PART 1: TYPE THEORY · CHAPTER 4

Quantifiers

Alexander V. Gheorghiu
COMP0009 Logic · University College London

CHAPTER 4

Quantifiers

How to say every *and* some*, and what it costs.*

OPENING

Monday morning: a screen recording

CASE: THE RECORDING

Ivanova writes drivers for a phone maker. A researcher sends her thirty seconds of video. An app with no permissions asks for a camera and is handed one. A moment later it is reading from the microphone, which it was never granted.

OPENING

What went wrong

- Every handle in the kernel is the same sort of thing: a small token with `read`, `write` and `close`.
- Behind each token is a private record that only its own driver understands.
- The camera call set to work on the record *as though it were a camera's*. It never asked.

The bytes were a microphone handle's.

OPENING

What did the token say?

A handle says that the holder may operate on *some* resource. It doesn't say *which kind*.

TYPE CONFUSION

A value of one type used where another was meant.

CWE-843. A staple of kernel exploits.

Ivanova knew the record was a camera's. The code had nowhere to write it down.

OPENING

Ivanova's first thought

If the letter p doesn't say *camera*, write a letter that does.

LETTER	READS
p_{cam}	the holder has a camera capability
q_{cam}	the holder has a camera handle
r_{cam}	the holder may read a camera
$p_{\text{mic}},\ q_{\text{mic}},\ r_{\text{mic}}$	the same, for the microphone; and so on

OPENING

One rule per kind

$$p_{\text{cam}} \to q_{\text{cam}}, \quad q_{\text{cam}} \to r_{\text{cam}}, \quad p_{\text{mic}} \to q_{\text{mic}}, \quad q_{\text{mic}} \to r_{\text{mic}}, \quad \dots$$

"a camera capability gives a camera handle, and a camera handle lets you read the camera; the same for the microphone; and so on"

A camera handle proves r_{cam}, not r_{mic}, so the wrong-kind read fails. So far so good.

OPENING

Three ways it fails

- **Size.** Dozens of device kinds: dozens of letters, and every rule copied forty times.
- **Staleness.** A new sensor arrives on Tuesday, and Monday's rule is a kind short.
- **Blindness.** The clerk compares symbols. To him p_{cam} and r_{cam} are as unrelated as p and r.

The kind is inside *the letter, where the clerk can't look.*

OPENING

Every kind

What the driver theory says is one sentence: *every kind of handle supports the same three operations*. Every kind, including Tuesday's sensor.

Nothing so far can say every*.*

OPENING

Every, elsewhere

WHO SAID IT	WHAT THEY SAID
Baker's runbook	what happens on *every* payment
a sorting routine	*every* element is at most the next
a sale	the discount applies on *every* day of it

None of these is a finite list.

OPENING

Where we are going

SECTION	THE QUESTION
4.1	Predicates and quantifiers: how do you say *every* and *some*?
4.2	NJ with quantifiers: four phrases, four rules, eight worked proofs.
4.3	Side conditions: what goes wrong without them.
4.4	Dependent types: put the programs back.
4.5	Curry–Howard with quantifiers.
4.6	Theories: what proving *from* one buys you.
4.7	History: the quantifier, invented twice.
4.8	The λ-cube: eight type systems in one picture.

SECTION 4.1

Predicates and quantifiers

Pull the kind out of the letter.

Pull them apart

$$p_{\text{cam}} \quad \text{becomes} \quad \mathsf{Cap}(\mathsf{cam})$$

"the holder has a capability for the camera"

- Cap is a predicate: a claim with a hole in it. cam is a name that fills the hole.
- Cap(cam) and Read(cam) now visibly share a part, and the clerk can see it.

4.1 · PREDICATES AND QUANTIFIERS

Ivanova's words: the names

Names for things: the device kinds, a driver, and a way of naming a driver from a kind.

SYMBOL	NAMES
cam, mic, net	the camera, the microphone, the network: three kinds
v4l	the camera driver, `v4l`
owner(k)	the driver that owns kind k, whichever that is

4.1 · PREDICATES AND QUANTIFIERS

Ivanova's words: the predicates

Claims with holes in them, five of them:

PREDICATE	READS
$\mathsf{Cap}(k)$	the holder has a capability for kind k
$\mathsf{Hand}(k)$	the holder has an open handle to kind k
$\mathsf{Read}(k)$	the holder may read from kind k
$\mathsf{Reg}(d, k)$	driver d has registered kind k
$\mathsf{Live}(k)$	kind k has a driver loaded

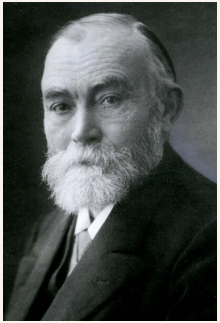
Ivanova's signature

Those symbols are Ivanova's signature: the vocabulary her claims are written in.

- Kinds and drivers are one sort of thing as far as the language goes, so $\mathsf{Cap}(\mathsf{v4l})$ is a formula, though nobody would assert it.
- Everything else in this section is built from these words, and every formula can be read aloud in them.

Frege, 1879

- Before Frege: *subject* and *predicate*, "the camera" and "is live". The grammar of Latin.
- Frege: *function* and *argument*, Live( ) and cam. The grammar of mathematics.



Gottlob Frege, 1848–1925

Frege, in his preface

"I believe that the replacement of the concepts *subject* and *predicate* by *argument* and *function*, respectively, will stand the test of time."

Frege, *Begriffsschrift*, 1879

A hole has to be closed

$$\mathsf{Cap}(k) \to \mathsf{Hand}(k)$$

"if the holder has a capability for kind k, it has a handle to k"

Which k? Until that's settled this isn't a claim, any more than $x + 1$ is a number. There are two ways to settle it.

For every way of filling it: the long *and*

$$(\mathsf{Cap}(\mathsf{cam}) \to \mathsf{Hand}(\mathsf{cam})) \land (\mathsf{Cap}(\mathsf{mic}) \to \mathsf{Hand}(\mathsf{mic})) \land \cdots$$

"a camera capability gives a camera handle, and a microphone capability gives a microphone handle, and …"

One conjunct per kind, and a kind short on Tuesday.

For every way of filling it: *for all*

$$\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))$$

"for every kind k: if the holder has a capability for k, it has a handle to k"

For all *is an* and *with a case for every thing, written once with a parameter.*

For some way of filling it: the long *or*

$$\mathsf{Reg}(\mathsf{v4l}, \mathsf{cam}) \lor \mathsf{Reg}(\mathsf{usb}, \mathsf{cam}) \lor \cdots$$

"`v4l` has registered the camera, or the USB driver has, or …"

One disjunct per driver.

For some way of filling it: *there is*

$$\exists d\, \mathsf{Reg}(d, \mathsf{cam})$$

"there is a driver d that has registered the camera"

There is *is an* or *with a case for every thing. The thing is the tag, the witness.*

4.1 · PREDICATES AND QUANTIFIERS

You have met both as a programmer

IN CODE	IN LOGIC
a test suite	the long *and*, its cases written out
a function body, written for a parameter	*for all*
a single failing test	*there is*, with the input exhibited

The quantifiers are $\wedge$ and $\vee$ with the list of cases made unbounded.

Why it mattered

- For two thousand years logic meant Aristotle's syllogisms: one *every* or *some* per sentence, never one inside another.
- Kant, 1787: logic "has reached its completion".
- Ninety-two years later, Frege's eighty-eight pages: "perhaps the most important single work ever written in logic" (van Heijenoort).

An argument no syllogism can reach

> Every horse is an animal. So every head of a horse is the head of an animal.
>
> De Morgan, 1840s

A some *inside an* every.

The horse, as a formula

$$\forall y \left(\exists x \left(\mathsf{Horse}(x) \wedge \mathsf{HeadOf}(y, x) \right) \rightarrow \exists x \left(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y, x) \right) \right)$$

"for every y: if there is a horse x whose head is y, then there is an animal x whose head is y"

4.1 · PREDICATES AND QUANTIFIERS

Nesting is everywhere

$$\forall x \, \exists y \, (x < y)$$

"for every number x there is a number y larger than it"

- An SQL query with `EXISTS` and `NOT EXISTS` is a formula with quantifiers.
- Frege's notation raised the question Turing answered in 1936: can a machine decide what follows from what?

Example 4.23 derives the horse. First, the general definitions.

4.1 · PREDICATES AND QUANTIFIERS

From Ivanova's words to any words

Ivanova's signature is one signature. A sorting library has another ($<$), arithmetic another (0, $+$, $<$).

The next three definitions say what any signature, term and formula is. Every example is still Ivanova's.

4.1 · PREDICATES AND QUANTIFIERS

Definition 4.1: first-order language

The symbols are of two kinds.

1. The logical symbols, the same in every language: $\to$, $\wedge$, $\vee$, $\bot$; the quantifiers $\forall$, $\exists$; variables $x, y, z, \ldots$
2. The signature, which changes from language to language: constant symbols $c, d, \ldots$; function symbols $f, g, \ldots$; predicate symbols $P, Q, \ldots$, each with an arity.

Ivanova's: constants cam, mic, net, v4l; one function symbol owner, arity 1; predicates Cap, Hand, Read, Live of arity 1 and Reg of arity 2.

Definition 4.2: terms

1. every variable is a term;
2. every constant symbol is a term;
3. if f has arity n and $t_1, \ldots, t_n$ are terms, $f(t_1, \ldots, t_n)$ is a term;
4. nothing else is a term.

A term names *a thing:* cam, k, $\mathsf{owner}(\mathsf{cam})$. *With no variables it is closed.*

Definition 4.3: formulas

1. $\bot$ is a formula;
2. if P has arity n and $t_1, \ldots, t_n$ are terms, $P(t_1, \ldots, t_n)$ is a formula;
3. if A, B are formulas, so are $(A \to B)$, $(A \wedge B)$, $(A \vee B)$;
4. if A is a formula and x a variable, $\forall x\, A$ and $\exists x\, A$ are formulas;
5. nothing else is a formula.

Clauses 1 and 2 give the atomic formulas, such as $\mathsf{Reg}(\mathsf{v4l}, \mathsf{cam})$. *A formula makes a* claim.

4.1 · PREDICATES AND QUANTIFIERS

Three conventions

- $\neg A$ abbreviates $A \to \bot$.
- Familiar symbols go between their arguments: $x + 1 < y$ is $<(+(x, 1), y)$, a term inside a formula.
- $\neg$ and the quantifiers bind tightest, so $\forall x\, A \to B$ is $(\forall x\, A) \to B$. A wider scope is bracketed: $\forall x\, (A \to B)$.

4.1 · PREDICATES AND QUANTIFIERS

The driver theory, Γ_{pol}

AS A SENTENCE	IN WORDS
$\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))$	a capability for a kind opens a handle to it
$\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))$	a handle to a kind may be read
$\forall d\,\forall k\,(\mathsf{Reg}(d,k) \to \mathsf{Live}(k))$	a registered kind has a driver loaded

Three sentences, about every kind there is or will be.

The open rule

$$\exists d\, \mathsf{Reg}(d, k) \;\wedge\; \mathsf{Cap}(\mathsf{owner}(k))$$

"some driver has registered kind k, and its owner has a capability for it"

$\mathsf{owner}(k)$ names a driver without saying which. That's what a function symbol is for.

Order matters

$$\forall k\, \exists d\, \mathsf{Reg}(d, k)$$

"every kind has some driver that registered it, perhaps a different one each time"

$$\exists d\, \forall k\, \mathsf{Reg}(d, k)$$

"there is one driver that registered every kind"

Which connective goes under the quantifier

- Under *for all*, a condition goes on the left of an arrow.
- Under *there is*, the description is joined with $\wedge$.

Five sentences from Ivanova's week follow, each with the formula that's easy to write instead.

Example 4.4: a rule about every kind

CASE: THE DOCUMENTATION

A kind with no driver loaded can't be read.

$$\forall k\,(\neg \mathsf{Live}(k) \to \neg \mathsf{Read}(k))$$

"for every kind k: if k has no driver loaded, the holder may not read k"

Example 4.4: the easy mistake

$$\forall k\,(\neg \mathsf{Live}(k) \land \neg \mathsf{Read}(k))$$

"every kind has no driver loaded, and no kind may be read"

A phone that's switched off. The condition belongs on the left of an arrow.

Example 4.5: a report about some kind

CASE: THE RESEARCHER'S REPORT

Some kind can be read by an app that holds no capability for it.

$$\exists k \, (\mathsf{Read}(k) \land \neg \mathsf{Cap}(k))$$

"there is a kind k that the holder may read, and the holder has no capability for k"

Example 4.5: the easy mistake

$$\exists k\,(\neg \mathsf{Cap}(k) \rightarrow \mathsf{Read}(k))$$

"there is a kind k such that, if the holder has no capability for k, it may read k"

Any kind the app *does* hold a capability for makes this true, so an app that reads nothing establishes it. The description belongs under $\wedge$.

Example 4.6: a named driver and an unnamed one

CASE: TWO SUMMARIES OF THE BOOT SEQUENCE

One engineer writes *every kind is registered by its owner*; the other, *every kind is registered by some driver*.

$$\forall k\, \mathsf{Reg}(\mathsf{owner}(k), k)$$

"for every kind k, the owner of k has registered k"

$$\forall k\, \exists d\, \mathsf{Reg}(d, k)$$

"for every kind k there is a driver d that has registered k"

Example 4.6: what each says

- The first names the driver, by a rule anyone can apply: hand it the camera and it says $\mathsf{owner}(\mathsf{cam})$.
- The second says only that there is a driver, perhaps a different one each time.
- The first gives the second (Example 4.20). Not the other way round.

Example 4.7: where the *not* goes

CASE: A SECURITY REVIEW

Two lines about an untrusted app: *not every kind may be read*, and *no kind may be read*.

$$\neg \forall k \, \mathsf{Read}(k)$$

"it is not the case that the holder may read every kind"

$$\forall k \, \neg \mathsf{Read}(k)$$

"for every kind k, the holder may not read k"

Example 4.7: a complaint and a sandbox

- The first is a mild complaint: perhaps one kind is protected. The second is a sandbox.
- The sandbox is also $\neg\exists k\,\mathsf{Read}(k)$: "there's no kind it may read".
- $\neg\forall k\,\mathsf{Read}(k)$ doesn't quite say $\exists k\,\neg\mathsf{Read}(k)$ here: a proof of *there is* must name the kind.

Example 4.8: a *some* that means *every*

CASE: THE INCIDENT RUNBOOK

If any kind can be read without a capability, page the security team.

Write s for "the team is paged" and $\mathsf{Bad}(k)$ for $\mathsf{Read}(k) \wedge \neg \mathsf{Cap}(k)$.

Example 4.8: two ways to write it

$$\exists k\, \mathsf{Bad}(k) \to s$$

“if there is a kind that can be read without a capability, the team is paged”

the runbook’s rule

$$\forall k\, (\mathsf{Bad}(k) \to s)$$

“for every kind k: if k can be read without a capability, the team is paged”

the monitoring code’s rule

Example 4.8: the same rule, and a different one

An any *in the condition of a rule is an* every *over the whole rule (Example 4.22).*

$$\exists k\,(\mathsf{Bad}(k) \to s)$$

"there is a kind k such that, if k can be read without a capability, the team is paged"

Not the same: any one kind known to be safe makes it true.

Definition 4.9: free variables

In $\forall x\, A$ and $\exists x\, A$, A is the scope. An occurrence of x is bound if it lies in the scope of a $\forall x$ or $\exists x$, and free otherwise.

$$\mathrm{FV}(A) := \begin{cases} \emptyset & \text{if } A \text{ is } \bot, \\ \text{the variables in } t_1, \ldots, t_n & \text{if } A \text{ is } P(t_1, \ldots, t_n), \\ \mathrm{FV}(B) \cup \mathrm{FV}(C) & \text{if } A \text{ is } B \circ C \ (\circ \text{ any of } \to, \wedge, \vee), \\ \mathrm{FV}(B) \setminus \{x\} & \text{if } A \text{ is } Q x\, B \ (Q \text{ either quantifier}). \end{cases}$$

Free variables: a sentence

$$\forall x \, \exists y \, (x < y)$$

"every number has a larger one"

Every occurrence is bound: a sentence. It makes a claim on its own.

Free variables: not a sentence

$$\forall x\,(x < y) \to \exists y\,(x < y)$$

"if every number is below y, then some number is above x"

The third x and the first y are free. The claim depends on which x and y you mean.

Everything in Γ_{pol} is a sentence. The open rule is not: its k is free.

Definition 4.11: substitution

$A[t/x]$ puts the term t in for the *free* occurrences of x in A, and nothing else.

1. For a term s, $s[t/x]$ is s with every x replaced by t.
2. t is substitutable for x in A if no free x in A lies in the scope of a quantifier on a variable of t.

Definition 4.11, continued

If t is substitutable for x in A:

$$A[t/x] := \begin{cases} \bot & \text{if } A \text{ is } \bot, \\ P(t_1[t/x], \dots, t_n[t/x]) & \text{if } A \text{ is } P(t_1, \dots, t_n), \\ B[t/x] \circ C[t/x] & \text{if } A \text{ is } B \circ C, \\ Q x\, B & \text{if } A \text{ is } Q x\, B, \\ Q y\, B[t/x] & \text{if } A \text{ is } Q y\, B \text{ with } y \neq x. \end{cases}$$

Substitutable, or not

$$\exists y\,(x < y)$$

"some number is larger than x"

- Put in z for x: "some number is larger than z". Fine.
- Put in y for x: $\exists y\,(y < y)$, "some number is larger than itself". The meaning changed: y is not substitutable.

When a term isn't substitutable, rename the bound variable first.

SECTION 4.2

NJ with quantifiers

Four phrases you have said aloud.

4.2 · NJ WITH QUANTIFIERS

Two arguments you have heard

Gentzen got the rules for the connectives by listening to people argue. The quantifiers got theirs the same way.

EVERY KIND WITH A CAPABILITY MAY BE READ

"Let u be any kind, and suppose a capability for u. Then there's a handle to u, so u may be read. And u was any kind."

THE REGISTER RULE, USED

"Some driver registered the camera; call it y. Then by the rule, at y, the camera is live."

Four phrases carry the quantifiers in these two arguments. The four phrases are the four rules.

4.2 · NJ WITH QUANTIFIERS

Four phrases

SAID ALOUD	FROM	TO
let y be arbitrary	$A[y/x]$, nothing assumed about y	$\forall x\, A$
in particular, for t	$\forall x\, A$	$A[t/x]$
take x to be t	$A[t/x]$	$\exists x\, A$
let y be such a thing; then …	$\exists x\, A$; and C, supposing $A[y/x]$	C

4.2 · NJ WITH QUANTIFIERS

Two of them come with a condition

- **Arbitrary:** nothing assumed says anything about y.
- **Such a thing:** y is a name made up for the argument, so C, which outlives the argument, mustn't mention it.

In each case the condition is what the phrase means. Section 4.3 takes the conditions away to see what breaks.

4.2 · NJ WITH QUANTIFIERS

Where you have heard them

A MATHEMATICIAN SAYS	A PROGRAMMER DOES
let x be arbitrary	writes a body for a parameter, not knowing which value will arrive
in particular, for t	calls the function on the value in hand
take x to be t	writes a failing test: one input, exhibited
let y be such a thing	works with “some driver registered it” without being told which

4.2 · NJ WITH QUANTIFIERS

NJ was always first-order

Gentzen had all four rules in 1934. The propositional NJ of Chapter 3 is the half of his system without quantifiers.

He called the restricted variable the *Eigenvariable*: the variable proper to that step.

4.2 · NJ WITH QUANTIFIERS

Figure 4.1: the quantifier rules of NJ

INTRODUCTION: TO PROVE IT

$$\frac{A[y/x]}{\forall x\, A}\ \forall\mathrm{I}$$

y free in no open hypothesis, nor in $\forall x\, A$

ELIMINATION: TO USE IT

$$\frac{\forall x\, A}{A[t/x]}\ \forall\mathrm{E}$$

$$\frac{A[t/x]}{\exists x\, A}\ \exists\mathrm{I}$$

$$\frac{\exists x\, A \qquad \begin{array}{c}[A[y/x]]\\ \vdots\\ C\end{array}}{C}\ \exists\mathrm{E}$$

y free in no other open hypothesis, nor in C or $\exists x\, A$

4.2 · NJ WITH QUANTIFIERS

The rules in words: the two without a condition

- ($\forall$E) is *in particular*: from $\forall x\, A$, conclude A with any term t put in for x.
- ($\exists$I) is *take x to be t*: from A with t put in, conclude $\exists x\, A$.

The term t is written on the bar: the witness, or the case being used.

4.2 · NJ WITH QUANTIFIERS

The rules in words: the two with a condition

- ($\forall$I) is *let y be arbitrary*: from A with a variable y put in, conclude $\forall x\, A$, provided nothing still assumed mentions y.
- ($\exists$E) is *let y be such a thing*: to get C out of $\exists x\, A$, get it from the supposition that A holds of a new name y, which C mustn't mention.

($\exists E$) discharges its supposition, like ($\vee E$). It is ($\vee E$) with one case for every thing there is.

4.2 · NJ WITH QUANTIFIERS

One tree, read aloud

The argument we began with, from the first two sentences of the driver theory:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}{\mathsf{Hand}(u) \to \mathsf{Read}(u)}\,\forall\mathrm{E} \quad \dfrac{\dfrac{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}{\mathsf{Cap}(u) \to \mathsf{Hand}(u)}\,\forall\mathrm{E} \quad [\mathsf{Cap}(u)]^{1}}{\mathsf{Hand}(u)}\,{\to}\mathrm{E}}{\mathsf{Read}(u)}\,{\to}\mathrm{E}}{\mathsf{Cap}(u) \to \mathsf{Read}(u)}\,{\to}\mathrm{I},\,1}{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Read}(k))}\,\forall\mathrm{I}$$

4.2 · NJ WITH QUANTIFIERS

Reading it aloud

- Let u be any kind, and suppose a capability for it.
- The first sentence, *in particular* for u, gives a handle; the second, in particular for u, gives the right to read.
- Discharge the supposition: if a capability for u, then u may be read.
- And u was any kind. The leaves still open are the two sentences of the theory, and neither mentions u.

The definition says in general what this tree did.

4.2 · NJ WITH QUANTIFIERS

Definition 4.13: derivations, with quantifiers

The nine propositional clauses as before, and four more. Throughout, t is a term and y a variable.

9. ($\forall$E) If $\mathcal{D}$ concludes $\forall x\, A$ and t is substitutable for x in A, then

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \forall x\, A\end{array}}{A[t/x]}\,\forall\mathrm{E} \qquad \text{is a derivation.}$$

10. ($\forall$I) If $\mathcal{D}$ concludes $A[y/x]$, where y is substitutable for x in A, and y **is free neither in $\forall x\, A$ nor in any formula of** $\mathrm{OH}(\mathcal{D})$, then

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A[y/x]\end{array}\quad \textit{no open leaf has } y \textit{ free}}{\forall x\, A}\,\forall\mathrm{I} \qquad \text{is a derivation, with } \mathrm{OH} = \mathrm{OH}(\mathcal{D}).$$

Definition 4.13, continued: the existential clauses

11. ($\exists$I) If $\mathcal{D}$ concludes $A[t/x]$ and t is substitutable for x in A, then

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A[t/x]\end{array}}{\exists x\, A}\,\exists\mathrm{I}$$

is a derivation.

12. ($\exists$E) If $\mathcal{D}_0$ concludes $\exists x\, A$ and $\mathcal{D}_1$ concludes C, with y substitutable for x in A and **free in none of C, $\exists x\, A$, and $\mathrm{OH}(\mathcal{D}_1) \setminus \{A[y/x]\}$**, then

$$\dfrac{\exists x\, A \qquad \begin{array}{c}[A[y/x]]\\ \mathcal{D}_1\\ C\end{array}}{C}\,\exists\mathrm{E}$$

no other open leaf has y free

is a derivation.

Definition 4.13, continued

13. Nothing else is a derivation.

The variable y of $(\forall I)$ and $(\exists E)$ is the eigenvariable of the step. Two clauses looked at $\mathrm{OH}(\mathcal{D})$, the open hypotheses: a function of the tree's shape, as in Chapter 3, with one line per new rule.

4.2 · NJ WITH QUANTIFIERS

Definition 4.14: open hypotheses; derivability

By cases on the last rule, $\mathcal{D}'$ the one subderivation of a one-premiss rule, $\mathcal{D}_0$, $\mathcal{D}_1$ the two of ($\exists$E):

$$\mathrm{OH}(\mathcal{D}) := \begin{cases} \text{as in Chapter 3} & \text{a leaf or an old rule,} \\ \mathrm{OH}(\mathcal{D}') & (\forall\mathrm{E}),\ (\forall\mathrm{I}),\ (\exists\mathrm{I}), \\ \mathrm{OH}(\mathcal{D}_0) \cup (\mathrm{OH}(\mathcal{D}_1) \setminus \{A[y/x]\}) & (\exists\mathrm{E}), \text{ discharging } A[y/x]. \end{cases}$$

DERIVABLE IN NJ

$\Gamma \vdash_{\mathrm{NJ}} A$ when there is a derivation $\mathcal{D}$ whose conclusion is A and whose open hypotheses are among Γ: $\mathrm{OH}(\mathcal{D}) \subseteq \Gamma$.

4.2 · NJ WITH QUANTIFIERS

Three bars discharge

($\to$I), ($\vee$E) and now ($\exists$E). Chapter 3's conventions carry over.

- A bar takes every open copy of the formula it discharges: many leaves, or none.
- A leaf is discharged by the first bar below it, on its path to the root, that can discharge it there.
- For ($\exists$E), *there* is the second premiss. Nothing in the first is discharged.

4.2 · NJ WITH QUANTIFIERS

What the two conditions check

(∀I): ARBITRARY, TWO CHECKS

y is free in no open hypothesis, read off the leaves; and not in $\forall x\, A$.

(∃E): SUCH A THING, THREE CHECKS

y is free neither in C, which outlives it, nor in $\exists x\, A$, nor in any other open hypothesis of the branch.

So nothing already in play is quietly being identified with y.

4.2 · NJ WITH QUANTIFIERS

The clause beside the turnstile

Write Δ for the open hypotheses above the bar. On the left, the tree; on the right, what it witnesses.

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A[y/x]\end{array}}{\forall x\, A}\ \forall\mathrm{I} \qquad\qquad \dfrac{\Delta \vdash_{\mathrm{NJ}} A[y/x]}{\Delta \vdash_{\mathrm{NJ}} \forall x\, A}$$

with y free in no formula of Δ, nor in $\forall x\, A$

Read on the turnstile, the conditions are one: nothing to the left of it mentions y.

4.2 · NJ WITH QUANTIFIERS

Worked derivations

Seven of Ivanova's questions, and one of De Morgan's. Each is done in the same order: the situation, the claim, the reasoning in words, the tree.

How trees like these are found is Section 4.5's business. For now, read them.

Example 4.16: the boot log

CASE: THE BOOT LOG

The log has the line `v4l: registered camera`. Is the camera live?

$$\forall d\,\forall k\,(\mathsf{Reg}(d,k) \to \mathsf{Live}(k)),\ \ \mathsf{Reg}(\mathsf{v4l},\mathsf{cam}) \ \vdash_{\mathrm{NJ}}\ \ \mathsf{Live}(\mathsf{cam})$$

"given the register rule, and the log line saying `v4l` registered the camera: the camera is live"

Example 4.16: the argument

The register rule holds for every driver, in particular `v4l`*; and for every kind, in particular the camera; and the log says its condition is met.*

Example 4.16: the tree

$$\dfrac{\dfrac{\dfrac{\forall d\,\forall k\,(\mathsf{Reg}(d,k) \to \mathsf{Live}(k))}{\forall k\,(\mathsf{Reg}(\mathsf{v4l},k) \to \mathsf{Live}(k))}\,\forall\mathrm{E}}{\mathsf{Reg}(\mathsf{v4l},\mathsf{cam}) \to \mathsf{Live}(\mathsf{cam})}\,\forall\mathrm{E} \qquad \mathsf{Reg}(\mathsf{v4l},\mathsf{cam})}{\mathsf{Live}(\mathsf{cam})}\,{\to}\mathrm{E}$$

The two ($\forall E$) bars come off outermost first: $t := \mathsf{v4l}$, then $t := \mathsf{cam}$.

Example 4.17: the driver guide

CASE: THE DRIVER GUIDE

Ivanova wants one sentence: *a capability for a kind gives you a handle to it and the right to read it.* Does the model support it?

$$\Gamma_{\text{pol}} \vdash_{\text{NJ}} \forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k) \wedge \mathsf{Read}(k))$$

"from the driver theory: for every kind, a capability for it gives a handle to it and the right to read it"

Example 4.17: the argument

Let u be any kind, and suppose a capability for it. Then a handle, by the first sentence, and the right to read, by the second. Both halves; discharge; generalise.

Example 4.17: a piece used twice

The handle is needed twice. Call the subtree that makes it $\mathcal{H}$:

$$\mathcal{H} \;=\; \dfrac{\dfrac{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}{\mathsf{Cap}(u) \to \mathsf{Hand}(u)}\,\forall\mathrm{E} \qquad [\mathsf{Cap}(u)]^{1}}{\mathsf{Hand}(u)}\,\to\mathrm{E}$$

Example 4.17: the whole tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}{\mathsf{Hand}(u) \to \mathsf{Read}(u)}\,\forall\mathrm{E} \qquad \mathcal{H}\ \mathsf{Hand}(u)}{\mathsf{Read}(u)}\,{\to}\mathrm{E} \qquad \mathcal{H}\ \mathsf{Hand}(u)}{\mathsf{Hand}(u) \wedge \mathsf{Read}(u)}\,\wedge\mathrm{I}}{\mathsf{Cap}(u) \to \mathsf{Hand}(u) \wedge \mathsf{Read}(u)}\,{\to}\mathrm{I},1}{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k) \wedge \mathsf{Read}(k))}\,\forall\mathrm{I}$$

open hypotheses $\{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k)),\ \forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))\} \subseteq \Gamma_{\mathrm{pol}}$

4.2 · NJ WITH QUANTIFIERS

Example 4.17: two things to notice

FLAG: MULTIPLE DISCHARGE, THEN GENERALISE

The bar marked 1 discharges both copies of $[\mathsf{Cap}(u)]^1$, one in each $\mathcal{H}$. Only below it is ($\forall$I) legal: above it, $\mathsf{Cap}(u)$ is open, and it mentions u.

Example 4.18: anything at all?

CASE: A REVIEWER'S QUESTION

Can the camera app read anything at all? It holds a camera capability.

$$\Gamma_{\text{pol}},\ \mathsf{Cap}(\mathsf{cam}) \vdash_{\text{NJ}} \exists k\, \mathsf{Read}(k)$$

"from the driver theory and a camera capability: some kind may be read"

Example 4.18: the argument

The camera has a handle, so it can be read, so some kind can: namely the camera.

Example 4.18: the tree

$$\dfrac{\dfrac{\dfrac{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}{\mathsf{Hand}(\mathsf{cam}) \to \mathsf{Read}(\mathsf{cam})}\,\forall\mathrm{E} \qquad \dfrac{\dfrac{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}{\mathsf{Cap}(\mathsf{cam}) \to \mathsf{Hand}(\mathsf{cam})}\,\forall\mathrm{E} \qquad \mathsf{Cap}(\mathsf{cam})}{\mathsf{Hand}(\mathsf{cam})}\,\to\mathrm{E}}{\mathsf{Read}(\mathsf{cam})}\,\to\mathrm{E}}{\exists k\,\mathsf{Read}(k)}\,\exists\mathrm{I}$$

The last bar takes $t := \mathsf{cam}$*: the witness is written on that bar.*

Example 4.19: a cut-off manifest

CASE: A CUT-OFF MANIFEST

Another app's manifest records a capability for *some* kind; the line saying which is missing. Can the app read anything?

$$\Gamma_{\text{pol}},\ \exists k\, \mathsf{Cap}(k)\ \vdash_{\text{NJ}}\ \exists k\, \mathsf{Read}(k)$$

"from the driver theory and a capability for some kind or other: some kind may be read"

Example 4.19: the argument

Call the kind y. Then y has a handle and can be read, so some kind can be read. That conclusion doesn't mention y, so it follows whichever kind y was.

Example 4.19: the branch where y is supposed

Call it $\mathcal{B}$:

$$\dfrac{\dfrac{\dfrac{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}{\mathsf{Hand}(y) \to \mathsf{Read}(y)}\,\forall\mathrm{E} \qquad \dfrac{\dfrac{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}{\mathsf{Cap}(y) \to \mathsf{Hand}(y)}\,\forall\mathrm{E} \qquad [\mathsf{Cap}(y)]^1}{\mathsf{Hand}(y)}\,{\to}\mathrm{E}}{\mathsf{Read}(y)}\,{\to}\mathrm{E}}{\exists k\,\mathsf{Read}(k)}\,\exists\mathrm{I}$$

Example 4.19: opening the hypothesis

$$\dfrac{\exists k\,\mathsf{Cap}(k) \qquad \begin{array}{c}[\mathsf{Cap}(y)]\\ \mathcal{B}\\ \exists k\,\mathsf{Read}(k)\end{array}}{\exists k\,\mathsf{Read}(k)}\ \exists\mathrm{E},1$$

open hypotheses $\{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k)),\ \forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k)),\ \exists k\,\mathsf{Cap}(k)\} \subseteq \Gamma_{\mathrm{pol}} \cup \{\exists k\,\mathsf{Cap}(k)\}$

Example 4.19: the eigenvariable

FLAG: THE EIGENVARIABLE OF AN EXISTENTIAL

All three checks are read off the tree. y is free neither in the conclusion $\exists k\,\mathsf{Read}(k)$, nor in $\exists k\,\mathsf{Cap}(k)$, nor in the two sentences of the theory, the open leaves of $\mathcal{B}$ other than the discharged $\mathsf{Cap}(y)$.

Example 4.20: the two summaries again

CASE: THE TWO SUMMARIES

Does *every kind is registered by its owner* give *every kind is registered by some driver*?

$$\forall k\, \mathsf{Reg}(\mathsf{owner}(k), k) \;\vdash_{\mathrm{NJ}}\; \forall k\, \exists d\, \mathsf{Reg}(d, k)$$

"from *every kind is registered by its owner*: every kind is registered by some driver"

Example 4.20: the argument

Let u be any kind. Its owner registered it, so some driver did, namely $\mathsf{owner}(u)$*. And u was any kind.*

Example 4.20: the tree

$$\dfrac{\dfrac{\dfrac{\forall k\,\mathsf{Reg}(\mathsf{owner}(k),k)}{\mathsf{Reg}(\mathsf{owner}(u),u)}\,\forall\mathrm{E}}{\exists d\,\mathsf{Reg}(d,u)}\,\exists\mathrm{I}}{\forall k\,\exists d\,\mathsf{Reg}(d,k)}\,\forall\mathrm{I}$$

- ($\exists$I) takes $t := \mathsf{owner}(u)$: a witness needn't be a constant.
- ($\forall$I) is legal: the one open hypothesis is a sentence.

Example 4.21: the sandbox

CASE: AN UNTRUSTED APP

Its sandbox says $\neg\exists k\,\mathsf{Read}(k)$. The kernel checks one kind at a time.

$$\neg\exists k\,\mathsf{Read}(k) \;\vdash_{\mathrm{NJ}}\; \forall k\,\neg\mathsf{Read}(k)$$

"from *no kind may be read*: for every kind, it may not be read"

Example 4.21: the argument

Let u be any kind, and suppose the app may read u. Then some kind may be read, which the sandbox denies.

Example 4.21: the tree

$$\dfrac{\dfrac{\dfrac{\neg\exists k\,\mathsf{Read}(k) \qquad \dfrac{[\mathsf{Read}(u)]^{1}}{\exists k\,\mathsf{Read}(k)}\,\exists\mathrm{I}}{\bot}\,\to\mathrm{E}}{\neg\mathsf{Read}(u)}\,\to\mathrm{I},1}{\forall k\,\neg\mathsf{Read}(k)}\,\forall\mathrm{I}$$

FLAG: DISCHARGE BEFORE YOU GENERALISE

($\forall$I) comes last. Below ($\to$I), $\mathsf{Read}(u)$ is no longer open; only the sandbox is.

Example 4.22: the runbook and the code

Example 4.8 claimed the runbook's rule and the monitoring code's rule say the same thing. That is two derivations, one each way.

$$\exists k\, \mathsf{Bad}(k) \to s \;\vdash_{\mathrm{NJ}}\; \forall k\, (\mathsf{Bad}(k) \to s)$$

"the runbook's rule gives the code's rule"

$$\forall k\, (\mathsf{Bad}(k) \to s) \;\vdash_{\mathrm{NJ}}\; \exists k\, \mathsf{Bad}(k) \to s$$

"the code's rule gives the runbook's rule"

Example 4.22: the runbook gives the code

Let u be any kind, and suppose it can be read without a capability. Then some kind can, so the runbook says page.

$$\dfrac{\dfrac{\dfrac{\exists k\,\mathsf{Bad}(k) \to s \qquad \dfrac{[\mathsf{Bad}(u)]^{1}}{\exists k\,\mathsf{Bad}(k)}\,\exists\mathrm{I}}{s}\,{\to}\mathrm{E}}{\mathsf{Bad}(u) \to s}\,{\to}\mathrm{I},1}{\forall k\,(\mathsf{Bad}(k) \to s)}\,\forall\mathrm{I}$$

The sandbox tree again, with s where it had $\bot$.

4.2 · NJ WITH QUANTIFIERS

Example 4.22: the code gives the runbook

Suppose some kind can be read without a capability, and call it y. The code's rule at y says page, and paging doesn't mention y.

$$\dfrac{\dfrac{[\exists k\,\mathsf{Bad}(k)]^2 \qquad \dfrac{\dfrac{\forall k\,(\mathsf{Bad}(k) \to s)}{\mathsf{Bad}(y) \to s}\,\forall\mathrm{E} \qquad [\mathsf{Bad}(y)]^1}{s}\,{\to}\mathrm{E}}{s}\,\exists\mathrm{E},1}{\exists k\,\mathsf{Bad}(k) \to s}\,{\to}\mathrm{I},2$$

($\exists E$) is legal: y is free in none of s, $\exists k\,\mathsf{Bad}(k)$ and the code's rule.

Example 4.23: the head of a horse

CASE: DE MORGAN'S ARGUMENT

Every horse is an animal, so every head of a horse is the head of an animal. The argument two thousand years of syllogisms couldn't make.

$$H(z) \;:=\; \mathsf{Horse}(z) \wedge \mathsf{HeadOf}(y, z)$$

"z is a horse, and y is its head"

Example 4.23: in words

- The conclusion starts with $\forall$: *let y be any thing.*
- Under that is an arrow: *suppose y is the head of some horse.*
- That starts with $\exists$, so open it: *call the horse z.*
- Then z is an animal, and y is still its head. So y is the head of some animal, namely z.

Example 4.23: the middle, where z is known

Call it $\mathcal{Z}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x\,(\mathsf{Horse}(x) \to \mathsf{Animal}(x))}{\mathsf{Horse}(z) \to \mathsf{Animal}(z)}\,\forall\mathrm{E} \quad \dfrac{[H(z)]^1}{\mathsf{Horse}(z)}\,\wedge\mathrm{E}_1}{\mathsf{Animal}(z)}\,\to\mathrm{E} \quad \dfrac{[H(z)]^1}{\mathsf{HeadOf}(y,z)}\,\wedge\mathrm{E}_2}{\mathsf{Animal}(z) \wedge \mathsf{HeadOf}(y,z)}\,\wedge\mathrm{I}}{\exists x\,(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y,x))}\,\exists\mathrm{I}$$

Example 4.23: three more bars

One bar each for *call the horse* z, *suppose*, and *let* y *be any thing*:

$$\dfrac{\dfrac{\dfrac{[\exists x\,(\mathsf{Horse}(x) \wedge \mathsf{HeadOf}(y,x))]^2 \qquad \overset{z}{\exists x\,(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y,x))}}{\exists x\,(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y,x))}\,\exists\mathrm{E},1}{\exists x\,(\mathsf{Horse}(x) \wedge \mathsf{HeadOf}(y,x)) \to \exists x\,(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y,x))}\,{\to}\mathrm{I},2}{\forall y\,\Big(\exists x\,(\mathsf{Horse}(x) \wedge \mathsf{HeadOf}(y,x)) \to \exists x\,(\mathsf{Animal}(x) \wedge \mathsf{HeadOf}(y,x))\Big)}\,\forall\mathrm{I}$$

open hypotheses $\{\forall x\,(\mathsf{Horse}(x) \to \mathsf{Animal}(x))\} \subseteq \{\forall x\,(\mathsf{Horse}(x) \to \mathsf{Animal}(x))\}$

Example 4.23: why the bars are legal

- ($\exists$E): z is free in none of its conclusion, $\exists x\,(\mathsf{Horse}(x) \wedge \mathsf{HeadOf}(y, x))$, and the open leaves of $\mathcal{Z}$ other than $H(z)$.
- ($\forall$I): after both discharges the only open hypothesis is a sentence.

Nine bars, and every one of them is a rule of NJ.

4.2 · NJ WITH QUANTIFIERS

Checking a tree: the old checks

- Every bar has premisses of the shape its clause requires.
- Every bracket carries a number, and the bar with that number is the first below the leaf that can discharge it.
- The leaves left without brackets are the open hypotheses. The tree witnesses $\Gamma \vdash_{\mathrm{NJ}} A$ for every Γ that contains them.

4.2 · NJ WITH QUANTIFIERS

Checking a tree: the new checks

- ($\forall$I): y free neither in $\forall x\, A$ nor in a leaf open above the bar.
- ($\exists$E): y free in none of C, $\exists x\, A$, and the open leaves of the second premiss other than $A[y/x]$.
- Every term put in is substitutable.

4.2 · NJ WITH QUANTIFIERS

Theorem 4.24: consistency

THEOREM 4.24 · CONSISTENCY

No atomic formula is derivable from no hypotheses: $\nvdash_{\mathrm{NJ}} \bot$, and $\nvdash_{\mathrm{NJ}} P(t_1, \ldots, t_n)$.

The rules add nothing of their own.

4.2 · NJ WITH QUANTIFIERS

Corollaries 4.24 and 4.25

COROLLARIES 4.24 AND 4.25 · DISJUNCTION PROPERTY; EXISTENCE PROPERTY

4.21 If $\vdash_{\mathrm{NJ}} A \vee B$, then $\vdash_{\mathrm{NJ}} A$ or $\vdash_{\mathrm{NJ}} B$.
4.22 If $\vdash_{\mathrm{NJ}} \exists x\, A$, then there is a term t with $\vdash_{\mathrm{NJ}} A[t/x]$.

A theorem there is *can be made to say* what. *The proofs come in Section 4.5, from the programs.*

SECTION 4.3

Side conditions

Three bugs you have met.

4.3 · SIDE CONDITIONS

Three things can go wrong

Each is a bug you've met in a program, and each is refused by a clause of Definition 4.13.

BUG	RULE	THE COMPILER SAYS (CHAPTER 5)
generalising over a fixed variable	($\forall$I)	*rigid type variable*
an escaping witness	($\exists$E)	*would escape its scope*
variable capture	($\forall$E)	a refactoring that renames the wrong thing

4.3 · SIDE CONDITIONS

Bug 1: generalising over a fixed variable

CASE: A REQUEST FOR KIND u

The hypothesis in force is $\mathsf{Cap}(u)$: *this* caller has a capability for u. One ($\forall$I) would give $\forall k\, \mathsf{Cap}(k)$, a capability for every kind, and so the right to read every kind.

4.3 · SIDE CONDITIONS

Bug 1: where it fails

FLAG: THE CONDITION ON GENERALISING

u is free in the open hypothesis $\mathsf{Cap}(u)$, so u isn't arbitrary, whatever the argument calls it.

Discharge first, by ($\to$I), and generalising becomes legal. What you generalise then is $\mathsf{Cap}(u) \to \ldots$*, which is harmless.*

4.3 · SIDE CONDITIONS

Bug 2: an escaping witness

CASE: THE REGISTER RECORD

"Some driver registered the camera; call it y; so y registered the camera."

FLAG: THE FIRST CONDITION ON NAMING A WITNESS

The conclusion $\mathsf{Reg}(y, \mathsf{cam})$ has the made-up name y free, and would outlive the argument.

4.3 · SIDE CONDITIONS

Bug 2: the other two conditions

- The fresh name mustn't be a variable that $\exists x\, A$ already mentions,
- nor one that some other open hypothesis already says something about.
- Either way, "call it y" would be naming a thing already in play.

Together the three say: you may use the thing, and may not learn which thing it is. That's what a module boundary says.

4.3 · SIDE CONDITIONS

Bug 3: variable capture

Drop substitutability from (∀E), and NJ proves $\bot$ from two things any programmer would accept of the natural numbers:

- *every number has a larger one*, $\forall x\, \exists y\, (x < y)$;
- *no number is less than itself*, $\forall x\, \neg(x < x)$.

4.3 · SIDE CONDITIONS

Bug 3: run to $\bot$

Take t to be y, which is *not* substitutable for x in $\exists y\,(x < y)$, and pretend the rule didn't care:

$$\dfrac{\dfrac{\forall x\,\exists y\,(x < y)}{\exists y\,(y < y)}\,\forall\mathrm{E} \qquad \dfrac{\dfrac{\forall x\,\neg(x < x)}{\neg(z < z)}\,\forall\mathrm{E} \quad [z < z]^{1}}{\bot}\,{\to}\mathrm{E}}{\bot}\,\exists\mathrm{E},1$$

FLAG: SUBSTITUTABILITY

The term y would land under $\exists y$. Every other bar is legal.

4.3 · SIDE CONDITIONS

Bug 3: what went wrong

The y we put in meant "some given number". Under $\exists y$ it came to mean "whichever y the quantifier picks". So $\exists y\,(y < y)$ says some number is less than itself, which the hypothesis never said.

Rename first: the hypothesis is equally $\forall x\,\exists z\,(x < z)$, its instance at y is $\exists z\,(y < z)$, and nothing follows but what should.

4.3 · SIDE CONDITIONS

The same bug on the program side

Running a program is substitution, and it needs the same condition. Apply the keep-the-first-argument function to a free y, and run it blindly:

$$(\lambda x.\, \lambda y.\, x)\, y \;\twoheadrightarrow\; \lambda y.\, y$$

- The free y has been captured. The constant function has become the identity.
- Rename first, $(\lambda x.\, \lambda z.\, x)\, y \twoheadrightarrow \lambda z.\, y$, and the function we meant comes out.

4.3 · SIDE CONDITIONS

One bug, one fix

With the free y of type B and the parameter of type A, the program before the step proves $A \to B$ from B. The blind result $\lambda y.\ y$ proves $A \to A$ from nothing.

Capture swaps one theorem for another. The logic side has been known for a century; what Curry–Howard adds is that it's the same condition.

SECTION 4.4

Dependent types

Put the programs back.

4.4 · DEPENDENT TYPES

What type should the open call have?

In the kernel's old code it was an arrow from `Capability` to `Handle`, and we've seen what that cost. It has to do two things the old type couldn't say:

- work for *every* kind, Tuesday's included;
- say in its result *which* kind it was given, so that the handle it makes for the camera is a handle for the camera.

4.4 · DEPENDENT TYPES

Ivanova's new declarations

```
cam : Kind
camCap : Capability cam                    -- granted at install
openHandle : (k : Kind) -> Capability k -> Handle k
readHandle : (k : Kind) -> Handle k -> Bytes k
record : (k : Kind) -> Handle k -> IO ()
```

Read the third line slowly. It takes a kind k, then a capability for k, and returns a handle to k. The result type mentions the argument.

4.4 · DEPENDENT TYPES

What the bug was, in types

OLD CODE

```
h :: Handle
```

The kind and the token were one value. That was the bug.

NEW CODE

```
cam : Kind    h : Handle cam
```

Two values: the kind, and a token whose *type* has the kind in it.

An arrow whose right side depends on what went in on the left. No type system so far can write that.

4.4 · DEPENDENT TYPES

A note on the code

The code is Haskell with one change: an argument the type mentions is named on the arrow.

Haskell doesn't accept it by default, because its types may not mention values. That's a design decision, not a limit of the idea.

4.4 · DEPENDENT TYPES

The long *and*, as data

The long *and* is a record with one field per kind: forty handles. A function from kinds to handles holds the same information: hand it the camera, get the camera's field.

When you don't know how many fields there will be, a function is the only way to write the record down. A proof of for all *is a function from things.*

4.4 · DEPENDENT TYPES

The long *or*, as data

The long *or* is a forty-way `Either`: a value with a tag saying which case it is. Here the tags are drivers, so the value is a driver paired with evidence about that driver.

A proof of there is *is a pair of a thing and evidence about it.*

4.4 · DEPENDENT TYPES

A change of letters

WHO WRITES IT	FOR ALL	THERE IS	AND	OR
logic	$\forall$	$\exists$	$\wedge$	$\vee$
type theory	Π	Σ	$\times$	$+$

Sections 4.2 and 4.3 used the logician's letters. This one uses the type theorist's.

4.4 · DEPENDENT TYPES

Problem 4.24

Decorate the four quantifier rules of NJ with programs, as Chapter 3 decorated ($\to$I) and ($\to$E) with λ and application. What is the program for each of *let x be arbitrary*, *in particular, for t*, *take x to be t* and *let y be such a thing*? And what becomes of the eigenvariable?

The hint: a function from things, and a pair. The next two definitions are the answer.

4.4 · DEPENDENT TYPES

Definition 4.28: types

Fix a signature. Its terms t name things; variables $x, y, z, \ldots$ stand for things. A second stock, $h, k, \ldots$, stands for evidence. The **types** are the formulas, in the type theorist's letters:

$$A, B, C ::= P(t_1, \ldots, t_n) \mid \bot \mid A \to B \mid A \times B \mid A + B \mid \Pi x\, A \mid \Sigma x\, A$$

$\Pi x\, A$ is the dependent function type; $\Sigma x\, A$ the dependent pair type.

4.4 · DEPENDENT TYPES

Definition 4.28, continued: terms

The terms add four forms to the propositional ones:

$$M, N ::= \cdots \mid \lambda x.\, M \mid M\, t \mid \langle t, M \rangle \mid \mathbf{let}\ \langle x, h \rangle = M\ \mathbf{in}\ N$$

An abstraction over a thing; an application to a thing; a pair of a thing and evidence; and the match that opens such a pair.

4.4 · DEPENDENT TYPES

Definition 4.28, continued: contexts

CONTEXTS

A **context** Γ is a finite list of entries: a variable x on its own, which *declares* a thing, or a typing $h : A$, which declares evidence. Nothing is declared twice, and every variable free in a type is declared *earlier* than the entry that mentions it.

A declared x is a thing nobody has chosen yet, a parameter. The four new rules all turn on it.

4.4 · DEPENDENT TYPES

Definition 4.31: typing rules, the old ones

The judgement is $\Gamma \vdash_{\lambda P} M : A$, after λP, its name on the cube of Section 4.7. The propositional rules carry over:

$$\frac{}{\Gamma \vdash_{\lambda P} h : A}\ \text{(var) if } h : A \text{ is in } \Gamma \qquad \frac{\Gamma,\ h : A \vdash_{\lambda P} M : B}{\Gamma \vdash_{\lambda P} \lambda h.\, M : A \to B}\ \text{(abs)}$$

$$\frac{\Gamma \vdash_{\lambda P} M : A \to B \qquad \Gamma \vdash_{\lambda P} N : A}{\Gamma \vdash_{\lambda P} M\, N : B}\ \text{(app)}$$

and those for $\times$, $+$ and $\bot$, as before.

4.4 · DEPENDENT TYPES

Definition 4.31: the rules for Π

$$\frac{\Gamma,\ x \vdash_{\lambda P} M : A}{\Gamma \vdash_{\lambda P} \lambda x.\, M : \Pi x\, A}\ (\Pi\mathrm{I}) \qquad \frac{\Gamma \vdash_{\lambda P} M : \Pi x\, A}{\Gamma \vdash_{\lambda P} M\, t : A[t/x]}\ (\Pi\mathrm{E})$$

(ΠE) if t is in scope: its variables are declared in Γ

- (ΠE) is a function call: a program for every x, called on the t in hand.
- (ΠI) is a definition finished: *proved for an arbitrary x, hence for all.*

Definition 4.31: the rules for Σ

$$\frac{\Gamma \vdash_{\lambda P} M : A[t/x]}{\Gamma \vdash_{\lambda P} \langle t, M \rangle : \Sigma x\, A}\ (\Sigma\mathrm{I})$$

$$\frac{\Gamma \vdash_{\lambda P} M : \Sigma x\, A \qquad \Gamma,\ x,\ h : A \vdash_{\lambda P} N : C}{\Gamma \vdash_{\lambda P} \mathbf{let}\ \langle x, h \rangle = M\ \mathbf{in}\ N : C}\ (\Sigma\mathrm{E})$$

(ΣI) if t is in scope; (ΣE) if $x \notin \mathrm{FV}(C)$

- (ΣI) is the constructor: a thing and evidence about it.
- (ΣE) is the pattern match: name the thing x and use it, but the *type* of the result may not mention x.

4.4 · DEPENDENT TYPES

Why the letters are Π and Σ

The rules for $\times$ and $+$ are these four with the cases cut down to two: a pair with two fixed fields instead of a function from things; a tag from $\{Left, Right\}$ instead of a thing.

A function from things is a product *with one factor per thing. A tagged pair is a member of a* sum *with one summand per thing.*

Peirce's letters, from 1885. Logic traded them for $\forall$ and $\exists$; type theory took them back.

4.4 · DEPENDENT TYPES

Where did the side condition go?

The ($\forall$I) clause asked that its eigenvariable be free in no open hypothesis. Yet (ΠI) carries no proviso.

- In Γ, x the variable x is the *last* thing declared, and an entry may mention only what was declared before it.
- So nothing in Γ can say anything about x.

4.4 · DEPENDENT TYPES

A context is a stack

To generalise over x while a hypothesis about x is in force, discharge the hypothesis first, with a λh. Only then is x on top. That's bug 1 of Section 4.3, refused in another form.

You can only pop x when everything pushed after it has gone.

The one condition the stack can't enforce is written on (ΣE): the result type C isn't in the context. It's the condition a module system enforces.

4.4 · DEPENDENT TYPES

Ivanova's declarations, as one context

The kinds are constants, always in scope. What's left to declare is the evidence:

NAME	TYPE
ch	$\Pi k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))$
hr	$\Pi k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))$
lv	$\Pi d\,\Pi k\,(\mathsf{Reg}(d,k) \to \mathsf{Live}(k))$
rec	$\Pi k\,(\mathsf{Read}(k) \to \mathit{IO}\,())$
cc	$\mathsf{Cap}(\mathsf{cam})$

This is Γ in every tree below; whatever a rule adds to it is written out.

4.4 · DEPENDENT TYPES

Example 4.32: the camera handle

First the handle, from the theory's first sentence and the capability:

$$\dfrac{\dfrac{\overline{\Gamma \vdash_{\lambda P} \mathsf{ch} : \Pi k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}^{\text{var}}}{\Gamma \vdash_{\lambda P} \mathsf{ch}\,\mathsf{cam} : \mathsf{Cap}(\mathsf{cam}) \to \mathsf{Hand}(\mathsf{cam})}^{\Pi\text{E}} \qquad \overline{\Gamma \vdash_{\lambda P} \mathsf{cc} : \mathsf{Cap}(\mathsf{cam})}^{\text{var}}}{\Gamma \vdash_{\lambda P} \mathsf{ch}\,\mathsf{cam}\,\mathsf{cc} : \mathsf{Hand}(\mathsf{cam})}^{\text{app}}$$

The (ΠE) *bar picks the kind:* $t := \mathsf{cam}$. *Then (app) hands it the capability.*

4.4 · DEPENDENT TYPES

Example 4.32: the read ticket

Then the ticket, from the theory's second sentence and the handle, written (1):

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{\lambda P} \mathsf{hr} : \Pi k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}\,\text{var}}{\Gamma \vdash_{\lambda P} \mathsf{hr}\,\mathsf{cam} : \mathsf{Hand}(\mathsf{cam}) \to \mathsf{Read}(\mathsf{cam})}\,\Pi\text{E} \qquad (1)}{\Gamma \vdash_{\lambda P} \mathsf{hr}\,\mathsf{cam}\,(\mathsf{ch}\,\mathsf{cam}\,\mathsf{cc}) : \mathsf{Read}(\mathsf{cam})}\,\text{app}$$

The program at the root is the ticket, and its type has the camera's kind in it.

Example 4.33: the wrong-kind read

The app's handler passes the camera ticket to `record mic`. The checker gets as far as the ticket:

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda P} \mathsf{rec} : \Pi k\,(\mathsf{Read}(k) \to IO\,())}{\Gamma \vdash_{\lambda P} \mathsf{rec}\,\mathsf{mic} : \mathsf{Read}(\mathsf{mic}) \to IO\,()}\ \Pi\mathrm{E} \qquad \Gamma \vdash_{\lambda P} \mathit{ticket} : \mathsf{Read}(\mathsf{cam})}{\text{fails: expected } \mathsf{Read}(\mathsf{mic}),\ \text{got } \mathsf{Read}(\mathsf{cam})}\ \text{app}$$

The type confusion is a type error, and the error says which kind the handle was for.

4.4 · DEPENDENT TYPES

Example 4.34: every kind may be read

In the context $\Gamma, k, h : \mathsf{Cap}(k)$ the camera trees, with k for cam and h for cc, give $\mathsf{hr}\, k\, (\mathsf{ch}\, k\, h) : \mathsf{Read}(k)$. Two bars close it:

$$\dfrac{\dfrac{\Gamma,\ k,\ h : \mathsf{Cap}(k) \vdash_{\lambda P} \mathsf{hr}\, k\, (\mathsf{ch}\, k\, h) : \mathsf{Read}(k)}{\Gamma,\ k \vdash_{\lambda P} \lambda h.\, \mathsf{hr}\, k\, (\mathsf{ch}\, k\, h) : \mathsf{Cap}(k) \to \mathsf{Read}(k)}\ \text{abs}}{\Gamma \vdash_{\lambda P} \lambda k.\, \lambda h.\, \mathsf{hr}\, k\, (\mathsf{ch}\, k\, h) : \Pi k\, (\mathsf{Cap}(k) \to \mathsf{Read}(k))}\ \Pi\mathrm{I}$$

4.4 · DEPENDENT TYPES

Example 4.34: the order is the side condition

- The (abs) bar pops h, the hypothesis about k. Then k is on top, and (III) pops it.
- The other way round, (III) has nothing to apply to: k isn't the last entry.

Function composition with a kind threaded through it. As a sentence: every kind with a capability may be read.

Example 4.35: the register rule

CASE: BOOT

The `v4l` driver registers the camera, and the kernel records $\langle \mathsf{v4l}, \mathsf{rc} \rangle : \Sigma d\, \mathsf{Reg}(d, \mathsf{cam})$: the driver and the evidence packed together, in a value whose type says only that *some* driver registered the camera.

Before it hands out camera handles the kernel needs $\mathsf{Live}(\mathsf{cam})$. *The register rule gives it, once the pair is opened.*

4.4 · DEPENDENT TYPES

Example 4.35: opening the pair

Write S for $\Sigma d\,\mathsf{Reg}(d, \mathsf{cam})$, with the record held as a. Step (2) calls the rule at y and cam and applies it to z:

$$\dfrac{\dfrac{}{\Gamma,\ a : S \vdash_{\lambda P} a : S}\ \text{var} \qquad (2)\ \Gamma,\ a : S,\ y,\ z : \mathsf{Reg}(y, \mathsf{cam}) \vdash_{\lambda P} \mathsf{lv}\, y\, \mathsf{cam}\, z : \mathsf{Live}(\mathsf{cam})}{\Gamma,\ a : S \vdash_{\lambda P} \mathbf{let}\ \langle y, z\rangle = a\ \mathbf{in}\ \mathsf{lv}\, y\, \mathsf{cam}\, z : \mathsf{Live}(\mathsf{cam})}\ \Sigma\text{E}$$

The side condition holds: $\mathsf{Live}(\mathsf{cam})$ *doesn't mention* y.

4.4 · DEPENDENT TYPES

Example 4.35: the escape

Suppose the handler tried to conclude which driver registered the camera:

$$\textbf{let } \langle y, z \rangle = a \textbf{ in } z \ : \ \mathsf{Reg}(y, \mathsf{cam}) \qquad \text{refused: } y \text{ is free in } \mathsf{Reg}(y, \mathsf{cam})$$

- Inside the match, $z : \mathsf{Reg}(y, \mathsf{cam})$ is perfectly good.
- Outside it there's no y. A type that mentions it mentions nothing.

Bug 2 of Section 4.3, caught by the result type.

4.4 · DEPENDENT TYPES

Definition 4.36: β-reduction

A step rewrites an introduction met by its elimination, anywhere inside a term. Two new rules join Chapter 3's, with a rule for each place a subterm can sit:

$$\frac{}{(\lambda x.\, P)\, t \;\rightsquigarrow\; P[t/x]}\;\beta \qquad\qquad \frac{}{\mathbf{let}\ \langle x, h\rangle = \langle t, P\rangle\ \mathbf{in}\ Q \;\rightsquigarrow\; Q[t/x][P/h]}\;\mathbf{let}$$

- Call a function on a thing: the thing goes in for the parameter, as in β.
- Open a freshly packed pair: thing and evidence go in for the two names, as in a `case` on a tag.

4.4 · DEPENDENT TYPES

The central table gains two rows

CONNECTIVE	TYPE	CONSTRUCTOR	PATTERN MATCH
$A \to B$	$A \to B$	$\lambda h.\, M$	application $M\, N$
$A \wedge B$	$A \times B$	(M, N)	fst, snd
$A \vee B$	$A + B$	$Left\, M$, $Right\, M$	`case` with two branches
$\bot$	$\bot$	—	$absurd$
$\forall x\, A$	$\Pi x\, A$	$\lambda x.\, M$	application $M\, t$
$\exists x\, A$	$\Sigma x\, A$	$\langle t, M \rangle$	`let` with two names

4.4 · DEPENDENT TYPES

De Bruijn, 1967

- Types that mention values weren't invented for Ivanova.
- In 1967 de Bruijn, at Eindhoven, set out to have a machine check a whole mathematics textbook.
- It couldn't be done unless the types could mention the things the theorems were about.

Nicolaas de Bruijn, 1918–2012

4.4 · DEPENDENT TYPES

Automath checks Landau

His language, Automath, has the rules above. In 1977 his student Jutting fed it the whole of Landau's *Foundations of Analysis*, and it checked.

> "Please forget everything you have learned in school; for you haven't learned it."
>
> Landau's preface for the student. A clerk could ask for no better source.

SECTION 4.5

Curry–Howard with quantifiers

Rub out the programs, and rename.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Rub out the programs

Delete the programs from the four typing rules, keep contexts and formulas, and rename $\Pi \mapsto \forall$, $\Sigma \mapsto \exists$:

$$\frac{\Gamma,\ x \vdash_{\mathrm{NJ}} A}{\Gamma \vdash_{\mathrm{NJ}} \forall x\, A} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} \forall x\, A}{\Gamma \vdash_{\mathrm{NJ}} A[t/x]} \qquad \frac{\Gamma \vdash_{\mathrm{NJ}} A[t/x]}{\Gamma \vdash_{\mathrm{NJ}} \exists x\, A}$$

$$\frac{\Gamma \vdash_{\mathrm{NJ}} \exists x\, A \qquad \Gamma,\ x,\ A \vdash_{\mathrm{NJ}} C}{\Gamma \vdash_{\mathrm{NJ}} C}$$

side conditions unchanged; a declaration x is kept, since it's a name and not evidence

The four rules of NJ, written on the turnstile. Chapter 3's correspondence extends.

Theorem 4.37: Curry–Howard, with quantifiers

THEOREM 4.37 · CURRY–HOWARD, WITH QUANTIFIERS

Erasing the terms from a typing, and renaming, turns it into a derivation of NJ; decorating a derivation with terms turns it into a typing. Writing $|\Gamma|$ for the types of the evidence entries of Γ,

$$\Gamma \vdash_{\lambda P} M : A \text{ for some } M \quad \text{if and only if} \quad |\Gamma| \vdash_{\mathrm{NJ}} A,$$

for every context Γ that declares the variables free in A.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Two kinds of variable

- Erasing $\lambda h.\, M$ deletes h: evidence has nothing to say once the evidence is gone.
- Erasing $\lambda x.\, M$ does *not* delete x: a declared thing is a name, and $\forall x\, A$ still needs it.

A logic keeps its variables when it loses its programs.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

What the tree records instead of the stack

A tree has no declarations. It records what a context records by computing OH.

THE TYPE SYSTEM SAID

x is the last thing declared.

THE ($\forall$I) CLAUSE SAYS

y is free in no open hypothesis: the same fact, read off the leaves.

The tree says it more sharply: free in no hypothesis used, *not merely* available. *Nothing provable changes.*

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Example 4.38: the worked derivations, decorated

By the theorem each worked derivation has a program. The theory's sentences are ch, hr, lv; cc is the camera capability.

EXAMPLE	PROGRAM
4.12, the boot log	$\mathsf{lv}\,\mathsf{v4l}\,\mathsf{cam}\,r$, with r the log line
4.13, the driver guide	$\lambda u.\,\lambda h.\,(\mathsf{ch}\,u\,h,\ \mathsf{hr}\,u\,(\mathsf{ch}\,u\,h))$
4.14, anything at all?	$\langle \mathsf{cam},\ \mathsf{hr}\,\mathsf{cam}\,(\mathsf{ch}\,\mathsf{cam}\,\mathsf{cc})\rangle$
4.15, the manifest	$\mathbf{let}\ \langle y, h\rangle = a\ \mathbf{in}\ \langle y,\ \mathsf{hr}\,y\,(\mathsf{ch}\,y\,h)\rangle$

Example 4.38, continued

EXAMPLE	PROGRAM
4.16, every kind has a driver	$\lambda u.\ \langle \mathsf{owner}(u),\ o\,u\rangle$
4.17, the sandbox	$\lambda u.\,\lambda h.\ n\,\langle u, h\rangle$
4.18, the pager	$\lambda u.\,\lambda h.\ f\,\langle u, h\rangle$ and $\lambda a.\ \mathbf{let}\ \langle y, h\rangle = a\ \mathbf{in}\ g\,y\,h$
4.19, the horse	$\lambda y.\,\lambda a.\ \mathbf{let}\ \langle z, p\rangle = a\ \mathbf{in}\ \langle z,\ (h a\, z\,(f s t\, p),\ s n d\, p)\rangle$

Each says in a line what its tree said in a slide.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Read down the column

- Two (∀E) bars and a (→E) are a function called on two things and then on evidence.
- A proof of *there is* is a pair whose first half is the witness: ask 4.14 *which kind?* and it says cam.
- A hypothesis *there is* is opened by a **let**, which uses the kind it finds and never learns which it was.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Read down the column, continued

- The sandbox packs a kind and a supposed proof together, and the specification turns any such pair into $\bot$.
- The pager's two programs are a function on pairs and its curried form: currying, one level up.
- The horse: open the pair, keep the horse, swap *it's a horse* for *it's an animal*, and pack it up again.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Detours, with quantifiers

The theorems about running programs carry over, each with two more cases.

- A lemma proved for an arbitrary x and then used at t: ($\forall$I) met by ($\forall$E), removed by $(\lambda x.\, M)\, t \rightsquigarrow M[t/x]$.
- A witness packed and at once unpacked: ($\exists$I) met by ($\exists$E), removed by the step for **let**.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Theorem 4.39: subject reduction

THEOREM 4.39 · SUBJECT REDUCTION

If $\Gamma \vdash_{\lambda P} M : A$ and $M \twoheadrightarrow N$, then $\Gamma \vdash_{\lambda P} N : A$.

Running a program never changes its type.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Theorem 4.40: normalisation

THEOREM 4.40 · NORMALISATION

Every typed term runs to a normal form. Equivalently, every derivation of NJ can be rewritten into one without detours. Prawitz 1965

Without $+$ and Σ, proved by counting, as before: a term of the signature names a thing and is never an introduction form, so putting it in creates no redex.

Σ breaks the count as $+$ does, for the same reason. The rest is on loan.

Proposition 4.41: closed normal forms

PROPOSITION 4.41 · CLOSED NORMAL FORMS

A closed typed term in normal form is an introduction form: $\lambda h.\, M$, (M, N), $\mathit{Left}\, M$, $\mathit{Right}\, M$, $\lambda x.\, M$ or $\langle t, M \rangle$.

So a closed derivation runs to a closed normal term of the same type: an introduction form. Now look at what the introduction forms can be.

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Theorem 4.24 and its corollaries, proved

- No introduction form has type $\bot$ or an atomic type: **consistency**, Theorem 4.24.
- A closed normal term of type $A + B$ is $\mathit{Left}\, M$ or $\mathit{Right}\, M$, and its tag says which: the **disjunction property**.
- A closed normal term of type $\Sigma x\, A$ is $\langle t, M \rangle$, and its first half is the witness: the **existence property**.

A proof of there is *names the thing. From a* hypothesis $\exists x\, A$ *you can't: a variable names nothing.*

4.5 · CURRY–HOWARD WITH QUANTIFIERS

Finding a tree, with quantifiers

A normal derivation has introductions near the root, over eliminations that take hypotheses apart. Chapter 3's method gains a line for each quantifier:

IF	THEN
the conclusion starts with $\forall$	the last bar is probably ($\forall$I): *let u be any kind*
the conclusion starts with $\exists$	the last bar is probably ($\exists$I); leave the witness till last, as with ($\vee$I)
a hypothesis starts with $\exists$	open it early with ($\exists$E) and name the thing, as with ($\vee$E)

Example 4.19, found that way

$$\Gamma_{\text{pol}},\ \exists k\,\mathsf{Cap}(k)\ \vdash_{\text{NJ}}\ \exists k\,\mathsf{Read}(k)$$

"from the driver theory and a capability for some kind or other: some kind may be read"

- A hypothesis starts with $\exists$: the ($\exists$E) bar is the root, the first decision made.
- The witness y for the ($\exists$I) bar was chosen only after that bar had named it.

SECTION 4.6

Theories

Where everything about the camera comes from.

4.6 · THEORIES

Read the consistency theorem with names in it

NJ proves no atomic sentence: neither $\mathsf{Cap}(\mathsf{cam})$ nor $\mathsf{Read}(\mathsf{mic})$ nor $2 < 3$. Whatever it proves outright is about how claims relate.

Every fact about a particular kind, element or day has to come from the hypotheses.

4.6 · THEORIES

Theories

SPECIFICATION

A set of sentences taken as hypotheses. What we prove, we prove *from* it.

Three, in increasing order of interest: the driver theory, a comparator's contract, a sale.

4.6 · THEORIES

Example 4.43: the driver theory

From Γ_{pol} and $\mathsf{Cap}(\mathsf{cam})$, the camera tree with its programs rubbed out witnesses

$$\Gamma_{\text{pol}},\ \mathsf{Cap}(\mathsf{cam}) \vdash_{\text{NJ}} \mathsf{Read}(\mathsf{cam})$$

"from the driver theory and a camera capability: the holder may read the camera"

- The tree is a certificate. An auditor needn't rerun the kernel; they check four bars.
- Since NJ proves no atomic sentence on its own, $\mathsf{Read}(\mathsf{cam})$ came from the driver theory and nowhere else.

4.6 · THEORIES

Example 4.43: every kind

From Γ_{pol} alone, the tree read aloud in Section 4.2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall k\,(\mathsf{Hand}(k) \to \mathsf{Read}(k))}{\mathsf{Hand}(u) \to \mathsf{Read}(u)}\,\forall\mathrm{E} \quad \dfrac{\dfrac{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Hand}(k))}{\mathsf{Cap}(u) \to \mathsf{Hand}(u)}\,\forall\mathrm{E} \quad [\mathsf{Cap}(u)]^{1}}{\mathsf{Hand}(u)}\,{\to}\mathrm{E}}{\mathsf{Read}(u)}\,{\to}\mathrm{E}}{\mathsf{Cap}(u) \to \mathsf{Read}(u)}\,{\to}\mathrm{I},\,1}{\forall k\,(\mathsf{Cap}(k) \to \mathsf{Read}(k))}\,\forall\mathrm{I}$$

No runtime check settles this by evaluation: it quantifies over every kind at once. It is Example 4.34 with the programs gone, bar for bar.

4.6 · THEORIES

Example 4.44: a comparator

Insertion sort, with the comparison a parameter, because a library can't know what you're sorting:

```
sort(lt, xs):
    out = []
    for x in xs:
        i = 0
        while i < length(out) and lt(out[i], x): i = i + 1
        insert x into out at position i
    return out
```

$$sort : \Pi a \left((a \to a \to Bool) \to [a] \to [a] \right)$$

"for every type a: given a comparison on a and a list of a, `sort` returns a list of a"

4.6 · THEORIES

What the type certifies

The checker proves that type, so it has certified something real: no string added to a number, no non-function called.

About sorting it has certified nothing whatever.

4.6 · THEORIES

The gap is the parameter

CASE: THE COMMONEST MISTAKE

Give `sort` a comparison that returns `True` for equal arguments: `<=` where `<` was meant. The type doesn't change. The program compiles, runs, and sometimes returns a list that isn't sorted.

AN EXCEPTION JAVA'S SORT REALLY THROWS

```
IllegalArgumentException: Comparison method violates its general
contract!
```

4.6 · THEORIES

What the exception means

The sort hasn't checked the contract; that would cost more than the sort. It noticed its own bookkeeping had become impossible: a *consequence* of the contract had failed.

To say what that means, state the contract and derive the consequence.

4.6 · THEORIES

The contract, Γ_{ord}

What `sort` needs of `lt` is a *strict weak order*. With $x < y$ for `lt(x,y)`:

WHAT THE LIBRARY NEEDS	AS A SENTENCE
nothing is less than itself	$\forall x\, \neg(x < x)$
if $x < y$ and $y < z$ then $x < z$	$\forall x\, \forall y\, \forall z\, (x < y \wedge y < z \to x < z)$
if $x < y$, every z is above x or below y	$\forall x\, \forall y\, \forall z\, (x < y \to x < z \vee z < y)$

They say for all. *No propositional type could even write them down.*

4.6 · THEORIES

The consequence: asymmetry

A comparator built from `<=` reports $a < b$ and $b < a$ for equal elements. No comparator obeying the contract can, because

$$\Gamma_{\text{ord}} \;\vdash_{\text{NJ}}\; \forall x\, \forall y\, \neg(x < y \wedge y < x)$$

"from the contract: no two elements are each less than the other"

Informally, one line: if $u < v$ and $v < u$, then by transitivity $u < u$, which irreflexivity forbids.

4.6 · THEORIES

Asymmetry, derived: the first piece

$\mathcal{D}$ takes transitivity down to $u < u$, on the supposition:

$$\mathcal{D} \;=\; \dfrac{\dfrac{\dfrac{\dfrac{\forall x\,\forall y\,\forall z\,(x<y \wedge y<z \to x<z)}{\forall y\,\forall z\,(u<y \wedge y<z \to u<z)}\,\forall\mathrm{E}}{\forall z\,(u<v \wedge v<z \to u<z)}\,\forall\mathrm{E}}{u<v \wedge v<u \to u<u}\,\forall\mathrm{E} \qquad [u<v \wedge v<u]^{1}}{u<u}\,{\to}\mathrm{E}$$

Three ($\forall E$) bars choose $t := u$, v, u. The last choice is the step that says "then $u < u$".

4.6 · THEORIES

Asymmetry, derived: the second piece

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x\,\neg(x<x)}{\neg(u<u)}\,\forall\mathrm{E} \qquad \begin{array}{c}\mathcal{D}\\ u<u\end{array}}{\bot}\,\to\mathrm{E}}{\neg(u<v\wedge v<u)}\,\to\mathrm{I},\,1}{\forall y\,\neg(u<y\wedge y<u)}\,\forall\mathrm{I}}{\forall x\,\forall y\,\neg(x<y\wedge y<x)}\,\forall\mathrm{I}$$

The order is forced. ($\forall I$) for v can't come above the discharge: v is free in the supposition while it is open. The stack again.

4.6 · THEORIES

Now decorate it

$$\lambda u.\, \lambda v.\, \lambda h.\;\; irrefl\, u\, (trans\, u\, v\, u\, h) \;:\; \Pi x\, \Pi y\, \neg (x < y \times y < x)$$

"a function that turns any two elements, and evidence that each is below the other, into a contradiction"

- Given two elements and a claimed pair of comparisons h, call transitivity at u, v, u on h, and hand the proof of $u < u$ to irreflexivity.
- Nine bars of proof, one line of program: three λs, two for *let … be arbitrary* and one for the supposition.

4.6 · THEORIES

The exception, read again

When the sort catches a comparator reporting both $a < b$ and $b < a$, the exception is reporting a theorem *being violated.*

The type checker couldn't state it. A verifier, a type checker whose types may say *for all*, can state it and check this proof.

4.6 · THEORIES

Example 4.45: the sale

CASE: THE PATCH

A bug report said “discount not applied on the last day of the sale”. A patch changed `<=` to `<` at both ends. The tests passed, and the discount quietly disappeared from the first day.

4.6 · THEORIES

Example 4.45: what a sale is

With $\mathsf{Disc}(d)$ for "the discount applies on day d", one sentence:

$$\forall d\,(start \leq d \wedge d \leq end \rightarrow \mathsf{Disc}(d))$$

"on every day from the first day of the sale to the last, both included, the discount applies"

- The patched code fails the sentence at $d := start$.
- Nobody had written the sentence down, which is how the patch got approved.

Once it's written, "is the patch correct?" has an answer a checker can verify. The answer is no.

4.6 · THEORIES

Witnesses

“Is this file readable by anyone outside the team?” is an existence question. A *yes* that can’t say *who* is no use to a security engineer.

In NJ, a yes *proved from no hypotheses comes with the name attached.*

4.6 · THEORIES

The price of a name

The insistence on a name is also worth distrusting.

- *Either the program halts or it doesn't*: you'd like to assert it without a halting run or a proof there's none.
- *Some input breaks this function*: you'd like to assert it without producing the input.

Here you may assert neither. Virtue or restriction? A question for Part 2.

SECTION 4.7

History

Invented twice, like NJ.

4.7 · HISTORY

Twice

FREGE, 1879

The quantifier, in two dimensions. Almost nobody read the book. The notation lost; the analysis won.

PEIRCE AND MITCHELL, 1883–85

Π and Σ, knowing nothing of Frege. Carried by Schröder to Peano, and by Peano to Russell.

4.7 · HISTORY

Then the letters came back

- Peano turned Σ into $\exists$ in 1897; Gentzen added $\forall$ in 1934.
- Hilbert and Ackermann's textbook of 1928 set out first-order logic on its own, and asked the question Part 2 answers.
- Then Π and Σ came back as types: Howard, de Bruijn, Martin-Löf, Girard and Reynolds.

4.7 · HISTORY

Footnote: June 1902

With volume two of his *Basic Laws* at the printer, Frege received a polite letter from the young Bertrand Russell, showing that one of its laws led to a contradiction.

> "Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished."
>
> Frege's appendix. The quantifier survived; the edifice didn't.

4.7 · HISTORY

Footnote: Peirce

Some historians think the student taught the teacher.

Peirce's one university post was the lectureship that gave him Mitchell. He lost it in 1884, and died poor in 1914.

4.7 · HISTORY

Footnote: September 1930

Hilbert, at Königsberg, on the radio: *Wir müssen wissen. Wir werden wissen.* We must know; we will know. It's on his gravestone.

The day before, at a round table in the same meeting, a twenty-four-year-old named Gödel had announced a theorem.

4.7 · HISTORY

Who did what, 1879–1934

WHO	WHEN	WHAT
Frege	1879	function and argument; the quantifier, in two dimensions
Peirce, Mitchell	1883–85	the quantifier again, as Π and Σ
Peano	1897	$\exists$
Hilbert & Ackermann	1928	first-order logic as a system of its own
Gentzen	1934	$\forall$; NJ's quantifier rules, with their side conditions

4.7 · HISTORY

Who did what, 1965–1991

WHO	WHEN	WHAT
Prawitz	1965	normalisation, quantifiers included
de Bruijn	1968	Automath: types that mention things, in a checker
Howard	1969	the quantifiers as a function from things and a pair
Girard; Reynolds	1971; 1974	Π over types: generics
Martin-Löf	1972–	Π and Σ; a foundation on dependent types
Mitchell & Plotkin	1988	modules as existential types
Barendregt	1991	the λ-cube

SECTION 4.7

The λ-cube

Eight type systems in one picture.

4.7 · THE λ-CUBE

A dozen calculi, one difference

By the end of the 1980s there were a dozen typed λ-calculi, each with its own notation, rules and normalisation proof.

> *Barendregt saw that they differ in one respect only: what is allowed to depend on what.*

4.7 · THE λ-CUBE

What may depend on what

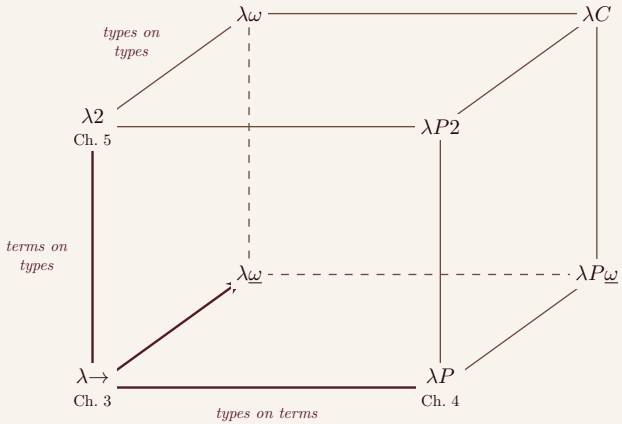
A term can always depend on a term: that's a function. Three more dependencies are possible, and you've met each.

DEPENDENCY	LOOKS LIKE	WHERE YOU MET IT
terms on types	$\Lambda a.\, M$, a program that takes a type	generics: Chapter 5
types on terms	$\mathsf{Hand}(k)$, a type that takes a thing	dependent types, Section 4.4
types on types	$[a]$, a type that takes a type	every `List<String>` you've written

Each can be switched on or off: $2^3 = 8$ *calculi, the corners of a cube.*

4.7 · THE λ-CUBE

The cube



4.7 · THE λ-CUBE

Walking the edges: the near corners

- $\lambda{\to}$, the corner nearest you: the simply typed λ-calculus, Church's of 1940.
- Up: $\lambda 2$, Girard's and Reynolds's System F, the subject of Chapter 5.
- Right: λP, types that mention things. This chapter, and the heart of Automath and of the logical framework LF (1987).

4.7 · THE λ-CUBE

Walking the edges: the far corners

- Back: $\lambda\underline{\omega}$, types that take types. Up and back: $\lambda\omega$, Girard's Fω.
- The far corner, all three on: λC, the calculus of constructions (Coquand and Huet 1988).

λC is the core of a proof assistant thousands now use.

4.7 · THE λ-CUBE

The cube of logics

THROUGH CURRY–HOWARD

Every corner is a logic too. $\lambda{\to}$ is the logic of implication; λP is first-order NJ, this chapter's; $\lambda 2$ is second-order propositional logic, Chapter 5's; λC is higher-order logic of every kind at once.

Chapter 3 stood at one corner. This chapter walked one edge; Chapter 5 walks another.

4.7 · THE λ-CUBE

More than tidying up

Every corner is a part of λC. So one proof that the programs of λC all stop covers all eight calculi at once, and with it the consistency of all eight logics.

And the cube marks its own edge. One step beyond, let the type of all types be a type, $\mathsf{Type} : \mathsf{Type}$*, and everything is provable: Girard's paradox, in Chapter 5.*

4.7 · THE λ-CUBE

A name

The proof assistant was called Coq: after the calculus of constructions, after Thierry Coquand, and after the rooster.

In 2025 it was renamed Rocq. Anyone who has said the old name aloud in an English-speaking lecture theatre can guess at the unofficial reasons.

SUMMARY

Chapter 4, the logic

- Frege's **quantifiers** put a *some* inside an *every*: $\forall x\, A$ is an *and* with a case for every thing, $\exists x\, A$ an *or* whose tags are things.
- NJ's four quantifier rules are four phrases. Two carry a condition on their **eigenvariable**, and breaking the conditions gives three bugs you've met.
- NJ proves no atomic sentence, and a proof of *there is* names the thing.

SUMMARY

Chapter 4, the programs

- A proof of *for all* is a **function from things**; a proof of *there is* a **pair** of a thing and evidence. The context, a stack, enforces every freshness condition but one.
- Erase the programs and the rules are NJ's, bar for bar (Theorem 4.37). Every program stops, and consistency comes from that.
- A **specification** is a set of sentences. A verifier is a type checker whose types may say *for all*.

LOOKING AHEAD

Chapter 5

Let a quantifier range over *propositions*, and every schema becomes a single theorem, $\forall X\,(X \to X)$.

Put the programs back and out comes *generics*: the `a` in `length :: [a] -> Int`.

A puzzle: write a program of type `((a -> b) -> a) -> a`, without recursion and without `undefined`.

CHAPTER 5

Second-order logic; then generics, and its programs.

READING

Sources

- Sørensen and Urzyczyn (2006), *Lectures on the Curry–Howard Isomorphism*, chapter 8. The first-order correspondence at this level.
- Howard (1980), its second half; de Bruijn (1970). The quantifiers first read as a function and a pair; a checker whose types mention things.
- Mitchell and Plotkin (1988); Wadler (2015). Modules as existential types; the story told to programmers.
- Barendregt (1991); van Heijenoort (1967). The cube in its inventor's words; Frege to Gödel in English.

Thank you.