COMP0009 Logic

Lecture notes · 2026–27

Part 1: Type Theory

Chapter 2

Hilbert Proofs

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

2. Hilbert Proofs 1
2.1. Formulas, axioms, and proofs 2
2.2. The Deduction Theorem 7
2.3. Proofs are programs 10
2.4. More connectives: and, or, and not 15
History 23
Exercises 27

2. Hilbert Proofs

You come home after dark and the windows are black, and before the key is out of the lock you've decided that nobody is in. That's reasoning. We all do it, all day, and mostly without noticing: the doctor goes from the rash to the illness, the mechanic from the rattle to the part, the reader of a detective story from the mud on the boots to the butler. What interests me in this chapter is writing reasoning down so completely that somebody who understood none of it could still check it.

Here is the walk from the dark windows to the dog, written out that way:

1. The lights are off.	seen
2. If the lights are off, nobody is home.	known
3. So nobody is home.	from 2 and 1
4. If nobody is home, the dog has not been fed.	known
5. So the dog has not been fed.	from 4 and 3

Three of the lines are starting points, things seen or already known. The other two each follow from two earlier lines by a single move, used twice: from "if this then that" and "this", conclude "that". That's the whole of it, and, give or take some notation, it's what the mathematician David Hilbert said a proof was.[1]

Why write reasoning out so completely? Because reasoning goes wrong in ways that are hard to see from the inside. Around 1900 mathematicians set out to write down the principles of mathematical reasoning itself, and found that principles which looked harmless led to contradictions.[2] Rules agreed in advance don't prevent such a thing by themselves, but they turn every argument into something anyone can check. A bank works the same way. A cashier who pays out a large cheque doesn't judge whether the payment is wise: she checks that it carries two signatures from the list the bank holds, and if it does, the cheque is paid. What is striking about Hilbert's proposal is how little it asks of whoever does the checking, and so how easy the checking is.

To check the walk to the dog you don't need to know what a dog is, or that it needs feeding. You need only see that line 3 is the right-hand half of line 2 and that line 1 is its left-hand half. Let's call whoever does the checking, knowing nothing of the subject, *the clerk*.[3] The clerk understands nothing and checks everything. They can see whether a line has one of a few permitted shapes, and they can compare two lines symbol by symbol,

[1] How he came to say it is told at the end of the chapter (§2.4).

[2] The best known is Russell's paradox, which broke Frege's system in 1902 (§2.4).

[3] Turing, in 1936, didn't describe a machine; he described a person (Turing, 1937): a clerk with a pencil, a supply of squared paper, a finite alphabet of symbols and a small number of states of mind, who at each moment looks at one square and, according to what is there and the state he is in, writes a symbol or rubs one out, moves one square left or right, and changes state. That's all the clerk may do, and Turing claimed that whatever a human being can calculate, a clerk so restricted can calculate. The machine that bears his name is the clerk with the person taken out; in 1936 a *computer* was a job, a person, often a woman, paid to do arithmetic to a fixed procedure in a calculating office, and Turing's paper uses the word in that sense throughout. Emil Post, the same year and independently, described a "worker" moving along a row of boxes, marking and unmarking them (Post, 1936).

but they have no idea what any of the symbols mean, and need none. The clerk could just as well be a machine.

2.1. **Formulas, axioms, and proofs.** Written out in full, reasoning is a sequence of sentences, each step taken under a fixed discipline. English is a poor medium for this, because its grammar and its idioms hide the logical point. So we work instead with *propositions*, the thoughts that sentences express, and keep only their logical connections. A lease might say: "Should the tenant fail to pay the rent in full by the first day of the month, the landlord shall be entitled to retain the deposit." For all its length, the clause relates two propositions: if the rent is late, then the landlord may keep the deposit. Write p for the first and q for the second, and the clause is $p \to q$. Formulas such as $p \to q$ are given by a small grammar.

Definition 2.1 (formulas)**.** Fix a stock of *propositional variables* $p, q, r, \ldots$, called *letters* for short. The *formulas* are defined inductively:

1. every letter is a formula;
2. if φ and ψ are formulas, so is $\varphi \to \psi$, read "if φ then ψ", or "φ implies ψ";
3. nothing else is a formula.

In one line,

$$\varphi, \psi \ ::= \ p \mid \varphi \to \psi,$$

with the arrow grouping to the right, so that $\varphi \to \psi \to \chi$ is $\varphi \to (\psi \to \chi)$.[4]

Example 2.2 (Hughes: the release bot)**.** Hughes has just become a maintainer of an open-source library, one of those that half the internet depends on without knowing it. Her first job is to set up the release bot, the program that publishes new versions to the package registry. The project's policy document, MAINTAINERS.md, lists a maintainer called Smith, and says two more things: that "every maintainer has write access to the repository", and that "anyone with write access may publish a release".

On her first Monday two requests arrive. Smith asks for version 2.3 to be released. A contributor called Jones asks the bot to rotate the signing key, which only a maintainer may do.

Five propositions matter, and each gets a letter:

- p: Smith is a maintainer;
- q: Smith has write access;
- r: Smith may publish a release;
- j: Jones is a maintainer;
- k: Jones has write access.

The policy, said of Smith, is the two formulas $p \to q$ and $q \to r$, and MAINTAINERS.md adds the letter p. Said of Jones, the first sentence of the policy is $j \to k$.

[4] Łukasiewicz found the brackets so annoying that in 1924 he got rid of them: write the connective *first*, Cpq for $p \to q$, and $p \to (q \to r)$ and $(p \to q) \to r$ become $CpCqr$ and $CCpqr$, with nothing left to group. Hewlett-Packard built the idea, back to front, into its calculators from 1972 on, and an HP calculator still has no bracket keys.

Formulas can say more than the policy does. By the grouping convention, $p \to q \to r$ is $p \to (q \to r)$: if Smith is a maintainer, then, if he has write access, he may publish. The formula $(p \to q) \to r$, with its brackets, says something else: if being a maintainer brings Smith write access, then he may publish.

What do we mean by "if ..., then ..."? At the least this: whenever the *if* part holds, the *then* part may be concluded. That is a rule, and we write it with a bar, as the rules of Definition 1.2 were written:

$$\frac{\varphi \to \psi \qquad \varphi}{\psi}\ \text{MP}$$

Whenever what is above the bar holds, so does what is below: from $\varphi \to \psi$ and φ, conclude ψ. The implication is the left premiss. The rule's Latin name is *modus ponens*, the mood that affirms.[5]

Is that all there is to say about implication? No. Some implications hold whatever we have been told. "If the lights are off, then the lights are off" is true, and nobody should need telling. If such things are to come out of the logic rather than be put into it, the rule needs starting points to work on, and two will do. Each is written with the Greek letters φ, ψ, χ standing for arbitrary formulas, so that each is a shape shared by infinitely many axioms.

The first says that a truth is implied by anything: if φ holds, then φ holds whatever else, say ψ, does.

$$\varphi \to \psi \to \varphi$$

The second is modus ponens carried out *under a hypothesis*: if, given φ, you have $\psi \to \chi$, and, given φ, you have ψ, then, given φ, you have χ.

$$(\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$$

A reader of Chapter 1 may recognise these shapes. That is no accident, and §2.3 says why. Why these two are the right ones is the business of §2.2, and where they came from is told in §2.4.

Definition 2.3 (axioms)**.** The *axioms* are all instances of the two schemas

$$\text{(I)} \quad \varphi \to \psi \to \varphi$$
$$\text{(II)} \quad (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$$

where φ, ψ and χ stand for arbitrary formulas.

A schema stands for infinitely many axioms, all of one shape. Both of

$$p \to q \to p \qquad \text{and} \qquad (r \to s) \to p \to (r \to s)$$

are instances of (I), the first with p and q for φ and ψ, the second with $r \to s$ and p, and a simple parser can test whether a given formula matches one of the two shapes.

A proof is then something the clerk can check one line at a time: each line is an axiom, or one of the things we were told, or follows from two earlier lines by modus ponens.

[5]The name is medieval; the rule is a good deal older. In the third century BC the Stoics, Chrysippus above all, listed five argument shapes that need no justification, the *indemonstrables*, and the first of them is this one: "if it is day, it is light; it is day; so it is light". The fifth, which a dog is supposed to be able to manage, turns up in Exercise 2.6.

Definition 2.4 (proof from hypotheses)**.** Let Γ be a set of formulas, the *hypotheses*. A *proof of* φ *from* Γ is a finite sequence of formulas

$$\varphi_1,\ \varphi_2,\ \ldots,\ \varphi_n$$

whose last formula φ_n is φ, and in which each formula φ_k is justified in one of three ways:

1. (*axiom*) φ_k is an axiom;
2. (*hypothesis*) φ_k is a member of Γ;
3. (*modus ponens*) there are two earlier formulas φ_i and φ_j, with $i, j < k$, such that φ_i is the formula $\varphi_j \to \varphi_k$.

To check a modus ponens the clerk finds the two formulas it cites and confirms, symbol for symbol, that one of them reads ⟨the other⟩ $\to$ ⟨the new formula⟩. That is the whole job.[6]

Most of the time what matters about a proof is only that there is one, and that claim gets a notation of its own.

Definition 2.5 (provability; theorem)**.** Let Γ be a set of formulas and φ a formula. If there is a proof of φ from Γ, we write $\Gamma \vdash_{\mathrm{HJ}} \varphi$, read "$\varphi$ is provable from Γ", and say that the proof *witnesses* $\Gamma \vdash_{\mathrm{HJ}} \varphi$. A formula φ with $\varnothing \vdash_{\mathrm{HJ}} \varphi$, written $\vdash_{\mathrm{HJ}} \varphi$, is a *theorem*.

The subscript names the system, HJ: H for Hilbert, and a J that Part 2 explains.

We set a proof out as a box, with the hypotheses Γ above the line and the formulas below it, one to a numbered line, each with its justification on the right:

	Γ	
1	φ_1	*reason*
2	φ_2	*reason*
	$\vdots$	
n	φ_n	*reason*

The box keeps in view everything the clerk needs: the hypotheses, the order of the formulas, and the reason for each. A line got by modus ponens gives its reason as "MP i, j", citing the implication first. So an argument whose every step is the one move can be written as a box, and an argument with a bad step is caught at that step.

Example 2.6 (Hughes, cont'd: two boxes)**.** Hughes doesn't grant Smith's request on a hunch. She reasons from what MAINTAINERS.md told her:

1. Smith is a maintainer. told
2. If Smith is a maintainer, Smith has write access. told

[6] Why is modus ponens a rule, something the clerk does, rather than a third shape on the list? Lewis Carroll answered in 1895 with a tortoise who grants φ and $\varphi \to \psi$ but won't grant ψ until "if φ and $\varphi \to \psi$, then ψ" has been written down as a further premiss; and, that granted, asks for the next; and never stops. Achilles fills a notebook. The clerk escapes the tortoise because nothing on the list persuades them of anything: a rule isn't something they're told.

3. So Smith has write access. from 2 and 1
4. If Smith has write access, Smith may publish a release. told
5. So Smith may publish a release. from 4 and 3

and lets the bot go ahead.

In the letters of Example 2.2, with "hypothesis" for "told" and "MP" for "from", her argument is a proof in the sense of Definition 2.4:

	$p,\ p \to q,\ q \to r$	
1	p	hypothesis
2	$p \to q$	hypothesis
3	q	MP 2, 1
4	$q \to r$	hypothesis
5	r	MP 4, 3

It witnesses $\{p,\ p \to q,\ q \to r\} \vdash_{\mathrm{HJ}} r$.

Jones has an argument ready. Maintainers have write access, he says; I have write access; so I am a maintainer. Hughes is fairly sure something is wrong with that, and the box says what:

	$j \to k,\ k$		
1	$j \to k$	hypothesis	
2	k	hypothesis	
3	j	MP 1, 2	*refused*

Line 3 claims to be a modus ponens on lines 1 and 2, which would need line 1 to read $\langle \text{line } 2\rangle \to j$, that is, $k \to j$. It reads $j \to k$. So the clause for modus ponens doesn't apply, and no other clause puts j on a line.

Nothing in that refusal depended on what j and k stand for. Jones's mistake is a mistake of *shape*.

Proofs in a system like this are not always easy to find. Modus ponens says how to *use* an implication, but nothing in the system says directly how to *prove* one; and some implications, such as "if the lights are off, then the lights are off", ought to come for free. The simplest of them already takes five lines.

Lemma 2.7 (identity)**.** *For every formula φ, $\vdash_{\mathrm{HJ}} \varphi \to \varphi$.*

Proof. The following proof, from no hypotheses, witnesses the claim for a propositional letter p. The square brackets give the formulas put for the schema's Greek letters, in the order φ, ψ, χ.

	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

Lines 1, 2 and 4 are matched against the two shapes, and lines 3 and 5 against the two lines they cite. Nothing in the five lines depends on what p is, so the same lines with φ in place of p prove $\varphi \to \varphi$ for every formula φ. □

Remark (finding the first line). Each line of that proof is trivial to *check*. Line 1 is the hard one to *find*, and the finding is worth watching, because proofs are found backwards from the goal.[7] Take a letter p for φ. The goal $p \to p$ isn't an axiom, so it must come by modus ponens, from some χ and $\chi \to (p \to p)$.

- Could the second of these be an instance of (I)? Then χ would be p, and we would need a proof of p on its own, which there can't be: p is a bare letter, and nothing in the system proves a bare letter.
- So try (II). Its last part, $\varphi \to \chi$, must be $p \to p$, which fixes $\varphi = \chi = p$ and leaves ψ free; and both premisses, $p \to (\psi \to p)$ and $p \to \psi$, will be instances of (I) if we choose ψ to be $p \to p$.

That's line 1. The search took a paragraph; the check takes none.

Example 2.8 (Hughes, cont'd: filing a rule)**.** Hughes would like to file the rule that a maintainer may publish, $p \to r$, so that she never again has to go through the two steps for each maintainer. A person would suppose that Smith is a maintainer and conclude the rule from the policy. The clerk has no line for "suppose", so the implication has to be built from the axioms. The claim is $\{p \to q,\ q \to r\} \vdash_{\mathrm{HJ}} p \to r$, and here is a proof that witnesses it.

	$p \to q,\ q \to r$	
1	$q \to r$	hypothesis
2	$(q \to r) \to p \to (q \to r)$	(I) $[q \to r,\ p]$
3	$p \to (q \to r)$	MP 2, 1
4	$(p \to (q \to r)) \to (p \to q) \to p \to r$	(II) $[p,\ q,\ r]$
5	$(p \to q) \to p \to r$	MP 4, 3
6	$p \to q$	hypothesis
7	$p \to r$	MP 5, 6

Read the blocks, not the symbols.

- Lines 1–3 take the hypothesis $q \to r$ and weaken it, by (I), to $p \to (q \to r)$: a statement that says less, since it only promises $q \to r$ once p is given.
- Lines 4–5 put that through (II), which turns "given p, q implies r" into "if p gives q then p gives r".

[7] Including a machine's. In 1956 Newell, Simon and Shaw wrote the Logic Theorist, the first program anyone called artificial intelligence (1957); it searched backwards from the goal, as this paragraph does, and proved 38 of the first 52 theorems of *Principia*'s chapter on propositional logic. For one of them it found a shorter proof than Whitehead and Russell's. Simon sent it to Russell, who by Simon's account was delighted. The *Journal of Symbolic Logic* declined a paper on the proof, a new proof of an elementary theorem, reportedly without remarking that one of the authors was a program.

- Line 7 fires the result on the other hypothesis.

The rule that chains two implications in this way is called *hypothetical syllogism*.

Every Hilbert proof has this character: a few ideas, spelled out at a length the clerk can follow. The identity takes five lines and the chaining of two implications seven, each built on an instance of (II) that nobody would write unprompted. Yet both implications are plain to a person, who supposes the *if*, reaches the *then*, and is done. Making that move legitimate takes some work, and the work is the next section.

2.2. **The Deduction Theorem.** What is the difference between $\{\varphi\} \vdash_{\mathrm{HJ}} \psi$ and $\vdash_{\mathrm{HJ}} \varphi \to \psi$? The first says that there is a proof which takes φ as a hypothesis and ends with ψ. The second says that there is a proof, from no hypotheses at all, of the formula $\varphi \to \psi$. These are different claims about different lists. Yet ask what "φ implies ψ" means, in terms of reasoning, and most of us would answer with the first: assume φ, and you can deduce ψ. So the two ought to come to the same thing. The main result of this section, the Deduction Theorem, says that they do.

To get there we treat proofs as objects in their own right, things we can prove theorems *about*, as one proves theorems about numbers or triangles; that is what makes proof theory a subject.[8] Three easy facts come first.

Proposition 2.9 (structural facts)**.** *For all sets of formulas Γ and Δ and all formulas φ and ψ:*

1. *(*reflexivity*) if $\varphi \in \Gamma$ then $\Gamma \vdash_{\mathrm{HJ}} \varphi$;*
2. *(*monotonicity*) if $\Gamma \subseteq \Delta$ and $\Gamma \vdash_{\mathrm{HJ}} \varphi$ then $\Delta \vdash_{\mathrm{HJ}} \varphi$;*
3. *(*transitivity*) if $\Gamma \vdash_{\mathrm{HJ}} \varphi$ and $\Delta \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi$ then $\Gamma \cup \Delta \vdash_{\mathrm{HJ}} \psi$.*

Proof. In each case the job is to exhibit a proof of the required kind, built from the proofs we are given; nothing is said about what the formulas mean, only about lists.

(1) The one-line sequence φ is a proof of φ from Γ: its only line is justified as a hypothesis, because φ is a member of Γ.

(2) A proof of φ from Γ is already a proof of φ from Δ: every line justified as a member of Γ is a member of Δ too, because $\Gamma \subseteq \Delta$, and the other two kinds of line do not mention Γ.

(3) Write out the proof of φ from Γ, and after it the proof of ψ from $\Delta \cup \{\varphi\}$ with every line justified as "hypothesis φ" deleted; each modus ponens that cited one of the deleted lines now cites the last line of the first proof, which is φ. Every remaining hypothesis is in Γ or in Δ, so the result is a proof of ψ from $\Gamma \cup \Delta$.

Only the third item rewrites a proof; in the others the list we are given, or the one line φ, does the job as it stands. □

In words: what you were told, you may use; being told more never takes anything away; and a result proved once is as good as one you were given.

[8] The idea was Hilbert's (§2.4).

The third is the one to remember, because it says that proofs *compose*: a theorem, once proved, may be cited as a line in any later proof, with the proof spliced in by the clerk if anyone ever asks. From here on a line may be justified by the name of a fact already proved, and the box is still a proof.

Example 2.10 (Hughes, cont'd: two proofs spliced)**.** Hughes's argument of Example 2.6 can be built from two shorter ones. The first gets Smith his write access, and the second gets him from write access to publishing:

$$\{p,\ p \to q\} \vdash_{\mathrm{HJ}} q \qquad \text{and} \qquad \{q,\ q \to r\} \vdash_{\mathrm{HJ}} r.$$

Each is witnessed by a three-line proof:

	$p,\ p \to q$	
1	p	hypothesis
2	$p \to q$	hypothesis
3	q	MP 2, 1

and

	$q,\ q \to r$	
1	q	hypothesis
2	$q \to r$	hypothesis
3	r	MP 2, 1

Transitivity, with $\Gamma = \{p,\ p \to q\}$, $\Delta = \{q \to r\}$ and q for φ, gives $\{p,\ p \to q,\ q \to r\} \vdash_{\mathrm{HJ}} r$. Its proof says how to splice: write the second proof after the first, without its line "q, hypothesis", and let the modus ponens that cited that line cite line 3, the last line of the first proof. The result is the box of Example 2.6, line for line.

Example 2.11 (Hughes, cont'd: Monday, again)**.** Hughes has filed the rule of Example 2.8, and Monday's argument, five lines from the three things she was told, now takes three:

	$p,\ p \to q,\ q \to r$	
1	$p \to r$	hypothetical syllogism (Example 2.8)
2	p	hypothesis
3	r	MP 1, 2

Line 1 isn't an axiom and isn't a hypothesis; it is justified by a fact about proofs. Example 2.8 proves $p \to r$ from $\{p \to q,\ q \to r\}$, and by monotonicity that proof is also a proof from the three hypotheses here. Transitivity then lets the citation stand for the seven lines it names. Spliced in, the box has nine lines; nobody need write them.

The Deduction Theorem is a fact of the same kind, and a deeper one:

$$\Gamma \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi \quad \text{if and only if} \quad \Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi.$$

Read from left to right, as a licence, it says: to prove $\varphi \to \psi$, suppose φ, reach ψ, and then dispense with the supposition, or *discharge* it. It is about the most desirable property a proof system could have. It also looks, on the

face of it, far too good: the definition of a proof says nothing of the kind, and nothing in (I), (II) and modus ponens mentions discharging anything.

One direction is easy, and needs only modus ponens and transitivity. The other needs an algorithm: one that takes a proof of ψ from Γ and φ and turns it, line by line, into a proof of $\varphi \to \psi$ from Γ alone. The theorem says that the algorithm works, and the proof of the theorem is the algorithm.[9]

Theorem 2.12 (Deduction Theorem)**.** *For every set of formulas* Γ *and all formulas* φ *and* ψ*,*

$$\Gamma \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi \quad \Longleftrightarrow \quad \Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi.$$

Proof. Right to left. Suppose $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi$. A three-line proof witnesses $\{\varphi \to \psi,\ \varphi\} \vdash_{\mathrm{HJ}} \psi$:

	$\varphi \to \psi,\ \varphi$	
1	$\varphi \to \psi$	hypothesis
2	φ	hypothesis
3	ψ	MP 1, 2

Transitivity (Proposition 2.9), with $\varphi \to \psi$ for its φ and $\{\varphi\}$ for its Δ, gives $\Gamma \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi$.

Left to right. Let $\varphi_1, \ldots, \varphi_n$ be a proof of ψ from $\Gamma \cup \{\varphi\}$, so that φ_n is ψ. For each k from 1 to n, write $P(k)$ for the claim

$$\Gamma \vdash_{\mathrm{HJ}} \varphi \to \varphi_k.$$

We prove $P(k)$ for every k by induction on k. The theorem is $P(n)$.

By Definition 2.4, each line φ_k is an axiom, a hypothesis or a modus ponens, and a hypothesis is either a member of Γ or φ itself. Axioms and members of Γ are handled alike, so there are three kinds of line:

(a) φ_k is an axiom or a member of Γ;

(b) φ_k is φ itself;

(c) φ_k follows by modus ponens from two earlier lines.

Base case, $k = 1$. The first line has no earlier lines to cite, so it is of kind (a) or (b). The proofs of those two cases below use no earlier lines, so they cover the base case.

Inductive step. Let $k > 1$, and suppose, as the *induction hypothesis*, that $P(j)$ holds for every $j < k$: proofs from Γ of $\varphi \to \varphi_j$ have already been written for all the lines above line k. We show $P(k)$, one case for each kind of line.

- *(a)* φ_k *is an axiom or a member of* Γ. Then φ_k may stand as a line of a proof from Γ. Write

φ_k	
$\varphi_k \to \varphi \to \varphi_k$	an instance of (I), with φ_k and φ for its two letters
$\varphi \to \varphi_k$	MP on those two lines.

  Three lines, and the induction hypothesis isn't needed.

[9] The algorithm is due to Jacques Herbrand (§2.4).

- *(b) φ_k is φ itself.* Then $\varphi \to \varphi_k$ is $\varphi \to \varphi$, which Lemma 2.7 proves from no hypotheses at all, so in particular from Γ. Again the induction hypothesis isn't needed.
- *(c) φ_k follows by modus ponens.* Then there are earlier lines φ_j and φ_i, where φ_i is $\varphi_j \to \varphi_k$. By the induction hypothesis, applied to j and to i, proofs from Γ of $\varphi \to \varphi_j$ and of $\varphi \to \varphi_j \to \varphi_k$ have already been written. Write the instance of (II) with φ, φ_j and φ_k for its three letters, and fire it twice:

$(\varphi \to \varphi_j \to \varphi_k) \to (\varphi \to \varphi_j) \to \varphi \to \varphi_k$	(II)
$(\varphi \to \varphi_j) \to \varphi \to \varphi_k$	MP with the proof for i
$\varphi \to \varphi_k$	MP with the proof for j.

  Three lines, on top of the proofs already written.

So $P(k)$ holds for every k. Concatenating the pieces in order gives a proof from Γ whose last line is $\varphi \to \varphi_n$, that is, $\varphi \to \psi$. □

Look at what the proof used. Only two schemas were needed, and each case needed the one it used:

- case (a) took a line that didn't depend on the supposition and weakened it, by (I), to one that mentions the supposition anyway;
- case (c) distributed the supposition over the two premisses of a modus ponens, by (II);
- case (b) used the proof of $\varphi \to \varphi$ of Lemma 2.7, which is one (II) and two (I)s.

Nothing else was needed, and this is why we chose these two shapes. Other axioms would have done as well,[10] and from them one could derive (I) and (II) and then argue as in the proof of Theorem 2.12. We have taken the direct route: (I) and (II) are exactly what it takes to push a supposition into a formula, one line at a time.

Together with transitivity, the theorem gives a way of writing proofs that a person can find. To prove $\varphi \to \psi$ from Γ, prove ψ from Γ and φ, and discharge φ. To use a result already proved, cite it as a line, and let transitivity splice its proof in. The construction is not free: each line of the given proof becomes three lines, or five for the supposition itself, so each discharge, carried out, roughly triples the proof it is applied to.

The two shapes earned their place by doing a job. They also have a past, which the History tells,[11] and a second identity, which is the subject of the next section.

2.3. **Proofs are programs.** Look again at the types that schemas (K) and (S) of Definition 1.24 give K and S. They are the axioms (I) and (II), with

[10] Hilbert and Ackermann's textbook took *or* and *not* as basic, wrote $\varphi \to \psi$ for $\neg\varphi \vee \psi$, and started from four axioms of *Principia*: $\varphi \vee \varphi \to \varphi$, $\varphi \to \varphi \vee \psi$, $\varphi \vee \psi \to \psi \vee \varphi$ and $(\varphi \to \psi) \to (\chi \vee \varphi \to \chi \vee \psi)$. *Principia* had a fifth, which Bernays showed in 1926 to follow from the other four.

[11] Letter for letter, they are the first two of the starting points that Gottlob Frege gave for *if* in 1879, and modus ponens is his rule.

Latin letters for Greek.[12] The match calls for an explanation, and it doesn't stop at the axioms. The four rules for typing are the rules for proof, with the programs left off:

- schema (I) is the type of K with the K left off;
- schema (II) is the type of S with the S left off;
- a hypothesis is a declaration $x : A$ above the line with the x left off;
- modus ponens, from $\varphi \to \psi$ and φ conclude ψ, is (app), from $M : A \to B$ and $N : A$ conclude $M\,N : B$, with the M and the N left off.

Read the other way, the list says something stronger. A proof is a list of justified lines, and its justifications say how each line was got from the ones it cites. Fold the justifications up, writing K for each use of (I), S for each use of (II), a variable for each hypothesis and an application for each modus ponens, and the whole proof becomes a single term. That term is a program, and it is the proof in compact form: every justification is in it, in the order the proof used them. In this sense a well-typed program *is* a proof.

Take the typing of $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$ at $A \to A$ in Example 1.27 and rub out the programs.

- Line 1 loses its S and is the formula
$$(A \to (A \to A) \to A) \to (A \to (A \to A)) \to A \to A,$$
an instance of (II);
- line 3 loses its "(app) 1, 2" and gains "MP 1, 2";
- with φ for A the whole box is the proof of Lemma 2.7, line for line, down to the first line that nobody would write unprompted.

The identity function *is* the proof that p implies p.

Now fold it up the other way. Go down the five lines of the proof of Lemma 2.7 writing K beside each use of (I), S beside each use of (II), and an application beside each modus ponens, and when you reach the bottom you've written $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$.

This happens with every proof and every typing, for a reason that is almost embarrassingly simple: the two definitions are the same definition with a column added. Here is the dictionary.

Line of a typing	**Line of a proof**
$\mathsf{K} : B \to C \to B$, by (K)	$B \to C \to B$, an instance of (I)
$\mathsf{S} : (B \to C \to D) \to \cdots$, by (S)	$(B \to C \to D) \to \cdots$, an instance of (II)
$x : B$, by (var), $x : B$ being in Γ	B, a hypothesis, B being in $\lvert\Gamma\rvert$
$M\,N : C$, by (app) i, j	C, by MP i, j

Here Γ is a context and $\lvert\Gamma\rvert$ is the set of types declared in it, read as formulas: if Γ is $x_1 : A_1, \ldots, x_n : A_n$ then $\lvert\Gamma\rvert$ is $\{A_1, \ldots, A_n\}$. Read left to right, the table rubs programs out; read right to left, it writes them in.

[12] Curry noticed this in 1934 (Curry, 1934). He had studied the combinators to take substitution apart, and nothing in that purpose suggested proof; yet the types they had to have turned out to be Frege's first two axioms, the starting points of his account of proof (§2.4).

Theorem 2.13 (Curry's correspondence)**.** *Let* Γ *be a context and* $|\Gamma|$ *the set of types declared in it.*

*(i) (*Erasure.*) Take a typing that witnesses* $\Gamma \vdash_{\mathrm{KS}} M : A$*, delete the terms, and justify each line by (I), (II), "hypothesis" or "MP* i*,* j*" where the typing used (K), (S), (var) or "(app)* i*,* j*". The result is a proof of* A *from* $|\Gamma|$*.*

*(ii) (*Decoration.*) Take a proof of* φ *from* $|\Gamma|$*, where* Γ *declares a distinct variable for each hypothesis. Working downwards, write on each line a term:* K *on an instance of (I),* S *on an instance of (II), the declared variable on a hypothesis, and* $M_i\,M_j$ *on a line "MP* i*,* j*", where* M_i *and* M_j *are the terms on lines* i *and* j*. The result is a typing that witnesses* $\Gamma \vdash_{\mathrm{KS}} M : \varphi$*, where* M *is the term on the last line.*

In particular, $\Gamma \vdash_{\mathrm{KS}} M : A$ *for some term* M *if and only if* $|\Gamma| \vdash_{\mathrm{HJ}} A$*.*

Proof. All that needs checking is that each line's justification survives the change, and the table says it does; here are the four cases spelled out.

For (i), every line of the typing has the same context Γ (Definition 1.24), so take the four kinds of line in turn.

1. A line justified by (K) has the type $B \to C \to B$ for some B and C, which erased is an instance of (I).
2. A line justified by (S) has the type $(B \to C \to D) \to (B \to C) \to B \to D$, which erased is an instance of (II).
3. A line justified by (var) has a type listed in Γ, which erased is a member of $|\Gamma|$: a hypothesis.
4. A line justified by "(app) i, j" has type C, where line i has type $B \to C$ and line j has type B. Erased, line i is the formula $B \to C$ and line j is B, so "MP i, j" is a correct justification for C.

Each line of the result is justified, and the last is A.

For (ii), run the same four cases upwards.

- An instance of (I) or (II) is a type that (K) or (S) assigns, so the decorated line is justified by that rule.
- A hypothesis is in $|\Gamma|$, and Γ gives it a variable, so the decorated line is justified by (var).
- A line "MP i, j" concludes χ from $\psi \to \chi$ at line i and ψ at line j, which by then carry terms $M_i : \psi \to \chi$ and $M_j : \psi$, so $M_i\,M_j : \chi$ is justified by "(app) i, j".

□

I want to be careful about what the theorem says, because "analogy" is the wrong word for it. A typing (Definition 1.24) and a proof (Definition 2.4) are the same list, the one with an extra column; and the four rules the compiler was checking your code against are, column removed, the rules written down in 1879 for what it is to prove something (§2.4). The observation is now called the Curry–Howard correspondence.

Example 2.14 (Hughes, cont'd: the policy as types)**.** Monday's reasoning was done by hand. Hughes would rather the release bot did it, and a bot is

a program, so she writes the policy in the one language the compiler reads: types. Each of the three propositions becomes a type, and each of the two policy sentences a function.

```
smith            :: Maintainer                    -- MAINTAINERS.md
maintainersWrite :: Maintainer -> WriteAccess     -- policy, sentence 1
writeMayPublish  :: WriteAccess -> MayPublish     -- policy, sentence 2
publish          :: MayPublish -> Release -> IO ()
```

A value of type `Maintainer` is evidence that somebody is a maintainer, and `smith` is the evidence the MAINTAINERS file supplies. The two policy functions turn one piece of evidence into the next, and their bodies do nothing but that.

The point of the arrangement is the last line. `publish` doesn't take a yes or a no, which a careless workflow could skip asking for. It takes a value of type `MayPublish`, a *ticket*, and Hughes arranges that nothing in the program can make one except the policy functions, applied in the right order.[13]

To release 2.3, Hughes writes

```
publish (writeMayPublish (maintainersWrite smith)) rel
```

and the compiler types the ticket before anything runs. Here is its typing, a box as in Definition 1.24, with nothing abbreviated:

	`smith` : `Maintainer`	
	`maintainersWrite` : `Maintainer` → `WriteAccess`	
	`writeMayPublish` : `WriteAccess` → `MayPublish`	
1	`smith` : `Maintainer`	(var)
2	`maintainersWrite` : `Maintainer` → `WriteAccess`	(var)
3	`maintainersWrite smith` : `WriteAccess`	(app) 2, 1
4	`writeMayPublish` : `WriteAccess` → `MayPublish`	(var)
5	`writeMayPublish (maintainersWrite smith)` :`MayPublish`	(app) 4, 3

Now cover that column with your thumb, the programs and the colons, and read the types as sentences about Smith. What is left is Monday's reasoning, line for line:

1. a maintainer;
2. if a maintainer then write access;
3. so write access;
4. if write access then may publish;
5. so may publish.

Each (var) is a *told* and each (app) a *from*, so with p, q, r for `Maintainer`, `WriteAccess` and `MayPublish` this is the first box of Example 2.6 exactly.

The expression Hughes wrote to get Smith his ticket *is* the proof that Smith may have one. She set out to write a program, and the compiler, in accepting it, checked a proof.

[13] In Haskell she gets that by putting the three types in a module and exporting them without their constructors: outside the module the only way to make a `WriteAccess` is to apply `maintainersWrite` to a `Maintainer`.

The mistakes come out the same way. Write `publish smith rel`, skipping the policy, and the compiler stops: *couldn't match expected type 'MayPublish' with actual type 'Maintainer'*, the red line of Example 1.28. With the programs rubbed out, it is an attempt to conclude r from p with no implication to fire.

Jones has write access but isn't a maintainer, and he wants the bot to rotate the signing key: `rotateKey :: Maintainer -> IO ()`. What Hughes has for him is a `WriteAccess`, and nothing of type `WriteAccess -> Maintainer` exists, because the policy says no such thing. The compiler refuses Jones for the reason the clerk did: the line he needs isn't there.

Why are K, which throws an argument away, and S, which copies one, enough to do the work of every variable (Theorem 1.16)? The proof of that theorem is the proof of the Deduction Theorem, read with programs. Run the construction in the proof of Theorem 2.12 with the programs left on the lines, as Theorem 2.13 allows, and it compiles a definition with a parameter into K and S:

- case (a), a line that doesn't use the parameter, is wrapped in K, so as to ignore it;
- case (c), an application whose two halves both use the parameter, becomes an S, which copies the parameter to each half;
- case (b), the parameter itself, becomes $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$, the identity.

(I) and (II) are the types of the two constants left standing when it has finished. Chapter 1's "why did two combinators suffice?" and this chapter's "why these two axioms?" are one question.

This is where proof theory earns its keep in computer science.[14] Without the Deduction Theorem, a compiler that wanted a term of a given type would have to guess which K or S to start from, at a type like line 1 of the proof of Lemma 2.7, which nobody finds by thinking about it. With the theorem it starts from a declared parameter and works downwards:

- you write the function, with its parameter;
- the compiler, running the construction in the proof of Theorem 2.12 with the programs left on the lines, writes the combinators;
- (I) and (II) are the types of the two constants that survive when every parameter has been compiled away.

[14] Proofs earn their keep in access control too, and a ticket like the one of Example 2.14 is how it is done when it is done properly.

- Abadi et al. (1993) worked out a logic in which a policy is a set of formulas and a request is granted exactly when the resource asked for is provable from them;
- Appel and Felten (1999) went a step further and had the *request* carry its proof, so that the server, like the clerk, does nothing but check;
- Necula (1997) did the same for a program arriving over the network, which comes bundled with a proof that it is safe to run.

In every case whoever wants the access does the expensive part, finding the proof, and the trusted part is a few dozen lines of pattern matching.

That is the rule for parameters that the typings of Definition 1.24 lacked. It was never a rule. It was a theorem.[15]

What stands between the two definitions of §2.1 and a system you could deploy is expressiveness. A real policy says that two things both hold, that one of two things happened, that something is forbidden. It says who delegated what to whom, and it says things about every member of a group at once. The language of Definition 2.1 has one connective, and can say none of these. The connectives of the next section take care of the first three; saying things about every member of a group needs quantifiers.

2.4. **More connectives: and, or, and not.** Reasoning uses more connections between propositions than *if*, and the arrow alone can't express three of the commonest:

1. *Both.* "Both ψ and χ, given φ" could be written as two formulas, $\varphi \to \psi$ and $\varphi \to \chi$, and two will do wherever both are wanted; but there is no single claim "both", to be handed over as one hypothesis or proved as one conclusion.
2. *One or the other* can't be written at all. The formula $\varphi \to \psi$ says that if the first holds, so does the second; $\psi \to \varphi$ says the reverse; and $(\varphi \to \psi) \to \psi$ and its relatives say stranger things still. None of them says that one of the two holds, or lets anyone reason by cases from it.
3. *Never* is a prohibition, and an arrow only ever permits: it says that from one thing you may get another, never that something is forbidden. To say "never" we would need something to put on the right of the arrow that means "that cannot happen", and there is nothing there to put.

So we extend the language by two connectives, $\wedge$ and $\vee$, and one constant, $\bot$.

Definition 2.15 (formulas, extended)**.** Fix a stock of *letters* $p, q, r, \ldots$, as before. The *formulas* are defined inductively:

1. every letter is a formula;
2. $\bot$ is a formula, read "absurd" or "that cannot happen";
3. if φ and ψ are formulas, so is $\varphi \to \psi$, read "if φ then ψ";
4. if φ and ψ are formulas, so is $\varphi \wedge \psi$, read "φ and ψ";
5. if φ and ψ are formulas, so is $\varphi \vee \psi$, read "φ or ψ";
6. nothing else is a formula.

In one line,

$$\varphi, \psi \ ::= \ p \mid \bot \mid \varphi \to \psi \mid \varphi \wedge \psi \mid \varphi \vee \psi.$$

We write $\neg\varphi$, read "not φ", for $\varphi \to \bot$. It is not a sixth clause but a short way of writing an arrow into $\bot$. Negation takes precedence over $\wedge$ and $\vee$,

[15] And a machine has run it. David Turner ran the theorem's construction as the compiler for his language SASL in 1979, and in 1979–80 four people at Cambridge built SKIM, a processor whose instruction set was S, K and I, and ran programs on it (Clarke et al., 1980): the proof-transformation of Theorem 2.12 in silicon. The tripling is why Turner's compiler adds a few more combinators, B among them, to the two.

and they over $\to$; so $\neg\varphi \wedge \psi \to \chi$ is $((\neg\varphi) \wedge \psi) \to \chi$, and $\to$ still groups to the right.

The clauses for letters and for $\to$, and the closing clause, are those of Definition 2.1; the clauses for $\bot$, $\wedge$ and $\vee$ are new. Every formula of §2.1 is still a formula, read as it was.

Example 2.16 (Hughes, cont'd: three new sentences)**.** Hughes's policy has by now outgrown the arrow. It needs three new sentences, and Definition 2.15 can write each of them. With the letters of Example 2.2 and four more,

- a: Smith may edit the release notes;
- b: the scheduled job triggered the release;
- h: a maintainer pressed the button;
- u: the request comes from a fork,

the sentences become formulas:

(1) "people with write access may publish a release *and* edit the release notes" is $q \to r \wedge a$;

(2) "the release was triggered by the scheduled job *or* by a maintainer pressing the button" is $b \vee h$ (the audit log records that one of the two did it, and not which);

(3) "a request that comes from a fork may *not* publish a release" is $u \to \neg r$, that is, $u \to r \to \bot$.

By these conventions, $q \to r \wedge a$ is $q \to (r \wedge a)$.

How do we prove things in the richer language? Implication keeps its two schemas and its one rule, and modus ponens stays the only move. Each new connective gets schemas of its own, which say how to make a formula of that shape and how to use one:

- to make a conjunction you need both halves, and to use one you take a half;
- to make a disjunction you need one side, and to use one you must be ready for both;
- nothing makes $\bot$, and from $\bot$ anything follows.

Those are *both*, *one or the other* and *never*, turned into shapes a parser can match.[16]

[16]The names on the list are those of the Haskell functions whose types the schemas are, once $\varphi \wedge \psi$ is read as the pair type `(a, b)` and $\varphi \vee \psi$ as `Either a b`: `(,)` written as a function, `fst`, `snd`, `Left`, `Right`, `either`, and `absurd` from `Data.Void`; (I) and (II) could as well have been called (K) and (S). A type theorist writes the pair type $A \times B$ and the union $A + B$; the one connective this chapter types with, $\to$, is written the same way in both trades.

Definition 2.17 (axioms, extended)**.** From here on, the *axioms* are all instances of the nine schemas

$$\begin{aligned}
&\text{(I)} \quad \varphi \to \psi \to \varphi \\
&\text{(II)} \quad (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi \\
&\text{(pair)} \quad \varphi \to \psi \to \varphi \wedge \psi \\
&\text{(fst)} \quad \varphi \wedge \psi \to \varphi \\
&\text{(snd)} \quad \varphi \wedge \psi \to \psi \\
&\text{(left)} \quad \varphi \to \varphi \vee \psi \\
&\text{(right)} \quad \psi \to \varphi \vee \psi \\
&\text{(either)} \quad (\varphi \to \chi) \to (\psi \to \chi) \to \varphi \vee \psi \to \chi \\
&\text{(absurd)} \quad \bot \to \varphi
\end{aligned}$$

where φ, ψ and χ stand for arbitrary formulas.

Proofs, provability and theorems keep their definitions (Definitions 2.4 and 2.5): the three kinds of line are the same, word for word, and so is the rule; only the list of axioms is longer. Proofs are set out in boxes as before, and a line that is an instance of a schema is justified by the schema's name, with the formulas put for its Greek letters in square brackets, in the order φ, ψ, χ.

Every proof of the earlier sections is still a proof, and the proof theory of §2.2 still holds. The proofs of Proposition 2.9 and Theorem 2.12 look only at the three kinds of line, never at which schemas are on the list, so they go through unchanged (Exercise 2.9). In particular a supposition may be discharged as before.

Of the nine schemas, (either) is the one to read slowly. A disjunction $\varphi \vee \psi$ says that one of φ and ψ holds, without saying which. So to get anything out of it, say χ, you must be ready for both: you need a way from φ to χ and a way from ψ to χ. That is why (either) takes two implications before it takes the disjunction,

$$(\varphi \to \chi) \to (\psi \to \chi) \to \varphi \vee \psi \to \chi,$$

and why both implications end in the same χ: whichever way the disjunction was made, the conclusion is the same. It is proof by cases, written as a single formula. In a proof the two implications are usually got first, often by discharging a supposition, and the disjunction is used last.

Example 2.18 (Hughes, cont'd: proof by cases)**.** Hughes's bot may publish a release when the scheduled job triggers it, and it may publish when a maintainer presses the button. The audit log records that one of the two happened, and, being the kind of log it is, not which. May the release go ahead? Plainly yes: whichever of the two it was, it may.

The letters:

- b: the scheduled job triggered the release;
- h: a maintainer pressed the button;
- r: the release may go ahead.

She was told $b \to r$, $h \to r$ and $b \vee h$, and the claim is $\{b \to r,\ h \to r,\ b \vee h\} \vdash_{\mathrm{HJ}} r$.

Two rules with the same conclusion are as good as one rule whose condition is "the one or the other", and that is what (either) says. So make that rule and apply it to what the log says.

	$b \to r,\ h \to r,\ b \vee h$	
1	$b \to r$	hypothesis
2	$h \to r$	hypothesis
3	$(b \to r) \to (h \to r) \to b \vee h \to r$	(either) $[b,\ h,\ r]$
4	$(h \to r) \to b \vee h \to r$	MP 3, 1
5	$b \vee h \to r$	MP 4, 2
6	$b \vee h$	hypothesis
7	r	MP 5, 6

The box witnesses the claim.

Hughes never learns which of the two it was, and doesn't need to.

Example 2.19 (Hughes, cont'd: commutativity of $\vee$)**.** A colleague writes the log's sentence the other way round: "a maintainer pressed the button, or the job triggered it". Does it matter? It shouldn't, and it doesn't: each version follows from the other.

With b and h as before, the claim is $\vdash_{\mathrm{HJ}} b \vee h \to h \vee b$. Whichever of the two it was, the sentence the other way round holds, and (either) lets us say "whichever". Write ε for $h \vee b$, the thing wanted.

	no hypotheses	
1	$(b \to \varepsilon) \to (h \to \varepsilon) \to b \vee h \to \varepsilon$	(either) $[b,\ h,\ \varepsilon]$
2	$b \to \varepsilon$	(right) $[h,\ b]$
3	$(h \to \varepsilon) \to b \vee h \to \varepsilon$	MP 1, 2
4	$h \to \varepsilon$	(left) $[h,\ b]$
5	$b \vee h \to \varepsilon$	MP 3, 4

The box witnesses $\vdash_{\mathrm{HJ}} b \vee h \to h \vee b$.

No hypotheses and no discharge: the two cases were both axioms, and (either) joined them. The other direction is the same five lines with b and h exchanged.

Example 2.20 (Hughes, cont'd: from a contradiction, anything)**.** A careless edit leaves the policy saying both that Smith has write access and that he does not. What follows? Everything.

Let q be "Smith has write access"; the document now says q and $\neg q$, and the claim is that $\{q,\ \neg q\} \vdash_{\mathrm{HJ}} r$ for every r, "Jones may rotate the signing key", say. In words:

- "not q" means that q leads to what cannot happen;
- but q holds;
- so what cannot happen has happened;

- from that, anything.

	$q,\ \neg q$	
1	q	hypothesis
2	$q \to \bot$	hypothesis (that is, $\neg q$)
3	$\bot$	MP 2, 1
4	$\bot \to r$	(absurd) $[r]$
5	r	MP 4, 3

It witnesses $\{q,\ \neg q\} \vdash_{\mathrm{HJ}} r$.

So a conflict between an allow rule and a deny rule is an emergency, not a curiosity: from it, Jones may rotate the signing key.[17]

Example 2.21 (Hughes, cont'd: commutativity of $\wedge$)**.** The first of Hughes's three sentences was that people with write access may publish a release and edit the release notes. Her colleague, again, writes it as "edit the notes and publish".

The letters:

- r: Smith may publish a release;
- a: Smith may edit the release notes.

The claim, for now, is $\{r \wedge a\} \vdash_{\mathrm{HJ}} a \wedge r$: told both, she may conclude both the other way round. In words:

- take the second;
- take the first;
- put them together.

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

So the box witnesses $\{r \wedge a\} \vdash_{\mathrm{HJ}} a \wedge r$.

What Hughes would really like to file is the rule itself, $r \wedge a \to a \wedge r$, with nothing above the line. Theorem 2.12 gets it from this box in one step: discharge $r \wedge a$, and $\vdash_{\mathrm{HJ}} r \wedge a \to a \wedge r$.

[17] The story is told of Russell that a student, hearing that anything follows from a contradiction, challenged him to prove from $1 = 0$ that he was the Pope. Add one to each side: $2 = 1$. The Pope and I are two; therefore the Pope and I are one. Nobody has traced the story to a date or a room, which is in keeping with (absurd): once you've granted the premiss, any conclusion is as good as any other.

Example 2.22 (Hughes, cont'd: currying)**.** Hughes's two-factor rule says that people with write access who have two-factor authentication turned on may publish a release. A rule with two conditions can be written in two ways, and she wants to know that it doesn't matter which she picks.

The letters:

- q: Smith has write access;
- s: Smith has two-factor authentication on;
- r: Smith may publish a release.

- One way is $q \wedge s \to r$, "if write access and two-factor, then publish";
- the other is $q \to s \to r$, "if write access, then, if two-factor, publish".

She wants each to follow from the other.

First, told the *and* version of the rule, write access, and two-factor, pair the two conditions and apply the rule.

	$q \wedge s \to r,\ q,\ s$	
1	$q \wedge s \to r$	hypothesis
2	q	hypothesis
3	s	hypothesis
4	$q \to s \to q \wedge s$	(pair) $[q,\ s]$
5	$s \to q \wedge s$	MP 4, 2
6	$q \wedge s$	MP 5, 3
7	r	MP 1, 6

So the box witnesses $\{q \wedge s \to r,\ q,\ s\} \vdash_{\mathrm{HJ}} r$.

The other way: told the two-condition version and the pair, take each half and apply the rule twice.

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

So the box witnesses $\{q \to s \to r,\ q \wedge s\} \vdash_{\mathrm{HJ}} r$.

That isn't yet what she asked. She asked whether each rule follows from the other, and the two boxes say something weaker: that from either rule *and its conditions*, publish follows. Theorem 2.12 brings the hypotheses down one at a time:

- in the first box, discharge s, then q, then the rule, and $\vdash_{\mathrm{HJ}} (q \wedge s \to r) \to q \to s \to r$;

- in the second, discharge $q \wedge s$, then the rule, and $\vdash_{\mathrm{HJ}} (q \to s \to r) \to q \wedge s \to r$.

So a rule with two conditions may be written either way.

Example 2.23 (Hughes, cont'd: contraposition)**.** The checker refuses Smith a release: he has no write access. Hughes wants to say what that tells her, and what it tells her is that he isn't a maintainer.

The letters:

- p: Smith is a maintainer;
- q: Smith has write access.

The policy is $p \to q$, and what she wants is $\vdash_{\mathrm{HJ}} (p \to q) \to \neg q \to \neg p$: if being a maintainer gives you write access, then whoever lacks write access is not a maintainer.

In words:

- suppose the policy;
- suppose Smith has no write access, that is, his having it cannot happen;
- suppose, for the sake of argument, that he is a maintainer;
- then by the policy he has write access, which cannot happen;
- so his being a maintainer cannot happen either: that is $\neg p$.

The first three steps are suppositions. With them as hypotheses the clerk can follow her as far as "which cannot happen":

	$p \to q,\ \neg q,\ p$	
1	$p \to q$	hypothesis
2	$q \to \bot$	hypothesis (that is, $\neg q$)
3	p	hypothesis
4	q	MP 1, 3
5	$\bot$	MP 2, 4

So the box witnesses $\{p \to q,\ \neg q,\ p\} \vdash_{\mathrm{HJ}} \bot$: from the supposition that he is a maintainer, what cannot happen. That's what $\neg p$ *means*, $p \to \bot$, but here it has p above the line, not the formula $p \to \bot$ below it. Theorem 2.12 turns the one into the other. Discharge p, which gives $p \to \bot$, that is $\neg p$; then $\neg q$; then $p \to q$; and $\vdash_{\mathrm{HJ}} (p \to q) \to \neg q \to \neg p$.

Example 2.24 (Hughes, cont'd: distributing $\wedge$ over $\vee$)**.** Hughes's signing-key rule says that people with write access who are on the release rota or on the security rota may use the key. She would rather have two simple rules, one for people with write access on the release rota and one for people with write access on the security rota, and she wants to be sure that the two together cover everyone the single rule did.

The letters:

- w: has write access;
- e: is on the release rota;
- c: is on the security rota.
- The one rule's condition is $w \wedge (e \vee c)$;

- the two rules' conditions are $w \wedge e$ and $w \wedge c$.

What she wants is

$$\vdash_{\mathrm{HJ}} w \wedge (e \vee c) \to (w \wedge e) \vee (w \wedge c).$$

In words: take someone with write access who is on one rota or the other. See which.

- If the release rota, they have write access and are on the release rota;
- if the security rota, write access and the security rota.

Either way, they meet one of the two simple rules.

Write δ for $(w \wedge e) \vee (w \wedge c)$, the thing wanted. The two cases are two short lemmas, and here is the first: from write access and the release rota, conclude δ.

	$w,\ e$	
1	w	hypothesis
2	e	hypothesis
3	$w \to e \to w \wedge e$	(pair) $[w,\ e]$
4	$e \to w \wedge e$	MP 3, 1
5	$w \wedge e$	MP 4, 2
6	$w \wedge e \to \delta$	(left) $[w \wedge e,\ w \wedge c]$
7	δ	MP 6, 5

So $\{w, e\} \vdash_{\mathrm{HJ}} \delta$, and discharging e gives $\{w\} \vdash_{\mathrm{HJ}} e \to \delta$. The same seven lines with c for e, and (right) for (left), give $\{w\} \vdash_{\mathrm{HJ}} c \to \delta$.

Now the main proof, from the one rule's condition: take the two halves, and use the two lemmas to handle the two cases of the second half.

	$w \wedge (e \vee c)$	
1	$w \wedge (e \vee c)$	hypothesis
2	$w \wedge (e \vee c) \to w$	(fst) $[w,\ e \vee c]$
3	w	MP 2, 1
4	$w \wedge (e \vee c) \to e \vee c$	(snd) $[w,\ e \vee c]$
5	$e \vee c$	MP 4, 1
6	$e \to \delta$	first lemma, with line 3
7	$c \to \delta$	second lemma, likewise
8	$(e \to \delta) \to (c \to \delta) \to e \vee c \to \delta$	(either) $[e,\ c,\ \delta]$
9	$(c \to \delta) \to e \vee c \to \delta$	MP 8, 6
10	$e \vee c \to \delta$	MP 9, 7
11	δ	MP 10, 5

Lines 6 and 7 aren't single lines: each stands for the proof of a lemma, spliced in by transitivity, with its hypothesis w supplied by line 3. Discharge $w \wedge (e \vee c)$, and the theorem is proved: whoever the single rule admitted meets one of the two simple rules.

Carried out, each lemma becomes some twenty lines, the main proof with both spliced in about sixty, and the final discharge takes it to well over a hundred, every one of them checkable and almost none of them findable.

The Deduction Theorem is what keeps the proofs of this chapter short: every discharge in them, carried out, would roughly triple the proof.

History. Modern logic began with Gottlob Frege's *Begriffsschrift* of 1879 (Frege, 1879). Frege wanted to know how far arithmetic could be carried by pure inference, and found ordinary language too loose to tell him: an argument written in words can lean on something nobody stated, and nobody notices. So he built a notation in which every step must follow from earlier ones by a rule stated in advance. "To prevent anything intuitive from penetrating here unnoticed," he wrote, "I had to bend every effort to keep the chain of inferences free of gaps." A chain needs somewhere to start, and for *if* Frege gave three starting points. Schemas (I) and (II) are the first two, letter for letter, and modus ponens is how his chain goes on. The third was the swapping of two conditions,

$$(\varphi \to \psi \to \chi) \to \psi \to \varphi \to \chi,$$

and some fifty years later Łukasiewicz noticed that it follows from the other two by modus ponens, which Frege hadn't seen. That is why Definition 2.3 has two shapes and not three, and Exercise 2.3 asks you to do what Łukasiewicz did. The economy can be pushed further still: in 1953 Meredith (1953) found a single shape from which, with modus ponens alone, every instance of (I) and (II) follows, and nothing they don't already give,

$$((\varphi \to \psi) \to \chi) \to \theta \to (\psi \to \chi \to \tau) \to \psi \to \tau.$$

No shorter one is known.

Frege went on to derive arithmetic itself from a handful of logical laws, and in 1902, with the second volume of his *Basic Laws of Arithmetic* in press, he had a letter from Bertrand Russell. One of the laws let him form, for every property, the class of things that have it. Take the property of not being a member of oneself, and form the class R of all classes that are not members of themselves. Is R a member of R? If it is, it has the property that defines R, so it is not; and if it is not, it has that property, so it is. Russell later told the same story about a village barber who shaves exactly those villagers who don't shave themselves, and asked who shaves the barber. Frege's laws proved a contradiction, and from a contradiction anything follows, by (absurd).

Russell's repair, worked out with Alfred North Whitehead in the three volumes of *Principia Mathematica* (1910–13), was a theory of *types*: every class has a type, a class may have as members only things of lower type, and so the question whether a class is a member of itself can't even be asked. (The types of Chapter 1 are a distant descendant.) On that footing they wrote out the foundations of mathematics with every step by a stated rule and nothing left to the reader's good sense.

David Hilbert had his own reasons to care. In 1900, at the International Congress of Mathematicians in Paris, he set out twenty-three problems for

the new century, and the second asked for a proof that the axioms of arithmetic are consistent: that no proof from them ends in a contradiction. Russell's paradox made the question urgent. Hilbert did not adopt the philosophy of *Principia*, which was that mathematics is logic; what he took from it was the demonstration that mathematics could be written out in full, as formulas and rules. In lectures from 1917 on, with Paul Bernays beside him, he worked through the logic of *Principia* and recast it, and in the 1920s he proposed to study the result as mathematics in its own right (Hilbert, 1922). A proof is a finite sequence of formulas, each an axiom, or something one has been told, or got from earlier formulas by one of a short fixed list of rules. As a finite object it can be reasoned about, and Hilbert hoped to show, by reasoning about the lists alone, that no proof of arithmetic ends in $0 = 1$. That was the subject he called proof theory, and the plan is known as Hilbert's programme. Kurt Gödel showed in 1931 that it could not be carried out as Hilbert had hoped, which is the story Part 2 ends with.

The form used in this chapter, a short list of axioms, one rule, proofs as lists, is the one fixed in the textbook Hilbert wrote with Wilhelm Ackermann in 1928 (Hilbert and Ackermann, 1928), which grew out of those lectures, and that is why such systems carry his name. The same book asked a question that the clerk makes pressing: is there a procedure that decides, of any formula of the richer logic with *for all* and *there is* (Chapter 4 adds them), whether it is provable? Hilbert and Ackermann called it the main problem of mathematical logic, the *Entscheidungsproblem*. The answer, no, came in 1936, twice and independently. One of the two answers was Turing's, and his clerk was invented to give it; the other belongs to Chapter 3.

Meanwhile the cost of an implication had been noticed. In 1926 Łukasiewicz put it to his seminar in Warsaw: a mathematician *supposes* something, sees where it leads, and only then writes down an implication, and a logic ought to let the clerk do the same. Three years later Jacques Herbrand showed that a Hilbert system loses nothing by having no line for "suppose": the Deduction Theorem is in his doctoral thesis, finished in 1929 and printed in Warsaw in 1930, when he was twenty-two.[18] Alfred Tarski, also in Warsaw, said afterwards that he had known the theorem since 1921 and not written it down, which was an easy thing to do in 1921, a year before Hilbert made theorems *about* proofs a subject of their own. A logic whose rules suppose and discharge, as Łukasiewicz had asked, is the subject of Chapter 3.

Haskell Curry read *Principia* too, and saw in it something Hilbert hadn't been looking for: rules for rewriting formulas, which he took apart into combinators. When in 1934 he worked out what types K and S must have (Curry, 1934), he found Frege's first two axioms. Two questions that *Principia* had raised, Hilbert's of what a proof is and Curry's of what a function does, had arrived at the same place. Curry and Robert Feys put the correspondence into print in 1958, for exactly the system of this chapter (Curry and Feys, 1958); the second name it now carries is Chapter 3's. So the clerk's shapes are Frege's, their form is Hilbert's, the theorem that lets them suppose is

[18] A year later he was dead: on 27 July 1931 he fell while climbing at La Bérarde in the Massif des Écrins, aged twenty-three.

Year	Event
1879	Frege's *Begriffsschrift*: three axioms for *if*, the first two of them (I) and (II).
1900	Hilbert's Paris problems: the second asks for a proof that arithmetic is consistent.
1902	Russell's letter to Frege: the paradox.
1910	Whitehead and Russell, *Principia Mathematica* (three volumes, 1910–13): the foundations of mathematics written out rule by rule, on a theory of types.
1922	Hilbert, with Bernays: proof theory, with proofs as mathematical objects.
1926	Łukasiewicz's seminar: mathematicians suppose, and a logic should let them.
1928	Hilbert and Ackermann: the form of proof used in this chapter, and the Entscheidungsproblem.
1929	Herbrand's thesis: the Deduction Theorem (printed 1930).
1931	Gödel: Hilbert's programme cannot be carried out as planned.
1934	Curry: the types of K and S are Frege's first two axioms.
1936	The Entscheidungsproblem answered: no.
1953	Meredith: a single axiom does the work of (I) and (II).
1958	Curry and Feys print the correspondence for this chapter's system.

FIGURE 2.1. Timeline.

Herbrand's, and it was Curry who saw that the clerk had been a type checker all along. Figure 2.1 gives the dates.

Summary. A *proof* is a box of formulas, with the hypotheses above the line, each line an axiom, a hypothesis, or modus ponens from earlier lines; and checking one is pattern matching, which is why the clerk needs no understanding.

A typing is a proof with the programs written beside the lines (Theorem 2.13):

- the combinators K and S are the two schemas;
- application is modus ponens;
- a variable in the context is a hypothesis.

A program whose type is a policy's conclusion is such a typing, and a compiler that refuses one is refusing a bad proof.

The rule the compiler needed for a parameter is the **Deduction Theorem**, $\Gamma \cup \{\varphi\} \vdash_{\mathrm{HJ}} \psi$ iff $\Gamma \vdash_{\mathrm{HJ}} \varphi \to \psi$, whose proof is the algorithm that compiles the parameter away; and (I) and (II) are the types of the two constants that survive the compilation. When the language grew to *and*, *or* and *not*, seven schemas were added, each the type of a function you know once

$\wedge$ is read as a pair and $\vee$ as `Either`, and neither theorem changed. *How did the compiler know?* It checked a proof.

Exercises.

Exercise 2.1. Prove $\vdash_{\mathrm{HJ}} p \to p$ by the Deduction Theorem, in one line of thought. Then say which five lines the theorem writes on your behalf.

Exercise 2.2. Prove $\vdash_{\mathrm{HJ}} (p \to q) \to (q \to r) \to p \to r$, hypothetical syllogism with its antecedents the other way round, by the Deduction Theorem. Then start to prove it from the schemas alone by following the transformation in the proof of Theorem 2.12, and stop when you can see how long it will be.

Exercise 2.3. Flip, the function that swaps the order of two arguments, has the type $(p \to q \to r) \to q \to p \to r$. It is also, with its letters renamed, the third axiom about $\to$ in Frege's *Begriffsschrift*, the one Łukasiewicz later showed Frege hadn't needed (§2.4); this exercise is his redundancy, done by hand.

(1) Prove $\{p \to q \to r\} \vdash_{\mathrm{HJ}} q \to p \to r$ from the schemas and modus ponens, without the Deduction Theorem. (About nine lines; the instance of (II) you need has q as its first letter.)

(2) Prove the closed formula $\vdash_{\mathrm{HJ}} (p \to q \to r) \to q \to p \to r$ by the Deduction Theorem, and say roughly how long the proof of (1), put through the theorem's construction, would be.

Exercise 2.4. The first program of the clerk's kind was de Bruijn's Automath, built at Eindhoven from 1967 (de Bruijn, 1970). In 1977 van Benthem Jutting fed it the whole of Landau's *Grundlagen der Analysis*, a textbook that builds the real numbers from nothing, and it checked every line. De Bruijn's rule for such a program, now called the de Bruijn criterion, is that the checker must be small enough to be checked by hand, since the whole guarantee rests on it. Yours will be a few dozen lines.

1. Write a checker: a program that takes a list of formulas, each annotated "(I)", "(II)", "hypothesis" or "MP i, j", and a set of hypotheses, and returns whether the list is a proof from those hypotheses in the sense of Definition 2.4.
2. What is its running time in the length of the proof?
3. Then say which lines of your checker are rule (var) and which are rule (app) of a type checker, and what you would have to add to make it check typings instead.
4. Finally, extend it to the nine schemas of Definition 2.17: how many lines did that take?

Exercise 2.5. Prove $\vdash_{\mathrm{HJ}} \neg(\varphi \vee \psi) \to \neg\varphi \wedge \neg\psi$: if it cannot happen that one or the other holds, then neither holds.

Exercise 2.6. Chrysippus, in the third century BC, thought a dog could do this one. A hound following a scent comes to a fork of three paths; it sniffs the first, sniffs the second, finds nothing on either, and runs down the third without sniffing. "The first or the second or the third; not the first; not the second; so the third" is the fifth of the Stoics' indemonstrables (§2.1), and Sextus Empiricus, who reports the dog, wasn't impressed; but the dog had the schemas right.

Here's the two-path version: the scent is on the first path or the second, and not on the first. Prove $\vdash_{\mathrm{HJ}} (\varphi \vee \psi) \wedge \neg\varphi \to \psi$, and say where (absurd) is needed.

Exercise 2.7. Prove $\vdash_{\mathrm{HJ}} (\varphi \to \chi) \wedge (\psi \to \chi) \to \varphi \vee \psi \to \chi$, and say which Haskell function it is the type of, reading $\wedge$ as a pair and $\vee$ as `Either`.

Exercise 2.8. A release policy written in Haskell gains a rule: people with write access who have two-factor authentication turned on may publish a release. The policy already has a type `WriteAccess` of evidence that someone has write access, and a type `MayPublish` of tickets to publish.

1. Add a type `TwoFactor` for the new evidence.
2. Give the rule as a function in two ways:
   - taking a pair `(WriteAccess, TwoFactor)`;
   - curried.
3. Write the functions that turn each version into the other.
4. Say which proofs of Example 2.22 they are.

Exercise 2.9. Check that Proposition 2.9 and Theorem 2.12 hold for the proofs of Definition 2.17. Which steps of their proofs mention the list of schemas, and what do they need from it?

References.

Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS '99)*, pages 52–62, 1999.

Lewis Carroll. What the Tortoise said to Achilles. *Mind*, 4(14):278–280, 1895.

T. J. W. Clarke, P. J. S. Gladstone, C. D. MacLean, and A. C. Norman. SKIM—the S, K, I reduction machine. In *Proceedings of the 1980 ACM Conference on LISP and Functional Programming*, pages 128–135, 1980.

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

Haskell B. Curry and Robert Feys. *Combinatory Logic, Volume I.* North-Holland, Amsterdam, 1958.

N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer, Berlin, 1970.

Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Louis Nebert, Halle, 1879. English translation by S. Bauer-Mengelberg in J. van Heijenoort (ed.), *From Frege to Gödel*, Harvard University Press, 1967, pp. 1–82.

Jacques Herbrand. *Recherches sur la théorie de la démonstration.* PhD thesis, Université de Paris, 1930. Printed as Prace Towarzystwa Naukowego Warszawskiego, Wydział III, no. 33, Warsaw, 1930. English translation in *Logical Writings*, ed. W. D. Goldfarb, Harvard University Press, 1971.

David Hilbert. Neubegründung der Mathematik. Erste Mitteilung. *Abhandlungen aus dem Mathematischen Seminar der Hamburgischen Universität*, 1:157–177, 1922.

David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik.* Springer, Berlin, 1928. English translation of the second edition as *Principles of Mathematical Logic*, Chelsea, 1950.

Carew A. Meredith. A single axiom of positive logic. *The Journal of Computing Systems*, 1(3):169–170, 1953.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL)*, pages 106–119, 1997.

Allen Newell, J. C. Shaw, and Herbert A. Simon. Empirical explorations of the Logic Theory Machine: a case study in heuristics. In *Proceedings of the Western Joint Computer Conference*, pages 218–230, 1957.

Emil L. Post. Finite combinatory processes—formulation 1. *Journal of Symbolic Logic*, 1(3):103–105, 1936.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

David A. Turner. A new implementation technique for applicative languages. *Software: Practice and Experience*, 9(1):31–49, 1979.