Actema
or
"How to do proofs by hand"

Comment faire des preuves avec la main

Cambium @Inria Paris, Nov 28th 2024

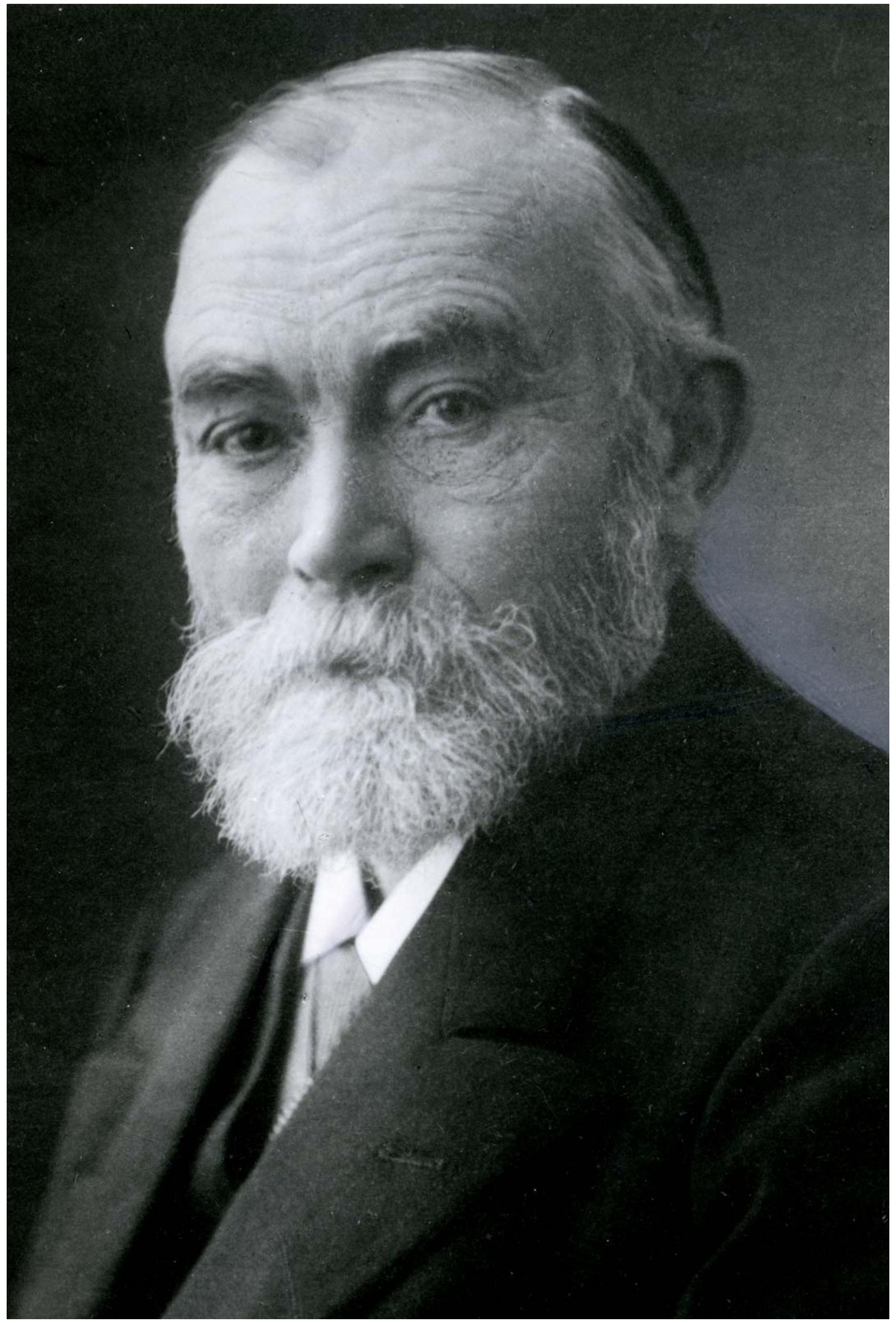
Mathis Bouverot-Dupuis, Kaustuv Chaudhuri, Pierre-Yves Strub, Benjamin Werner

LIX, Ecole polytechnique
EPC Partout, Inria-Saclay

Some preliminary remarks:

- You will see a protoype; one question is how to proceed from here on regarding software development
- Dual motivation:
  - More intuitive way to do proofs
  - Quicker way to do proofs (very often)
- We started looking at this through the HMI angle, and since then, we are excited !

Formal Proofs since 1879



Gottlob Frege
Begriffsschrift

A language of formulae
of pure thought,
imitated from the language of arithmetic

(and also Cantor, Hilbert, Russell, Gödel…)


350 PROLEGOMENA TO CARDINAL ARITHMETIC [PART II

*52·42. $\vdash :. \alpha \epsilon 1 . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \cap \beta \epsilon 1$

Dem.

$\vdash . *51{\cdot}31 . \supset \vdash :. \exists ! \iota`x \cap \beta . \equiv . \iota`x \cap \beta = \iota`x :.$

$[*20{\cdot}53] \supset \vdash :. \alpha = \iota`x . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \cap \beta = \iota`x :.$

$[*10{\cdot}11{\cdot}28] \supset \vdash :. (\exists x) . \alpha = \iota`x . \supset : (\exists x) : \exists ! \alpha \cap \beta . \equiv . \alpha \cap \beta = \iota`x :$

$[*10{\cdot}37] \supset : \exists ! \alpha \cap \beta . \supset . (\exists x) . \alpha \cap \beta = \iota`x$ (1)

$\vdash . (1) . *52{\cdot}1 . \supset \vdash :. \alpha \epsilon 1 . \supset : \exists ! \alpha \cap \beta . \supset . \alpha \cap \beta \epsilon 1$ (2)

$\vdash . *52{\cdot}16 . \supset \vdash : \alpha \cap \beta \epsilon 1 . \supset . \exists ! \alpha \cap \beta$ (3)

$\vdash . (2) . (3) . \supset \vdash .$ Prop

*52·43. $\vdash : \alpha \epsilon 1 . \exists ! \alpha \cap \beta . \equiv . \alpha \epsilon 1 . \alpha \cap \beta \epsilon 1$ [*52·42 . *5·32]

*52·44. $\vdash :. \alpha \epsilon 1 . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \subset \beta . \equiv . \alpha \cap \beta = \alpha$

Dem.

$\vdash . *51{\cdot}31 . \supset \vdash : \exists ! \iota`x \cap \beta . \equiv . \iota`x \subset \beta :$

$[*13{\cdot}13 . \text{Exp}] \supset \vdash :. \alpha = \iota`x . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \subset \beta :.$

$[*10{\cdot}11{\cdot}23] \supset \vdash :. (\exists x) . \alpha = \iota`x . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \subset \beta :.$

$[*52{\cdot}1] \supset \vdash :. \alpha \epsilon 1 . \supset : \exists ! \alpha \cap \beta . \equiv . \alpha \subset \beta$ (1)

$\vdash . (1) . *22{\cdot}621 . \supset \vdash .$ Prop

*52·45. $\vdash :: \alpha , \beta \epsilon 1 . \supset :. \alpha \subset \beta \cup \gamma . \equiv : \alpha = \beta . v . \alpha \subset \gamma$

Dem.

$\vdash . *51{\cdot}236 \frac{x, y, \gamma}{z, x, \beta} . \supset$

$\vdash :. x \epsilon \iota`y \cup \gamma . \equiv : x = y . v . x \epsilon \gamma :.$

$[*51{\cdot}2{\cdot}23] \supset \vdash :. \iota`x \subset \iota`y \cup \gamma . \equiv : \iota`x = \iota`y . v . \iota`x \subset \gamma :.$

$[*13{\cdot}21] \supset \vdash :: \alpha = \iota`x . \beta = \iota`y . \supset :. \alpha \subset \beta \cup \gamma . \equiv : \alpha = \beta . v . \alpha \subset \gamma ::$

$[*11{\cdot}11{\cdot}35] \supset \vdash :: (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset :. \alpha \subset \beta \cup \gamma . \equiv : \alpha = \beta . v . \alpha \subset \gamma$ (1)

$\vdash . (1) . *52{\cdot}1 . \supset \vdash .$ Prop

*52·46. $\vdash :. \alpha , \beta \epsilon 1 . \supset : \alpha \subset \beta . \equiv . \alpha = \beta . \equiv . \exists ! (\alpha \cap \beta)$

Dem.

$\vdash . *51{\cdot}2{\cdot}23 . \supset \vdash : \iota`x \subset \iota`y . \equiv . \iota`x = \iota`y$ (1)

$\vdash . (1) . *13{\cdot}21 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \subset \beta . \equiv . \alpha = \beta$ (2)

$\vdash . (2) . *11{\cdot}11{\cdot}35 . *52{\cdot}1 . \supset \vdash :. \alpha , \beta \epsilon 1 . \supset : \alpha \subset \beta . \equiv . \alpha = \beta$ (3)

$\vdash . (3) . *52{\cdot}44 \supset \vdash .$ Prop

*52·6. $\vdash :. \alpha \epsilon 1 . \supset : x \epsilon \alpha . \equiv . \iota`x = \alpha . \equiv . x = \breve{\iota}`\alpha$

Dem.

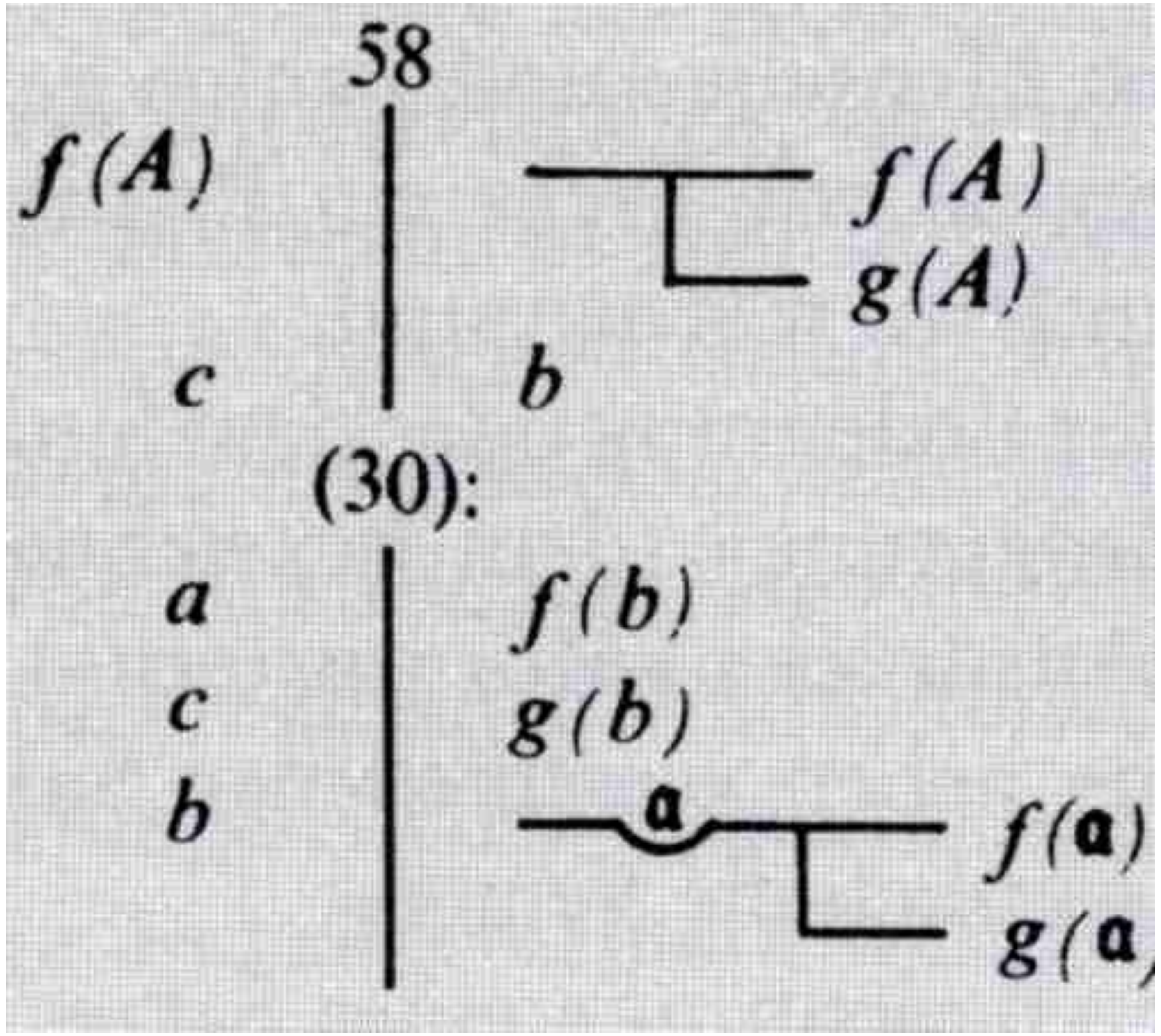
$\vdash . *51{\cdot}23 . \supset \vdash : x \epsilon \iota`y . \equiv . \iota`x = \iota`y :$

$[*13{\cdot}13 . \text{Exp}] \supset \vdash :. \alpha = \iota`y . \supset : x \epsilon \alpha . \equiv . \iota`x = \alpha :.$

$[*10{\cdot}11{\cdot}23 . *52{\cdot}1] \supset \vdash :. \alpha \epsilon 1 . \supset : x \epsilon \alpha . \equiv . \iota`x = \alpha .$ (1)

[*51·51] (2)

$\vdash . (1) . (2$



At the time: a branch of mathematics with no application

Enters the computer

1918-2012

N.G. de Bruijn 1967

Automath

First *proof system*

Formal objects and formal proofs are actually constructed.

They exist and are verified in the computer.

One modern proof-system : Coq

Why formal proofs ?

1. Because we can !

2. When we need to be sure !

   really really sure:

   - When human life is at stake,

   - When large amounts of money are at stake

Proofs that software is correct !

Formal proofs in practice

Two levels of formal language:

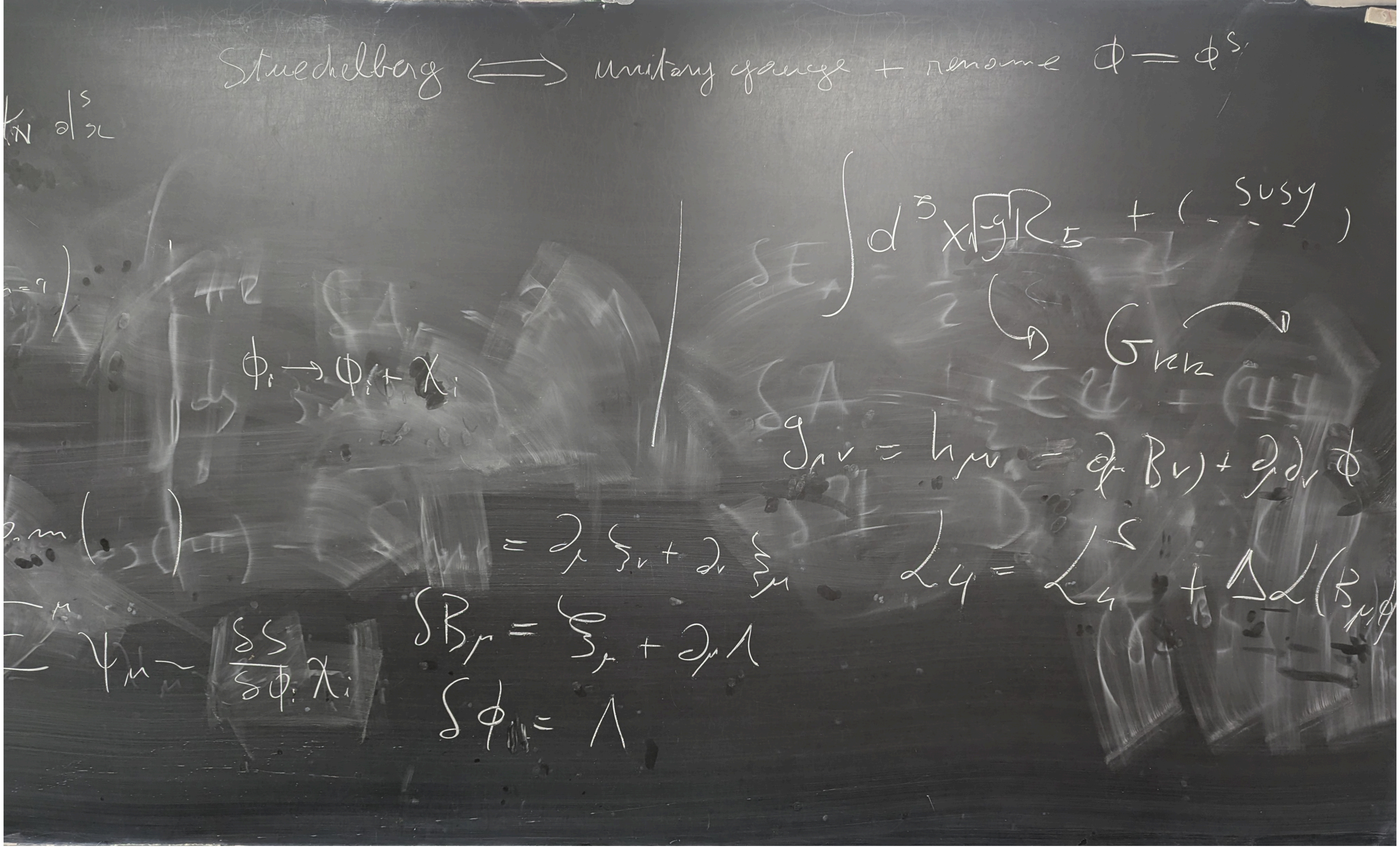
1. Writing mathematical propositions (and objects). Like $\forall$ a b $\in$ R, $(a+b)^2=a^2+2ab+b^2$
2. Writing the proofs themselves ("do an induction over *n*", "consider $\alpha \equiv \varepsilon^2 / 4$", "by Tychonov's theorem, it suffices to prove…")

The first level is quite structured, well-understood, readable (fortunately)

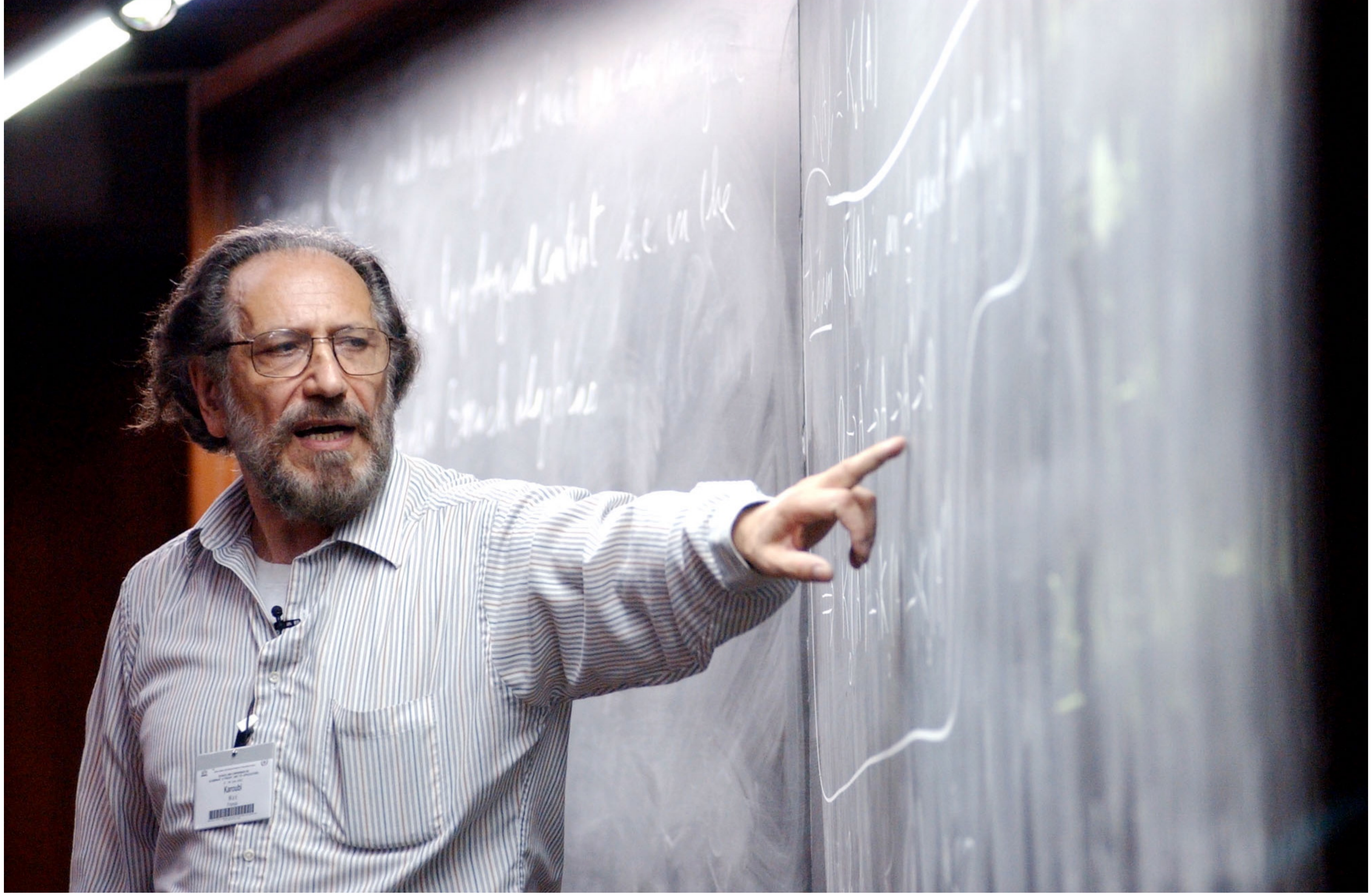
The second level is more messy: in proof-assistants, very often a kind of script language

$\Rightarrow$ do we really need text for the second level ?


Mais Gérard,
t'as qu'à cliquer!



Typical blackboard : mainly level 1 (propositions, facts…)



(Max Karoubi)

It is useful to *point* to *locations* in the text

Dual polarities

Why do we write a proposition ?

It is generally to state:

- either that we know this proposition at this stage (known lemma, hypothesis…)
- or that we *need* to prove this proposition

Let us chose a color code:

Known fact

Goal

Demo part 1

Paradigm: Handling Evidence

Human(Socrates)

Mortal(Socrates)

$\forall X . \mathrm{Human}(X) \Rightarrow \mathrm{Mortal}(X)$

Bring the evidence where it is needed $\quad A \dashrightarrow A$

But:

- evidence for $A \wedge B$ should also be evidence for A (and for B)
- evidence for A should also be evidence for $A \vee B$
- evidence for $A \Rightarrow B$ transforms B into A (or A into B)
- More generally, we can modify subexpressions of propositions

Let us play more with propositional logic

$$[\, A \Rightarrow \underline{B} \vdash C \vee (D \wedge \underline{B}) \,] \quad \Longrightarrow \quad C \vee (D \wedge A)$$

$$\blacktriangledown \ R\vee_2$$

$$C \vee [\, A \Rightarrow B \vdash (D \wedge B) \,]$$

*D*eep Inference
(Calculus of Structures : K. Chaudhuri)

$$\blacktriangledown$$

$$C \vee (D \wedge [\, A \Rightarrow B \vdash B \,])$$

$$\boxed{\text{Our set of rules subsumes all the given examples}}$$

$$\blacktriangledown$$

See CPP 22 article

$$C \vee (D \wedge A \wedge [\, B \vdash B \,]) \ \blacktriangleright \ C \vee (D \wedge A \wedge T\,) \ \blacktriangleright \ C \vee (D \wedge A)$$

Theoretical basis: Deep Inference (Guglielmi et. al.)

Quantifiers

$P(t) \vdash \exists x.\, P(x)$

T

$P(t) \vdash P(t)$

Two rules:

$$A \vdash \exists x.\, P(x) \quad \left\{ \begin{array}{l} \exists x.\, (A \vdash P(x)) \\ \text{or} \\ A \vdash P(t) \end{array} \right.$$

$\forall x.\, P(x) \vdash P(t)$

T

$P(t) \vdash P(t)$

$$\forall x.\, P(x) \vdash A \quad \left\{ \begin{array}{l} \exists x.\, (P(x) \vdash A) \\ \text{or} \\ P(t) \vdash A \end{array} \right.$$

(more rules for $\forall$ and $\exists$)

Equality

Equality, in a nutshell:

- A relation over any type (to write $u=t$, u and t must be of the same type)
- Reflexive ($t=t$ proved through a double click)
- If $t=u$, then t and u verify the same properties

$t=u$ allows to replace t by u:
- in the goal
- in other hypotheses

The rule: $u=t \vdash A[t] \blacktriangleright A[u]$

- also works on a hypothesis
- works like the axiom rule $A \vdash A \blacktriangleright T$ (thus benefits from deep inference)

$$\forall x, x \neq 0 \Rightarrow x/x = 1 \vdash \exists y\ z, R(y, z) \Rightarrow P(f(y/y))$$

$$\blacktriangleright \quad \exists y\ z, R(y, z) \Rightarrow y \neq 0 \Rightarrow P(f(1))$$

Handling the objects

The objects of Coq (of Type Theory) are basically pure functional programs, with data-types à la Caml/ML/Haskell…

Reasoning is *modulo computation*:

- 2+2 ▶ 4 (computes to 4)
- there is *no difference* between 2+2 and 4
- thus no difference between 2+2=4 and 4=4
- 2+2=4 or 201+199 = 400 are proved by reflexivity.

0+x ▶ x S x + y ▶ S (x+y)	but	$\left.\begin{array}{r} x+0 = x \\ x+(S\ y) = S(x+y) \end{array}\right\}$ proved by induction

This is regular Type Theory. What we gain is the ability to point where to perform computation, induction…

A few (very) simple examples

```
nat :=
 |  0 : nat
 |  S : nat → nat
```

```
ll :=
 |  nil : ll
 |  cons : nat → ll → ll
```

Properties of simple operators (+, \times, - …)
functions about lists (concatenation, sorting)

In general: handles well the first lessons of a course of formal proofs

Demo part 2

Some conclusions & remarks

- A new start for Gilles Kahn's last research project: "Proof-by-Pointing" in the 90ties (but with better software library infrastructure)
- But also better theoretical infrastructure: deep inference ⇒ real proof theory questions
- Positive features or directions appear as the system is used
- Many possible directions (manipulating real analysis expressions, navigating in proofs…)
- Software development / architecture questions
  - Splitting features between front-end and back-end
  - Requires varied skills: logic, ML, front-end/JS…
  - The aim is essentially to deliver a nice & usable product vs. academic schedule