any other line.* It cites earlier lines and subboxes, which already have trees. Put those trees above a bar for the rule that justifies line k. Two

facts about scope keep the open hypotheses inside Δ_k. A line i in scope at line k stands in no box that does not also contain line k, so $\Delta_i \subseteq \Delta_k$. The last line j of a subbox in scope at line k, with supposition χ, has $\Delta_j \subseteq \Delta_k \cup \{\chi\}$, and the bar removes χ again: for ($\to$I) from the one subbox, for ($\vee$E) from each branch its own supposition.

The last line of the proof has context Γ, so its tree witnesses $\Gamma \vdash_{\mathrm{NJ}} \varphi$.

Both constructions add one line for each bar, or one bar for each line, under the same rule, which is the last part of the theorem. □

Programs can be written in the boxes too, and then (abs) is a box inside a box. To type $\lambda x.\, M$:

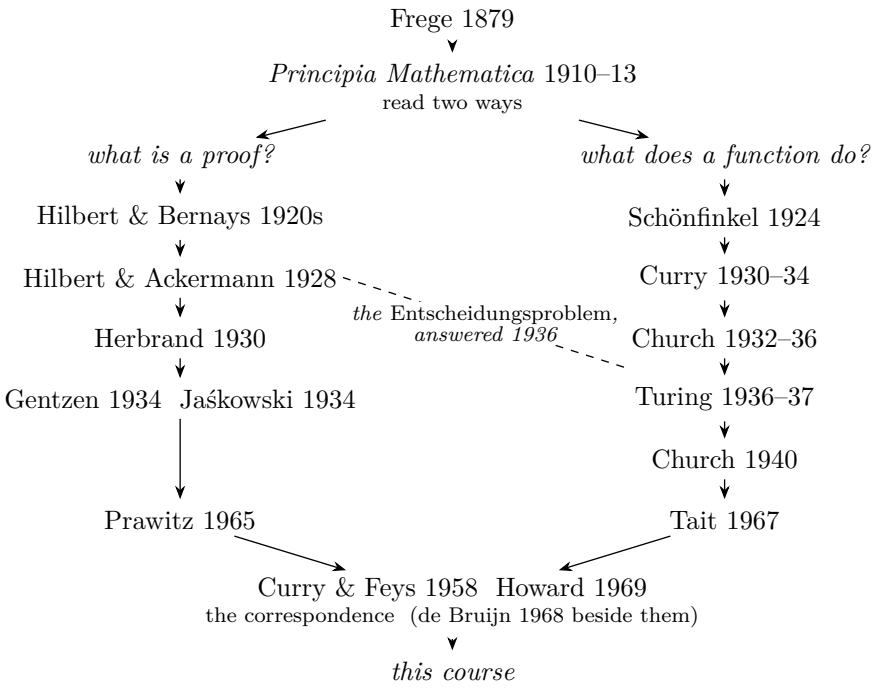
1. open an inner box whose declaration is $x : A$;
2. type the body M inside it;
3. close the box.

The line after it, $\lambda x.\, M : A \to B$, is written *outside* the box, because outside the box x is not in scope. A line may use a declaration from its own box or from each box around it, and nothing written inside a box that has been closed. That's the scope rule of a programming language, and it is why proof assistants display proofs in this way.

For typing, though, these notes use the trees of Figure 3.3. In a box the context of a line is not written on the line: it has to be read off the boxes around it, and a term such as $f\,x$ inside a box has a free variable whose declaration is somewhere above. In sequent style every judgement carries its whole context, and each step can be checked without looking anywhere else.

Trees, sequent-style trees and boxes are three notations for one object, and the theorem is the licence to move between them without comment. We draw trees when the shape of an argument is the point, and boxes when a proof is long and best read from top to bottom.

History. Three chapters have told this history in pieces, which is the right way to read it and the wrong way to remember it. It has nearly twenty people in it, Curry and Church twice each, doing the same things under different names. As one picture it's a diamond, and it converges.



At the top is one book, read two ways. Hilbert read *Principia* and saw that a proof could be written down as a list and checked; so proof theory, the Deduction Theorem, and Gentzen's rebuilding of the list as a tree. Schönfinkel, Curry and Church read it and saw mechanical instructions for rewriting formulas, above all for handling variables, and asked what a function does when it is applied; so the combinators, the λ, and, once Turing had asked the question in his own way, the theory of computation.

On the left edge the work was done in Göttingen. Gerhard Gentzen wrote his thesis, *Untersuchungen über das logische Schließen* (Gentzen, 1934), there in 1932–33, under Paul Bernays, who had been Hilbert's collaborator on proof theory since 1917 and was writing the *Grundlagen der Mathematik* with him. In April 1933 Bernays was dismissed from the university under the new civil-service law, because he was a Jew; he left for Zürich the next year, and the thesis was examined by Hermann Weyl, who was himself about to leave. Gentzen had been set Hilbert's task, to prove theorems *about* proofs, and the trees of NJ were built for it.

His main theorem, the *Hauptsatz*, says that detours can be removed, and the same introduction as the passage quoted at the start of the chapter puts it this way:[20]

> The *Hauptsatz* says that every purely logical proof can be reduced to a definite, though not unique, normal form. Perhaps we may express the essential properties of such a normal proof by saying: it is not roundabout. No concepts enter into the proof other than those contained in its final result, and they are necessarily used in arriving at that result.

[20]Gentzen (1934, p. 69). His word is *Umweg*, a detour; *Hauptsatz* means the main theorem.

He could prove that every derivation can be made normal for NJ, but not for its classical version. To cover classical logic he invented a third presentation of the rules, the sequent calculus, and proved the *Hauptsatz* there. That proof covered the intuitionistic case as well, so he left the proof for NJ out of his thesis. In 2005 Jan von Plato found, among Bernays's papers in Zürich, a handwritten version of the thesis that contains a detailed proof of normalisation for NJ, in the form that later became standard (von Plato, 2008): Gentzen had the proof and set it aside, since the sequent calculus handled classical and intuitionistic logic at once. The proof for NJ that everyone learned was published by Dag Prawitz in 1965, in the book that made natural deduction a subject again.

From 1935 Gentzen was Hilbert's assistant, and in 1936 he proved the consistency of arithmetic, the result Chapter 12 comes back to. He went on writing to Bernays in Zürich for the rest of his life — drafts, the consistency proof, the ordinal that measures it — in letters that discuss nothing but mathematics, through the years in which he joined the SA and then the Party and in which Bernays could not have come back. The letters are in Menzler-Trott (2007), and I leave the reader to draw the conclusion.[21]

Natural deduction was invented twice in 1934. Stanisław Jaśkowski was in Warsaw, among Łukasiewicz's students, and the complaint of Łukasiewicz's seminar that Chapter 2's History records, that a logic ought to let one suppose, was his starting point. He showed a system to the First Polish Mathematical Congress in 1927, though the published proceedings record the talk by title only, and printed it, in English, in the first volume of *Studia Logica* (Jaśkowski, 1934). Gentzen wrote his proofs as trees and Jaśkowski wrote his in boxes; Gentzen's thesis appeared in the *Mathematische Zeitschrift*. The two men worked in different countries, wrote for different journals and moved in different circles, and were "never in contact with one another", in the words of the standard history (Pelletier, 2000).

The war then scattered the Warsaw school, and Jaśkowski's paper reached the wider world only afterwards, through textbooks. Quine's *Methods of Logic* set out its proofs in a form close to his in 1950, and Fitch's book of 1952, whose preface credits both Gentzen and Jaśkowski, made the boxes standard. By then Gentzen had been dead for seven years. So nobody chose between the tree and the box. Two people answered the same question separately, and Theorem 3.52 says that their answers agree.

On the right edge the combinators came first. Schönfinkel had them in 1924 (Schönfinkel, 1924), and Curry, finding them again in the 1920s, built a logic on them in his thesis of 1929 (Curry, 1930). In 1932 Alonzo Church, a Princeton mathematician not yet thirty, published a logic meant to carry

[21] Gentzen taught in Prague from 1943, was arrested there with the rest of the German faculty in May 1945, and died in the prison on 4 August 1945, of starvation; he is said to have gone on working, in his cell, on the consistency of analysis.

the whole of mathematics, built, like Curry's, from functions rather than from sets.[22]

Curry had done away with variables by building everything from constants. Church kept them, but only one kind: every variable in a formula had to be bound by a λ, and the free variable, the x left dangling in $x+1 = 1+x$, was to go. His paper calls this "the complete abandonment of the free variable", and a page later it states the spirit in which the whole system was offered:[23]

> We do not attach any character of uniqueness or absolute truth to any particular system of logic. The entities of formal logic are abstractions, invented because of their use in describing and systematizing facts of experience or observation ...

Just as well he didn't. In 1935 his own students, Kleene and Rosser, showed both his logic and Curry's inconsistent (Kleene and Rosser, 1935), by the self-application of the Monster. The repair was the same in both cases: types. Curry gave his combinators types in 1934 (Curry, 1934), and Church gave his calculus types in 1940 (Church, 1940).

What survived of Church's logic was the part that handled variables, the calculus of §3.5, and Church and his students went on to show that it could compute anything that can be computed at all. In 1936 Church proposed that "computable" should *mean* "definable by a λ-term", and used that definition to show that no procedure decides whether two terms have the same normal form (Church, 1936b). A two-page note followed (Church, 1936a): no procedure decides which formulas of the logic with *for all* and *there is* are provable. That was the *Entscheidungsproblem* of Chapter 2's History, answered. Turing, in Cambridge, had reached the same answer by way of his machines without knowing of Church's. That September he arrived at Princeton to study with Church, and in 1937 he proved that his machines and Church's terms compute the same functions (Turing, 1937a). Curry and his students showed the combinators equivalent to both.[24]

---

[22]Church was born in Washington in 1903 and spent almost his whole working life at two universities: Princeton, as student and then professor, until 1967, and then UCLA, where he taught until 1990, when he was eighty-seven. A childhood accident with an air gun had left him blind in one eye; his blackboard was nonetheless remembered as immaculate. He was exceedingly polite and exceedingly careful. He protected papers he valued with a coat of Duco cement, a clear household glue, and he read science-fiction magazines and wrote to their editors when they got something wrong. He was a founder of the *Journal of Symbolic Logic* in 1936 and edited its reviews until 1979, and he supervised thirty-one doctorates. Among them were Kleene, Rosser and Turing; Martin Davis; Michael Rabin and Dana Scott, who shared a Turing Award, a prize named after an earlier student of their supervisor; and Raymond Smullyan, of the mockingbirds, who had once performed in Chicago nightclubs as a stage magician called Five-Ace Merrill.

[23]Church (1932, pp. 347–348).

[24]Turing's doctoral thesis was written under Church (1938), and Church's two earlier students, Kleene and Rosser, had done most of the work of showing that the calculus could compute anything at all. Gödel, in Princeton at the time, was not persuaded that λ-definability captured "computable" until he read Turing's paper, and then was. The name "Turing machine" is Church's, from his review of Turing's paper (Church, 1937); Turing had called them computing machines.

So the two edges first touch in 1936, at the dashed strut in the picture: a question asked on the left, by Hilbert and Ackermann, answered on the right, by Church and Turing. They meet at the bottom, when Curry noticed that the types of his combinators were Hilbert's axioms, and Howard, thirty-five years later, that the typing rules for λ, pairs and unions were Gentzen's, and that running was normalising. Below that is this course, which has walked the diamond backwards.

The correspondence came with two notations, because its two halves were worked out by different people. A type theorist writes $A \times B$ for the type of pairs and $A + B$ for the type of tagged values, because a pair type has as many values as the product of the numbers of values of its halves, and a tagged type as many as their sum; hence *product* and *sum* types. A logician writes $A \wedge B$ and $A \vee B$, because those are the words. Nobody chose one notation for both, and most writing about programming languages uses the first; these notes use the second throughout, until Chapter 4 needs the type theorists' letters for its quantifiers.

Four pairs of names are confused more often than not, and one sentence each will settle them.

- *Curry and Church.* built two calculi for the same doomed logic, one with constants and one with λ; one paradox broke both, and each repaired his own with types, Curry in 1934 and Church in 1940.
- *Church and Turing.* gave two definitions of "computable" that are one theorem, and the thesis named after both is the claim that the theorem is about everything.[25]
- *Gentzen and Jaśkowski.* invented "suppose" in the same year without knowing of each other, one as a tree and one as a box.
- *Curry and Howard.* saw the two halves of one correspondence thirty-five years apart, Curry for the Hilbert system and the combinators, Howard for NJ and the λ-calculus — and Howard saw also that running was normalising, which Prawitz and Tait had just proved on the two sides, two years apart, with one proof.

Figure 3.5 collects the dates.

Summary. NJ is ordinary reasoning written as rules: nine of them, for proving each connective and for using it, and no way to start except by supposing. Its derivations are trees, built by combining smaller trees, and a bracket round a leaf is a receipt for a supposition paid off. NJ proves exactly what the Hilbert system proves (Theorem 3.14): each rule is a fact about Hilbert proofs, and the whole cost of the Deduction Theorem is the cost of discharging.

Decorated with programs, a derivation is a typing in the **simply typed λ-calculus** (Theorem 3.15):

- a hypothesis is a variable;
- $\to$I is λ;
- $\to$E is application;

[25] Neither of them named it. Kleene called it "Church's thesis" in 1952, and added Turing's name in 1967.

Who	**When**	**What**
Frege	1879	proofs as gap-free chains; axioms (I), (II)
Whitehead & Russell	1910–13	the whole of mathematics, by rule
Hilbert & Bernays	1922–	proofs as objects; proof theory
Schönfinkel	1924	K and S suffice
Hilbert & Ackermann	1928	the textbook; the *Entscheidungsproblem*
Herbrand	1930	the Deduction Theorem
Curry	1930–34	combinatory logic; the types of K, S
Church	1932–36	the λ-calculus; an unsolvable problem
Gentzen, Jaśkowski	1934	natural deduction; the Hauptsatz
Turing	1936–37	the machine; equivalence with λ
Church	1940	the simply typed λ-calculus
Curry & Feys	1958	types of combinators are axioms, in print
Prawitz; Tait	1965; 1967	normalisation, both sides
Howard	1969	the correspondence for λ and NJ

FIGURE 3.5. Natural deduction and the λ-calculus: who did what.

- the other connectives are your data types, their introduction rules the constructors and their elimination rules the destructors.

The untyped calculus compiles to K and S by the theorem's own construction and computes whatever a Turing machine computes.

Running a program is removing detours from a proof.

- What the checker said survives the run (subject reduction);
- every typed program stops (proved here by counting redexes, except for $\vee$);
- a normal derivation is what you look for when you look for one: introductions upwards from the goal, eliminations downwards from the hypotheses.

A stopped closed program is a constructor, so the logic proves no letter and no $\bot$ — the theorem Gentzen was hired to prove, by the route Howard noticed thirty-five years later — and a proof of *or* always says which.

Safe is not correct. Nothing we care about — a sorted list, a transitive order, a discount on every day of the sale — is proved from *no* hypotheses. By the consistency theorem, nothing with a bare claim at the top ever is. Such things are proved from a *specification*, a set of sentences about lists, orders or sales taken as hypotheses. And a specification says *for all* — every payment, not one customer's; every day of the sale — which no formula of this chapter can say.

Exercises.

Exercise 3.1. Give a derivation of each of the following, and say what it witnesses.

(a) $\vdash_{\mathrm{NJ}} \neg(p \wedge \neg p)$
(b) $\vdash_{\mathrm{NJ}} (p \to \neg p) \to \neg p$
(c) $\vdash_{\mathrm{NJ}} (p \to q) \wedge (p \to \neg q) \to \neg p$
(d) $\neg p \vdash_{\mathrm{NJ}} p \to q$
(e) $p \vdash_{\mathrm{NJ}} \neg(\neg p \wedge q)$
(f) $\vdash_{\mathrm{NJ}} (p \vee q \to r) \to (p \to r) \wedge (q \to r)$
(g) $\vdash_{\mathrm{NJ}} (p \wedge q) \vee r \to (p \vee r) \wedge (q \vee r)$
(h) $\vdash_{\mathrm{NJ}} (p \vee r) \wedge (q \vee r) \to (p \wedge q) \vee r$
(i) $\vdash_{\mathrm{NJ}} \neg((p \to \neg p) \wedge (\neg p \to p))$, that is, "$p$ is not equivalent to its own negation". (Harder.)

Then try each of $\neg\neg p \to p$, $((p \to q) \to p) \to p$, $(\neg q \to \neg p) \to (p \to q)$ and $p \vee \neg p$, and say where you get stuck. Keep the attempts: Section 3.6 explains why they stick, and Part 2 returns to all four.

Exercise 3.2. Draw derivations, in Gentzen's notation, of the schemas (fst) $\varphi \wedge \psi \to \varphi$, (either) $(\varphi \to \chi) \to (\psi \to \chi) \to \varphi \vee \psi \to \chi$ and (absurd) $\bot \to \varphi$ with no open hypotheses, and compute OH for each by Definition 3.3. Then decorate the derivation of (either) with a program, and say which Haskell function it is.

Exercise 3.3. An online order is collected in store or delivered. A shop's rules say that a collected order is marked complete, and that a delivered order is marked complete once the courier has confirmed it, which the courier has. Choose letters, draw the derivation that the order is marked complete, and compute its open hypotheses. Then give the program.

Exercise 3.4. Type $\lambda x.\, \lambda y.\, x$ and $\lambda x.\, \lambda y.\, \lambda z.\, x\, z\, (y\, z)$ again, this time by (var), (app) and (abs) with the contexts written out, and confirm that the types are $A \to B \to A$ and $(A \to B \to C) \to (A \to B) \to A \to C$, the ones Chapter 1 gave K and S. Then run $(\lambda x.\, \lambda y.\, \lambda z.\, x\, z\, (y\, z))\, M\, N\, P$ and count the β steps.

Exercise 3.5. Give derivations of $(p \to q \to r) \to q \to p \to r$ and of $(p \wedge q \to r) \to p \to q \to r$, as trees, and then write the program each one is.

Exercise 3.6. Find a closed program of type $\neg\neg(A \vee \neg A)$, and give the derivation it is, a proof of $\neg\neg(A \vee \neg A)$.

Exercise 3.7. Find a closed program of type $A \vee B \to \neg(\neg A \wedge \neg B)$, and say which formula it proves.

Exercise 3.8. Try to find a closed program of type $\neg\neg p \to p$, for a letter p. Say where every attempt sticks. Keep your attempt; Part 2 returns to this formula.

Exercise 3.9. The proof that NJ and the Hilbert system agree turns Chapter 2's Hilbert proof of hypothetical syllogism, $(q \to r) \to (p \to q) \to p \to r$, into a derivation; decorated, it is the program $\mathsf{S}\, (\mathsf{K}\, \mathsf{S})\, \mathsf{K}$ with S and K written

as λ-terms. That derivation has detours. Run the program to normal form and compare it with the composition function $\lambda f.\, \lambda g.\, \lambda x.\ f\,(g\,x)$.

Exercise 3.10. Proposition 3.28 needs the types. Give a term in which no step can be taken but which is not normal, and say which typing rule it breaks. Then give one whose destructor is *absurd*.

Exercise 3.11. Run the term $(\lambda x.\ x\,x)\,(\lambda x.\ x\,x)$ for three steps. Then run $(\lambda x.\, \lambda y.\ x)\,y$, first blindly and then renaming the bound y first, and say which answer is the one meant.

Exercise 3.12. Translate $\lambda f.\, \lambda x.\ f\,(f\,x)$, the numeral $\overline{2}$, into combinators by the three lines of $\langle x\rangle P$, innermost λ first, and check that the result applied to g and a runs to $g\,(g\,a)$.

References.

Henk Barendregt. The impact of the lambda calculus in logic and computer science. *Bulletin of Symbolic Logic*, 3(2):181–215, 1997.

Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 1936a.

Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936b.

Alonzo Church. Review of A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*. *Journal of Symbolic Logic*, 2(1):42–43, 1937.

Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

Haskell B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930.

Haskell B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the USA*, 20(11):584–590, 1934.

Frederic B. Fitch. *Symbolic Logic: An Introduction*. Ronald Press, New York, 1952.

Robin O. Gandy. An early proof of normalization by A. M. Turing. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 453–455. Academic Press, London, 1980.

Gerhard Gentzen. Untersuchungen über das logische Schließen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934. Appeared 1934–35; English translation in *The Collected Papers of Gerhard Gentzen*, ed. M. E. Szabo, North-Holland, 1969.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989. Free online.

Georges Gonthier. Formal proof—the four-color theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008.

Thomas Hales et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.

William A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, London, 1980. Written in 1969.

Stanisław Jaśkowski. On the rules of suppositions in formal logic. *Studia Logica*, 1: 5–32, 1934.

Stephen C. Kleene and J. Barkley Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36(3):630–636, 1935.

Gerwin Klein et al. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 207–220, 2009.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

Paul Lorenzen. *Einführung in die operative Logik und Mathematik*. Springer, Berlin, 1955.

John McCarthy. History of LISP. *ACM SIGPLAN Notices*, 13(8):217–223, 1978.

Eckart Menzler-Trott. *Logic's Lost Genius: The Life of Gerhard Gentzen*, volume 33 of *History of Mathematics*. American Mathematical Society and London Mathematical Society, 2007. Translated by Craig Smoryński and Edward Griffor.

Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

Francis Jeffry Pelletier. A history of natural deduction and elementary logic textbooks. In John Woods and Bryson Brown, editors, *Logical Consequence: Rival Approaches*, volume 1, pages 105–138. Hermes Science, Oxford, 2000.

Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almqvist & Wiksell, Stockholm, 1965.

J. Barkley Rosser. Highlights of the history of the lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.

William W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

Masako Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118(1):120–127, 1995.

Alan M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2(4): 153–163, 1937a.

Alan M. Turing. The p-function in λ-K-conversion. *Journal of Symbolic Logic*, 2 (4):164, 1937b.

Dirk van Dalen. *Logic and Structure*. Springer, London, fifth edition, 2013.

Jan von Plato. Gentzen's proof of normalization for natural deduction. *Bulletin of Symbolic Logic*, 14(2):240–257, 2008.