(towards) Intuitive Language for Formal Mathematics

Cezary Kaliszyk and Florian Rabe

U. Innsbruck and FAU Nuremberg-Erlangen

February 3, 2020

Background

Both of us

- highly interested in languages for mathematics
- but unsatisfied with existing languages
- extensive experience with multiple languages/systems/libraries

Now ready to design new language

- grant proposal under review
- some ideas presented here

Current State of Formal Math Languages

Major progress over the last few decades

- Strong proof support
- Big libraries available
- Flagship projects
- Growing interest from mathematicians

However

- Logic-based systems far from actual mathematical language
  - very difficult to read, write
  - mathematicians easily alienated
  - do not understand mathematical common sense

Buzzard: "give me LaTeX"

- Natural language-based systems less successful
  - smaller communities
  - smaller libraries
  - less proof automation

Why are Formal Logical Languages inappropriate?

	Formal Logic	Natural Math
types	primitive	emergent
decidable typing	necessary/important	irrelevant
subtyping	impossible/awkward	fundamental, critical
modules	primitive	emergent
inheritance/sharing	difficult	implicit
equality	undecidable	obvious, prove if necessary
identify up to iso	open problem	implicit, obvious
proofs	part of calculus	focus on convincing reader
granularity	full proof	as needed

Partial exception: Mizar (but community & automation issues)

Why do CNLs not have large libraries?

Lack of high-level structuring

- modules
- types
- tactics

No deep formalization

- Proofs relative to flexible assumptions

Formal text more efficient to write

CNLs focus more on math than software verification

- Software verifications means whole libraries formalized

Learning curve less steep but difference not that big

ForTheL and Mizar: middle of logic-CNL spectrum

ForTheL = a CNL trying to be close to logic

Mizar = a proof assistant trying to be close to CNL

- rich type system: intersection types, dependent types

  let S be n-dimensional vector space

- rich input:
  - contextual parsing
  - type guided disambiguation
  - implicit arguments
  - 100 meanings of "+"
- natural logic: close to FOL

(but other severe limitations)

Our Goal

Design ILF:

- a rigorous formal logic with its own type system, calculus etc.
- that captures structure of natural language mathematics

  Hales: "need language that is both human- and machine-readable"

ILF will be very different from

- natural languages: rigorous definition in its own right
- logical languages: more and different language features

Design of Current CNLs

Components

- Fragment of natural language (CNL)
- Formal Language (FL) for checking
  - Naproche-SAD: FOL+set theory
  - Hales/FAbstracts: Lean
- Translation from CNL to FL: parse + interpret

One-step design: translation defines

- syntax of CNL: parsing
- semantics of CNL: complex logical transformation

Design of Current CNLs

Components

- Fragment of natural language (CNL)
- Formal Language (FL) for checking
  - Naproche-SAD: FOL+set theory
  - Hales/FAbstracts: Lean
- Translation from CNL to FL: parse + interpret

One-step design: translation defines

- syntax of CNL: parsing
- semantics of CNL: complex logical transformation

Note: CNL can in principle be added to any FL

- not necessarily close to natural language
- readability may very

Our Envisioned Design

ILF: intermediate language (kernel)

- formal abstract syntax (grammar + inference system)
- concrete language unspecified

Concrete Syntax (frontend)

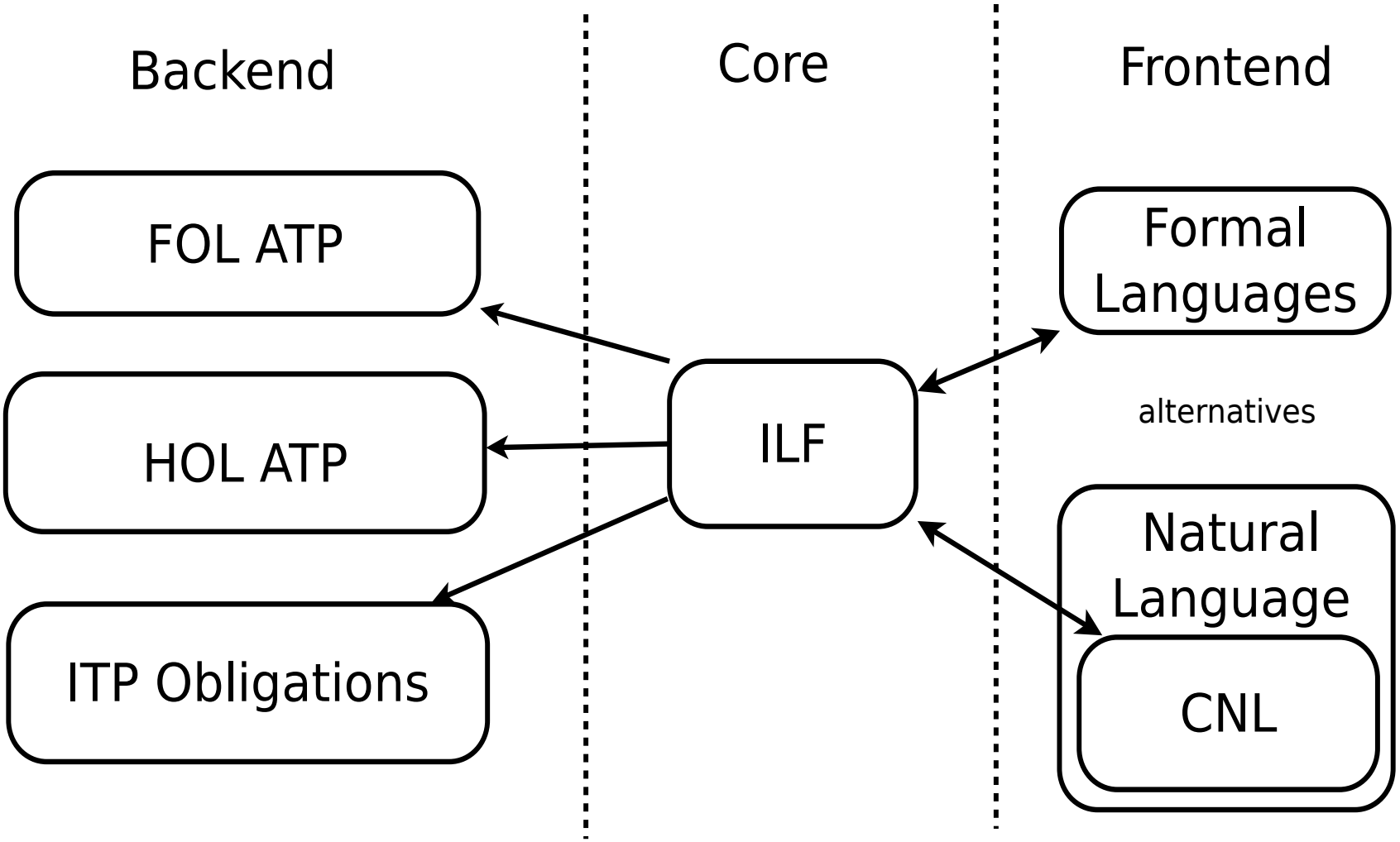
- compositional transformation abstract $\leftrightarrow$ concrete
- no non-trivial logical translations, encodings
- provide multiple alternatives
  - FL: formal logic-based (e.g., like Coq, Isabelle)
  - CNL: natural language-based (e.g., like ForTheL)

Theorem proving (backend)

- more specific logics optimized for proof support
- logical transformation ILF $\rightarrow$ ATP

possibly partial, possibly non-compositional

Architecture



Separation of Concerns

ILF

- captures structure of natural language

  better than both logics and CNLs

- formal, rigorous semantics

  complex task that needs attention in itself

  intermediate layer for two-step migration NL $\rightarrow$ FL

ILF $\leftrightarrow$ CNL

- captures concrete syntax, parsing user interface
- can reuse ideas from ForTheL, Mizar

ILF $\rightarrow$ ILF $\rightarrow$ FL

- ILF $\rightarrow$ ILF: logic translations to simplify the language
- ILF $\rightarrow$ (A)TP: provide proof support
- ILF $\rightarrow$ ITP: migrate NL content to proof assistant

Planned Implementation (major effort)

Implement ILF in multiple logical frameworks

- MMT: good for unrestricted experimentation
- Isabelle: good for quickly building proof support

Allow multiple frontends and provers

- implement parsers of FL and CNL concrete syntax as alternatives
- can be done in separate projects
- use interchange language to connect to implementations

Libraries

- conduct representative case studies to evaluate
- ensure building large library is feasible

Big Features we want in ILF

High-level structuring: modules, theories, records

- missing/weak in Mizar, ForTheL
- awkward solutions in Coq, Lean, Isabelle, ...

but unclear how to do better

Soft type system

- hard type systems have failed (Coq, Isabelle, ...)
- successful in Mizar, ForTheL
- but strong support must arise as emergent feature

e.g., dependent functions, subtypes, quotients

- must be undecidable and that's alright

Declarative proof system

- structured, assertion-based — successful in ForTheL, Mizar
- tactic-based proofs not readable
- must allow for gaps — so what if the prover fails to check it?

Small Features we want in ILF

small but subtle, and very difficult

Identification up to isomorphism

- demanded by Buzzard, Big Proof II
- very difficult, basically no prior work

Generated structures

e.g., $\mathbb{R}[X]$, $\mathrm{Group}\langle r, f | r^n, f^2, (rf)^2 \rangle$

- requires using terms as values
- but terms up to equality
- little prior work

Conversions

- use function-like objects like functions

  e.g., evaluating a polynomial
- use type-like objects like type e.g., use group like its universe
- seamlessly cast along forgetful functors, canonical projections/embeddings