Real World Coq Course
Sabancı University, Istanbul, August 12th 2022

Coq Fundamentals

Emilio Jesús Gallego Arias – Inria Paris

Hoşgeldiniz!

Welcome all to the 2022 Real World Coq course at Sabancı.
Before we start, some **administrativia**:

- Course is split in **two main sessions**:
  - **Morning** (8:40 – 12:10): presentation of core material
  - **Afternoon** (13:10 – 15:30): assisted exercise time
- Both sessions have a **30 mins break**
- Advanced topics for the last day **to be chosen by you!**

- **Day 1:** Coq and Type theory
- **Day 2:** Proof tactics and libraries
- **Day 3:** Mathematics
- **Day 4:** Software verification

Let me know **ASAP** if you have trouble with **the jsCoq demo page**

Online asynchronous help forum

https://coq.zulipchat.com/#narrow/stream/341461-Sabanc.C4.B1-Coq-Course---Sept-2022

Software that works: a hard task

Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it

Ariane 5 rocket

The rocket self-destructed 37 seconds after launch

Reason: A control software bug that went undetected

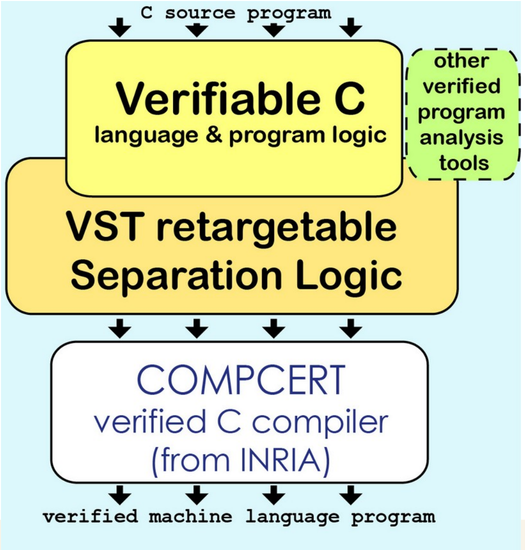
- Conversion from 64-bit floating point to 16-bit signed integer value
  an exception
  - The floating point number was larger than 32767 (max 16-bit signed integer)
- Efficiency considerations had led to the disabling of the exception handler.
- Program crashed rocket crashed

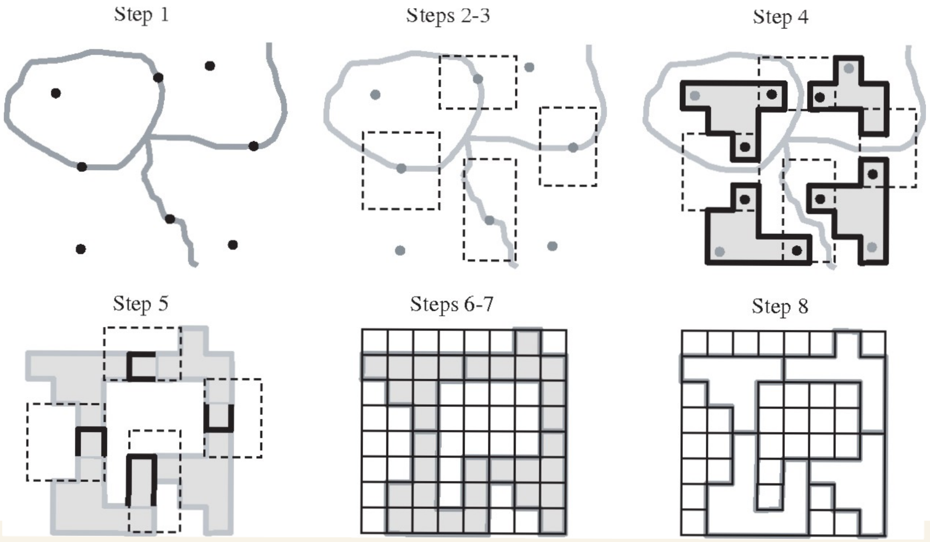
Total Cost: over $1 billion

Formal Methods: Successes

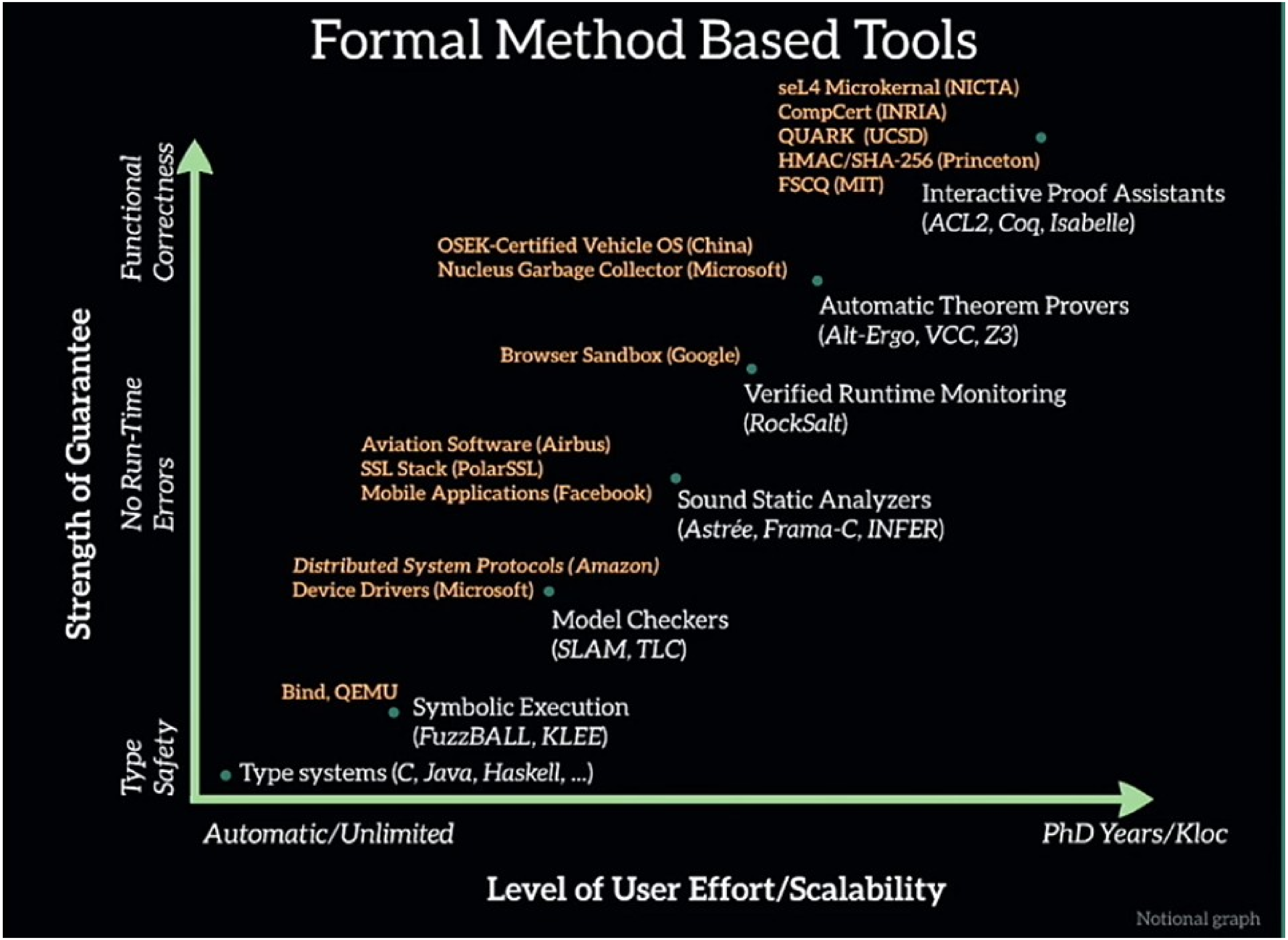
Formal Methods: Application of a <u>broad set of techniques</u> to problems in software and hardware verification and specification.

Many Success Stories: Verified compilers, Algorithms, OS kernels, Hardware, Protocols, Mathematical Proofs, …





More than **<u>50 years</u>** <u>of history</u>, but what's next?



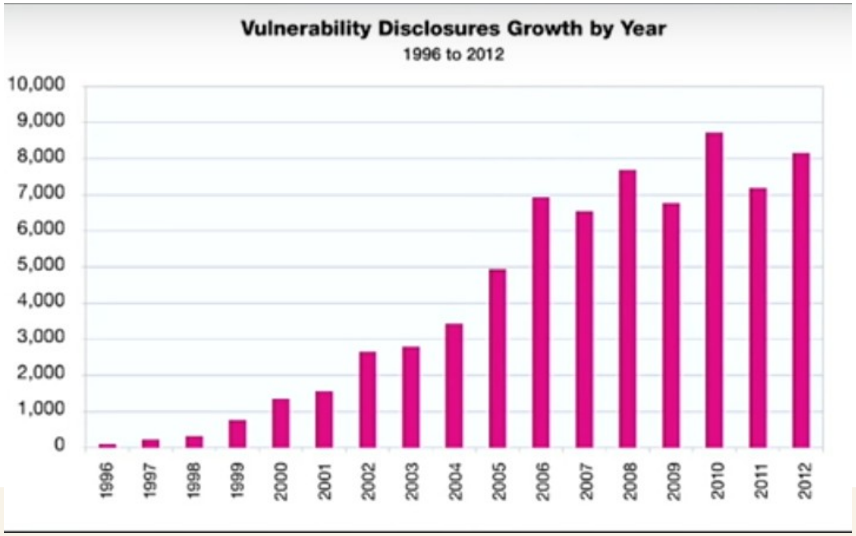
Credits: Kathleen Fisher

Formal Methods: Challenges

Key Challenges: Deep specification, new computing models (permision-less, quantum), effort, scalability, automation, education.

Common trend:

- *Horizontal* scaling: systems use <u>more components</u>
- *Vertical* scaling: components <u>grow larger</u>



2016-2019: Kernel +3.000.000 lines

Year	Linux Kernel	GCC
1996	640.000	100.000
2006	5.000.000	2.000.000
2016	20.000.000	15.000.000

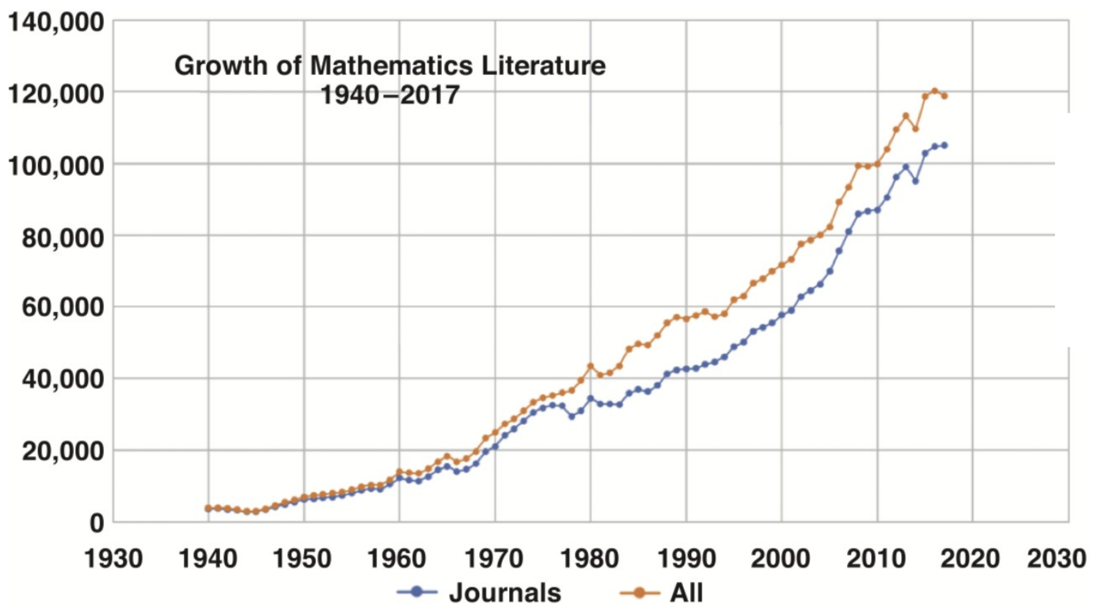
Figure – Core Software: Lines per Year

Complete Understanding of Systems Harder and Harder

The Growth of Scientific Knowledge

AMS: 3% increase in *math production per year*

400.000 papers *per year* by **2045**



E. Dunne, "Looking at the mathematics literature", *Notices of the American Mathematical Society*, vol. **66**, no. 2, pp. 227–230, 2019. ams.org/journals/notices/201902/rnoti-p227.pdf.

The proof that wasn't

Nick Scott explains the story of a mathematical proof that has sparked controversy, questioning how extremely complicated work can be validated if few understand it

AMERICAN MATHEMATICAL SOCIETY
MATHSCINET
MATHEMATICAL REVIEWS

4,067,699 **publications**
1,117,951 **authors**

About 11,800,000 results (0.41 seconds)

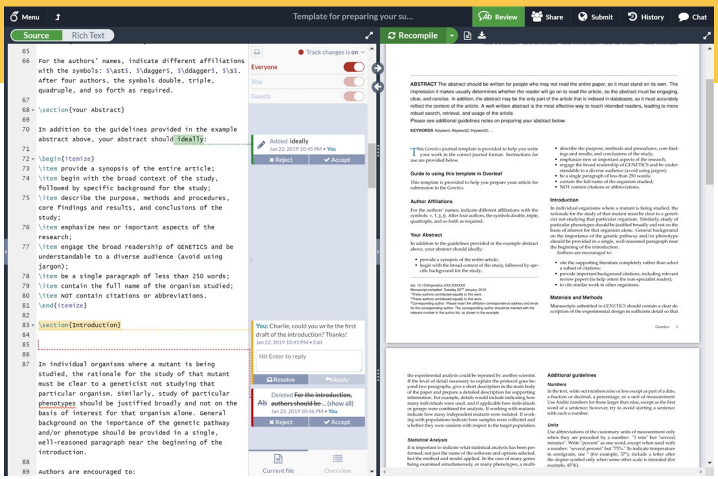
How to Resolve Merge Conflicts in Git?

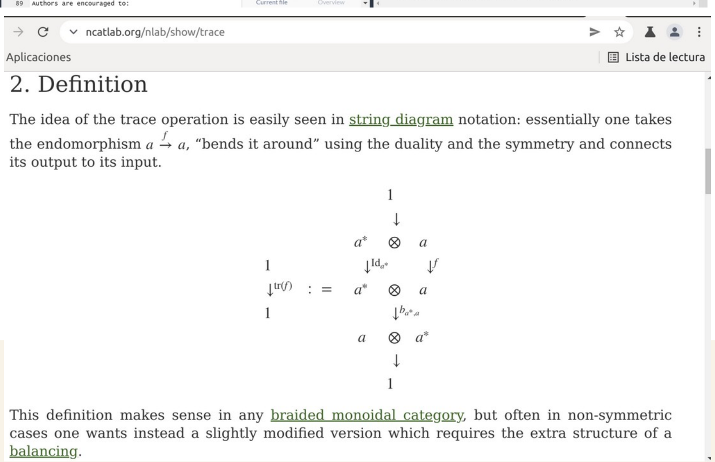
"If you think your job is getting ***harder****, you are* ***correct****. The mathematics literature is* *growing relentlessly**, and becoming* ***harder to figure out*** *along the way"*

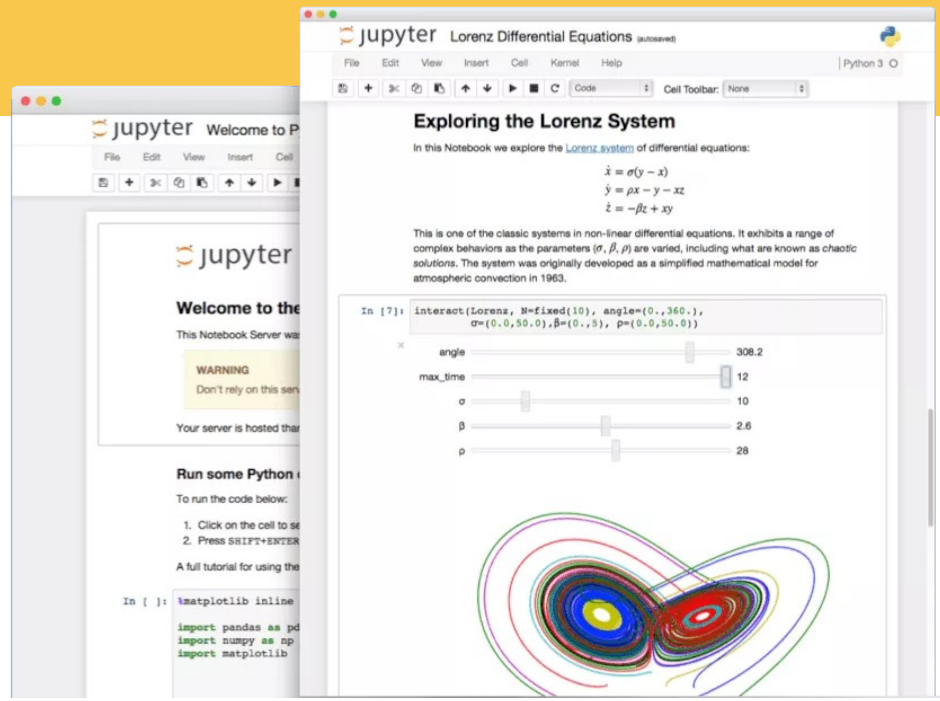
E. Dunne

Collaborative Mathematical Writing

WWW: ***diversification*** in knowledge *production*







Wikis / online editors very popular

New writing **media** brings new **opportunities**

More than **8.000.000** active **Jupyter** Notebooks on GitHub.com !

Coq and Type Theory: 2019-2022

2019: Environments for Large-Scale Proof Development
Focus on *advanced* proof engineers, multi-system

- Coq's **Continuous** Integration & **Industrial** Build Systems (creator & maintainer)
  > **3 million lines of Specs and Proofs**
- Complex **interop** with ***Mach. Learning / Soft. Eng***. : **document matters**!

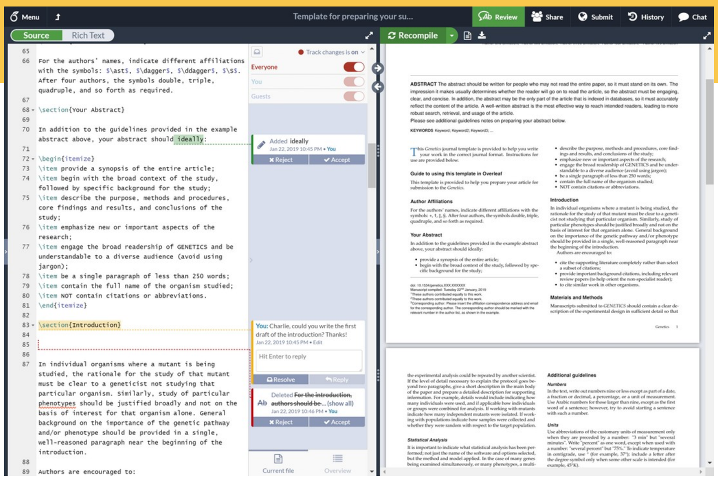
Online **Collaboration** + **Formal Mathematics** more important!

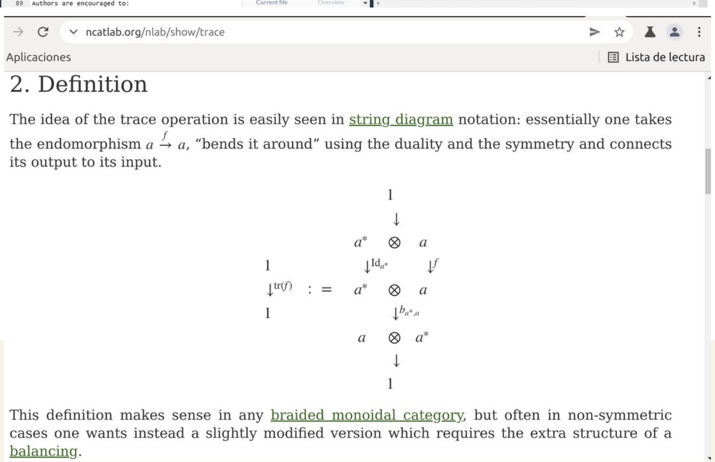
- From *advanced proof engineers* to **advanced mathematicians**
- Essential *feedback* from **Inria/IRIF**, **nLab** and **teaching** community

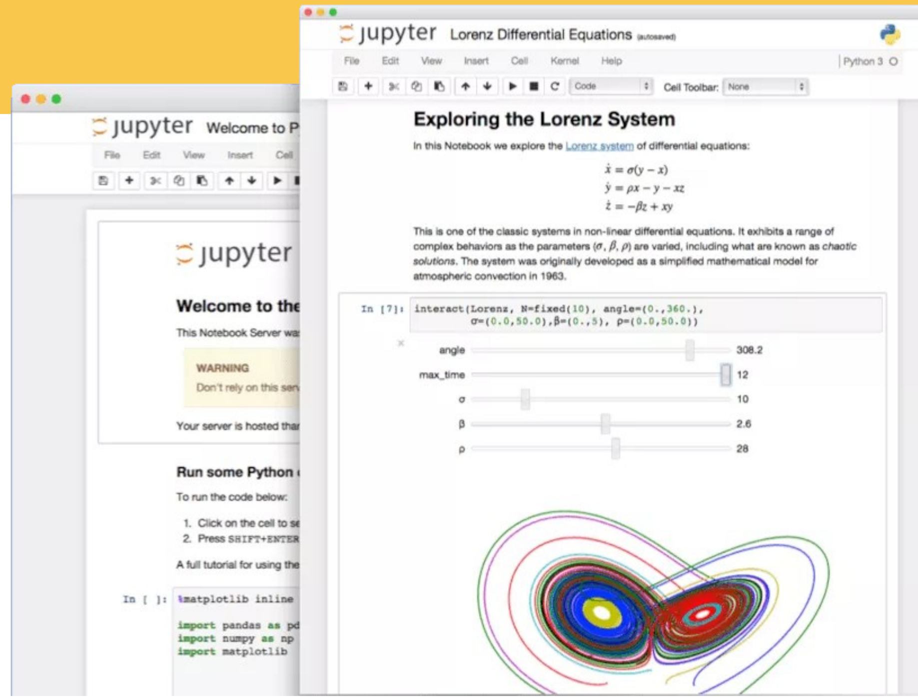
2022: From *mathematical* to ***formal*** documents
Focus on **collaboration**, **evolution** of documents

Collaborative Mathematical Writing

WWW: ***diversification*** in knowledge *production*







Wikis / online editors very popular

New writing **media** brings new **opportunities**

More than **8.000.000** active **Jupyter** Notebooks on GitHub.com !

Timeline of Type Theory

Goal: Build a foundation for mathematics!

- **19th century**: Cantor, Frege, Peano, Pierce, Brouwer
- **1903**: Russel's **Type Theory**
- **1930s**: Gentzen, Gödel; **consistency and incompleteness**
- **1940**: Church's simple theory of types; **lambda-calculus**
- **1950**: Gödel's System T; **Dialectica Interpretation**
- **1970**: Girard's **System F**, Martin-Lof **Type Theory**
- **1980**: Coquand's **Calculus of Constructions**

A stunning pace of development in a short period of time
(Many others Heyting, Kolmogorov, Kleene, Curry, Howard)

Coq: Foundational Theorem Proving

Foundational: Proofs written in *Minimal* yet *Expressive* Calculus
Verification: Proofs <u>automatically</u> checked by *trusted* small **kernels**

What's next?

- A **milestone** of 20th century logic and computer science
- Enabled a **very high degree of confidence** on many **key** mathematical results and **critical** software
- **The Coq Proof Assistant:** field leader (Inria, 1984-now) (myself: 2015-now)
- **2013: ACM System Software Award**
- **2022: Open Science Award** (collab dev)

Proofs as programs: very well suited for both <u>math</u> and <u>software</u> validation

The Coq Proof Assistant

Developed at **Inria** by T. **Coquand**, G. **Huet**, C. **Paulin**

- First usable version in **1985**
- Powerful **logical framework**, programming language
- Goal: mechanically-verified **programs** and **mathematical proofs** in a constructive meta-theory
- See *"Early history of Coq"* in the Coq's reference manual
- Constructing proofs mainly in "**interactive nature**"

Very successful project

The Calculus of Constructions

Theoretical basis of **Coq**
Higher-order dependently-typed calculus

- $\Gamma \vdash p : T$ "program **p** has type **T** under assumptions **Γ**"
- **T** is a **type**, or **proposition** to prove, examples:
  - $\exists\,(x : nat)\,,\ Turing\_machine_{32}(x) = 33$
  - $\forall\,(t_1\ t_2 : st)\,,\ t_1 \approx t_2 \Rightarrow \exists\, t'_1\ t'_2,\ t_1 \mapsto t'_1 \wedge t_2 \mapsto t'_2 \wedge t'_1 \approx t'_2$
- **p** is a **program**, or **proof** for **T**
  - "$p : A \Rightarrow B$" for any **input** of type **A**, output a **B**
  - "$p : \exists\, x\,,\ P(x)$" pair with a witness **w** and proof "**P(w)**"
  - "$p : A \wedge B$" pair with proofs "$p_1 : A$" and "$p_2 : B$"

The Calculus of Constructions

Given a proof **p** and a goal **T**, Coq will check "**p : T**"
Type-checking is described using "**inference**" rules

1. $$\frac{}{\Gamma \vdash P : T}$$

2. $$\frac{\Gamma \vdash A : K}{\Gamma, x : A \vdash x : A}$$

3. $$\frac{\Gamma, x : A \vdash B : K \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash (\lambda x : A.\, N) : (\forall x : A.\, B) : K}$$

4. $$\frac{\Gamma \vdash M : (\forall x : A.\, B) \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

5. $$\frac{\Gamma \vdash M : A \qquad A =_\beta B \qquad \Gamma \vdash B : K}{\Gamma \vdash M : B}$$

$$e ::= \mathbf{T} \mid \mathbf{P} \mid x \mid e\,e \mid \lambda x{:}e.\, e \mid \forall x{:}e.\, e$$

The Coq Proof Assistant

Core development team: **12 developers**

- A few dozens of **external contributors**
- Free software, open development model @ Github
- **1.000-10.000** regular users, including all target groups
- 10-100 active **research projects** in the world
- > 3.000.000 lines tested in the CI, in the wild 1 order more
- Fairly high degree of **maturity**, but active development

High-level Goals of Coq for 2020s

Improve how we

Produce
Organize
Interact with
Evolve
Collaborate on
Validate

scientific documents

Core Hypothesis:

Programming Languages
&
Interactive Theorem Provers

have reached a maturity point
where we can build upon them
formal, verifiable, hybrid documents
& theory for **collaboration** and **evolution**

We aim to develop a document model is
designed to **enable** many interesting
interactions with other research fields

Proving and verifying in the CoC

Strong points:

- Programs and proofs live at the same level
- Small, reliable kernel, good “Trusted Computing Base”
- **Foundational character**
- PL and theorem proving ideas do apply

Challenges:

- Writing ~~programs~~ proofs is very verbose, requires automation
- Language is low-level, notations and encoding needed
- Underlying logic very general, but not necessarily adapted to all domains

Beyond Programs: Data

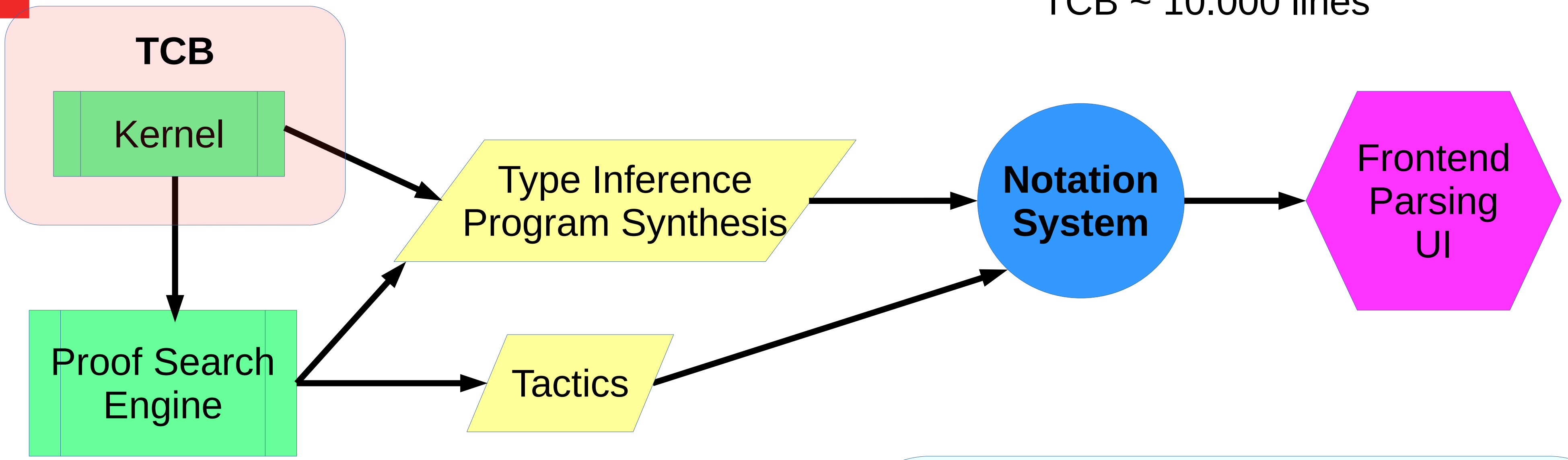
The Calculus of Inductive Constructions

- **Coq** also provides a powerful data-definition mechanism
- **“Inductive”** data types can encode arbitrary relations
- Well beyond safety, etc…
- However they have no **computational** content
- Commonly used to encode **transition systems**, rules, …
- Also the base for most logical connectives

Key to being friendly for PL verification.

Organization of Coq

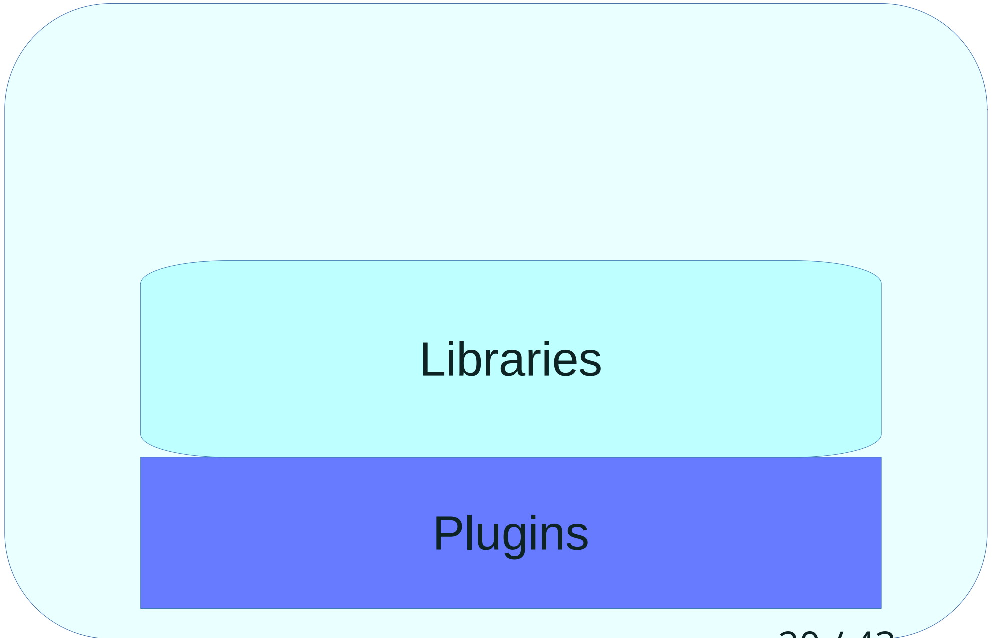
200.000 lines of OCaml code
120.000 lines of Coq code
5.000 lines of C
TCB ~ 10.000 lines



The Interactive Proof Cycle

- Users input documents in a high-level mathematical proof language
- High-level language is elaborated to the core calculus
- Tactics and type inference perform program search.
- Kernel checks correctness of the proofs

- System is extensible



Success Stories

- **Fundamental maths**: 4-color theorem, Feit-Thompson

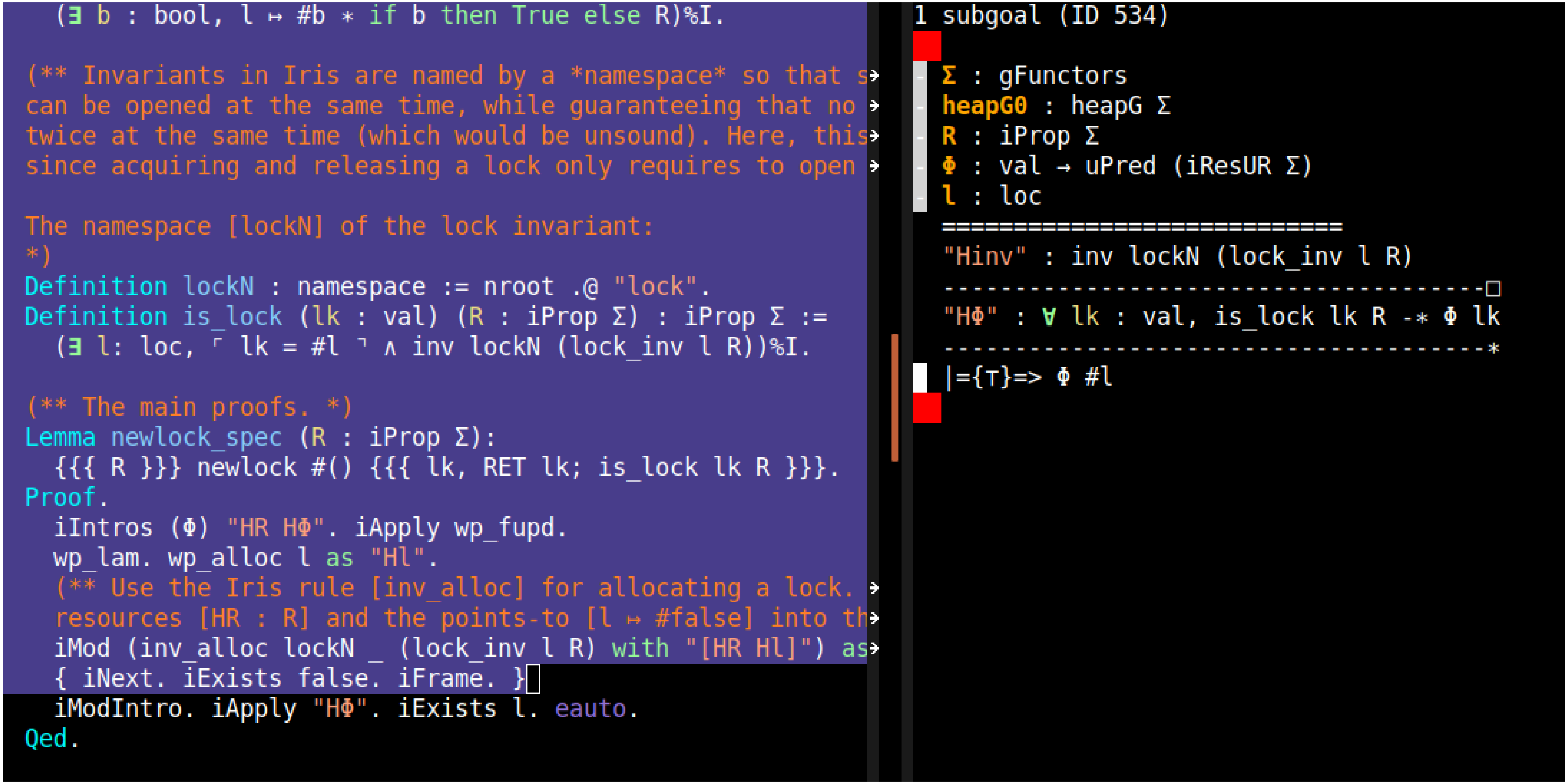
```
Qed.

Theorem Feit_Thompson (gT : finGroupType) (G : {group gT}) :
  odd #|G| -> solvable G.
Proof. exact: (minSimpleOdd_ind no_minSimple_odd_group). Qed.

Theorem simple_odd_group_prime (gT : finGroupType) (G : {group gT}
  odd #|G| -> simple G -> prime #|G|.
Proof. exact: (minSimpleOdd_prime no_minSimple_odd_group). Qed.
```

Success Stories

- **Software Verification:** CompCert, Fiat-Crypto, Deepspec, IRIS, blockchain, great impact at PL venues

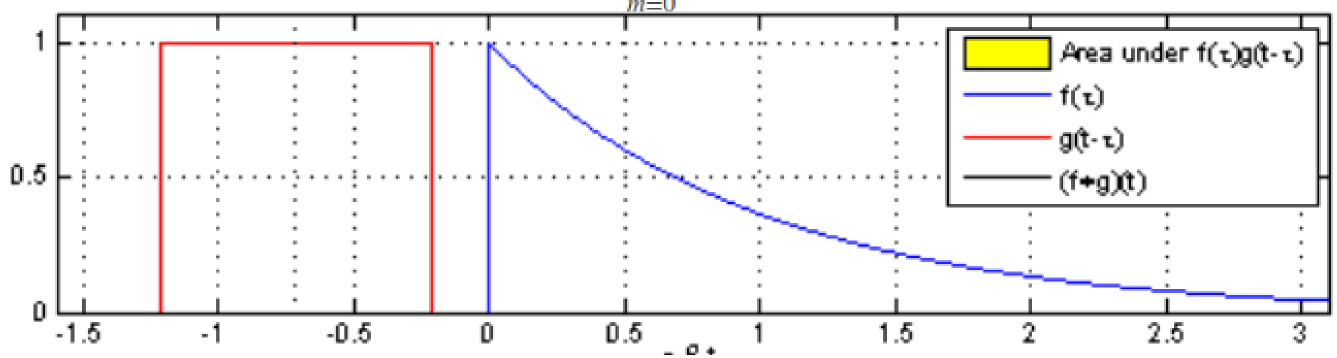


Success Stories

- **Teaching:** Software Foundations, CPDT, many schools and tutorials

Circular Convolution of two Signals

$$(x \circledast y)_n = \sum_{m=0}^{N-1} x(m)y(n-m)$$




```
Definition convs x y := \col_n \sum_m x m 0 * y (n-m) 0.
```

$$\begin{aligned}(x \circledast y)_n &= \sum_{m=0}^{N-1} x(m)y(n-m) = \sum_{l=n}^{n-(N-1)} x(n-l)y(l) \\ &= \sum_{l=0}^{N-1} y(l)x(n-l) \\ &= (y \circledast x)_n\end{aligned}$$

```
Lemma convsC : commutative convs.
Proof.
move=> x y; apply/matrixP=> n k; rewrite !mxE {k}.
rewrite (reindex inj (inj comp (addrI n) oppr inj)).
```

Links to resources
In the course
Webpage

Coq vs Other Systems

Coq both a **PL** and an **Interactive Theorem Prover**

- vs **traditional PL**:

  Notations, elaboration, implicit arguments, tactics, higher-order unif, partial evaluation, fp, interactive development, slower

- vs **Isabelle**: Different logic ; Isabelle interface much more user-friendly
- vs **Lean**:

  different development model and user base, different strengths implementation & compatibility. Punch with math community.

- Vs **Adga**: impredicativity, trust-base, tactics, less experimental features

Development needs to adapt, or risk becoming obsolete.
Huge legacy codebase difficults progress => research topic

Coq's Ecosystem

Large work in the last years to build a community

- **Zulip Forum**: Main forum, both users and devs
- **Coq Community**: collective maintenance
- StackOverflow, mailing lists
- GitHub project
- **Events**: 2 Workshops, 1 user and dev meetings, diversity, misc hackathons, schools....

Development model pretty unique among interactive theorem Provers, has pros and cons.

Coq's Vernacular Language

Type theory is a **very bare** language
Coq provides many **user-level** constructions to do math

```
Record abelian (V : Type) :=
 Mixin {
  zero : V;
  opp : V -> V;
  add : V -> V -> V;
  _ : associative add;
  _ : commutative add;
  _ : left_id zero add;
  _ : left_inverse zero opp add
 }.
```

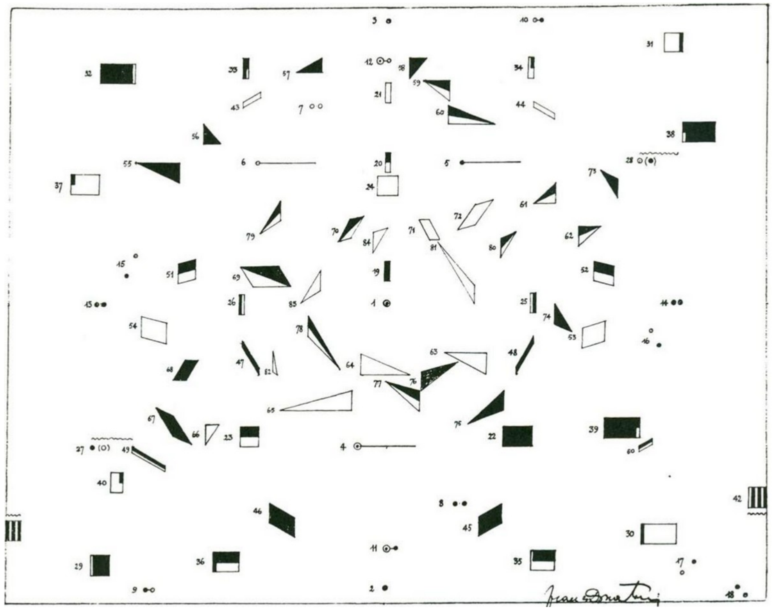
```
Instance abelian int :=
 {
  zero := 0;
  opp : -;
  add : +;
  addiA;
  addiC;
  add0i;
  ...
 }.
```

```
Lemma mulrnDl (T:abType) (x y:T) n: {morph (x => x *+ n) : x y / x + y}.
Proof. move=> x y; elim: n => [|n IHn]; rewrite ?addr0 // !mulrS.
by rewrite addrCA -!addrA -IHn -addrCA.
Qed.
```

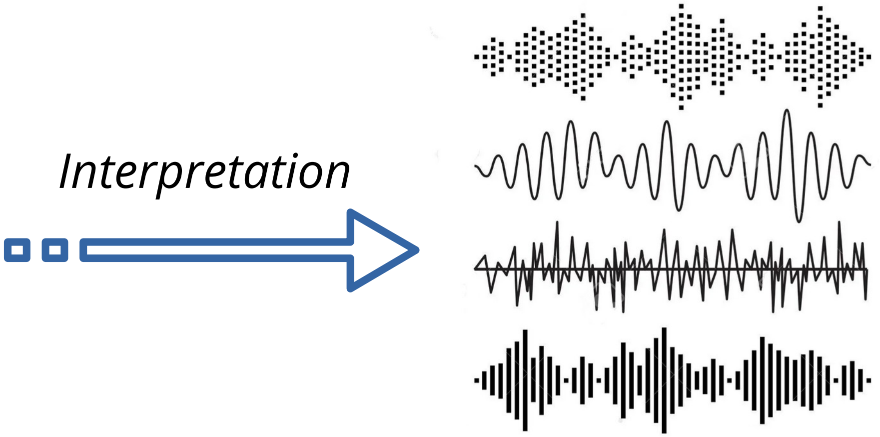
Of particular interest are **notations**, **tactics**, **structures**, **hints, definitions and modules, …** (over 200 vernaculars)

Doing Proofs: What is Hard?

With **high confidence** comes **high cost**
Proof assistants **notoriously difficult to use**



Franco Donatoni – Babài   Notations *(c) John Cage*

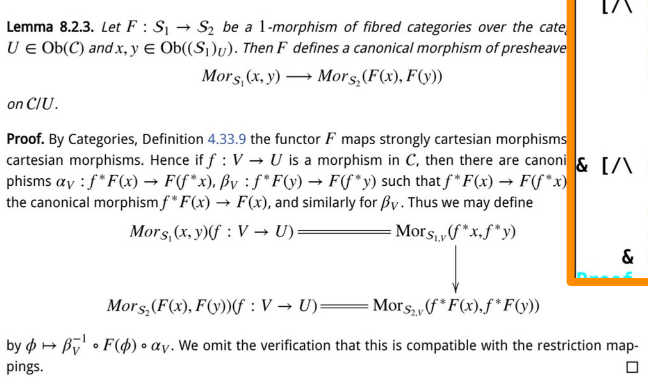


Human vs Machine *impedance*

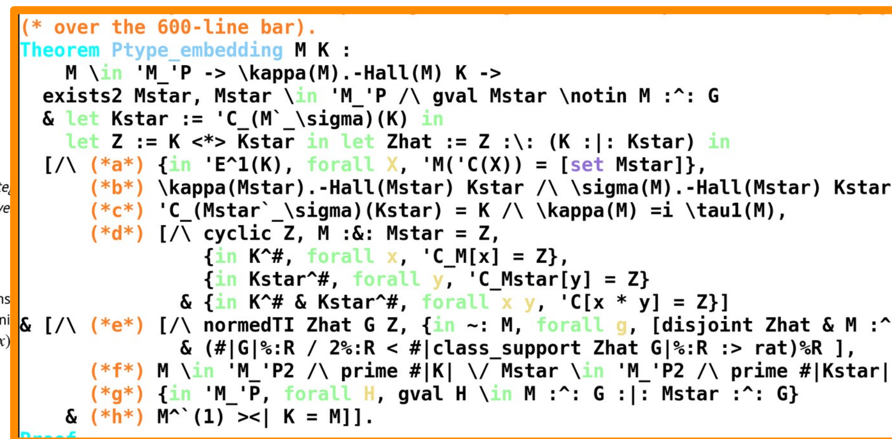
Doing Proofs: What is Hard?

With **high confidence** comes **high cost**

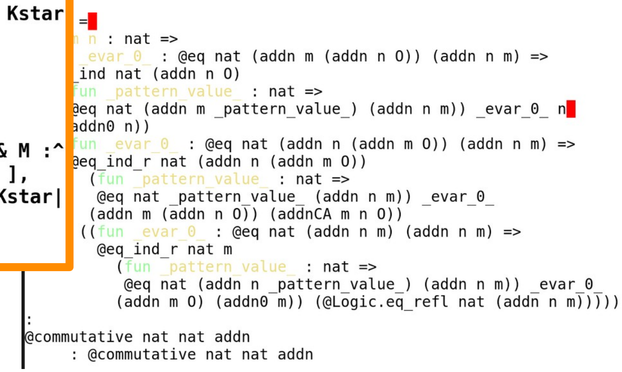
Proof assistants **notoriously difficult to use**



(c) The Stacks Project



Coq Document (Text-based)



Kernel-level Proof Term

Manual Translation + Interactive Interpretation

Human side: rich *natural*, *mathematical*, *graphical* language

Computer side: minimal "*assembler*" language of *proof terms*

Other Important Challenges

Installing things!
Libraries that don’t work / outdated proofs

- **Searching** for things without success
- Bad **display** / **notations**
- Boilerplate / **trivial** proofs
- Synchronization / **merging** problems
- Lack of **documentation**
- **Dumb** or outdated **interfaces**

A mix of Social, Research, and Engineering Problems!

Have we reached a Critical Point?

Recent times have seen a **proliferation** of **formal** and **semi-formal collaborative math writing systems**

- LaTeX / Literate Programming: **Stacks**
- Education for Maths: **Edukera, WaterProof**
- Semantic-Aware, Interactive: **Nota**, **ScholarPhi**
- Structure-Aware: **Hazelnut, Actema**
- Interactive Documentation: **Alectryon**
- Self-contained formal documents: **jsCoq, Holbert**

How far from an integral solution?

We have reached a Critical Point

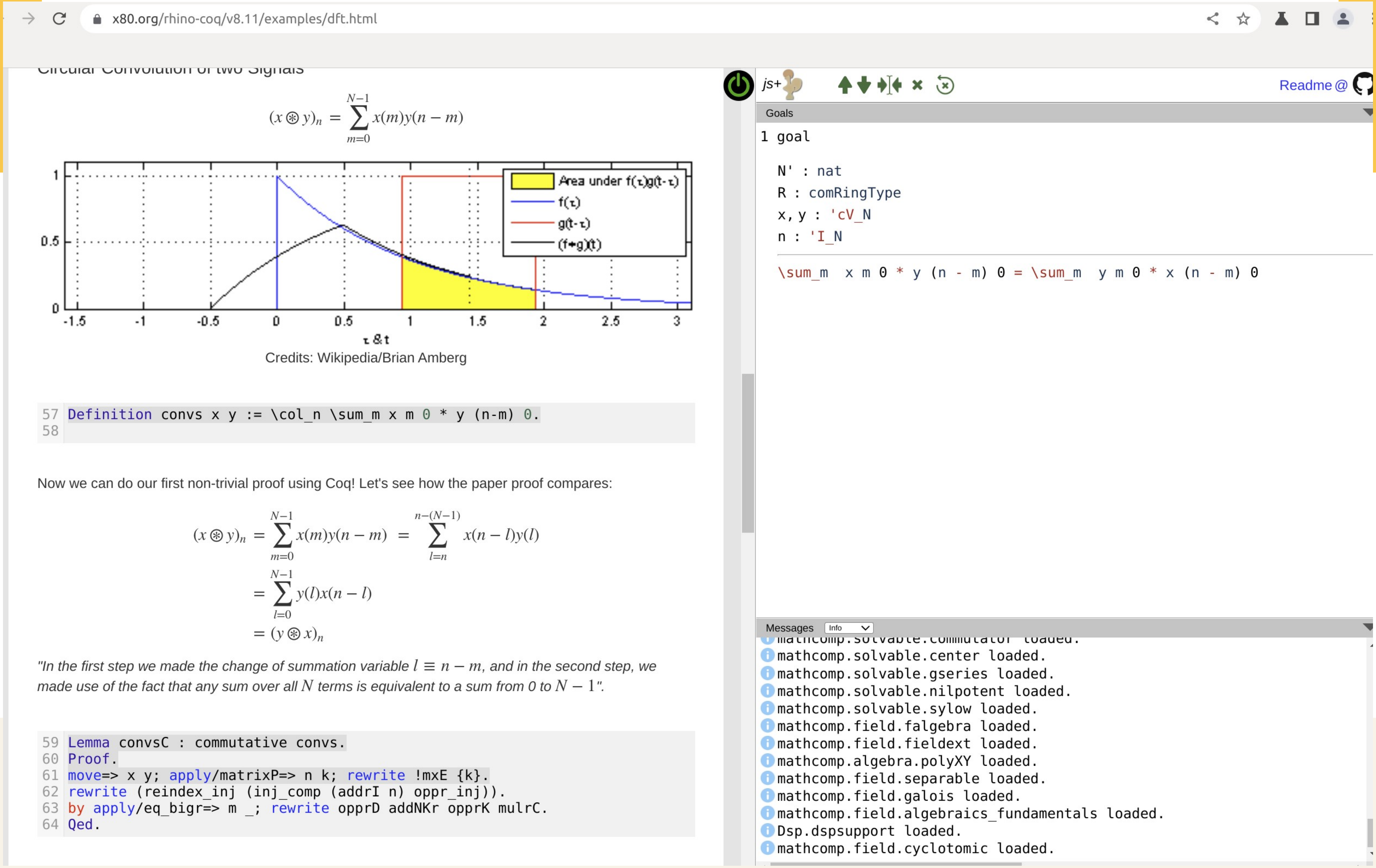
Current solutions don't address current needs

- **Jupyter Notebooks:** Great for computational content, falls short for general verified math and software
- **Overleaf, Wikis, Stacks:** Don't integrate with tools that can understand and validate content
- **Traditional ITPs (Coq, Lean, Isabelle,...):** Lack accessibility, collaboration features

The area has become a **very hot topic** in the last year

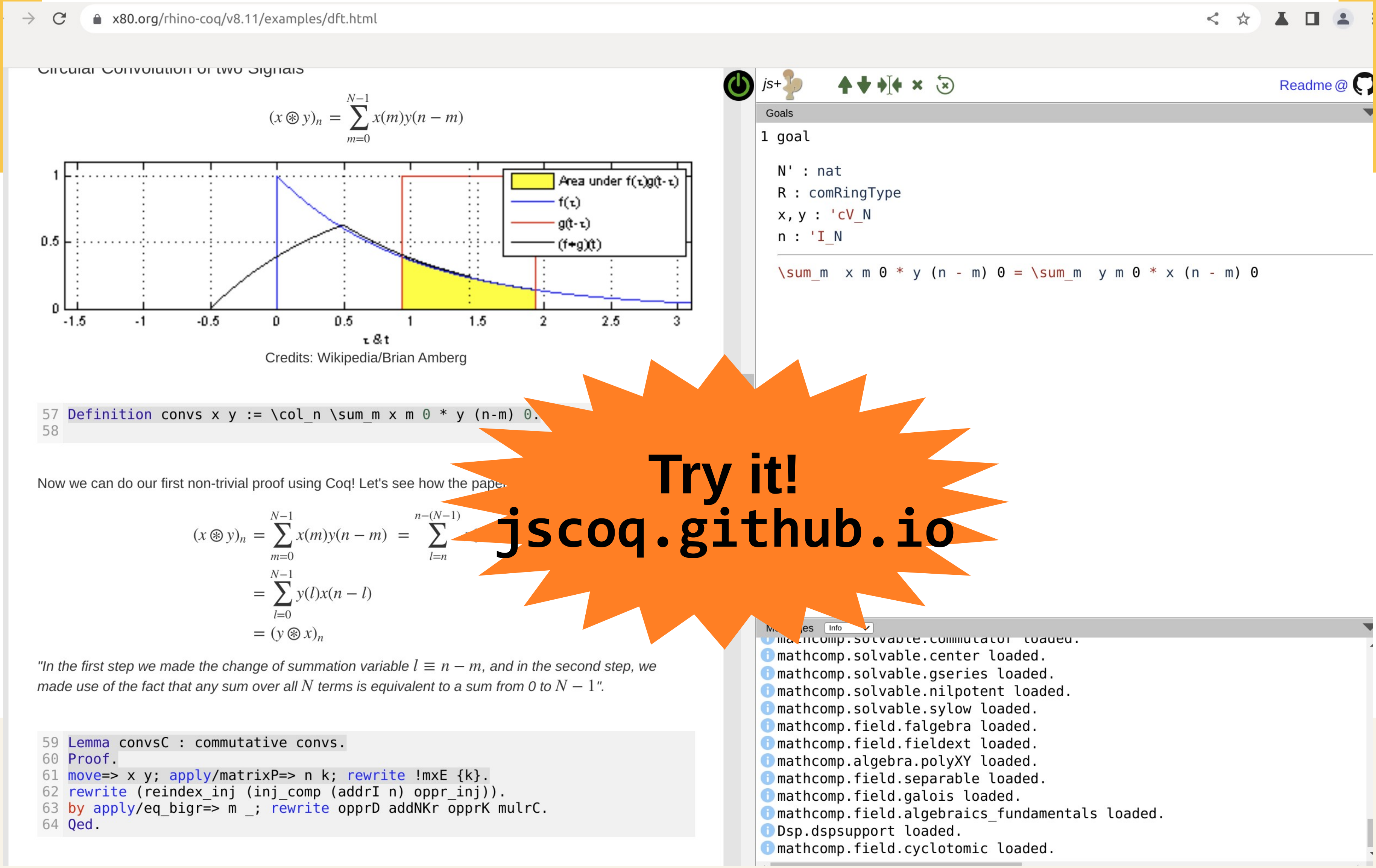
jscoq.wiki: a formally-verifiable Wiki!

jsCoq: Towards Hybrid Interfaces for Theorem Proving (UITP2016)



jscoq.wiki: a formally-verifiable Wiki!

jsCoq: Towards Hybrid Interfaces for Theorem Proving (UITP2016)



Try it!

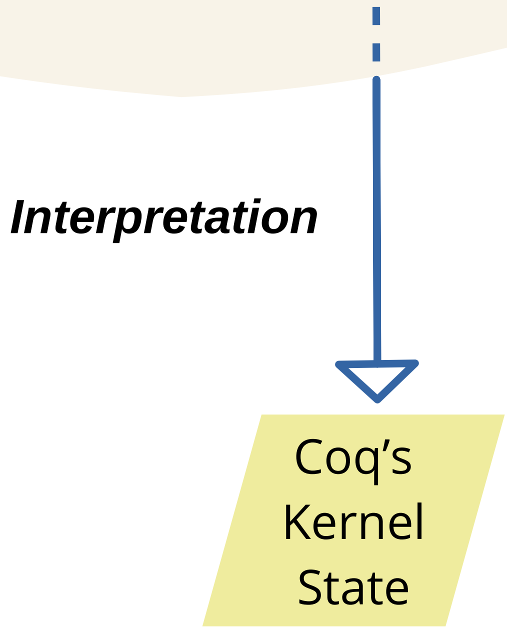
jscoq.github.io

Formal Hybrid Documents

Definition and **soundness** of *interpretation*

We now **show** that **+** is **commutative**:

```
Lemma addnC : commutative +.
Proof. elim=> [//|n iHn]. Qed.
```



The document calculus knows *3 kinds of objects*, and organizes them by **containment**:

- **semi-structured text:** free form, metadata can be updated and extracted
- **meta-logical objects:** objects that are formal, but are not seen by the kernel
- **logical objects:** objects that will be sent to the kernel, after interpretation

Theorem (*soundness*): The interpretation function **respects** the **logical structure** in the document.

We **cannot skip** sending a logical definition or theorem to the kernel.Formally:

$$\mathbf{I(L1 \oplus L2, M) = I(L1, M) \oplus I(L2, M)}$$

Note the document is **not checking correctly**, as the proof is incorrect

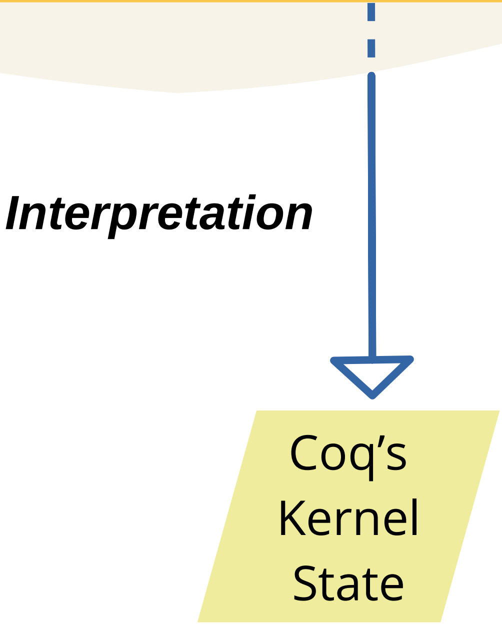
Formal Hybrid Documents

Gradual Document Interpretation: A **formal** theory of *Error Recovery*

We now **show** that **+** is **commutative**:

```
Lemma addnC : commutative +.
Proof. elim=> [//|n iHn]. Qed.

Definition bar := addnC. ✓
```



Proof development is best done by ***gradually*** refining **human-style specs** to their formal counterpart.

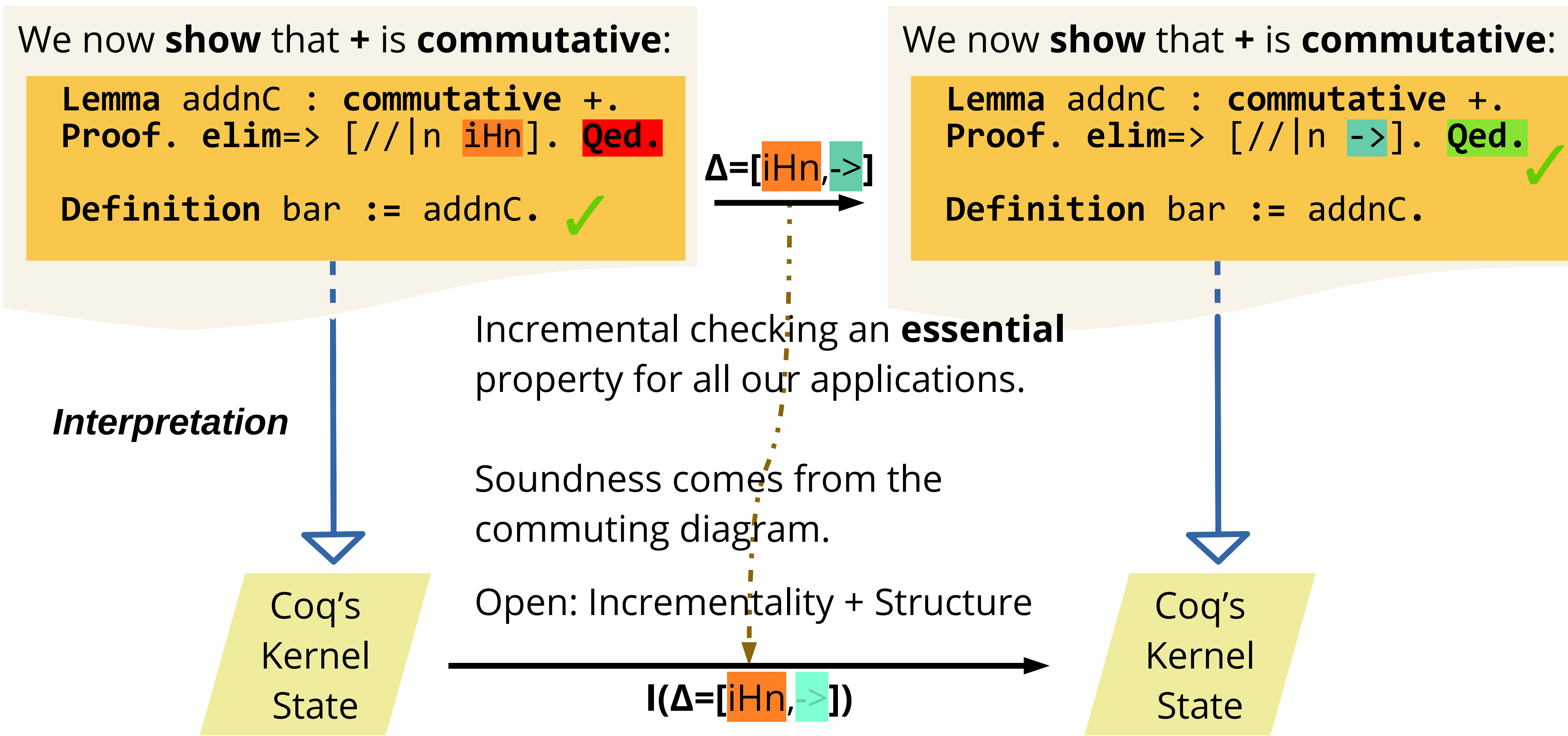
In this example, the proof of addnC is replaced by an unknown ?, which may only error if used. Error propagation can be contained **structurally**, to produce a better user experience. And bar can **still be checked**.

Gradual typing for Dependently Typed Systems is a very new area, and we will use it to formally model the **continuous process where a formal document evolves towards full validation**.

Theoretical Challenge: relation with interpretation soundness

Formal Hybrid Documents

Incremental Interpretation: Avoid **Re-Doing** Work



Incremental checking an **essential** property for all our applications.

Soundness comes from the commuting diagram.

Open: Incrementality + Structure

Shift from **study of proofs** to the **<u>study of evolution of proofs</u>**

Enabling Document-based Research

Document theory: **validity**, **distance**
Use checker as **Oracle** in "soft" experiments

"Mutation Testing for Coq"
(ASE2019)

- Documents as **source**: Indexing, Dataset Extraction
- Documents as **target:** Automatic "fuzzy" translation of mathematical texts, with feedback!

- More: **Structured access** provides an abstraction layer
- Foundation for **M.L. / S.E.** collaboration

A more fancy example: **constraint-aided conflict resolution**, SMT finds the **best resolution** w.r.t. doc soundness

SerAPI: Communicating with Coq

Enable other tools to interact easily with Coq

Not easy due to extensible nature; design constraints:

- **Low-effort:** cannot justify a large time sink
- **Lightweight**: neither can the users
- **Maintenable:** no use if it will stop working in 6 months
- **Robust:** API for clients should “resist change”
- **Machine-oriented:** Main use case is to talk to tools
- **User-driven:** convenience for users triumphs ideology
- Should be **easy to install**, work on unmodified Coq

Extensive use of OCaml’s meta-programming system

PPX

SerAPI: Interaction Protocol

control and **query** protocols

```
type cmd =
  | NewDoc  of newdoc_opts
  | Add     of add_opts   * string
  | Cancel  of Stateid.t list
  | Exec    of Stateid.t
  | Query   of query_opt * query_cmd
  | Print   of print_opt * coq_object

type coq_object =
  | CoqPp   of Pp.t
  | CoqLoc  of Loc.t
  | CoqTok  of Tok.t list
  | CoqAst  of Vernacexpr.vernac_control Loc.located

type query_cmd =
  | Option (** List of options Coq knows about *)
  | Goals  (** Current goals, in kernel form *)
  | Ast    (** Ast for the current sentence  *)
  | TypeOf of string
```

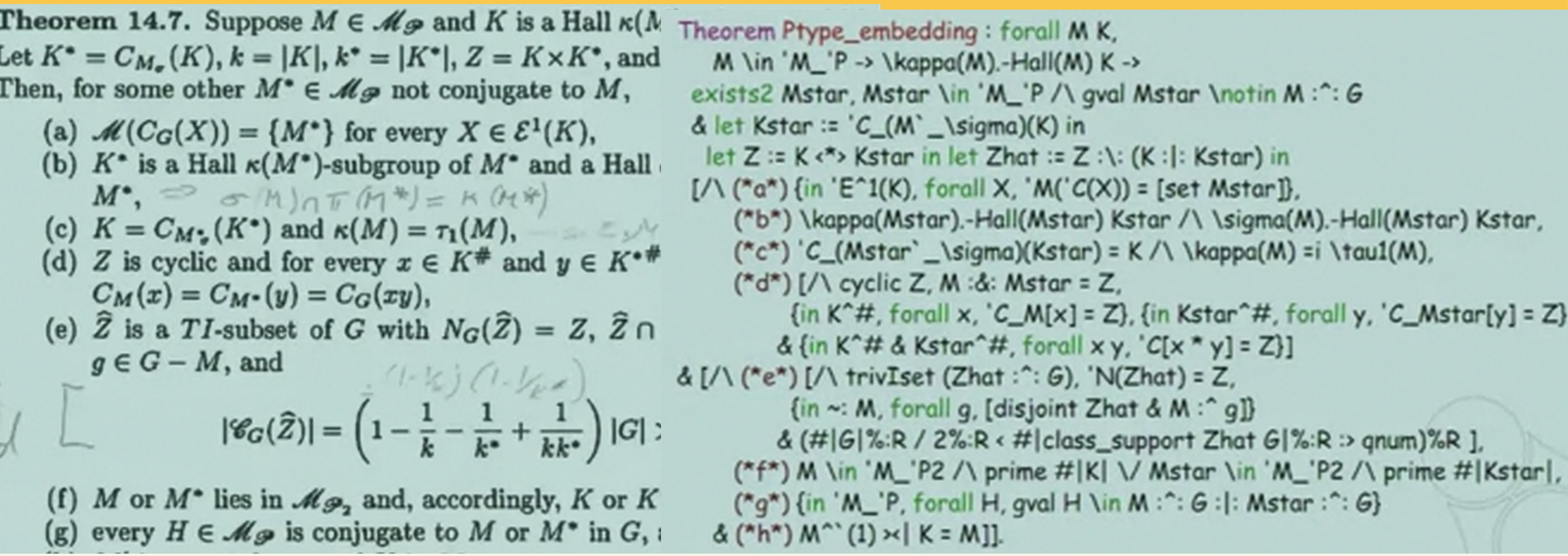
Then, these object definitions are serialized to JSON or Sexps

```
(Add ((ontop 3) (limit 3)) “Definition foo := 3.”)    (Query ((sid 3)) Ast)
```

Improving Coq's Printing

Coq's current printing system still **textual**

Roots on **console-based** interaction



Main problems: **1-dimensional layout**, lack of **meta-data**

The `BoxModel.t` printer

Adopt as **output** a **LaTeX/HTML box model**
Plus **attach semantic information** *à la Isabelle*

```
type t =
  | Variable of string
  | Constant of string
  | Identifier of Id.t
  | Sort of string list
  | App of { fn : t
           ; impl : t list
           ; argl : t list
           }
  | Abs of { kind : abs_kind; binderl : t list; v : t }
  | Let of { lhs : t; rhs : t; typ : t option; v : t }
  | Notation of
     { key : string
     ; args : t list
     ; raw : t
     }
```

```
module Id : sig
  type t =
    { relative : string
    ; absolute : string option
    }
end
```

Rendering to Web Components

Standard by Google, 2015, well **supported**
Allows to define **custom tags** in the DOM

- `<coq-notation raw=”...”></coq-notation>`
- `<coq-app>...</coq-app>`
- `<coq-binder-list> ... </coq-binder-list>`
- Reusable **components**, **shadow-DOM**
- Class based: extend `<coq-notation>` for your purposes!
- Programmable with **JavaScript / TypeScript**

In alpha stage, **collaboration** with **Actema** as to define an interactive, **2-way model**

Summary

A **wide-scope** project, potentially **large impact**
Great opportunity to **collaborate** with several CS areas

- Bringing together **mathematical writing**, **formal logic**, and **collaboration research**
- Trying to match **today's** demands
- Important little step to **improve validation in science**
- Co-enrichment between PL and a **few other fields**
- **State of the art** PL proposals (gradual, incremental, **differential**)...

Reflecting the **reality** of a more and more **multidisciplinary** setup in computer science.