Symbolic Informalization:
Fluent, Productive, Multilingual

MathCompLing, Institut Pascal, Orsay, 15 September 2025

(Extended from AITP, Aussois, 2 September 2025
also based on ENS Saclay April 2025 and others)

Aarne Ranta

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

aarne.ranta@cse.gu.se

Background:

How could AI solve mathematical problems?

Math, Inc.

A new company dedicated to autoformalization and the creation of verified superintelligence.

Introducing Gauss, an agent for autoformalization

Solve math, solve everything.

https://www.math.inc/

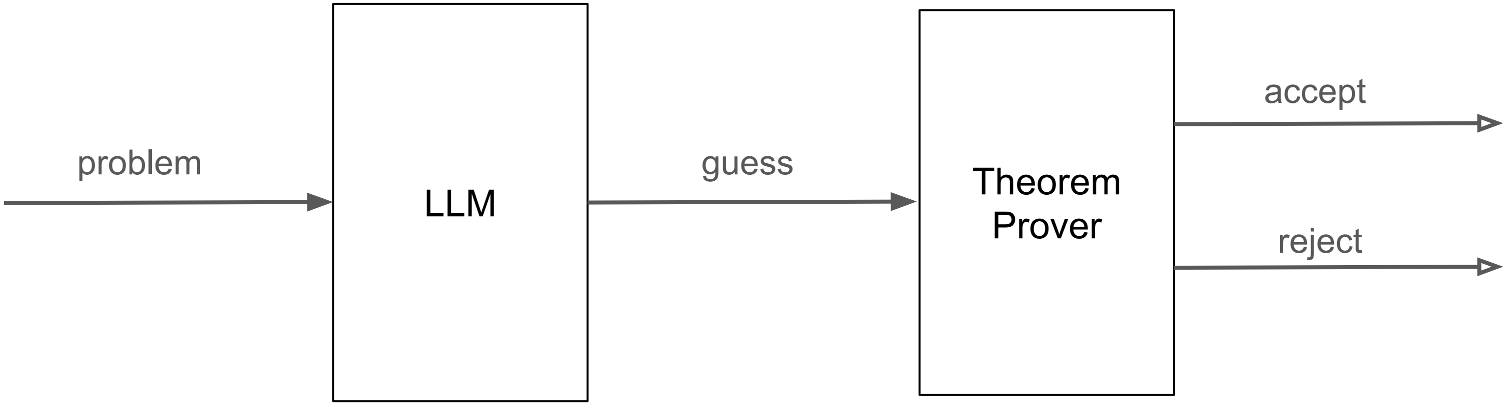
Symbolic AI (theorem provers)

- reliable: no "hallucinations"
- restricted problem solving capacity

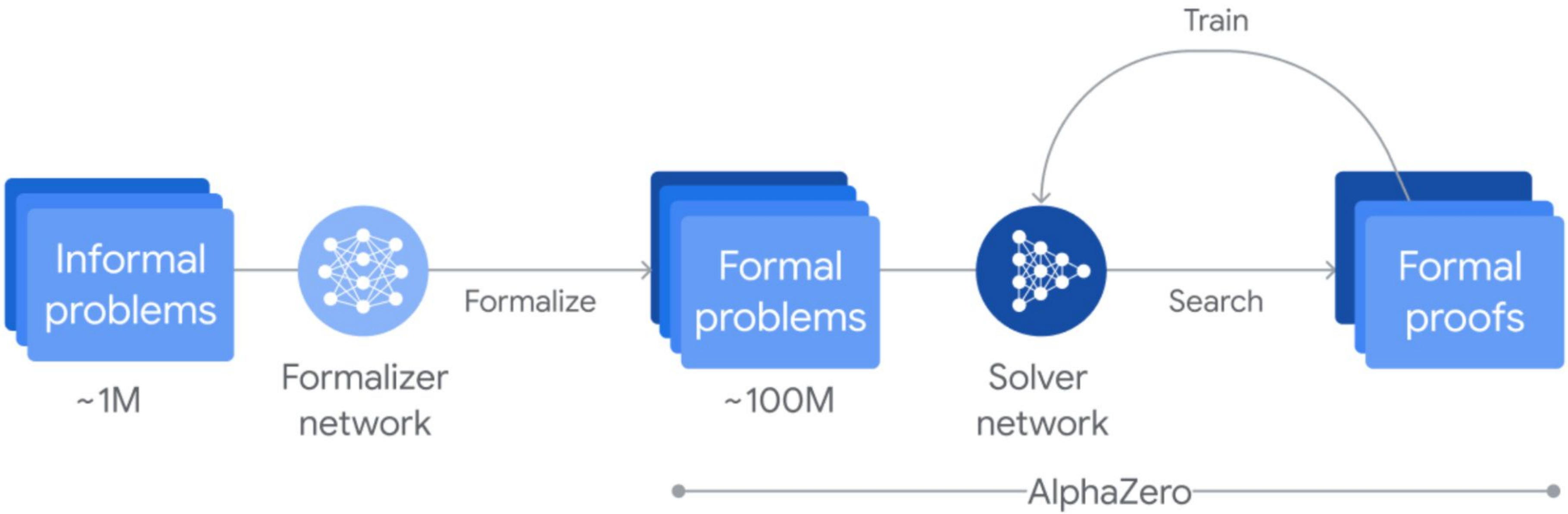
Neural AI (large language models)

- unreliable: "hallucinations"
- can find unexpected solutions

Federated systems

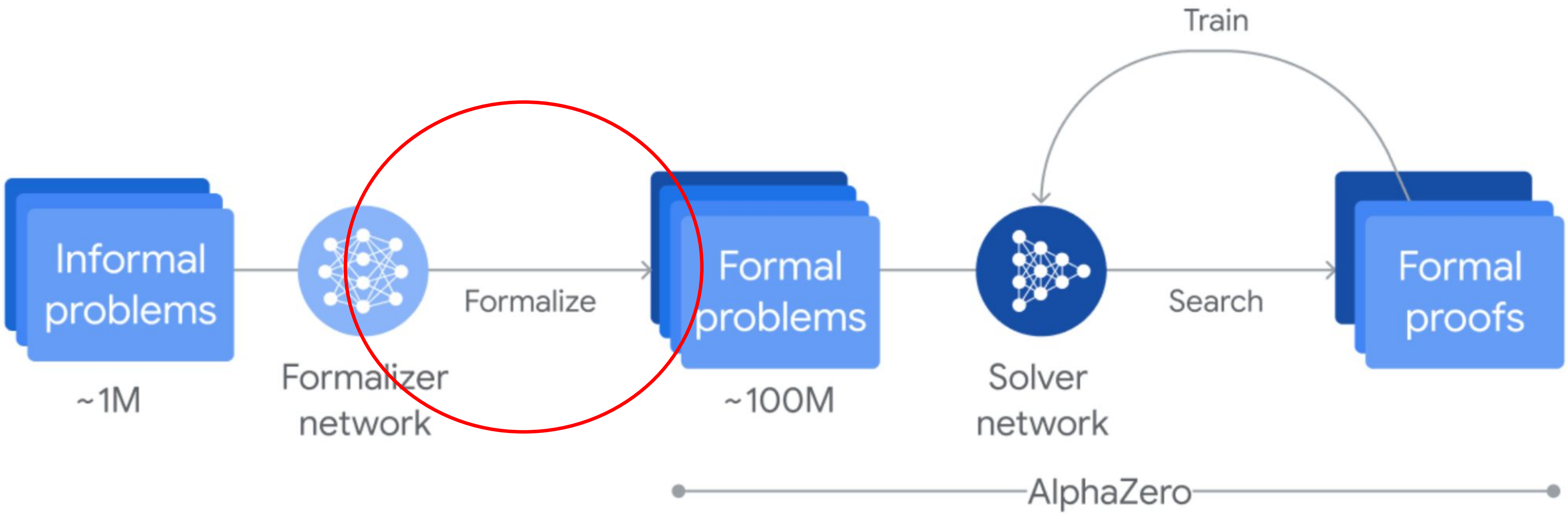


2024: "AI achieves silver-medal standard solving International Mathematical Olympiad problems"



https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

AI achieves silver-medal standard solving International Mathematical Olympiad problems



"First, the problems were manually translated into formal mathematical language for our systems to understand."

https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

Autoformalization

= automatic formalization

= automatic translation from informal to formal

A "hot topic" due to AI such as Google's AlphaProof

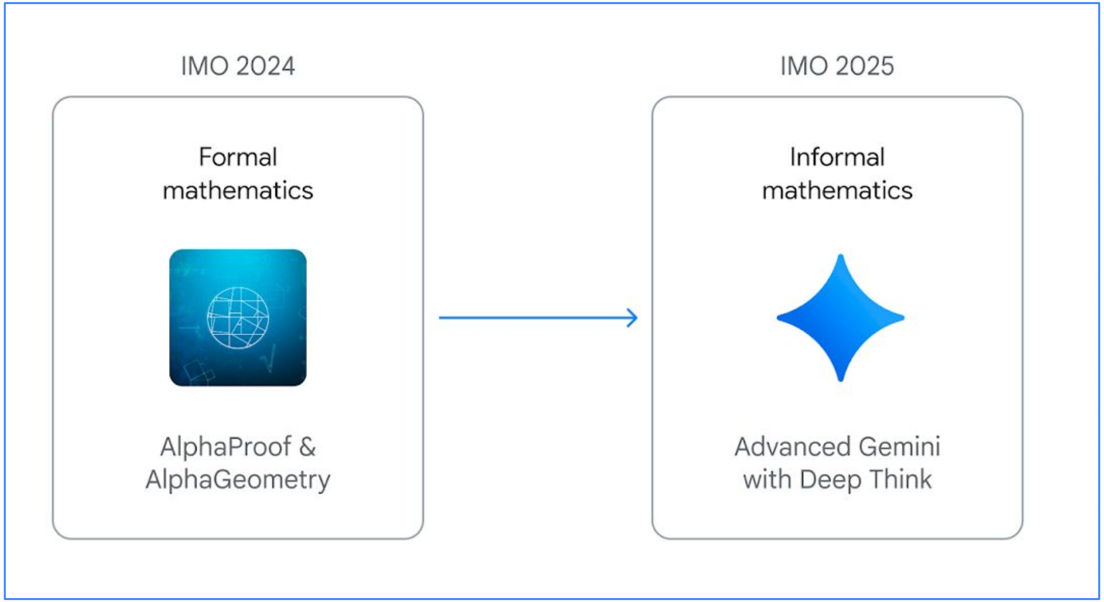
Symbolic autoformalization

- Controlled Natural Languages (CNLs)
- "brittle": only covers fragments of informal mathematics

Neural autoformalization

- LLMs "learn" to autoformalize from large amounts of data
- unreliable: typically 30% "adequate with minor corrections"
- problem: lack of training data (formal-informal pairs)

Side track: IMO 2025



At IMO 2024, AlphaGeometry and AlphaProof required experts to first translate problems from natural language into domain-specific languages, such as Lean, and vice-versa for the proofs… This year, our advanced Gemini model operated end-to-end in natural language.
https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/

Why a side track?

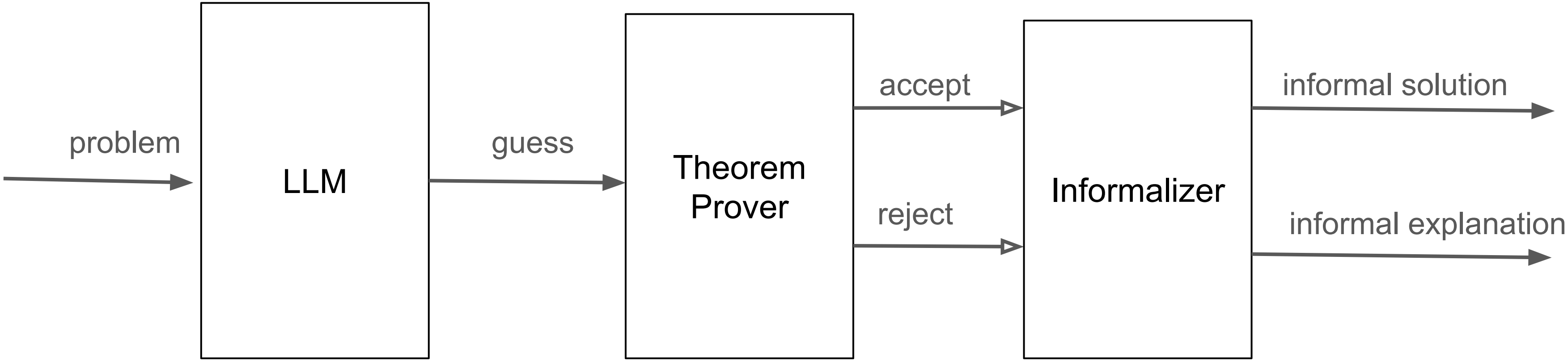
Solutions had to be checked by humans

- just like the human participants' solutions

This is natural in the context of IMO competitions

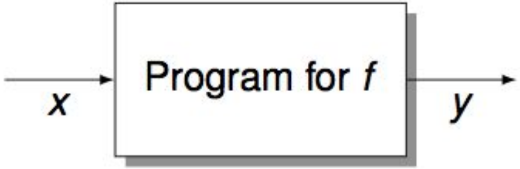
But it does not scale up to enable automated, reliable AI systems.

Checked informal solutions?



In a wider picture

The Problem



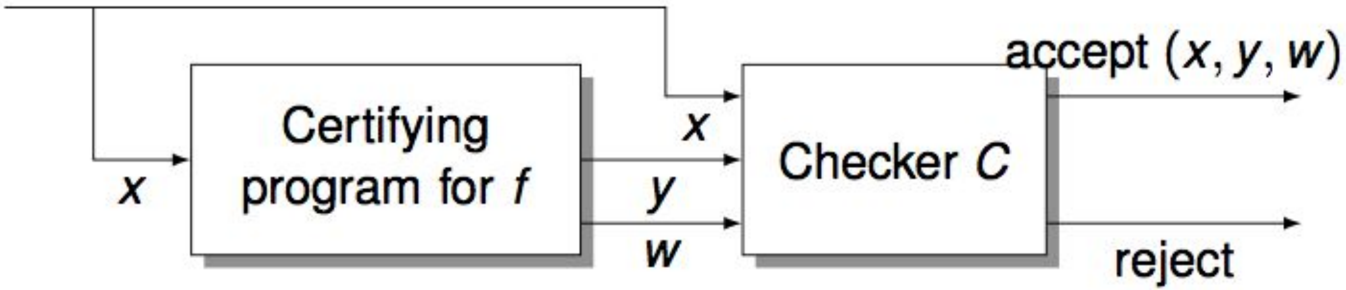
- A user feeds x to the program, the program returns y.
- How can the user be sure that, indeed,

$$y = f(x)?$$

The user has no way to know.

Kurt Mehlhorn, Certifying Algorithms
https://people.mpi-inf.mpg.de/~mehlhorn/ftp/SAPJuly2014.pdf

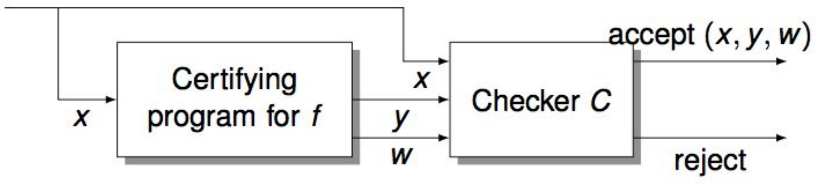
A Certifying Program for a Function f



- On input x, a certifying program returns the function value y and a certificate (witness) w
- w proves $y = f(x)$ even to a dummy,
- and there is a simple program C, the checker, that verifies the validity of the proof.

Kurt Mehlhorn, Certifying Algorithms
https://people.mpi-inf.mpg.de/~mehlhorn/ftp/SAPJuly2014.pdf

A Certifying Program for a Function *f*



- On input x, a certifying program returns the function value y and a certificate (witness) w
- w proves $y = f(x)$ even to a dummy,
- and there is a simple program C, the checker, that verifies the validity of the proof.

Function value *y:*
- informal proof from LLM

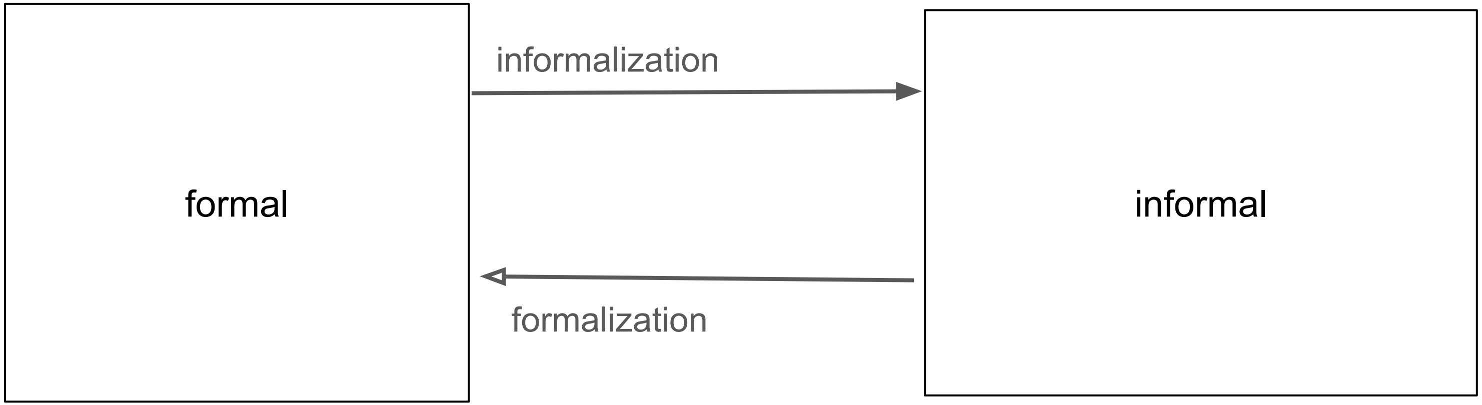
Proof *w*:
- the corresponding formal proof

Checker *C*:
- formal proof checker

Certifying program:
- jointly produces *y* and *w*

Remains to verify:
- **that the informal proof y really matches the formal proof w**

Our most important slide



total	$\longrightarrow$
partial	$\rightarrowtriangle$

Don't guess if you know.

- Kimmo Koskenniemi

- there is no essential need for non-symbolic (neural) informalization
- (except its allegedly low cost)

- However, (auto)formalization may require guessing

- symbolic informalization has things to contribute even there
  - synthetic data generation
  - verification feedback

The challenge

Multi-language Diversity Benefits Autoformalization

Albert Q. Jiang
University of Cambridge
qj213@cam.ac.uk

Wenda Li
University of Edinburgh
wenda.li@ed.ac.uk

Mateja Jamnik
University of Cambridge
mateja.jamnik@cl.cam.ac.uk

- "informalisation is much easier than formalisation"

- uses an GPT-4 to produce the dataset MMA to fine-tune LLaMA
    - ~70% "more or less acceptable"

- resulting autoformalization:
    - 16-18% "acceptable with minimal corrections"

- ~~symbolic informalization~~

NeurIPS 2024,

"symbolic informalisation tools

- result in natural language content that lacks the inherent diversity and flexibility in expression: they are rigid and not natural-language-like.

- symbolic informalisation tools are hard to design and implement

- They also differ a lot for different formal languages, hence the approach is not scalable for multiple formal languages. "

Our goal

Symbolic informalization that

has

- results in natural language content that ~~lacks~~ the inherent diversity and flexibility in expression: they are ~~rigid and not~~ natural-language-like.

  feasible

- symbolic informalisation tools are ~~hard~~ to design and implement ***with proper methods***

  can be shared

- They ~~also differ a lo~~t for different formal languages, hence the approach is ~~not~~ scalable for multiple formal languages. ***And even for multiple natural languages.***

Our goal

Symbolic informalization that

- results in natural language content that *has* ~~lacks~~ the inherent diversity and flexibility in expression: they are ~~rigid and not~~ natural-language-like. **FLUENT**
- symbolic informalisation tools are *feasible* ~~hard~~ to design and implement ***with proper methods*** **PRODUCTIVE**
- They *can be shared* ~~also differ a lo~~t for different formal languages, hence the approach is ~~not~~ scalable for multiple formal languages. ***And even for multiple natural languages.*** **MULTI-LINGUAL**

Symbolic informalization

CNL

Trybulec 1973: Mizar

Coscoy, Kahn & Théry 1994: extracting text from Coq proofs

Wenzel 1999: Isabelle-Isar

Hallgren & Ranta 2000: GF-Alfa (Agda)

Paskevich 2007: ForTheL

Cramer, Koepke & al 2009: Naproche

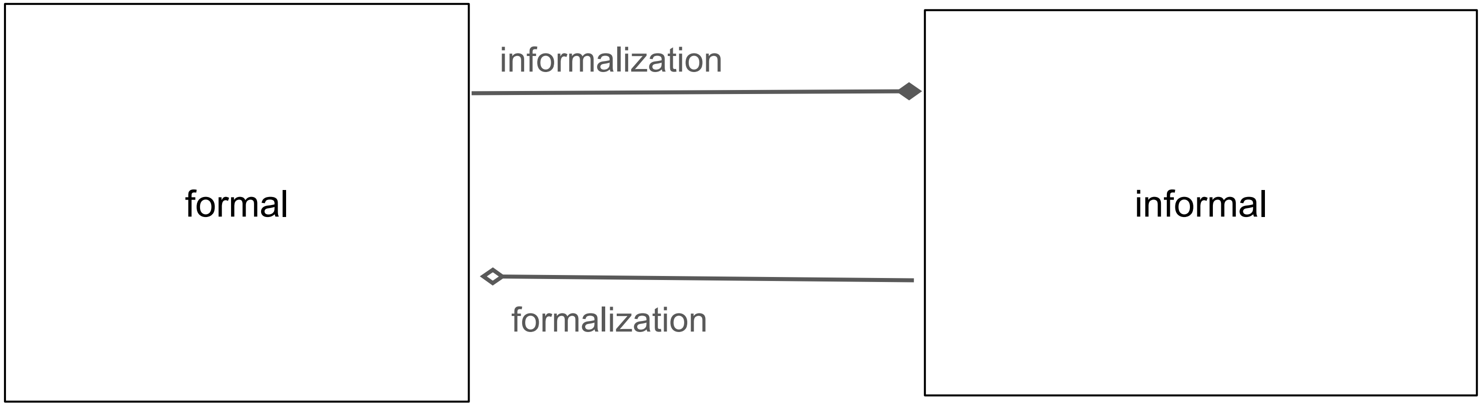
Humayoun & Raffalli 2011: MathNat

Pathak 2023: GF-Lean

Massot 2024: Verbose-Lean4

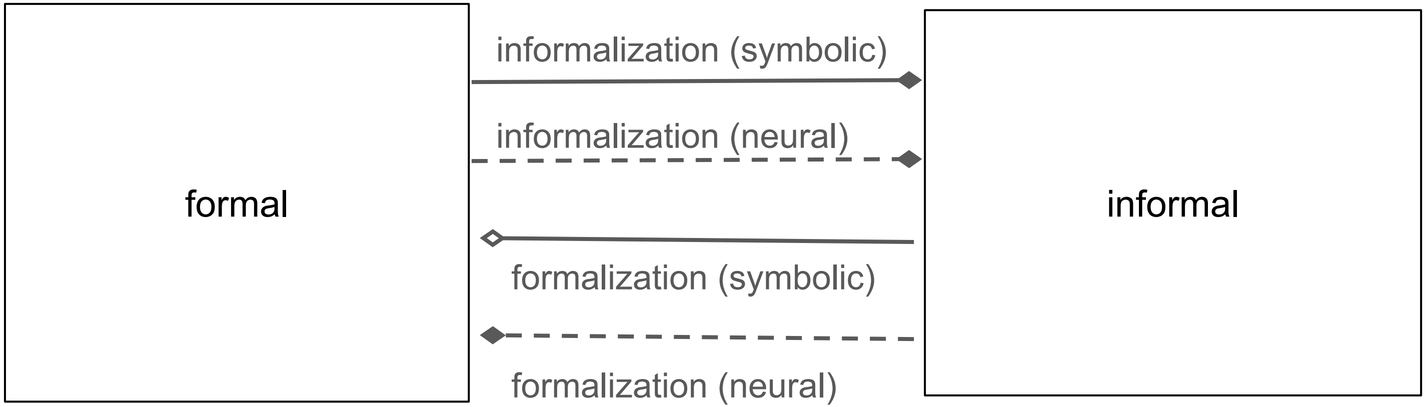
Kelber, Kohlhase, Schaefer & Schütz: Flexiformal mathematics, 2025

Extending pure CNL: from one to many



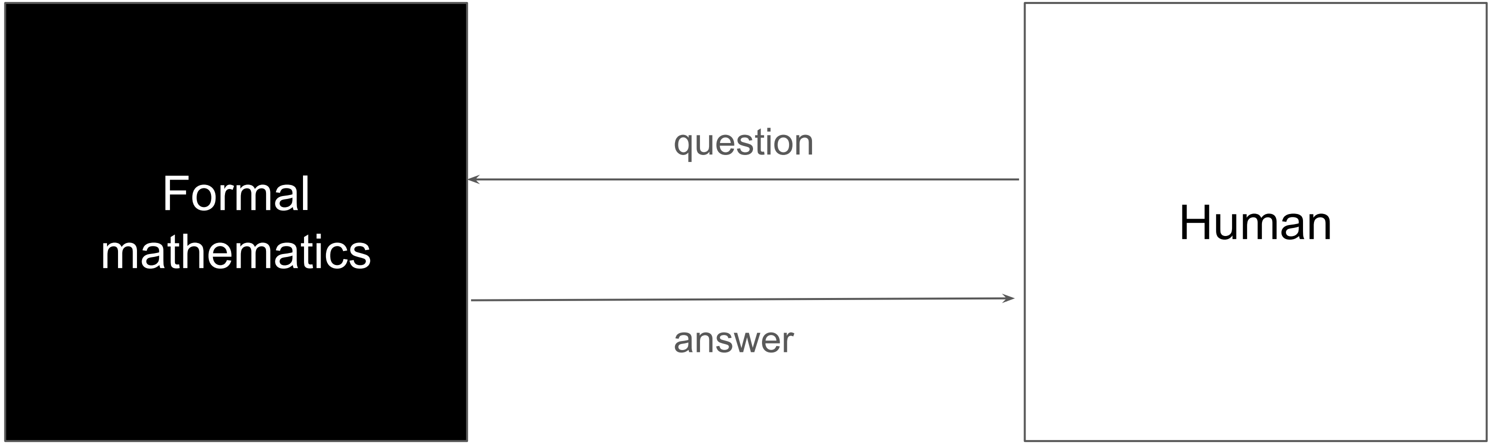
	to one	to many
total		
partial		

Symbolic vs. neural



	certain	uncertain
total		
partial		

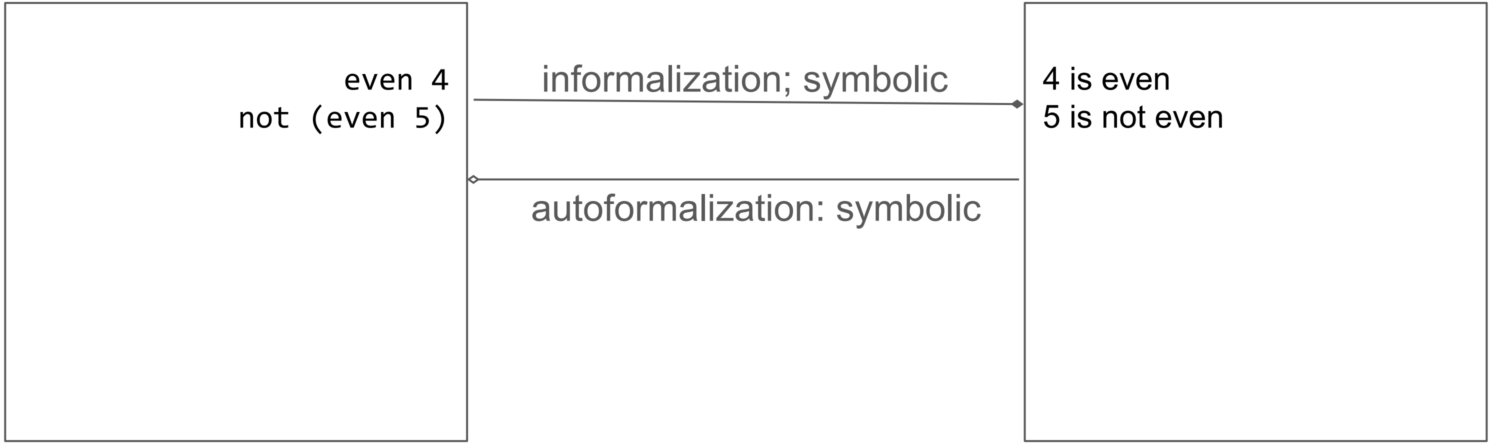
Verification feedback

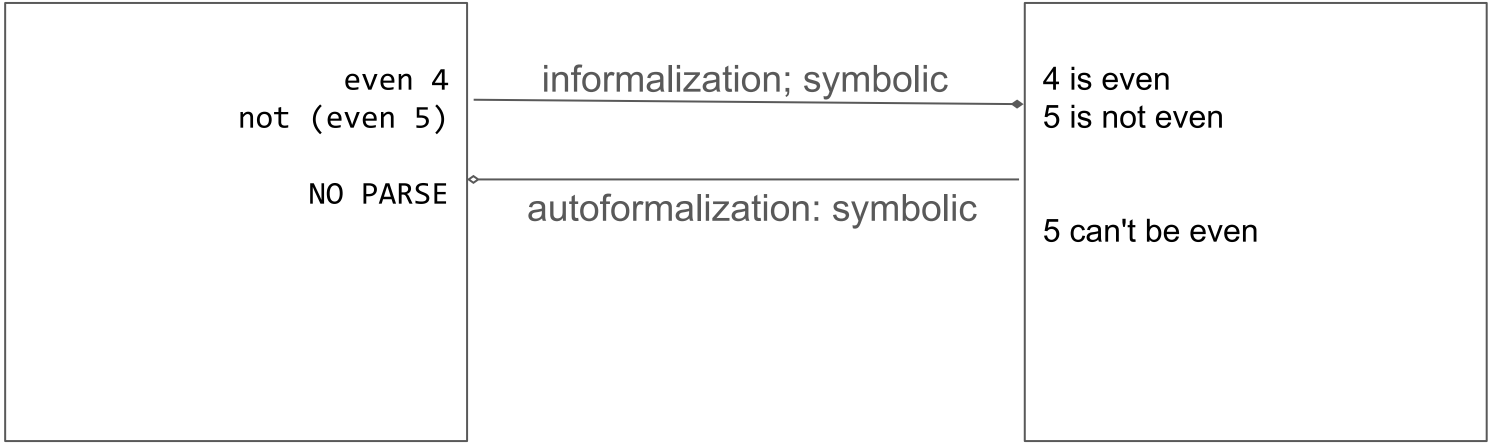


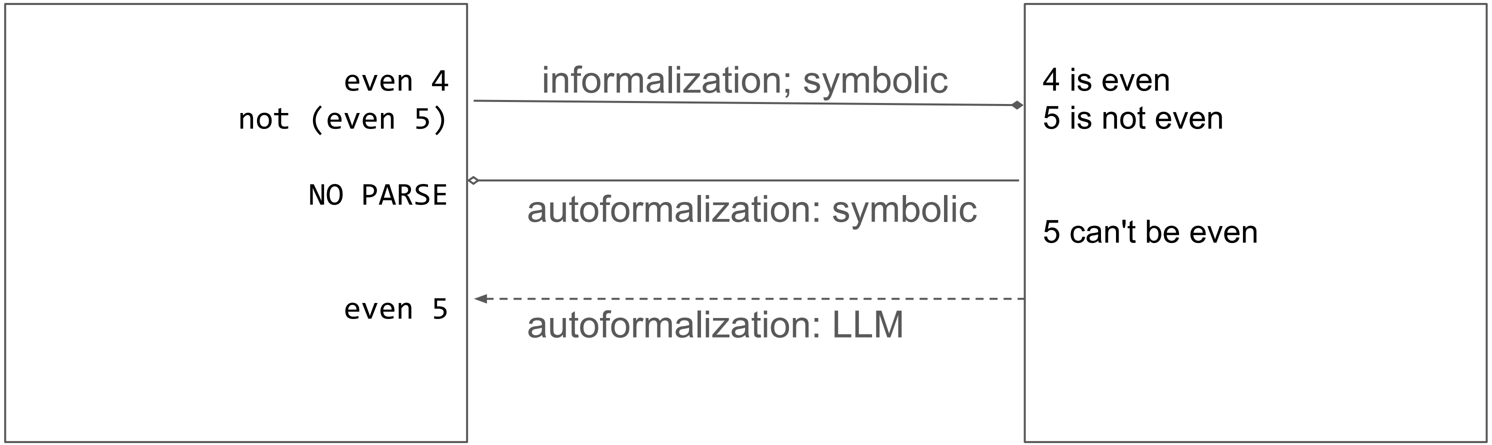
Vision:

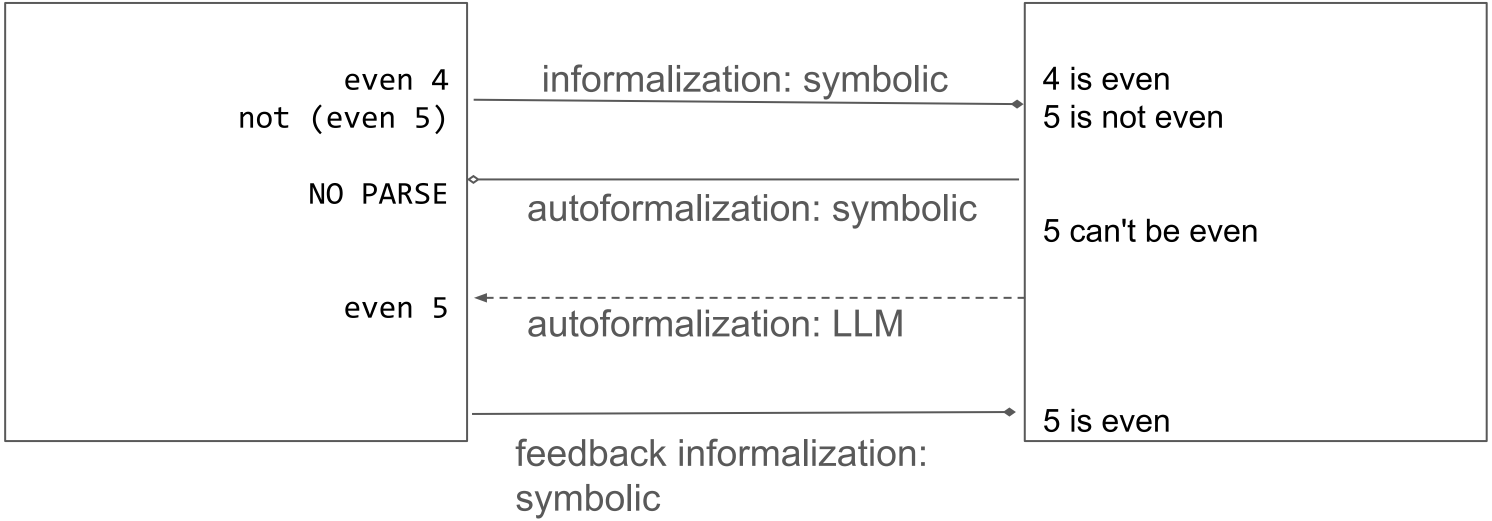
- the formal system is a black box that performs verification
- humans communicate with it in natural language

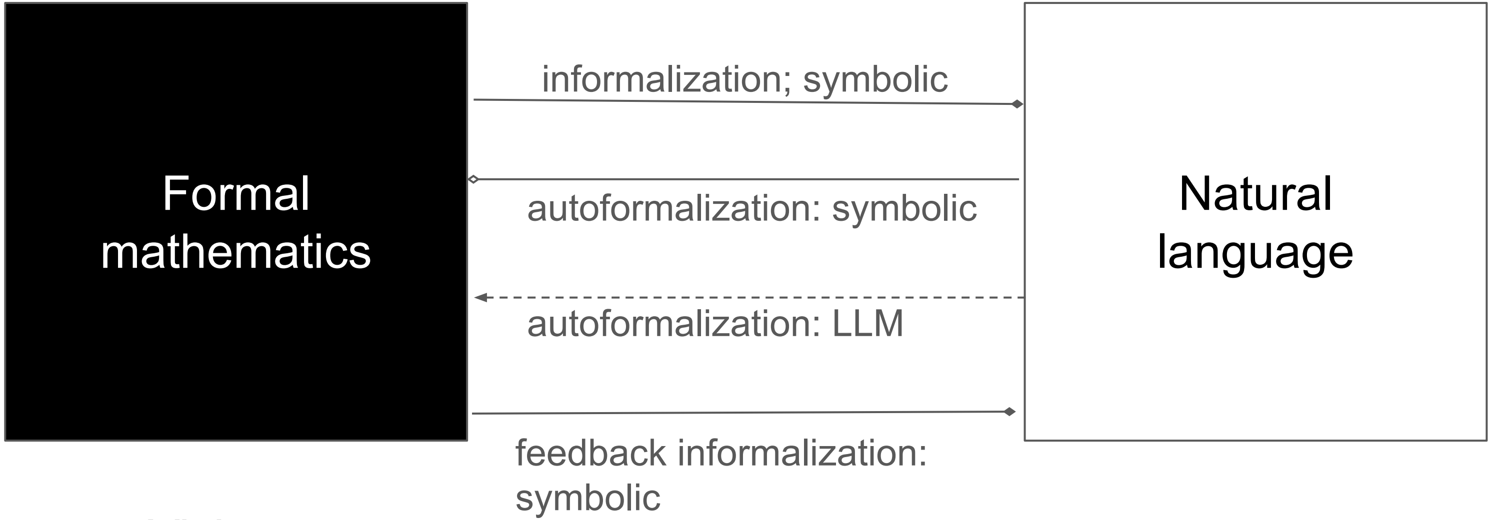
But how can they trust it?


even 4
not (even 5)
informalization; symbolic
4 is even
5 is not even
autoformalization: symbolic


even 4
not (even 5)
NO PARSE
informalization; symbolic
autoformalization: symbolic
4 is even
5 is not even
5 can't be even


even 4
not (even 5)
NO PARSE
even 5
informalization; symbolic
autoformalization: symbolic
autoformalization: LLM
4 is even
5 is not even
5 can't be even


even 4
not (even 5)
NO PARSE
even 5
informalization: symbolic
autoformalization: symbolic
autoformalization: LLM
feedback informalization: symbolic
4 is even
5 is not even
5 can't be even
5 is even



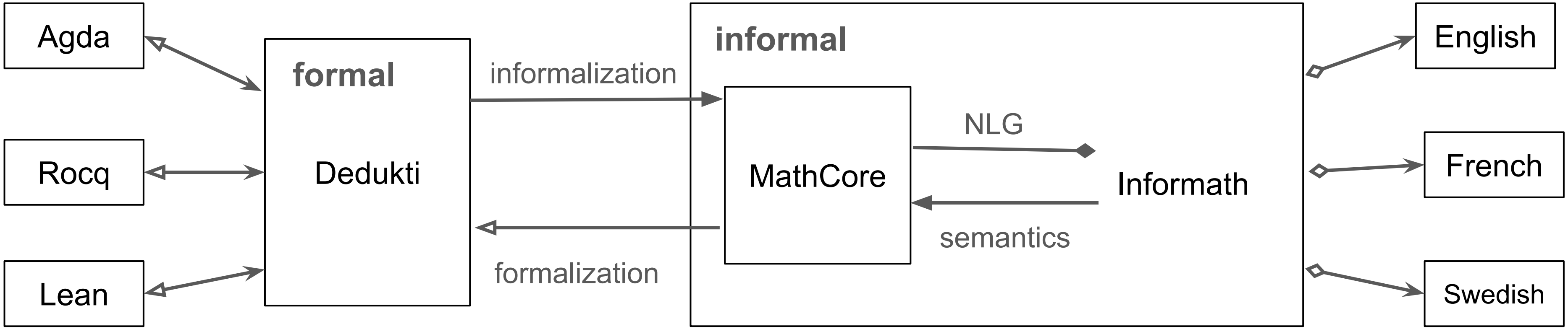
Vision:

- the formal system is a black box that performs verification
- humans communicate with it in natural language

Symbolic informalization is a certificate of the system's understanding.

Project Informath

- informalizing formal mathematics
  - multilingual, productive, fluent



	to one	to many
total		
partial		

https://github.com/GrammaticalFramework/informath

Multilingual

Agda:

```
postulate prop110 :
  (a : Int) -> (c : Int) ->
  and (odd a) (odd c) -> all Int (\ b ->
    even (plus (times a b) (times b c)))
```

Rocq:

```
prop110 : forall a : Int, forall c : Int,
  (odd a /\ odd c -> forall b : Int,
   even (a * b + b * c)) .
```

Lean:

```
prop110 (a c : Int) (x : odd a ∧ odd c)
:
  ∀ b : Int, even (a * b + b * c)
```

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then $ab + bc$ is even for all integers b.

Agda:

```
postulate prop110 :
  (a : Int) -> (c : Int) ->
  and (odd a) (odd c) -> all Int (\ b ->
    even (plus (times a b) (times b c)))
```

Rocq:

```
prop110 : forall a : Int, forall c : Int,
  (odd a /\ odd c -> forall b : Int,
   even (a * b + b * c)) .
```

Lean:

```
prop110 (a c : Int) (x : odd a ∧ odd c)
:
  ∀ b : Int, even (a * b + b * c)
```

Dedukti:

```
prop110 : (a : Elem Int) ->
  (c : Elem Int) ->
    Proof (and (odd a)
   (odd c)) ->
    Proof (forall Int
   (b => even (plus
   (times a b) (times b c)))).
```

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then $ab + bc$ is even for all integers b.

Dedukti

A "logical framework based on the λΠ-calculus modulo in which many theories and logics can be expressed"

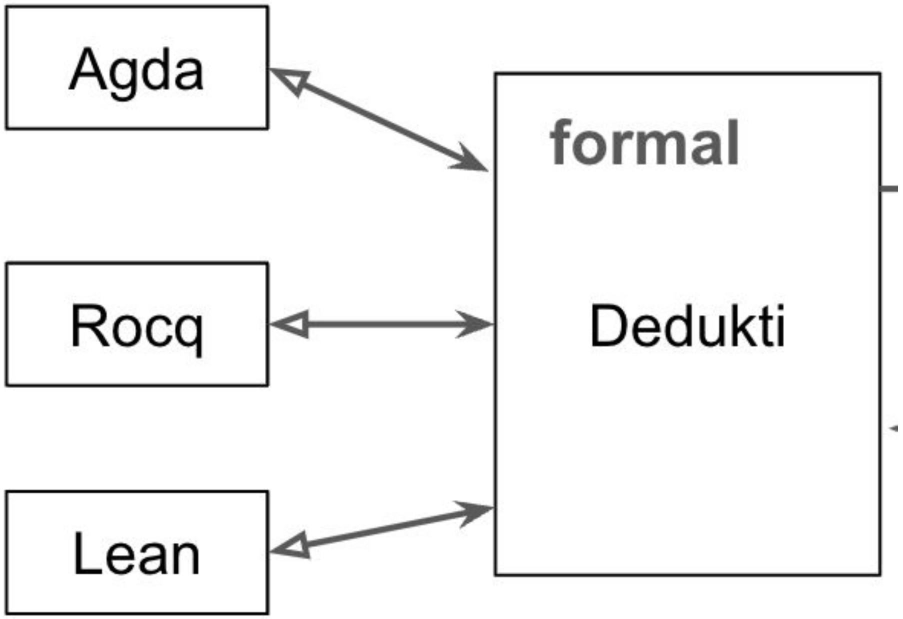
- Agda, HOL, Lean, Rocq (Coq), TPTP, …

Simpler but more powerful (i.e. more liberal) than any of these individually.

- dependent types and Pi types
- lambda abstracts
- rewrite rules
- (almost) no syntactic sugar

Similar to Martin-Löf's LF from the 1980s and to TWELF, ALF,... (for those who remember)

https://deducteam.github.io/



Agda:

```
postulate prop110 :
  (a : Int) -> (c : Int) ->
  and (odd a) (odd c) -> all Int (\ b ->
    even (plus (times a b) (times b c)))
```

Rocq:

```
prop110 : forall a : Int, for
  (odd a /\ odd c -> forall b
   even (a * b + b * c)) .
```

Dedukti:

```
prop110 : (a : Elem Int) ->
  (c : Elem Int) ->
    Proof (and (odd a)
   (odd c)) ->
    Proof (forall Int
   (b => even (plus
   (times a b) (times b c)))).
```

Lean:

```
prop110 (a c : Int) (x : odd a ∧ odd c)
:
  ∀ b : Int, even (a * b + b * c)
```

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then $ab + bc$ is even for all integers b.

Prop110. Soient $a, c \in Z$. Supposons que a et c sont impairs. Alors $ab + bc$ est pair pour tous les entiers b.

Prop110. Låt $a, c \in Z$. Anta att både a och c är udda. Då är $ab + bc$ jämnt för alla heltal b.

Agda:

```
postulate prop110 :
  (a : Int) -> (c : Int) ->
  and (odd a) (odd c) -> all Int (\ b ->
    even (plus (times a b) (times b c)))
```

Rocq:

```
prop110 : forall a : Int, for
  (odd a /\ odd c -> forall b
   even (a * b + b * c)) .
```

Dedukti:

```
prop110 : (a : Elem Int) ->
  (c : Elem Int) ->
    Proof (and (odd a)
   (odd c)) ->
    Proof (forall Int
   (b => even (plus
   (times a b) (times b c)))).
```

Lean:

```
prop110 (a c : Int) (x : odd a ∧ odd c)
:
  ∀ b : Int, even (a * b + b * c)
```

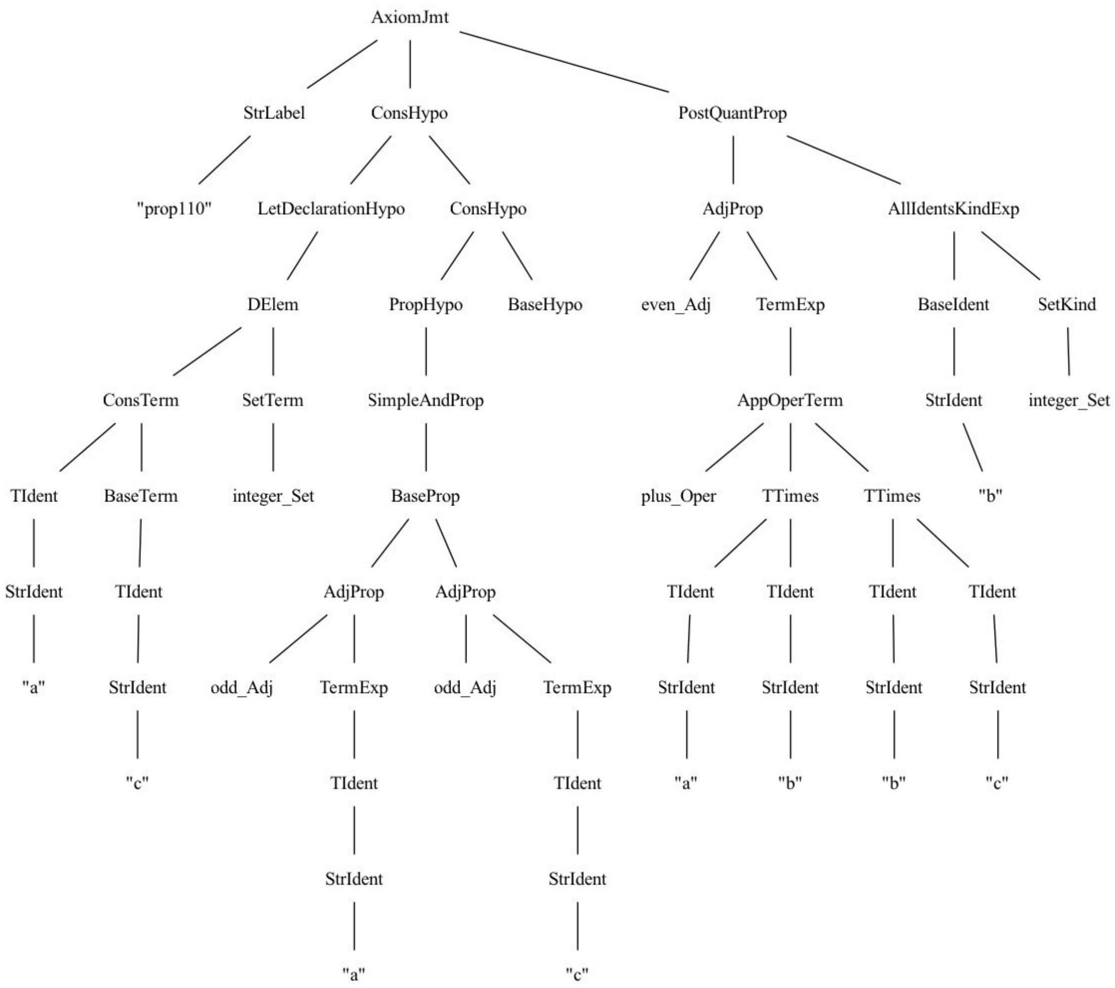
GF:

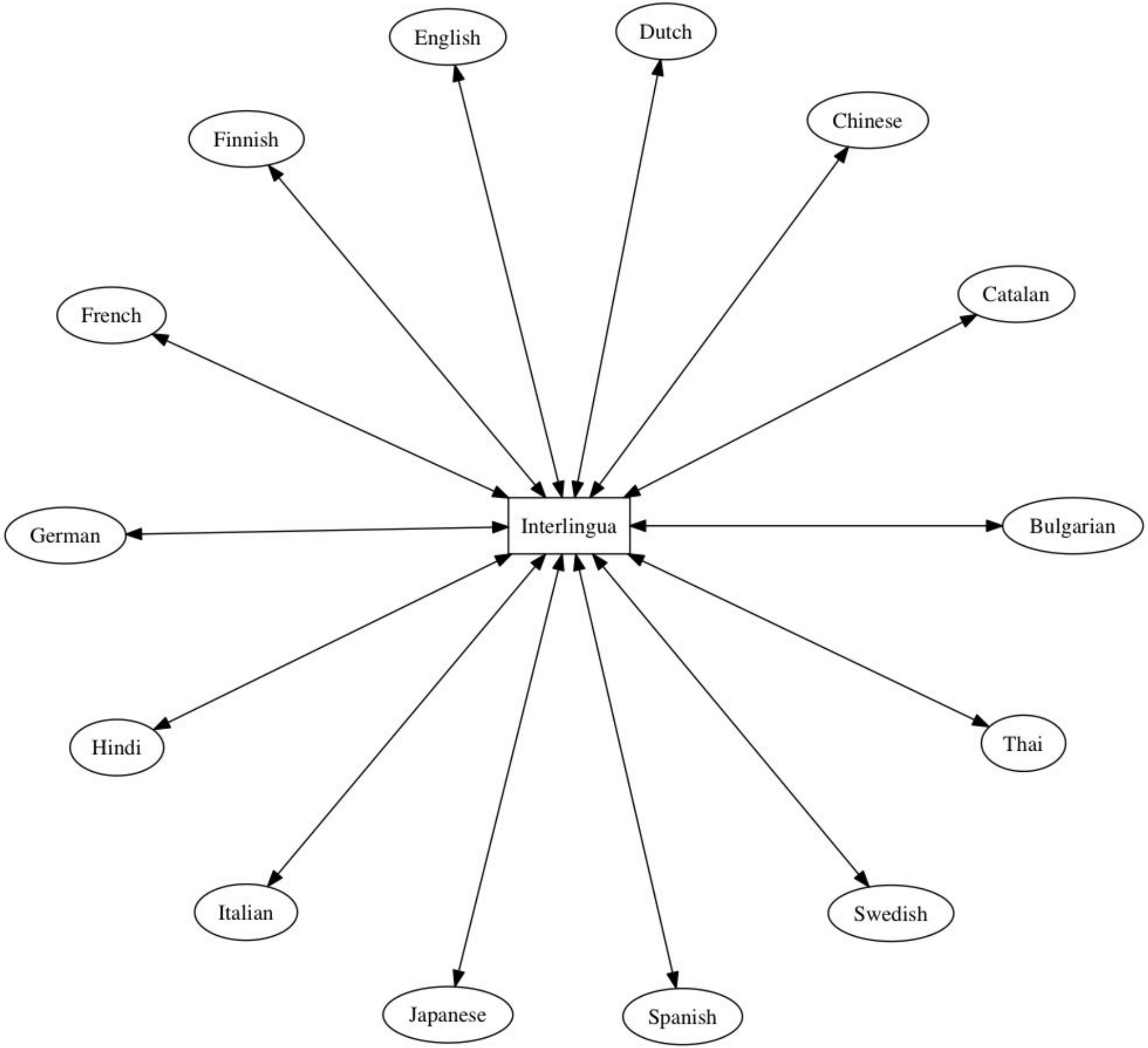
```
AxiomJmt (StrLabel "prop110")
(ConsHypo (LetFormulaHypo (FElem
(ConsTerm (TIdent (StrIdent "a"))
(BaseTerm (TIdent (StrIdent "c"))))
(SetTerm integer_Set))) (ConsHypo
(PropHypo (AdjProp odd_Adj (AndExp
(BaseExp (TermExp (TIdent (StrIdent
"a"))) (TermExp (TIdent (StrIdent
"c"))))))) BaseHypo)) (PostQuantProp
(AdjProp even_Adj (TermExp
(AppOperTerm plus_Oper (TTimes (TIdent
(StrIdent "a")) (TIdent (StrIdent
"b"))) (TTimes (TIdent (StrIdent "b"))
(TIdent (StrIdent "c")))))))
(AllIdentsKindExp (BaseIdent (StrIdent
"b")) (SetKind integer_Set)))
```

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then $ab + bc$ is even for all integers b.

…t $a, c \in Z$. Supposons que …airs. Alors $ab + bc$ est pair …ntiers b.

Prop110. Låt $a, c \in Z$. Anta att både a och c är udda. Då är $ab + bc$ jämnt för alla heltal b.


AxiomJmt
StrLabel
ConsHypo
PostQuantProp
"prop110"
LetDeclarationHypo
ConsHypo
AdjProp
AllIdentsKindExp
DElem
PropHypo
BaseHypo
even_Adj
TermExp
BaseIdent
SetKind
ConsTerm
SetTerm
SimpleAndProp
AppOperTerm
StrIdent
integer_Set
TIdent
BaseTerm
integer_Set
BaseProp
plus_Oper
TTimes
TTimes
"b"
StrIdent
TIdent
AdjProp
AdjProp
TIdent
TIdent
TIdent
TIdent
"a"
StrIdent
odd_Adj
TermExp
odd_Adj
TermExp
StrIdent
StrIdent
StrIdent
StrIdent
"c"
TIdent
TIdent
"a"
"b"
"b"
"c"
StrIdent
StrIdent
"a"
"c"


English
Dutch
Finnish
Chinese
Catalan
French
German
Interlingua
Bulgarian
Hindi
Thai
Italian
Swedish
Japanese
Spanish

Interlude: GF

GF = Grammatical Framework

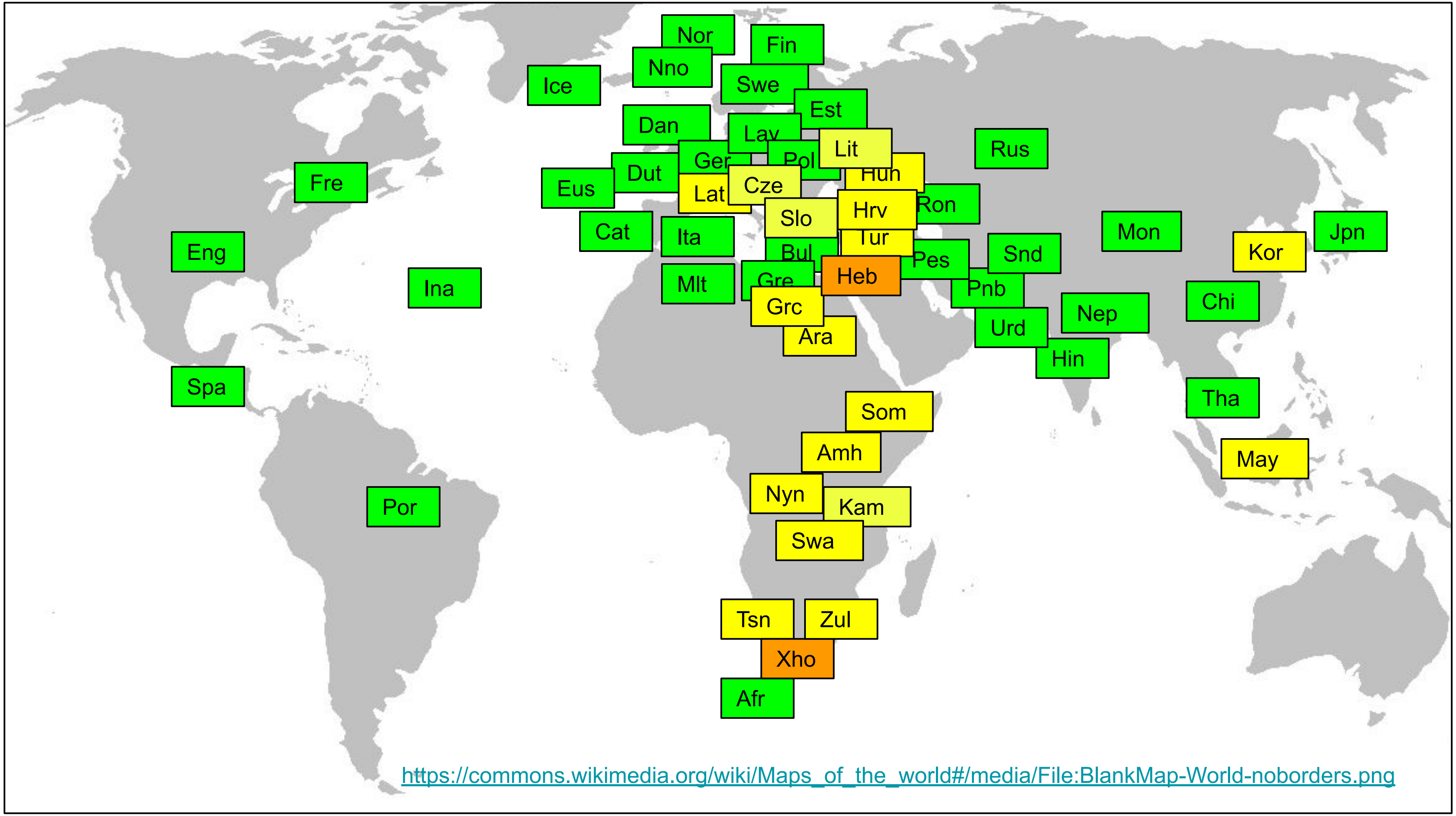
GF = Logical Framework + Grammar

First release 1998 at Xerox Research Centre Europe, Grenoble

Based on earlier work with ALF (Another LF, predecessor of Agda) 1992

Mission: *formalize the grammars of the world and make them available for computer applications.*

https://www.grammaticalframework.org/



RGL = Resource Grammars Library, created by the GF community 2001-2025

Abstract and concrete syntax: judgements

```
-- abstract syntax = LF

cat C  Γ

fun f : T

def t = u
```

```
-- concrete syntax

lincat C = L

lin f = t

param P = C | … | C

oper h : T = t
```

Abstract and concrete syntax: examples

```
-- abstract syntax = LF

cat Prop ; Term

fun commutative : Term -> Prop

def commutative f =
  forall Obj (\x, y ->
     Id Obj (f x y) (f y x)
```

```
-- concrete syntax

lincat Prop, Term = Str

lin commutative x =
     x ++ "is commutative"
```

Concrete syntax: parameters and operations

```
-- abstract syntax = LF

cat Prop ; Term

fun commutative : Term -> Prop
```

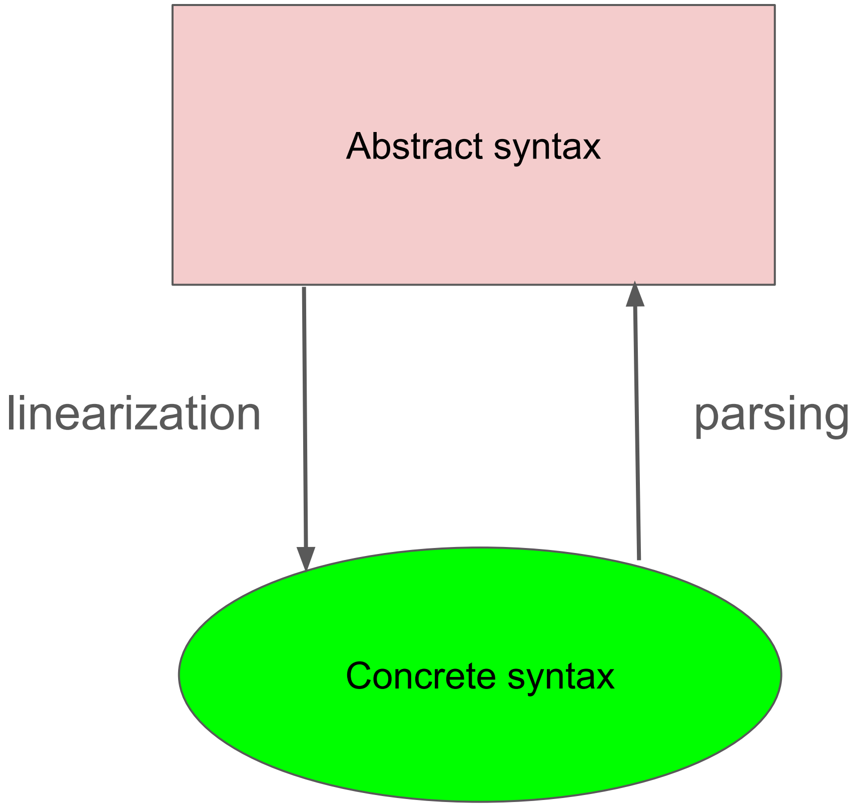
```
-- concrete syntax for English

lincat
  Prop = Str
  Term = {s : Str ; n : Number}

lin commutative x = x.s ++
   copula ! x.n ++ "commutative"

param
  Number = Sg | Pl

oper
  copula : Number => Str =
    table {Sg => "is" ; Pl => "are"}
```

Concrete syntax: parameters and operations

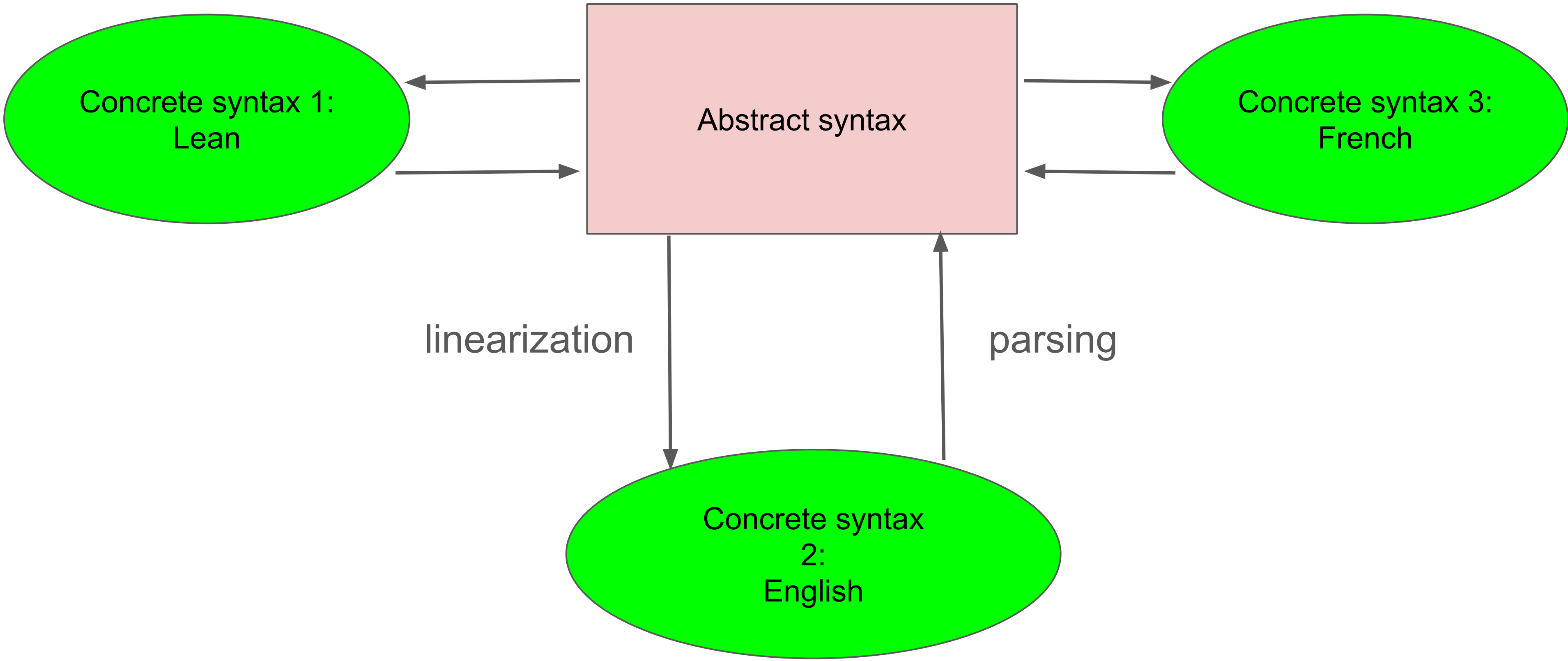
```
-- abstract syntax = LF

cat Prop ; Term

fun commutative : Term -> Prop
```

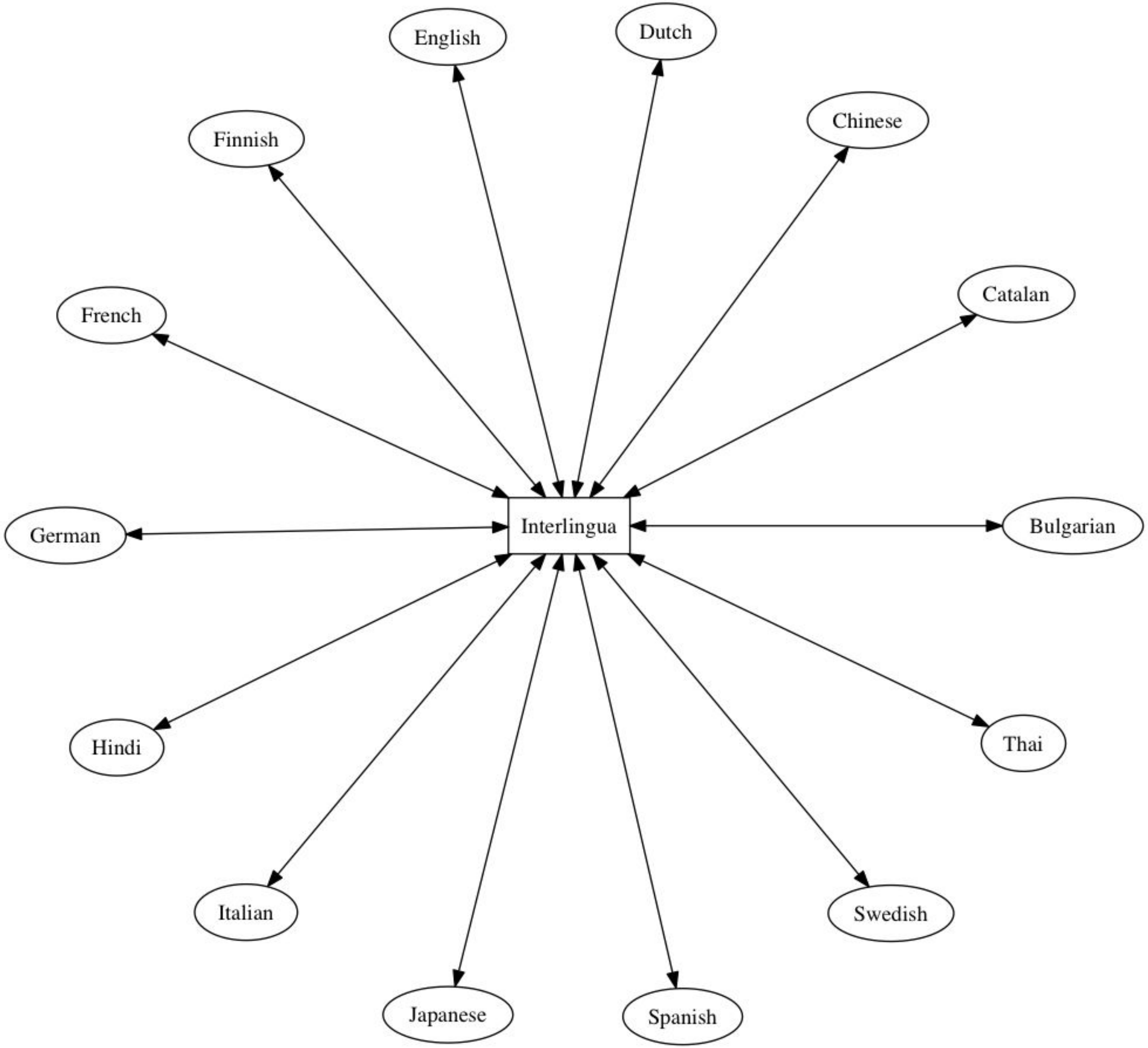
```
-- concrete syntax for French

lincat
  Prop = Mood => Str
  Term = {s : Str ; g : Gender ; n : Number}

lin commutative x = \\m => x.s ++
   copula ! m ! n ++
   mkA "commutatif" ! x.g ! x.n

param
   Number = Sg | Pl
   Gender = Masc | Fem
   Mood = Ind | Subj
oper
  mkA : Str -> Gender => Number = Str = ...
  copula : Mood => Number => Str = ...
```

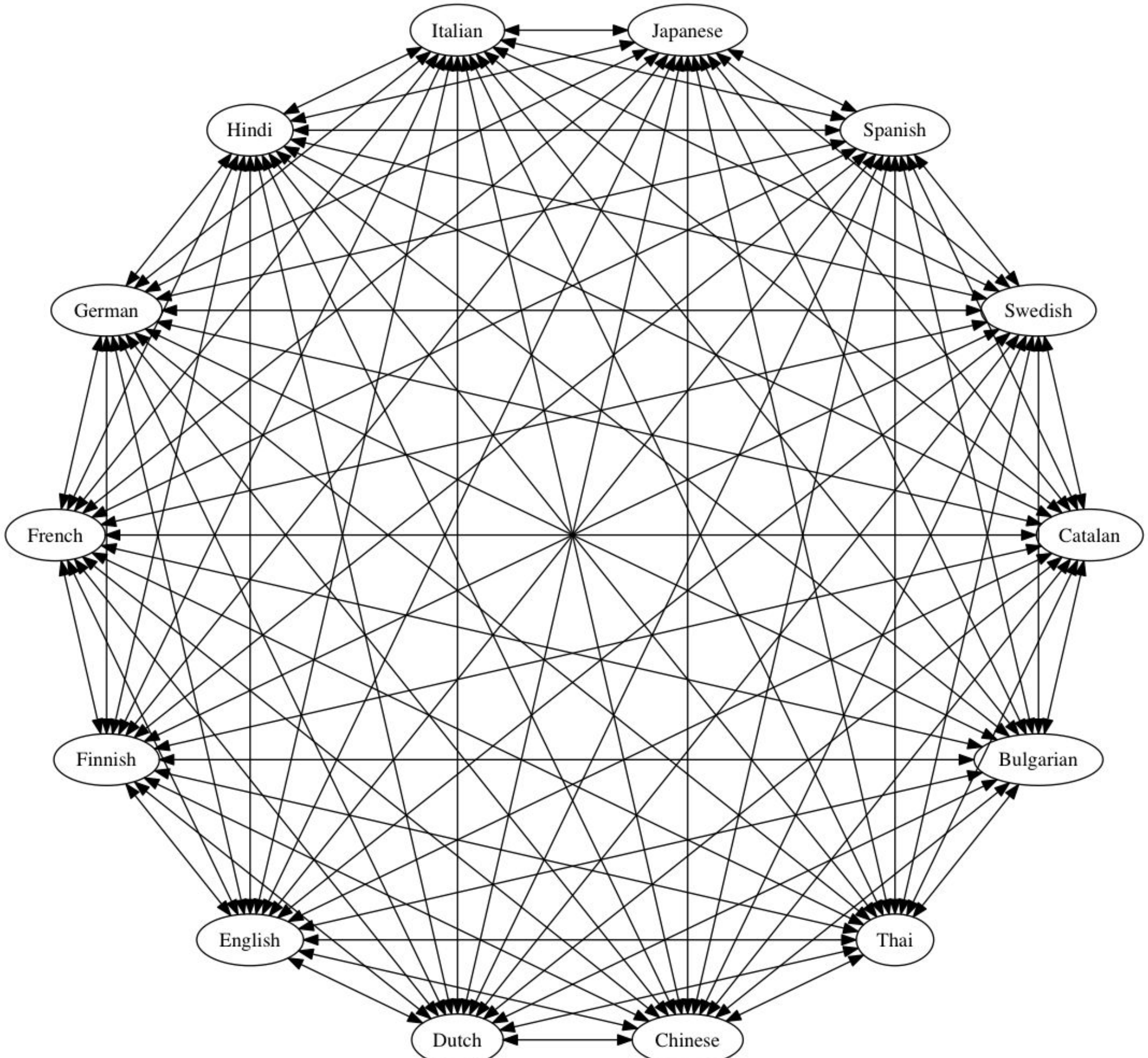
Reversible mappings



Multilingual grammars




English
Dutch
Finnish
Chinese
Catalan
French
German
Interlingua
Bulgarian
Hindi
Thai
Italian
Swedish
Japanese
Spanish


Italian
Japanese
Hindi
Spanish
German
Swedish
French
Catalan
Finnish
Bulgarian
English
Thai
Dutch
Chinese

Fluent

Productive

Multilingual

- Dedukti
- GF

Productive

RGL = Resource Grammar Library

morphology and syntax for ~50 languages

```
-- inflection of French adjectives, slightly simplified

mkA : Str -> A = \adj ->
    case adj of {
      _ + "eux"=> <adj, init adj + "se", adj, init adj + "ses"> ;
      _ + "al" => <adj, adj + "e", init adj + "ux", adj + "es"> ;
      _ + "en" => <adj, adj + "ne", adj + "s", adj + "nes"> ;
      _ + "el" => <adj, adj + "le", adj + "s", adj + "les"> ;
      x + "er" => <adj, x + "ère", adj + "s", x + "ères"> ;
      _ + "if" => <adj, init adj + "ve", adj + "s", init adj + "ves"> ;
      _ + "s"  => <adj, adj + "e", adj, adj + "es"> ;
      _ + "e"  => <adj, adj, adj + "s", adj + "s"> ;
      _        => <adj, adj + "e", adj + "s", adj + "es">
      } ;
```

http://www.grammaticalframework.org/lib/doc/synopsis/

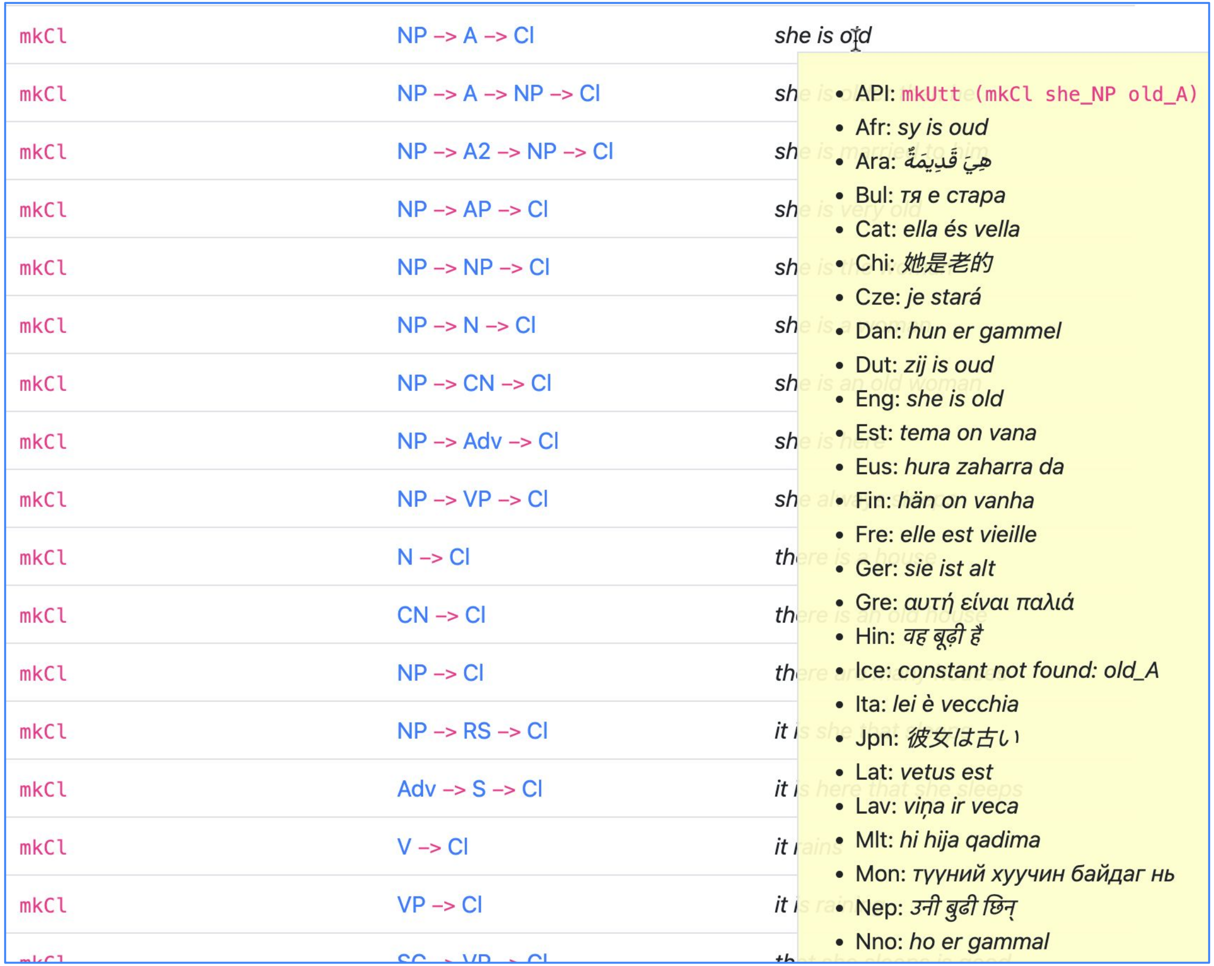
RGL

syntactic combination API

shared by all languages in the library

usable as functor interface + instances

http://www.grammaticalframework.org/lib/doc/synopsis/



mkCl	NP -> A -> Cl	she is old
mkCl	NP -> A -> NP -> Cl	
mkCl	NP -> A2 -> NP -> Cl	
mkCl	NP -> AP -> Cl	
mkCl	NP -> NP -> Cl	
mkCl	NP -> N -> Cl	
mkCl	NP -> CN -> Cl	
mkCl	NP -> Adv -> Cl	
mkCl	NP -> VP -> Cl	
mkCl	N -> Cl	
mkCl	CN -> Cl	
mkCl	NP -> Cl	
mkCl	NP -> RS -> Cl	
mkCl	Adv -> S -> Cl	
mkCl	V -> Cl	
mkCl	VP -> Cl	
mkCl	SC -> VP -> Cl	

- API: `mkUtt (mkCl she_NP old_A)`
- Afr: *sy is oud*
- Ara: *هِيَ قَدِيمَةٌ*
- Bul: *тя е стара*
- Cat: *ella és vella*
- Chi: *她是老的*
- Cze: *je stará*
- Dan: *hun er gammel*
- Dut: *zij is oud*
- Eng: *she is old*
- Est: *tema on vana*
- Eus: *hura zaharra da*
- Fin: *hän on vanha*
- Fre: *elle est vieille*
- Ger: *sie ist alt*
- Gre: *αυτή είναι παλιά*
- Hin: *वह बूढ़ी है*
- Ice: *constant not found: old_A*
- Ita: *lei è vecchia*
- Jpn: *彼女は古い*
- Lat: *vetus est*
- Lav: *viņa ir veca*
- Mlt: *hi hija qadima*
- Mon: *тууний хуучин байдаг нь*
- Nep: *उनी बुढी छिन्*
- Nno: *ho er gammal*

Concrete syntax: functor over the RGL

```
-- abstract syntax code

cat Prop ; Term
fun commutative : Term -> Prop


-- shared functor code

lincat
  Prop = Cl
  Term = NP

lin
  commutative x =
     mkCl x commutative_A
```

```
-- added code for each language

-- Eng
  commutative_A =
    mkA "commutative"

-- Fre
  commutative_A =
    mkA "commutatif"

-- Fin
  commutative_A =
    mkA "kommutatiivinen"
```

Context-free expansions of `'commutative : Term -> Prop'`

```
Prop_1_0  ::= Term_5 "is" "commutative"
Prop_1_0  ::= Term_6 "are" "commutative"
Prop_1_2  ::= "are" Term_6 "commutative"
Prop_1_2  ::= "is" Term_5 "commutative"
Prop_1_3  ::= Term_5 "is" "not" "commutative"
Prop_1_3  ::= Term_6 "are" "not" "commutative"
Prop_1_5  ::= "are" Term_6 "not" "commutative"
Prop_1_5  ::= "is" Term_5 "not" "commutative"
Prop_1_6  ::= Term_5 "isn't" "commutative"
Prop_1_6  ::= Term_6 "aren't" "commutative"
Prop_1_7  ::= Term_5 "isn't" "commutative"
Prop_1_7  ::= Term_6 "aren't" "commutative"
Prop_1_8  ::= "aren't" Term_6 "commutative"
Prop_1_8  ::= "isn't" Term_5 "commutative"
```

Context-free expansions of `'commutative : Term -> Prop'`

```
Prop_1_0  ::= Term_5 "is" "commutative"
Prop_1_0  ::= Term_6 "are" "commutative"
Prop_1_2  ::= "are" Term_6 "commutative"
Prop_1_2  ::= "is" Term_5 "commutative"
Prop_1_3  ::= Term_5 "is" "not" "commutative"
Prop_1_3  ::= Term_6 "are" "not" "commutative"
Prop_1_5  ::= "are" Term_6 "not" "commutative"
Prop_1_5  ::= "is" Term_5 "not" "commutative"
Prop_1_6  ::= Term_5 "isn't" "commutative"
Prop_1_6  ::= Term_6 "aren't" "commutative"
Prop_1_7  ::= Term_5 "isn't" "commutative"
Prop_1_7  ::= Term_6 "aren't" "commutative"
Prop_1_8  ::= "aren't" Term_6 "commutative"
Prop_1_8  ::= "isn't" Term_5 "commutative"
```

```
Prop_1_0  ::= Term_1 "est" "commutatif"
Prop_1_0  ::= Term_2 "n'est" "commutatif"
Prop_1_0  ::= Term_3 "sont" "commutatifs"
Prop_1_0  ::= Term_4 "ne" "sont" "commutatifs"
Prop_1_1  ::= Term_1 "soit" "commutatif"
Prop_1_1  ::= Term_2 "ne" "soit" "commutatif"
Prop_1_1  ::= Term_3 "soient" "commutatifs"
Prop_1_1  ::= Term_4 "ne" "soient" "commutatifs"
Prop_1_10 ::= "n'est" Term_1 "commutatif"
Prop_1_10 ::= "n'est" Term_2 "commutatif"
Prop_1_10 ::= "ne" "sont" Term_3 "commutatifs"
Prop_1_10 ::= "ne" "sont" Term_4 "commutatifs"
Prop_1_11 ::= "ne" "soient" Term_3 "commutatifs"
Prop_1_11 ::= "ne" "soient" Term_4 "commutatifs"
Prop_1_11 ::= "ne" "soit" Term_1 "commutatif"
Prop_1_11 ::= "ne" "soit" Term_2 "commutatif"
Prop_1_2  ::= Term_1 "n'est" "pas" "commutatif"
Prop_1_2  ::= Term_2 "n'est" "pas" "commutatif"
Prop_1_2  ::= Term_3 "ne" "sont" "pas" "commutatifs"
Prop_1_2  ::= Term_4 "ne" "sont" "pas" "commutatifs"
Prop_1_3  ::= Term_1 "ne" "soit" "pas" "commutatif"
Prop_1_3  ::= Term_2 "ne" "soit" "pas" "commutatif"
Prop_1_3  ::= Term_3 "ne" "soient" "pas" "commutatifs"
Prop_1_3  ::= Term_4 "ne" "soient" "pas" "commutatifs"
Prop_1_4  ::= Term_1 "n'est" "commutatif"
Prop_1_4  ::= Term_2 "n'est" "commutatif"
Prop_1_4  ::= Term_3 "ne" "sont" "commutatifs"
Prop_1_4  ::= Term_4 "ne" "sont" "commutatifs"
Prop_1_5  ::= Term_1 "ne" "soit" "commutatif"
Prop_1_5  ::= Term_2 "ne" "soit" "commutatif"
Prop_1_5  ::= Term_3 "ne" "soient" "commutatifs"
Prop_1_5  ::= Term_4 "ne" "soient" "commutatifs"
Prop_1_6  ::= "est" Term_1 "commutatif"
Prop_1_6  ::= "n'est" Term_2 "commutatif"
Prop_1_6  ::= "ne" "sont" Term_4 "commutatifs"
Prop_1_6  ::= "sont" Term_3 "commutatifs"
Prop_1_7  ::= "ne" "soient" Term_4 "commutatifs"
Prop_1_7  ::= "ne" "soit" Term_2 "commutatif"
Prop_1_7  ::= "soient" Term_3 "commutatifs"
Prop_1_7  ::= "soit" Term_1 "commutatif"
Prop_1_8  ::= "n'est" "pas" Term_1 "commutatif"
Prop_1_8  ::= "n'est" "pas" Term_2 "commutatif"
Prop_1_8  ::= "ne" "sont" "pas" Term_3 "commutatifs"
Prop_1_8  ::= "ne" "sont" "pas" Term_4 "commutatifs"
Prop_1_9  ::= "ne" "soient" "pas" Term_3 "commutatifs"
Prop_1_9  ::= "ne" "soient" "pas" Term_4 "commutatifs"
Prop_1_9  ::= "ne" "soit" "pas" Term_1 "commutatif"
Prop_1_9  ::= "ne" "soit" "pas" Term_2 "commutatif"
```

From Dedukti to GF

```
-- Dedukti.bnf

MJmts. Module ::= [Jmt] ;

terminator Jmt "" ;

comment "(;" ";)" ;
comment "#" ; ----

JStatic.  Jmt ::= QIdent ":" Exp "." ;
JDef.     Jmt ::= "def" QIdent MTyp MExp "." ;
JInj.     Jmt ::= "inj" QIdent MTyp MExp "." ;
JThm.     Jmt ::= "thm" QIdent MTyp MExp "." ;
JRules.   Jmt ::= [Rule] "." ;

RRule.  Rule ::= "[" [Pattbind] "]" Patt "-->" Exp ;
separator nonempty Rule "" ;

separator Pattbind "," ;

MTNone. MTyp ::= ;
MTExp.  MTyp ::= ":" Exp ;

MENone. MExp ::= ;
MEExp.  MExp ::= ":=" Exp ;

EIdent.  Exp9 ::= QIdent ;
EApp.    Exp5 ::= Exp5 Exp6 ;
EAbs.    Exp2 ::= Bind "=>" Exp2 ;
EFun.    Exp1 ::= Hypo "->" Exp1 ;

coercions Exp 9 ;

-- plus some rules for Hypo and Bind

token QIdent (letter | digit | '_' | '!' | '?' | '\'')+
('.' (letter | digit | '_' | '!' | '?' | '\'')+)? ;
```

```
-- MathCore.gf

abstract MathCore =
  Terms, UserConstants
  ** {
cat
  Jmt ;
  Exp ;
  Exps ;
  Prop ;
  Kind ;
  Hypo ;
  [Hypo] ;
  Proof ;
  Label ;
  -- plus more categories
fun
  ThmJmt : Label -> [Hypo] -> Prop -> Proof -> Jmt ;
  AxiomJmt : Label -> [Hypo] -> Prop -> Jmt ;
  DefPropJmt : Label -> [Hypo] -> Prop -> Prop -> Jmt ;
  DefKindJmt : Label -> [Hypo] -> Kind -> Kind -> Jmt ;
  DefExpJmt  : Label -> [Hypo] -> Exp -> Kind -> Exp -> Jmt ;
  AxiomPropJmt : Label -> [Hypo] -> Prop -> Jmt ;
  AxiomKindJmt : Label -> [Hypo] -> Kind -> Jmt ;
  AxiomExpJmt  : Label -> [Hypo] -> Exp -> Kind -> Jmt ;

  AppExp : Exp -> Exps -> Exp ;
  AbsExp : [Ident] -> Exp -> Exp ;
  TermExp : Term -> Exp ;
  KindExp : Kind -> Exp ;
  TypedExp : Exp -> Kind -> Exp ;

  AndProp : [Prop] -> Prop ;
  OrProp : [Prop] -> Prop ;
  IfProp : Prop -> Prop -> Prop ;
  IffProp : Prop -> Prop -> Prop ;
  NotProp : Prop -> Prop ;
  -- plus many more functions
```

```
-- Dedukti.bnf

JDef. Jmt ::= "def" QIdent MTyp MExp "." ;
```

```
-- MathCore.gf

DefPropJmt :
  Label -> [Hypo] -> Prop -> Prop -> Jmt ;

DefKindJmt :
  Label -> [Hypo] -> Kind -> Kind -> Jmt ;

DefExpJmt  :
  Label -> [Hypo] -> Exp -> Kind -> Exp -> Jmt ;

ThmJmt :
  Label -> [Hypo] -> Prop -> Proof -> Jmt ;
```

From formal Exp to linguistic categories

Dedukti Exp	GF category	linearization	linguistic category
`union A B`	`Exp`	*the union of A and B*	noun phrase
`Nat`	`Kind`	*natural number*	common noun
`divisible 9 3`	`Prop`	*9 is divisible by 3*	sentence
`oddS 0 evenZ`	`Proof`	*0 is even. Therefore 1 is odd.*	text

```
abstract BaseConstants = {

-- GF cat         usage                                example
--------------------------------------------------------------------
  Noun ;         -- Kind                             -- set
  Fam ;          -- Kind -> Kind                     -- list of integers
  Adj ;          -- Exp -> Prop                      -- even
  Verb ;         -- Exp -> Exp                       -- converge
  Reladj ;       -- Exp -> Exp -> Prop               -- divisible by
  Relverb ;      -- Exp -> Exp -> Prop               -- divide
  Relnoun ;      -- Exp -> Exp -> Prop               -- root of
  Name ;         -- Exp                              -- contradiction
  Fun ;          -- [Exp] -> Exp                     -- radius of
  Label ;        -- Exp                              -- theorem 1

  Set ;          -- Kind | Term                      -- integer, Z
  Const ;        -- Exp | Term                       -- the empty set, Ø
  Oper ;         -- Exp -> Exp -> Exp  | Term        -- the sum of, +
  Compar ;       -- Exp -> Exp -> Prop | Formula     -- greater than, >
  Comparnoun ;   -- Exp -> Exp -> Prop | Formula     -- a subset of, \sub
```

Symbol tables Dedukti ←→ GF

```
(; BaseConstants.dk ;)

(; constants defined in a lexicon ;)

Nat : Set.
Int : Set.
Rat : Set.
Real : Set.

Eq : Elem Real -> Elem Real -> Prop.
Lt : Elem Real -> Elem Real -> Prop.
Gt : Elem Real -> Elem Real -> Prop.

plus : (x : Elem Real) -> (y : Elem Real) -> Elem Real.
minus : Elem Real -> Elem Real -> Elem Real.
times : Elem Real -> Elem Real -> Elem Real.

even : Elem Int -> Prop.
def odd : Elem Int -> Prop := n => not (even n).
```

```
# base_constant_data.dkgf

# for translating between Dedukti and GF abstract syntax

Nat BASE Set natural_Set
Int BASE Set integer_Set
Rat BASE Set rational_Set
Real BASE Set real_Set

Eq BASE Compar Eq_Compar
Lt BASE Compar Lt_Compar
Gt BASE Compar Gt_Compar

plus BASE Oper plus_Oper
minus BASE Oper minus_Oper
times BASE Oper times_Oper

even BASE Adj even_Adj
odd BASE Adj odd_Adj

# for generating GF linearization rules

#LIN Eng natural_Set = mkSet "N" "natural" number_N
#LIN Fre natural_Set = mkSet L.natural_Set "naturel" nombre_N
#LIN Swe natural_Set = mkSet L.natural_Set "naturlig" tal_N

#LIN Eng even_Adj = mkAdj "even"
#LIN Fre even_Adj = mkAdj "pair"
#LIN Swe even_Adj = mkAdj "jämn"

# for converting identifiers from third-party projects

le ALIAS matita Leq
```

Lexicon extraction

```
def sphenic : Nat -> Prop
  := ...
(; GF: sphenic number ;)
```

lexical rule extraction

```
# from Wikidata

{"Q638185": {
  "pl": "Liczby sfeniczne",
  "de": "sphenische Zahl",
  "en": "sphenic number",
  "es": "número esfénico",
  "fr": "nombre sphénique",
  "zh": "楔形数",
  "sv": "sfeniskt tal",
  "ta": "ஸ்ஃபீனிக் எண்",
  }
}
```

```
sphenic Adj spenic_Adj

#LIN Eng sphenic_Adj = mkAdj "sphenic"
#LIN Fre sphenic_Adj = mkAdj "sphénique"
#LIN Swe sphenic_Adj = mkAdj "sfenisk"
```

AR, Building Grammar Libraries for Mathematics and Avoiding Manual Work.. Presentation at Hausdorff Center for Mathematics, 2024, https://www.youtube.com/watch?v=EQ-k_JQ7fDM&t=5s

```
def sphenic : (p : Elem Nat) -> Prop := p =>
  exists Nat (k => exists Nat (m => exists Nat (n =>
    and (and (and (prime k) (prime m)) (prime n))
      (and (and (Lt k m) (Lt m n))
        (Eq (times (times k m) n) p))))).
```

```
def sphenic : (p : Elem Nat) -> Prop := p =>
  exists Nat (k => exists Nat (m => exists Nat (n =>
    and (and (and (prime k) (prime m)) (prime n))
      (and (and (Lt k m) (Lt m n))
        (Eq (times (times k m) n) p))))).
```

Definition. Let p be a natural number. Then p is sphenic, if there exist natural numbers k, m and n, such that k, m and n are prime, $k < m < n$ and $kmn = p$.

```
def sphenic : (p : Elem Nat) -> Prop := p =>
  exists Nat (k => exists Nat (m => exists Nat (n =>
    and (and (and (prime k) (prime m)) (prime n))
      (and (and (Lt k m) (Lt m n))
        (Eq (times (times k m) n) p))))).
```

Definition. Let p be a natural number. Then p is sphenic, if there exist natural numbers k, m and n, such that k, m and n are prime, $k < m < n$ and $kmn = p$.

A sphenic number is a product *pqr* where *p*, *q*, and *r* are three distinct prime numbers.

https://en.wikipedia.org/wiki/Sphenic_number

Extraction functions for syntax (using the RGL)

```
AdjCN : AP -> CN -> CN ;               -- continuous function
CompoundN : N -> N -> N ;              -- function space
IntCompoundCN : Int -> CN -> CN ;      -- 13-cube
NameCompoundCN : PN -> CN -> CN ;      -- Lie group
NounIntCN : CN -> Int -> CN ;          -- Grinberg graph 42
NounPrepCN : CN -> Adv -> CN ;         -- ring of sets
NounGenCN : CN -> NP -> CN ;           -- bishop's graph

PositA : A -> AP ;                     -- uniform
AdAP : AdA -> AP -> AP ;               -- almost uniform
AAdAP : A -> AP -> AP ;                -- algebraically closed
PastPartAP : V -> AP ;                 -- connected

PrepNP : Prep -> NP -> Adv ;           -- (integration) by parts

-- plus some more functions, 21 functions in total
```

Terminology extraction from Wikidata with UD and RGL

language	labels covered		successful parses	
Eng	5188	96%	3872	74%
Fin	834	15%	328	39%
Fre	3230	60%	2199	68%
Ger	2956	54%	2609	88%
Ita	2019	37%	1390	68%
Por	2858	53%	1717	60%
Spa	2322	43%	1633	70%
Swe	1345	24%	826	61%

Adding a new language: ~2 minutes of CPU time

Fluent

- NLG, almost compositional functions
- GF RGL

Productive

- GF RGL
- lexicon and grammar extraction

Multilingual

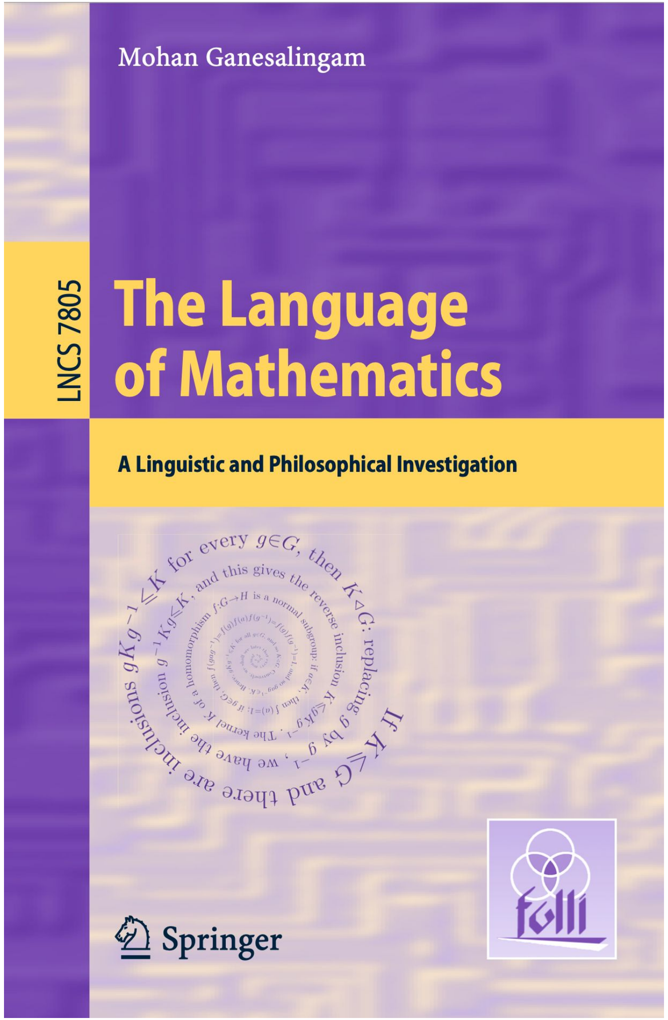
- Dedukti
- GF

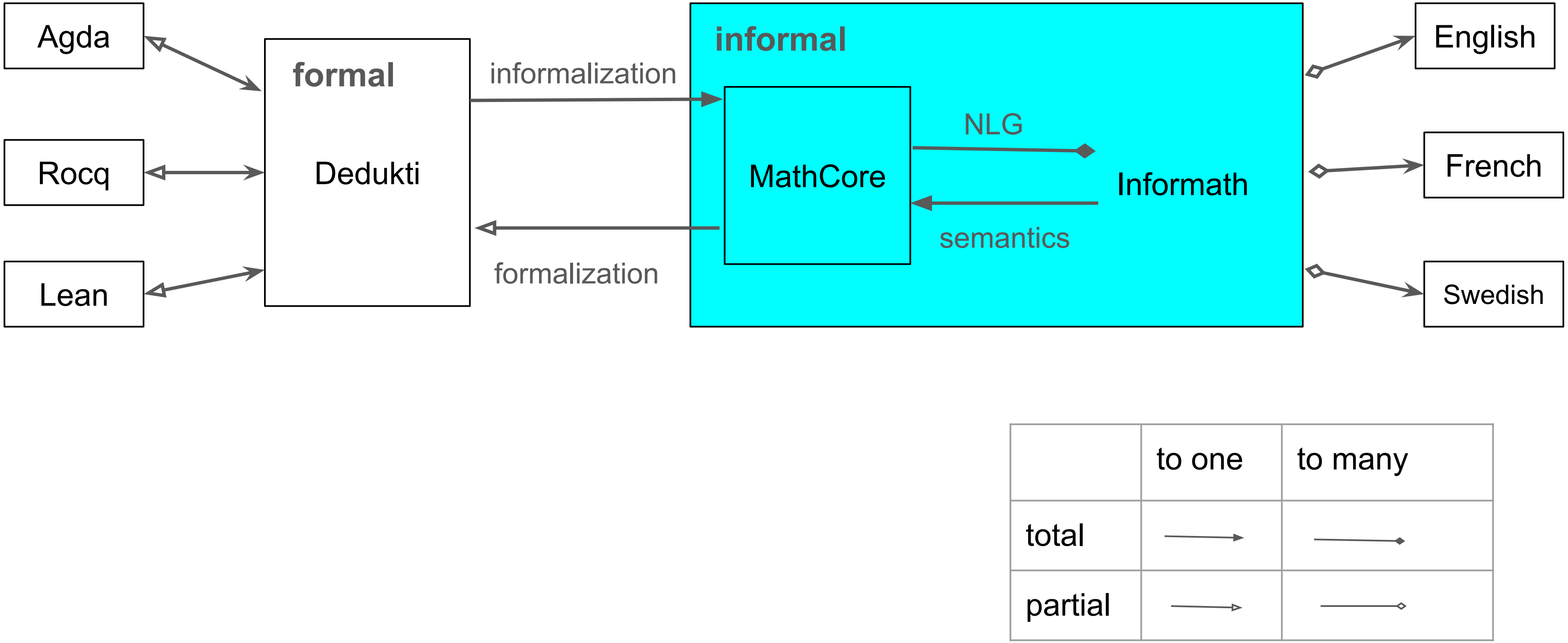
Fluent

has

natural language content that ~~lacks~~ the inherent diversity and flexibility in expression: they are ~~rigid and not~~ natural-language-like.

natural language content that ~~lacks~~ *has* the inherent diversity and flexibility in expression: they are ~~rigid and not~~ natural-language-like.




Agda
Rocq
Lean
formal
Dedukti
informalization
formalization
informal
MathCore
NLG
semantics
Informath
English
French
Swedish
to one
to many
total
partial

```
prop110 : (a : Elem Int) -> (c : Elem Int) ->
 Proof (and (odd a) (odd c)) -> Proof (forall
 Int (b => even (plus (times a b) (times b c)))).
```

Prop110. For all instances a and c of integers, if we can prove that a is odd and c is odd, then we can prove that for all integers b, the sum of the product of a and b and the product of b and c is even.

```
abstract Informath = MathCore ** {



fun
-- use symbolic expressions if possible
  FormulaProp : Formula -> Prop ;
  SetTerm : Set -> Term ;
  ConstTerm : Const -> Term ;
  ComparEqsign : Compar -> Eqsign ;




-- aggregation

  AndAdj : [Adj] -> Adj ;
  OrAdj : [Adj] -> Adj ;

  AndExp : [Exp] -> Exp ;
  OrExp : [Exp] -> Exp ;



-- post-quantification

  PostQuantProp : Prop -> Exp -> Prop ;


}
```

```
prop110 : (a : Elem Int) -> (c : Elem Int) ->
 Proof (and (odd a) (odd c)) -> Proof (forall
 Int (b => even (plus (times a b) (times b c)))).
```

Prop110. For all instances a and c of integers, if we can prove that a is odd and c is odd, then we can prove that for all integers b, the sum of the product of a and b and the product of b and c is even.

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then for all integers b, $ab + bc$ is even.

Prop110. Let $a, c \in Z$. Assume that both a and c are odd. Then $ab + bc$ is even for all integers b.

```
abstract Informath = MathCore ** {

 AndAdj : [Adj] -> Adj ;

 NoIdentsKindExp : [Ident] -> Kind -> Exp ;

 NoKindExp : Kind -> Exp ;

}
```

In situ quantification

$$(Q\ x : A)B(x) \Rightarrow B(Q\ A)$$

if x occurs exactly once in B:

The variable can optionally be omitted.

```
prop50 : Proof (forall Nat
  (n => not (and (even n) (odd n)))).
```

Prop50. We can prove that for all natural numbers n, it is not the case that n is even and n is odd.

Prop50. For all natural numbers n, n is not both even and odd.

Prop50. No natural number n is both even and odd.

Prop50. No natural number is both even and odd.

Scoring and ranking alternative phrases

```
data Scores = Scores {
  tree_length :: Int,
  tree_depth :: Int,
  characters :: Int,
  tokens :: Int,
  subsequent_dollars :: Int,
  initial_dollars :: Int,
  extra_parses :: Int
  }
```

These all are penalties
- minimize some linear combination of them
- users can give weights to each score (default = 1)

```
$ ./RunInformath -ranking -variations -test-ambiguity test/prop110.dk
## showing a sample from 87 results, first and last included

Prop110. Let $a , c \in Z$. Then if $a$ and $c$ are odd, then $a b + b c$ is even for
every integer $b$.

%% (Scores {tree_length = 55, tree_depth = 10, characters = 104, tokens = 40,
subsequent_dollars = 0, initial_dollars = 0, extra_parses = 1},210)


Prop110. Let $a$ and $c$ be integers. Assume that $a$ and $c$ are odd. Then for all
integers $b$, $a b + b c$ is even.

%% (Scores {tree_length = 55, tree_depth = 11, characters = 118, tokens = 43,
subsequent_dollars = 1, initial_dollars = 0, extra_parses = 0},228)


Prop110. Let $a$ and $c$ be instances of integers. Assume that we can prove that $a$
is odd and $c$ is odd. Then we can prove that for all integers $b$, the sum of the
product of $a$ and $b$ and the product of $b$ and $c$ is even.

%% (Scores {tree_length = 71, tree_depth = 14, characters = 230, tokens = 72,
subsequent_dollars = 0, initial_dollars = 0, extra_parses = 2},389)
```

Fluent

- NLG transformations
- GF RGL

Productive

- GF RGL
- lexicon and grammar extraction

Multilingual

- Dedukti
- GF .

Case studies

Wiedijk's "100 theorems" (a sample)

```
Thm01 : Proof (not (rational (sqrt 2))).

Thm20 : (p : Elem Nat) -> Proof (prime p) -> Proof (congruent p 1 4)
  -> Proof (exists Nat (x => exists Nat (y => Eq p (plus (square x) (square y))))).

Thm51wilson : (n : Elem Nat) ->
  Proof (iff (prime n) (congruent (factorial (minus n 1)) (neg 1)  n)).

Thm78 : (u : Elem Vector) -> (v : Elem Vector) ->
  Proof (if (orthogonal u v) (Eq (dotProduct u v) (nd 0))).

Thm91 : (u : Elem Vector) -> (v : Elem Vector) ->
  Proof (Leq (norm (vectorPlus u v)) (plus (norm u) (norm v))).
```

```
$ make lang=Eng top100

$ make lang=Fre top100
```

Towards math olympiad problems (only started)

Full of expressions with three dots - typically for sums

- first step: extract the summation term
- informalization of Sigma expressions produces ambiguous sequences

Theorem.

$$\sum_{n=1}^{9} \frac{1}{n} > 2.$$

Theorem.

$$\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{9} > 2.$$

Naproche-ZF (recently started)

A CNL designed to serve as input in formalization https://github.com/adelon/naproche-zf

1. extend Informath to parse Naproche-ZF
2. obtain Dedukti code and thereby Agda, Lean, Rocq
3. obtain paraphrases and thereby synthetic training data
4. increase the parsing that targets Naproche-ZF
5. translate to other Informath languages

Issues:

- undeclared variables and their types
- getting proof objects from proof texts

```
$ make lang=Eng naproche

$ make lang=Fre naproche
```

Generating synthetic data

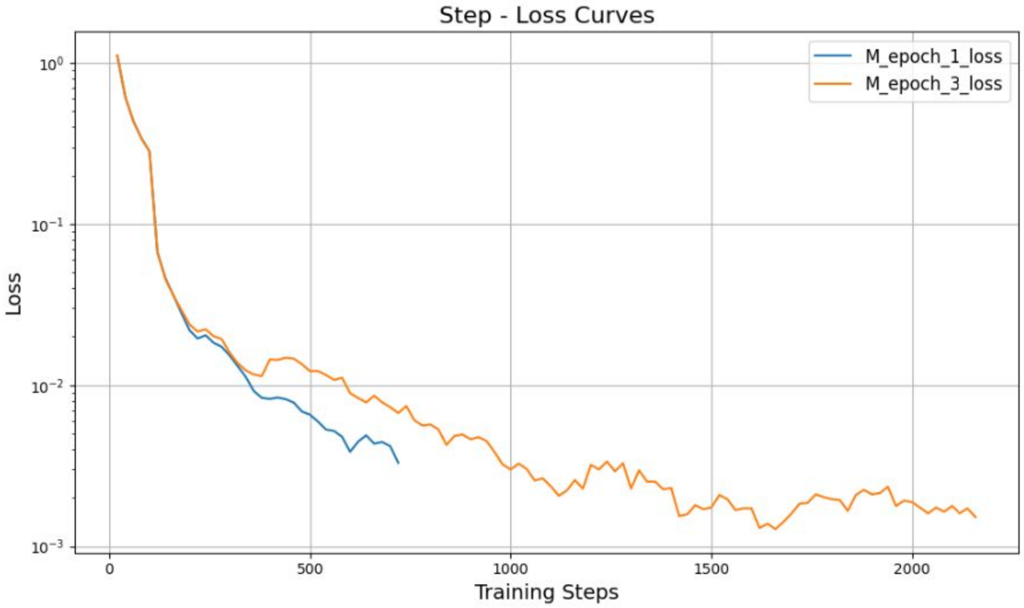


Figure 4.5: Training losses of fine-tuned models at different epochs.

Table 4.5: Model performance at different training epochs

Model	BLEU-4	ROUGE-1/2/L	Syntax Err.%	Score
Baseline	32.90	54.17 / 21.99 / 42.76	98.43	23.96
M_epoch_1	76.16	89.03 / 74.94 / 83.22	7.93	83.60
M_epoch_3	77.78	89.86 / 76.63 / 84.37	20.48	80.14

Fine-tuning an LLM:

- Qwen2.5-7B-instruct

Trained with ~1000 synthetic pairs of (dedukti,agda,coq,lean) - (English, French, Swedish) with

- arithmetic
- naive set theory
- concepts for 27 of the "100 theorems"

Tested with 57 natural native-speaker expressions of those theorems (by Nick Smallbone)

Pei Huang, *Autoformalization for Agda via Fine-tuning Large Language Models,* MSc thesis at Chalmers, 2025

Translate the following latex InformathEng translation problems (each is on one line) to deducti.
Do it analogously to the following extraction examples. Put the output all into one piece of text, preceded always by input (as in the examples).
BEGIN EXAMPLES
{"InformathEng": "thm58 . let $ A $ be a set . let $ n \\in N $. then $ | A | = n $, only if for all natural numbers $ k $, if $ k \\leq n $, then $ | \\binom{ A }{ k } | = \\binom{ n }{ k } $.", "dedukti": "thm58 : (A : Set) -> (n : Elem Nat) -> Proof (Eq (cardinality A) n) -> (k : Elem Nat) -> Proof (Leq k n) -> Proof (Eq (cardinality (combinationsFromSet A k)) (binomial n k)) ."}
{— …
END EXAMPLES
BEGIN TRANSLATION PROBLEMS
{"InformathEng": "Thm01f . there exists no natural number $ p $, such that there exists a natural number $ q $, such that the square of $ p $ is equal to the product of $ 2 $ and the square of $ q $.", "dedukti":
{"InformathEng": "thm52 . let $ A $ be a set . then we can prove that $ A $ is finite , only if we can prove that the cardinality of the power set of $ A $ is equal to the exponentiation of $ 2 $ and the cardinality of $ A $.", "dedukti":
— …
END TRANSLATION PROBLEMS

Sure, here are the translations of the given InformathEng statements to Dedukti:

```
input: {"InformathEng": "Thm01f . there exists no natural number $ p $ , such that there exists a natural number $ q $ , such that the square of $ p $ is equal to the product of $ 2 $ and the square of $ q $ .", "dedukti": "Thm01f : Proof ( not ( exists Nat ( p => exists Nat ( q => Eq ( square p ) ( times 2 ( square q ) ) ) ) ) ) ."}
— …
```

https://chat.mistral.ai/chat/60f44ab2-d86f-4ec2-b382-d94fd471eef0 by Josef Urban, 13 July 2025

Conclusion

Symbolic informalization can be

- natural and fluent
  - by extending CNL towards the full language of mathematics
- feasible to develop
  - by Dedukti, GF, and rule extraction
- shared by different formal and informal languages
  - by Dedukti and GF interlinguas
- inverted to autoformalization
  - natively, by reversilibility of GF
  - as backup, by fine-tuned LLM + feedback informalization

Symbolic informalization is

- based on well-understood compiler-like techniques
- potentially 100% reliable
- fast and energy-efficient
- a natural extension of formal proof techniques
- an integral part of reliable AI systems for mathematics
    - and other fields where formalization is possible

Some future work

Improve the informalization of *proofs*

- combine proof terms with scripts to identify crucial steps ?
- refactor proofs by creating lemmas !

```
-- the current syntax of proofs - minimal but complete

AbsProof : ListHypo -> Proof -> Proof ;
AppProof : ProofExp -> ListProof -> Proof ;

AppProofExp : ProofExp -> Exps -> ProofExp ;
LabelProofExp : Label -> ProofExp ;
```

Refine the evaluation criteria for autoformalization

- BLEU score and edit distance are too superficial
- logical equivalence is too liberal
- definitional equality is also too liberal

Create APIs to connect with proof systems

- use Informath as a library or a plugin component
- to enable natural language interaction and documentation
- GF is more powerful than mixfix and similar syntax extensions

Natural language is the ultimate syntactic sugar!

Exploit multilinguality

- to generate Wikipedia articles
- to translate Math Olympiad problems

thanks : Phrase

thanks
kiitos
merci
Danke
tack

https://github.com/aarneranta/informath