The Informath Deployment Stack

CHAIR Workshop, Gothenburg 21 May 2026
MALINCA Workshop, Nancy 23 June 2026

Aarne Ranta

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

aarne.ranta@cse.gu.se

Informath

NextReason

MALINCA

European Research Council
Established by the European Commission

Informath

The ultimate syntactic sugar: ordinary mathematical language

```
Thm04 : (u : Elem Vector) -> (v : Elem Vector) ->
  Proof (orthogonal u v) ->
    Proof (Eq Real (length (vectorPlus u v))
      (sqrt (plus (square (length u)) (square (length v))))).
```

Thm04. For all vectors u and v, if $u \perp v$, then

$$\|u + v\| = \sqrt{\|u\|^2 + \|v\|^2}.$$

Variations in ordinary mathematical language

```
Thm04 : (u : Elem Vector) -> (v : Elem Vector) ->
  Proof (orthogonal u v) ->
    Proof (Eq Real (length (vectorPlus u v))
      (sqrt (plus (square (length u)) (square (length v))))).
```

```
$ RunInformath -variations tmp/t4.dk | wc -l
    2542
```

Thm04. For all vectors u and v, if $u \perp v$, then

$$\|u+v\| = \sqrt{\|u\|^2 + \|v\|^2}.$$

Thm04. Let u and v be vectors. Assume that u is orthogonal to v. Then the length of the sum of u and v is equal to the square root of the sum of the square of the length of u and the square of the length of v.

Cf. Ganesalingam 2013

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)

Uses of informalization

Documenting formal mathematics

- Useful for documenting formal mathematics (Coscoy et al. 1994)

Formalizing informal mathematics

- via parsing with reversible grammars (CNL approach)
- via generating synthetic data for neural models (Wang et al. 2018)

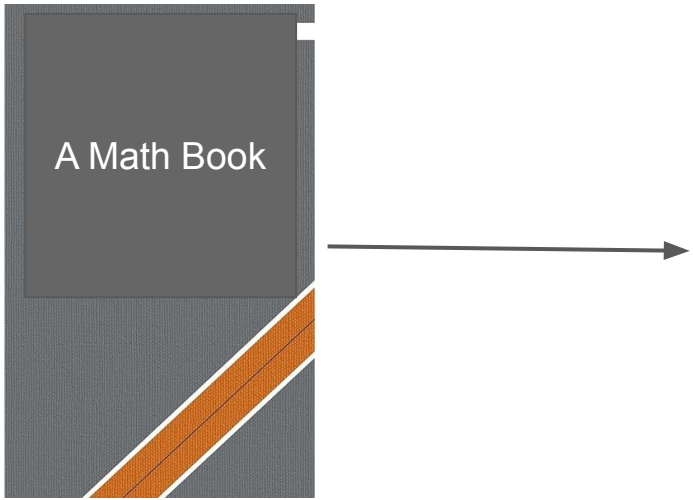
Uses of informalization

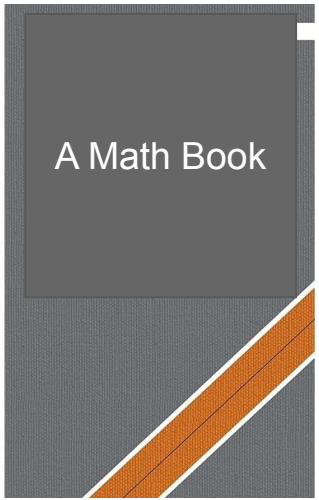
Documenting formal mathematics

- Useful for documenting formal mathematics (Coscoy et al. 1994)
- Indispensable for explaining AI generated code (Urban et al. 2026)

Formalizing informal mathematics

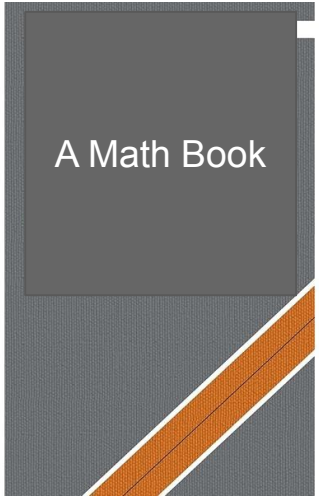
- via parsing with reversible grammars (CNL approach)
- via generating synthetic data for neural models (Wang et al. 2018)


A Math Book



LLM autoformalization

```
t0 a (t0 b c)) (t0 (t0 a b) c)) => match_Aop a1
a2 (plus Type2 (nd 1)) (__ : Aop a1 a2 => univ
Type2) (u0 : (__ : a1 -> __1 : a1 -> a1) => u1
: (a : a1 -> Eq (u0 a2 a) a) => u2 : (a : a1 ->
Eq (u0 a a2) a) => u3 : (a : a1 -> b : a1 -> c
: a1 -> Eq (u0 a (u0 b c)) (u0 (u0 a b) c)) =>
prod (plus Type1 (nd 1)) Type1 (univ Type1) (P
: Type1 => prod Type1 Type1 (prod Prop Type1
(Eq (R0 (prod Set Set a1 (__ : a1 => prod Set
Set a1 (__1 : a1 => a1))) t0) u0) (e0 : Eq (R0
(prod Set Set a1 (__ : a1 => prod Set Set a1
(__1 : a1 => a1))) t0) u0 => prod Prop Type1
(Eq (R1 (prod Set Set a1 (__ : a1 => prod Set
Set a1 (__1 : a1 => a1))) t0 (x_19 : (__ : a1
-> __1 : a1 -> a1) => _x_20 : Eq t0 x_19 => (x0
: (__ : a1 -> __1 : a1 -> a1) => p0 : Eq t0 x0
=> prod Set Prop a1 (a : a1 => Eq (x0 a2 a) a))
x_19 _x_20) t1 u0 e0) u1) (e1 : Eq (R1 (prod
Set Set a1 (__ : a1 => prod Set Set a1 (__1 :
a1 => a1))) t0 (x_19 : (__ : a1 -> __1 : a1 ->
a1) => _x_20 : Eq t0 x_19 => (x0 : (__ : a1 ->
__1 : a1 -> a1) => p0 : Eq t0 x0 => prod Set
Prop a1 (a : a1 => Eq (x0 a2 a) a)) x_19 _x_20)
t1 u0 e0) u1 => prod Prop Type1 (Eq (R2 (prod
Set Set a1 (__ : a1 => prod Set Set a1 (__1 :
a1 => a1))) t0 (x0 : (__ : a1 -> __1 : a1 ->
a1) => __ : Eq t0 x0 => (x01 : (__1 : a1 -> __2
: a1 -> a1) => p0 : Eq t0 x01 => prod Set Prop
a1 (a : a1 => Eq (x01 a2 a) a)) x0 __) t1 (x0 :
(__ : a1 -> __1 : a1 -> a1) => p0 : Eq t0 x0 =>
x1 : (x01 : (__ : a1 -> __1 : a1 -> a1) => p01
: Eq t0 x01 => prod Set Prop a1 (a : a1 => Eq
(x01 a2 a) a)) x0 p0 => __ : Eq (R1 (prod Set
Set a1 (__ : a1 => prod Set Set a1 (__1 : a1 =>
```



LLM autoformalization

```
t0 a (t0 b c)) (t0 (t0 a b) c)) => match_Aop a1
a2 (plus Type2 (nd 1)) (__ : Aop a1 a2 => univ
Type2) (u0 : (__ : a1 -> __1 : a1 -> a1) => u1
: (a : a1 -> Eq (u0 a2 a) a) => u2 : (a : a1 ->
Eq (u0 a a2) a) => u3 : (a : a1 -> b : a1 -> c
: a1 -> Eq (u0 a (u0 b c)) (u0 (u0 a b) c)) =>
prod (plus Type1 (nd 1)) Type1 (univ Type1) (P
: Type1 => prod Type1 Type1 (prod Prop Type1
(Eq (R0 (prod Set Set a1 (__ : a1 => prod Set
Set a1 (__1 : a1 => a1))) t0) u0) (e0 : Eq (R0
(prod Set Set a1 (__ : a1 => prod Set Set a1
(__1 : a1 => a1))) t0) u0 => prod Prop Type1
(Eq (R1 (prod Set Set a1 (__ : a1 => prod Set
Set a1 (__1 : a1 => a1))) t0 (x_19 : (__ : a1
-> __1 : a1 -> a1) => _x_20 : Eq t0 x_19 => (x0
: (__ : a1 -> __1 : a1 -> a1) => p0 : Eq t0 x0
=> prod Set Prop a1 (a : a1 => Eq (x0 a2 a) a))
x_19 _x_20) t1 u0 e0) u1) (e1 : Eq (R1 (prod
Set Set a1 (__ : a1 => prod Set Set a1 (__1 :
a1 => a1))) t0 (x_19 : (__ : a1 -> __1 : a1 ->
a1) => _x_20 : Eq t0 x_19 => (x0 : (__ : a1 ->
__1 : a1 -> a1) => p0 : Eq t0 x0 => prod Set
Prop a1 (a : a1 => Eq (x0 a2 a) a)) x_19 _x_20)
t1 u0 e0) u1 => prod Prop Type1 (Eq (R2 (prod
Set Set a1 (__ : a1 => prod Set Set a1 (__1 :
a1 => a1))) t0 (x0 : (__ : a1 -> __1 : a1 ->
a1) => __ : Eq t0 x0 => (x01 : (__1 : a1 -> __2
: a1 -> a1) => p0 : Eq t0 x01 => prod Set Prop
a1 (a : a1 => Eq (x01 a2 a) a)) x0 __) t1 (x0 :
(__ : a1 -> __1 : a1 -> a1) => p0 : Eq t0 x0 =>
x1 : (x01 : (__ : a1 -> __1 : a1 -> a1) => p01
: Eq t0 x01 => prod Set Prop a1 (a : a1 => Eq
(x01 a2 a) a)) x0 p0 => __ : Eq (R1 (prod Set
Set a1 (__ : a1 => prod Set Set a1 (__1 : a1 =>
```

symbolic informalization

Thm03. $\mathbb{Q}$ is denumerable.

Thm03a.

$$|\mathbb{N}| = |\mathbb{Q}|.$$

Thm04. Let u and v be vectors. Then $u \perp v$ implies $\|u+v\| = \sqrt{\|u\|^2 + \|v\|^2}$.

Thm07. Let $p, q \in \mathbb{N}$. Then if p and q are prime, then

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Thm09. Let c be a circle. Let r be a real number. Assume that $r = radius(c)$. Then the area of c is equal to $\pi \times r^2$.

Thm10FermatLittle. Let p be a natural number. Assume that p is prime. Let a be an integer. Then there exists an integer q, such that $a^p - a = p \times q$.

Thm11. $p \geq n$ and p is prime for a natural number p for all natural numbers n.

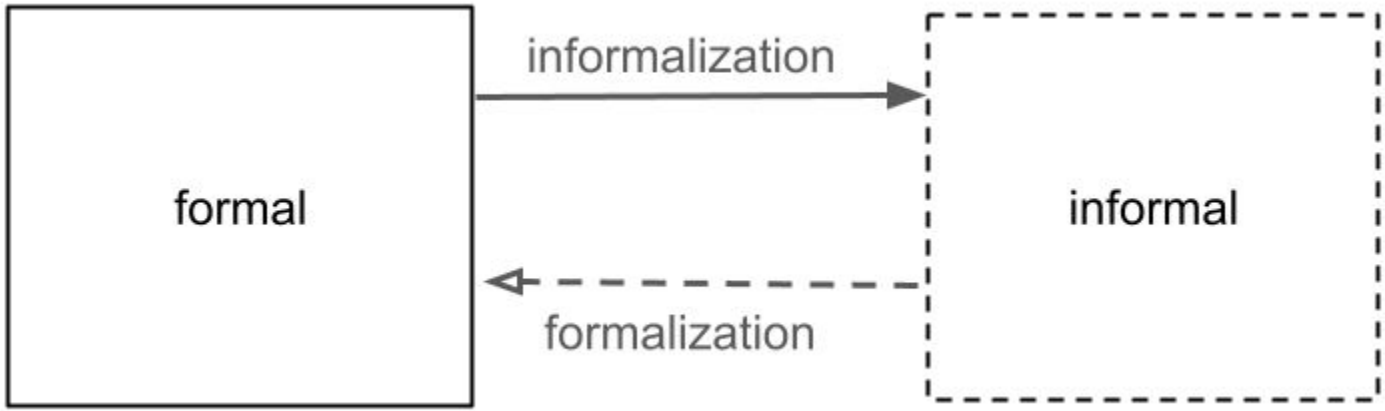
Thm19. $n = a^2 + b^2 + c^2 + d^2$ for some natural numbers a, b, c and d for all natural numbers n.

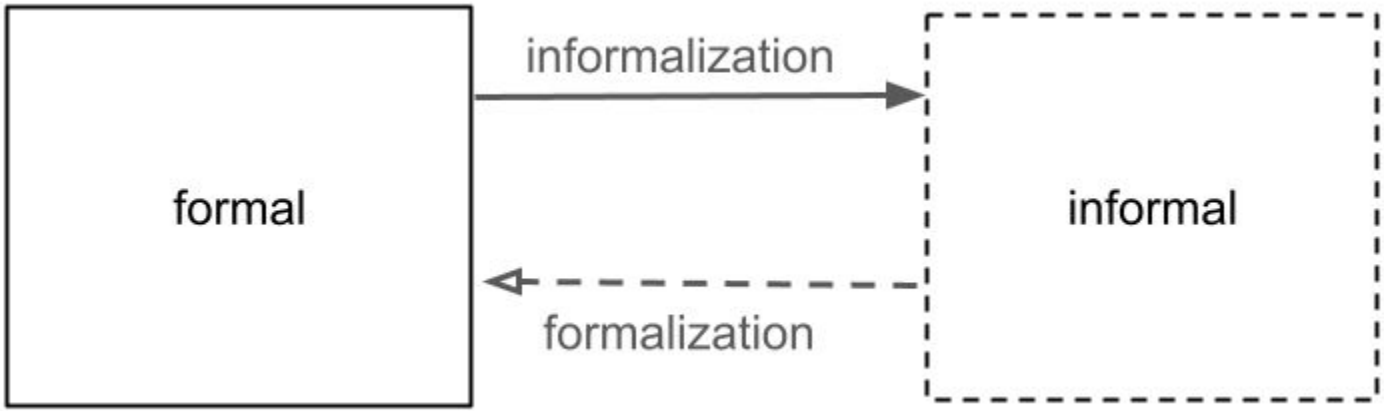
Thm20a. Let p be a natural number. Assume that p is prime. Let k be a natural number. Assume that $p = 4 \times k + 1$. Then there exists a natural number x, such that there exists a natural number y, such that $p = x^2 + y^2$.

Thm20b. Let p be a natural number. Assume that p is prime. Assume that $p \equiv 1 \bmod 4$. Then there exists a natural number x, such that there exists a natural number y, such that $p = x^2 + y^2$.

Thm22. It is not the case that $\mathbb{R}$ is denumerable.

Mock-up example: the code and the text do not correspond to each other.


informalization
formal
informal
formalization



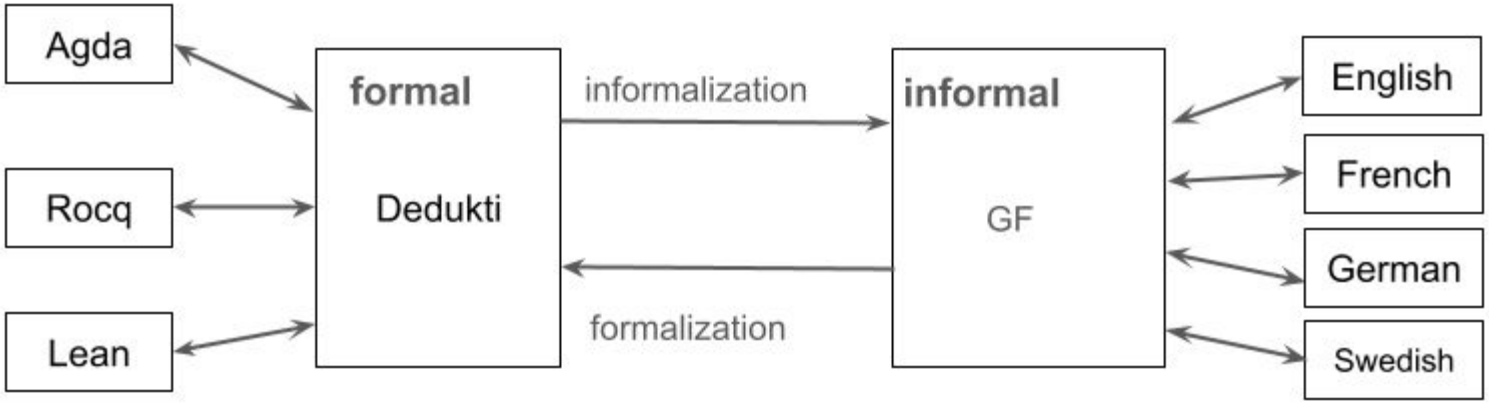
formal language
- precisely defined
- possible to check
- possible to cover completely

LLM formalization
- complete coverage
- results can be checked

rule-based informalization
- comprete coverage
- correct by construction

informal language
- not precisely defined
- not possible to check
- impossible to cover completely

The project Informath: structure



https://www.grammaticalframework.org/
https://github.com/GrammaticalFramework/informath

The project Informath: goals

Semantic completeness: everything expressible in type theory can be translated to Informath natural language.

Semantic soundness: everything expressible in Informath can be translated to type theory.

Linguistic completeness: everything attested in mathematical texts should be eventually covered by Informath.

Linguistic soundness: nothing should be added to Informath unless it is attested in mathematical texts.

Some statistics

Syntax rules: ~ 200 GF functions, extends to 87,000 BNF rules in English

Lexicon: ~ 4000 items, around 11,000 word forms in English

Languages: English, French, German, Swedish (common abstract syntax, implemented with a functor over GF Resource Grammar API)

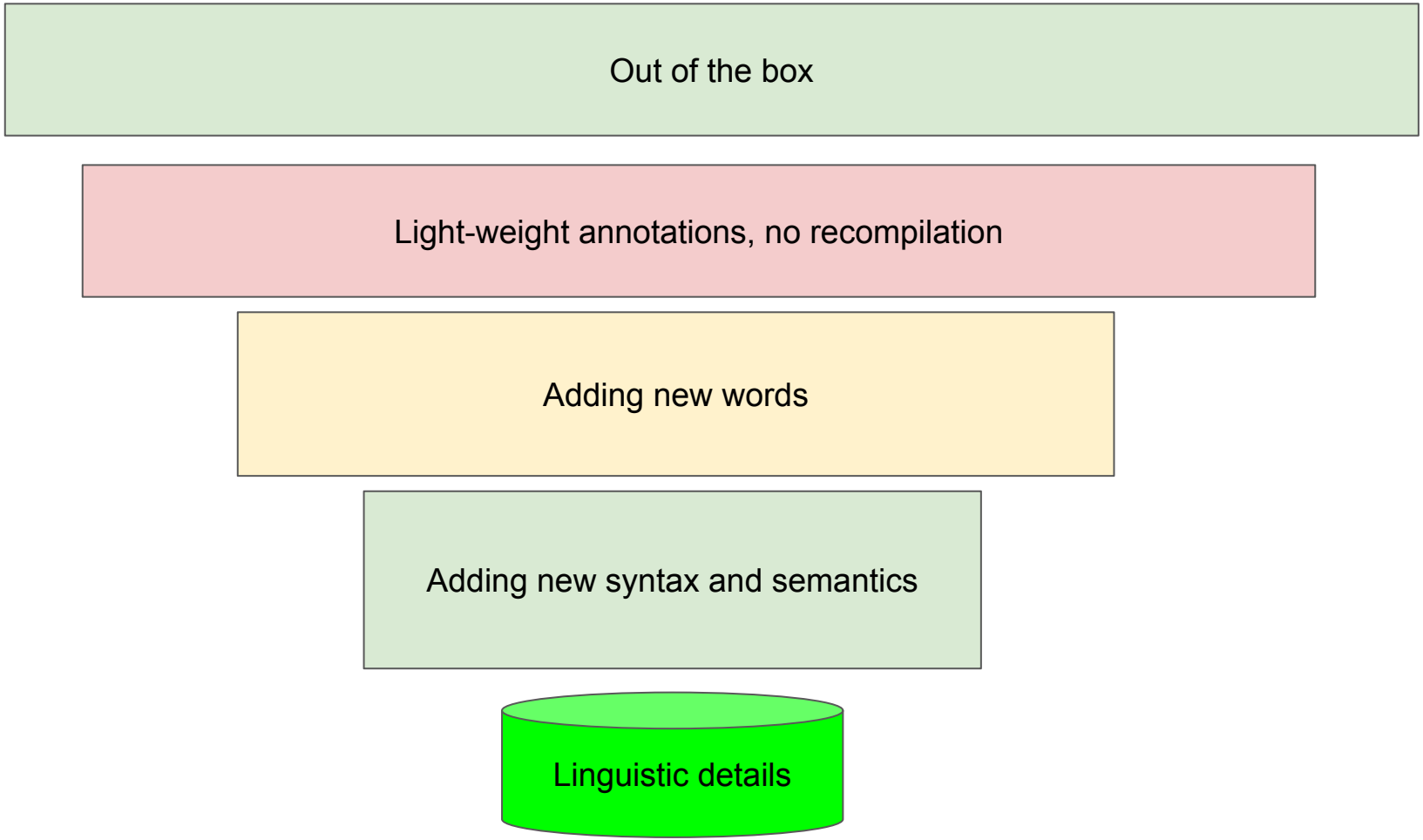
Formalisms: Dedukti; Agda, Lean, Rocq ; Megalodon and Mizar forthcoming

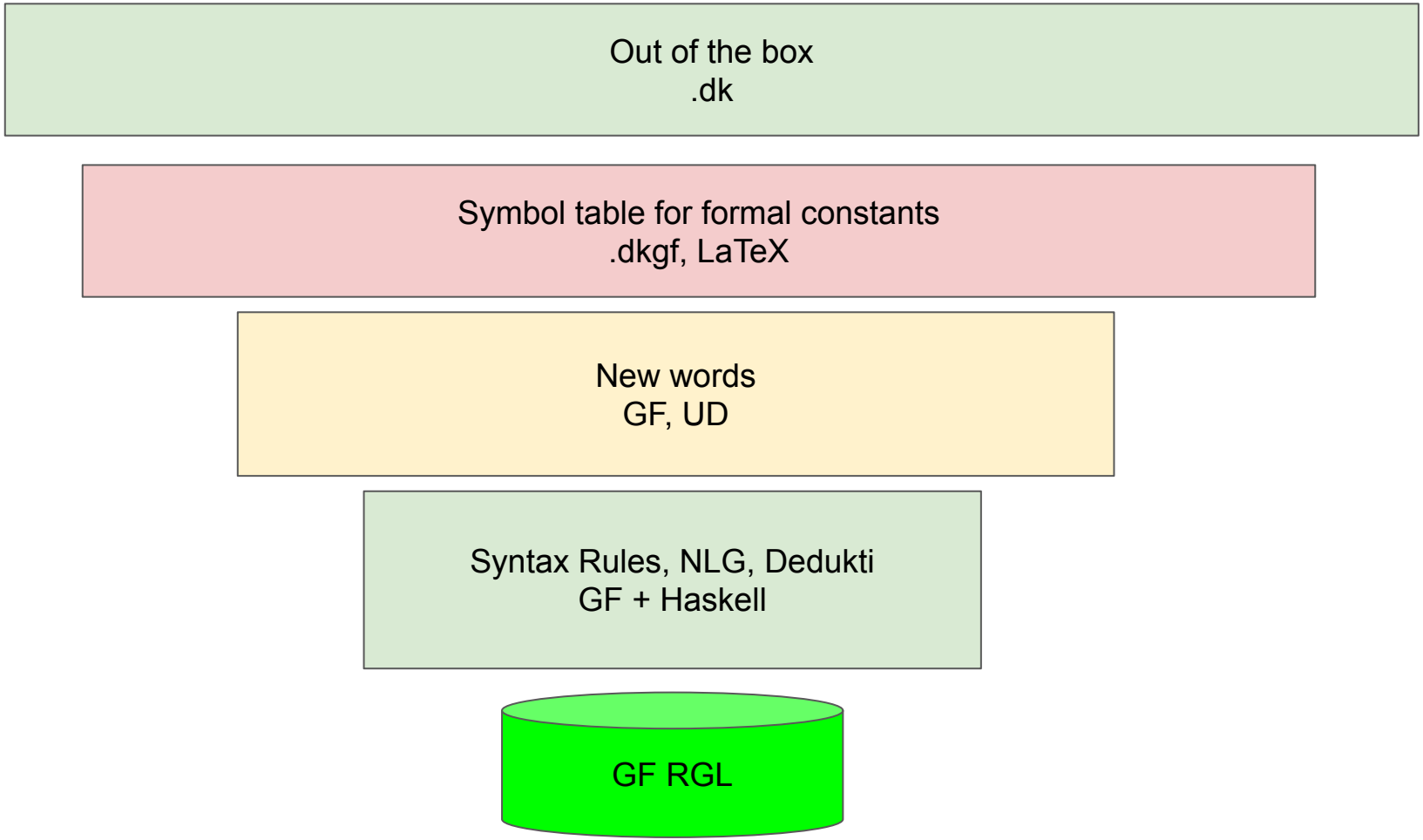
Examples:

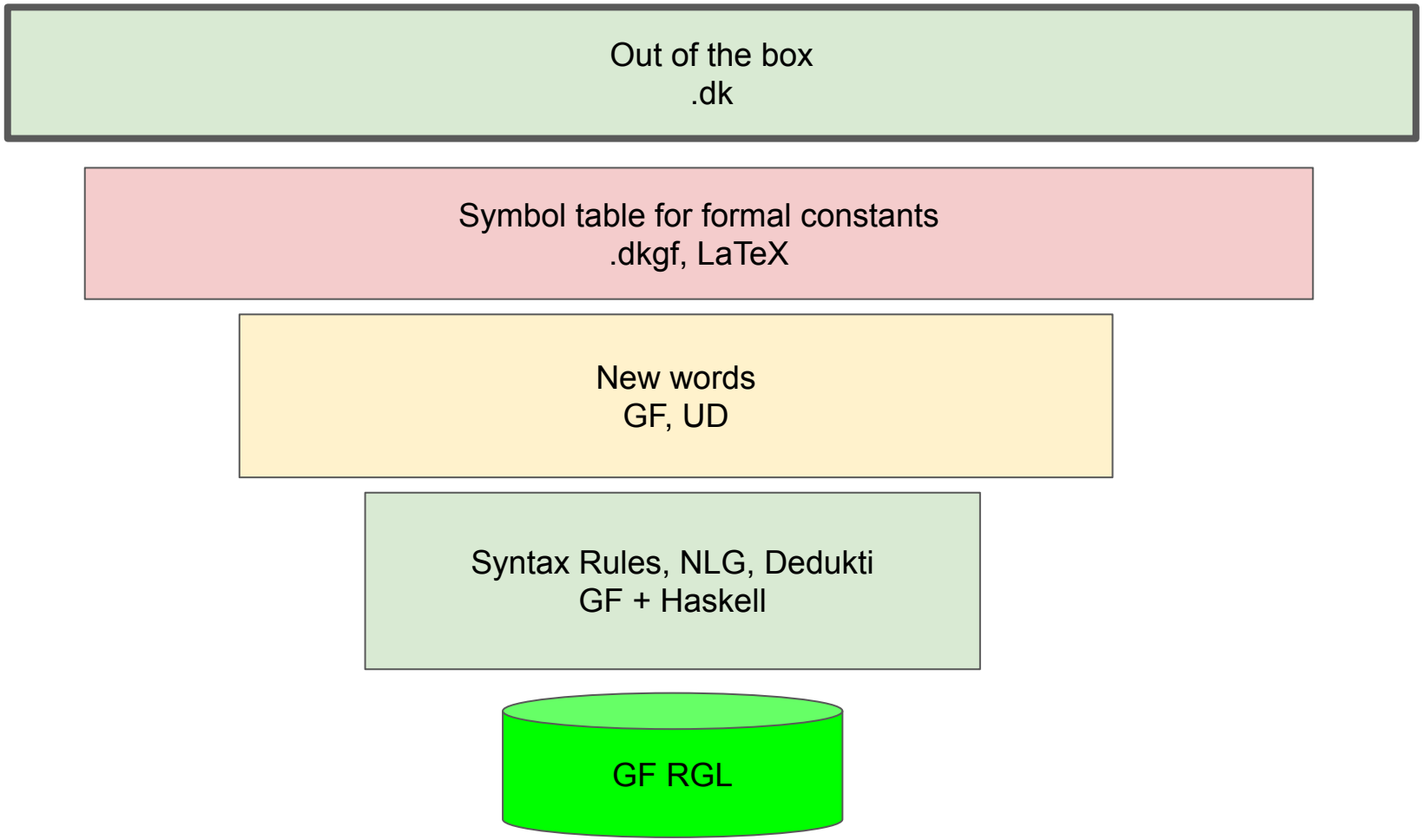
- arithmetic, algebra, set theory, topology, a sample of Wiedijk's "100 theorems", homotopy type theory;
- Dedukti dumps from Lean, Matita, Megalodon, Agda

Restrictions: proofs still very rudimentary

The Informath deployment stack


Out of the box
Light-weight annotations, no recompilation
Adding new words
Adding new syntax and semantics
Linguistic details


Out of the box
.dk
Symbol table for formal constants
.dkgf, LaTeX
New words
GF, UD
Syntax Rules, NLG, Dedukti
GF + Haskell
GF RGL


Out of the box
.dk
Symbol table for formal constants
.dkgf, LaTeX
New words
GF, UD
Syntax Rules, NLG, Dedukti
GF + Haskell
GF RGL

Installing Informath: binary

https://github.com/GrammaticalFramework/informath

https://github.com/GrammaticalFramework/informath/releases

Running the Informath binary: demo

```
$ make demo
```

Running the Informath binary: more demos

```
$ make sigma


$ make hott_demo


$ make naproche


$ make lang=Fre top100
```

Running the Informath binary: informalize a .dk file

```
$ RunInformath test/sets.dk
```

Running the Informath binary: inspect options

```
$ RunInformath -help
```

Running the Informath binary: some useful options

```
$ RunInformath -variations -nbest=11


$ RunInformath -to-latex-doc


$ RunInformath -to-lang=Fre


$ RunInformath -to-formalism=lean
```

Running the Informath binary: formalize a .tex file

```
$ RunInformath -test/naproche-zf-set.tex
```

Running the Informath binary on std-in

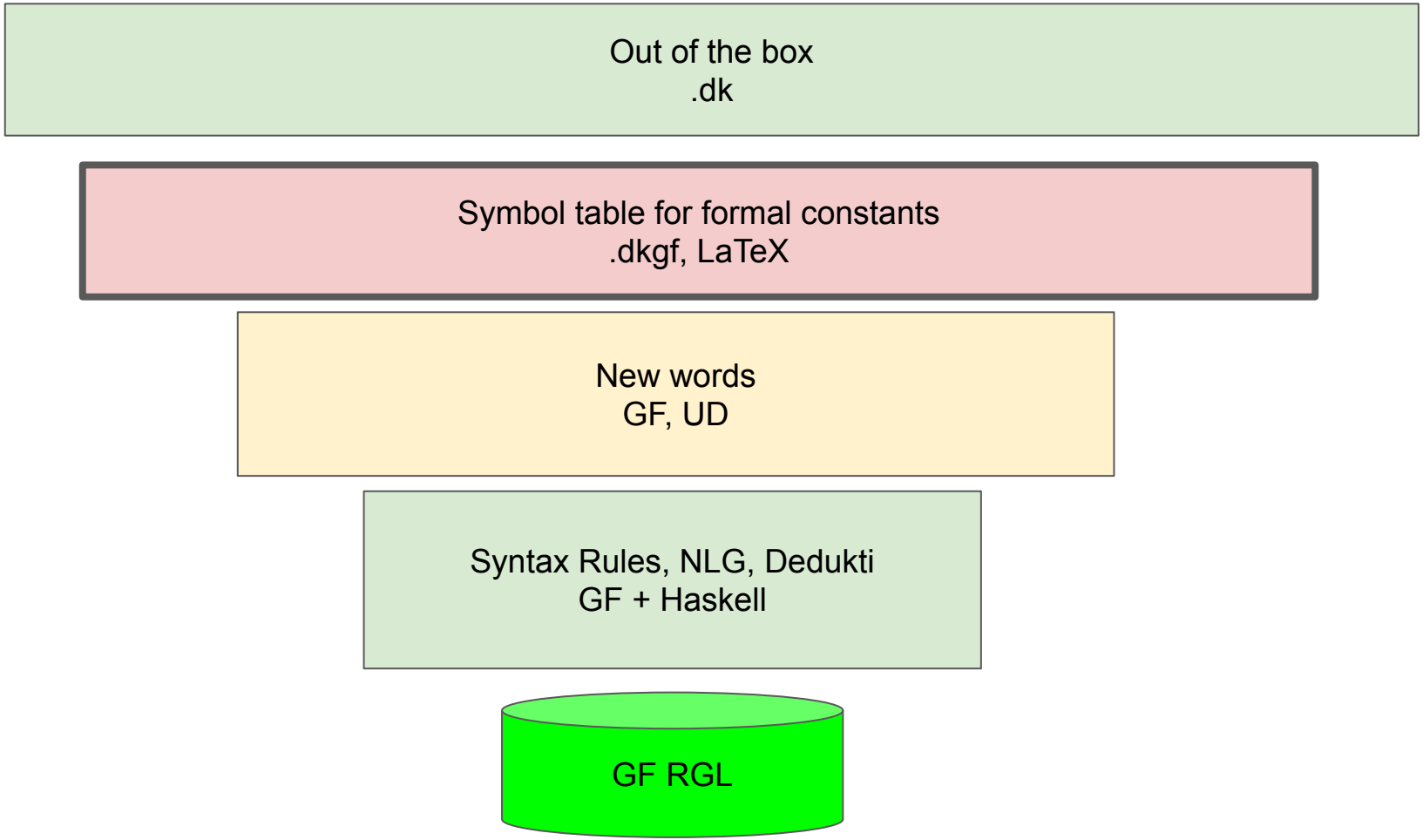
```
$ echo "c : Proof (exists Nat (n => divide n 7 ))." | RunInformath


$ echo "Every number is even or odd." | RunInformath -formalize
```

Running the Informath binary: analyse files

```
$ RunInformath -idents test/mini-matita.dk


$ RunInformath -unknown-idents test/mini-matita.dk


$ RunInformath-unknown-words ../rijke/rijke-tex/groups.tex
```

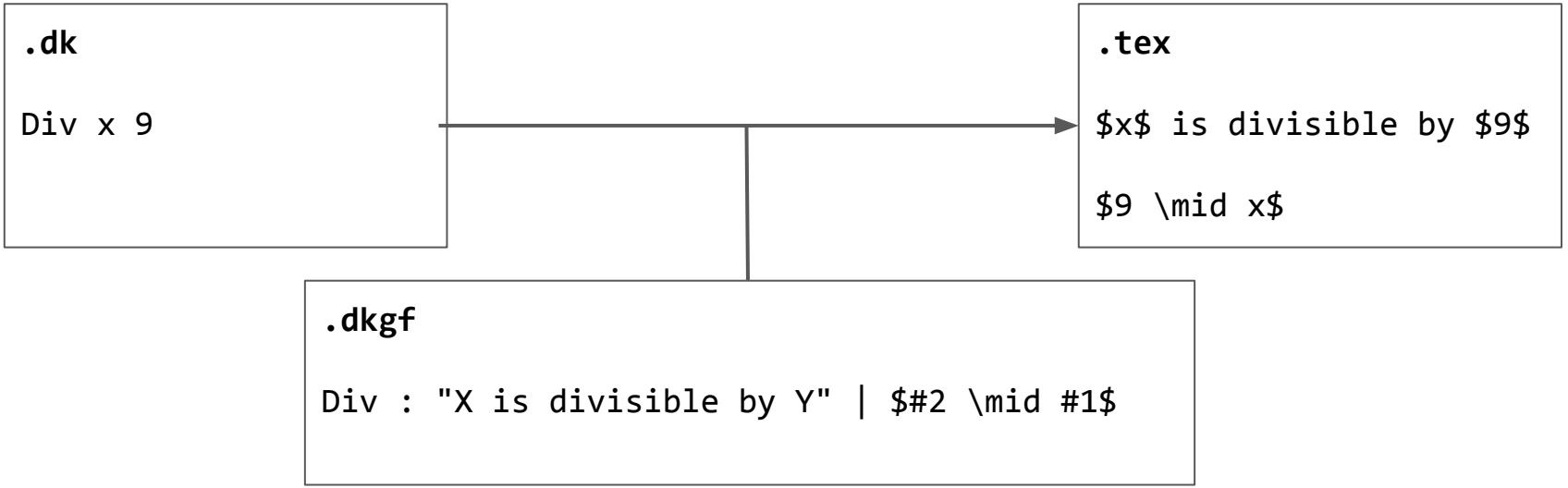
Running the Informath binary: select symbol tables

```
$ RunInformath test/hott_demo.dk

$ RunInformath -unknown-idents test/hott_demo.dk


$ RunInformath -symboltables=test/hott_demo.dkgf test/hott_demo.dk
```


Out of the box
.dk
Symbol table for formal constants
.dkgf, LaTeX
New words
GF, UD
Syntax Rules, NLG, Dedukti
GF + Haskell
GF RGL

Adapting to new concepts: symbol tables

```
-- file.dk

Div : Elem Nat -> Elem Nat -> Prop.




-- file.dkgf

Div: "X is divisible by Y" | $#2 \mid #1$
```


.dk
Div x 9
.tex
x is divisible by 9
$9 \mid x$
.dkgf
Div : "X is divisible by Y" | $#2 \mid #1$

GF abstract syntax and some derived structures

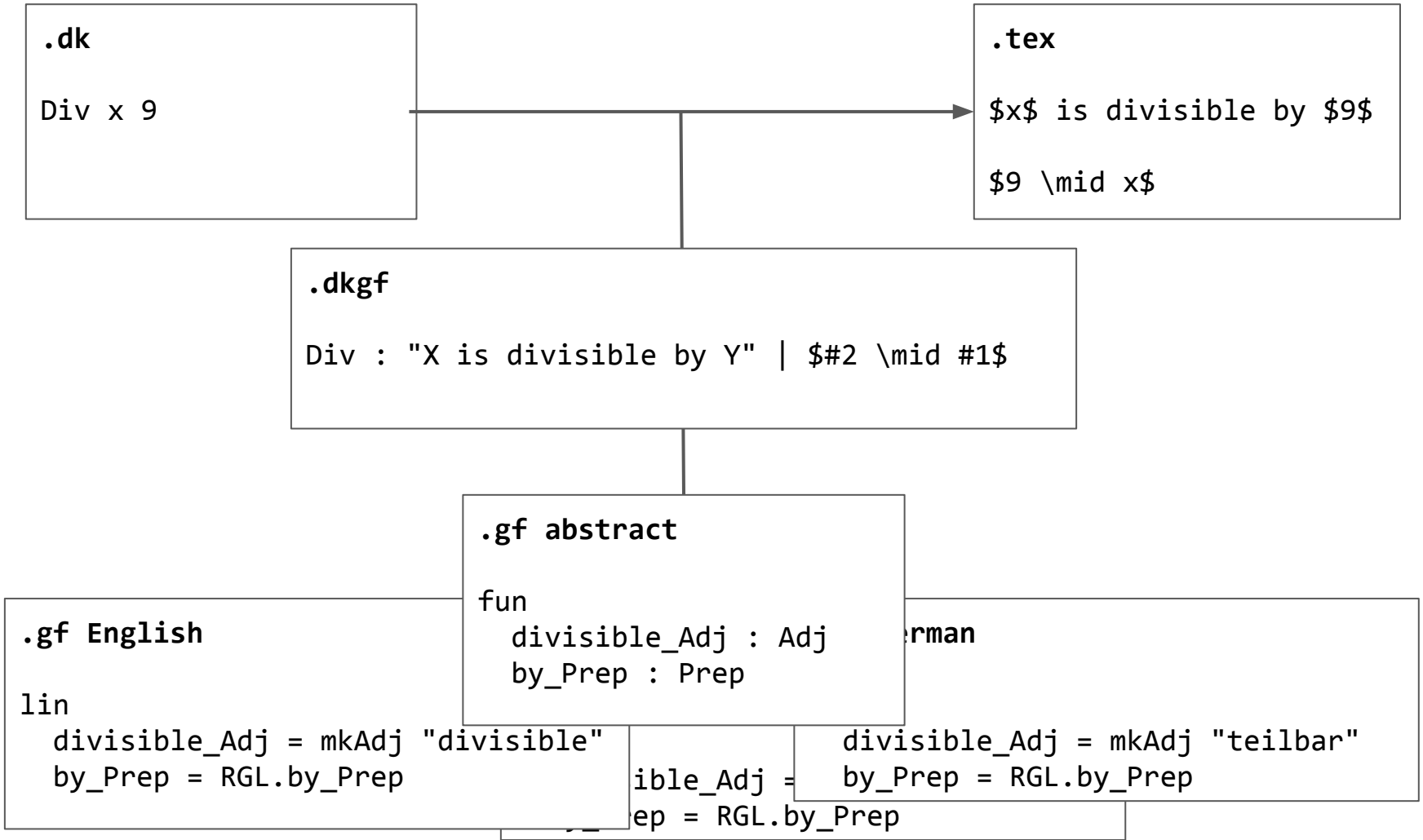
`Eq: "X is divisible by Y"`	→	`divisible_Adj2`
- *6 is not divisible by 4*		`not (Div 6 4)`
- *6 and 9 are divisible by 3*		`and (Div 6 3) (Div 9 3)`
- *x is even and divisible by 3*		`and (Even x) (Div x 3)`
- *let n be a number divisible by 4*		`(n : Elem Nat) -> Proof (Div n 4) ->`

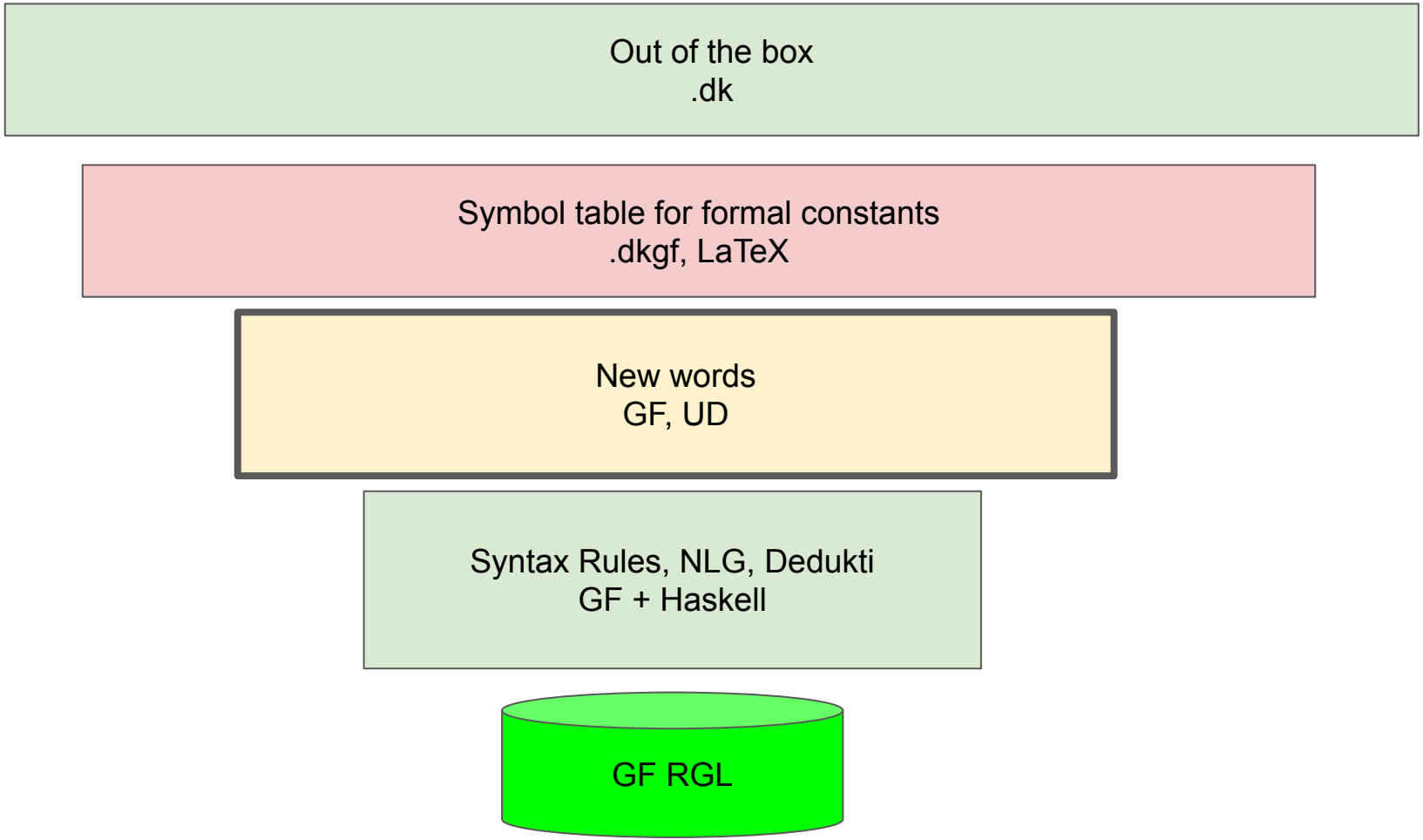
Running the Informath binary: testing examples

```
$ echo "X is divisible by Y" | RunInformath -parse-example


$ echo "X is a principal ultrafilter" | RunInformath -parse-example
```

Checking symbol tables

```
$ RunInformath -base=test/maps.dk test/maps.dkgf
```


.dk
Div x 9
.tex
x is divisible by 9
$9 \mid x$
.dkgf
Div : "X is divisible by Y" | $#2 \mid #1$
.gf abstract
fun
divisible_Adj : Adj
by_Prep : Prep
.gf English
lin
divisible_Adj = mkAdj "divisible"
by_Prep = RGL.by_Prep
rman
divisible_Adj = mkAdj "teilbar"
by_Prep = RGL.by_Prep


Out of the box
.dk
Symbol table for formal constants
.dkgf, LaTeX
New words
GF, UD
Syntax Rules, NLG, Dedukti
GF + Haskell
GF RGL

Manually in GF: files

```
abstract MyWords = Categories ** {

fun dictionary_Noun : Noun ;
fun explicit_Adj : Adj ;
fun put_Verb : Verb ;
}
```

```
concrete MyWordsEng = CategoriesEng **

open UtilitiesEng, ParadigmsEng, IrregEng in {

lin dictionary_Noun = mkNoun "dictionary" ;
lin explicit_Adj = mkAdj "explicit" ;
lin put_Verb = mkVerb put_V ;
}
```

Manually in GF: files

```
abstract MyWords = Categories ** {

fun dictionary_Noun : Noun ;
fun explicit_Adj : Adj ;
fun put_Verb : Verb ;
}
```

```
concrete MyWordsFre = CategoriesFre **
open UtilitiesFre, ParadigmsFre, IrregFre in {

lin dictionary_Noun =
  mkNoun "dictionnaire" masculine ;
lin explicit_Adj = mkAdj "explicite" ;
lin put_Verb = mkVerb mettre_V ;
}
```

Include and recompile

```
abstract UserExtensions =
  Categories,
  Naproche,
  NaprocheWords,
  HottWords,
  GodementWords,
  NaturalDeduction,
  ProofUnits,
  MyWords
  ;
```

```
concrete UserExtensionsEng of UserExtensions =
  CategoriesEng,
  NaprocheEng,
  NaprocheWordsEng,
  HottWordsEng,
  GodementWordsEng,
  NaturalDeductionEng,
  ProofUnitsEng,
  MyWordsEng
  ;
```

```
$ make english_grammar

$ make multi_grammar
```

Running a script on a word list

```
# file adjs.txt
explicit
implicit
elegant
hard
connected
```

```
$ python3 generate_gf.py Adj <adjs.txt
```

```
$ make multi.gpgrammar
```

Preparing a word list from UD trees

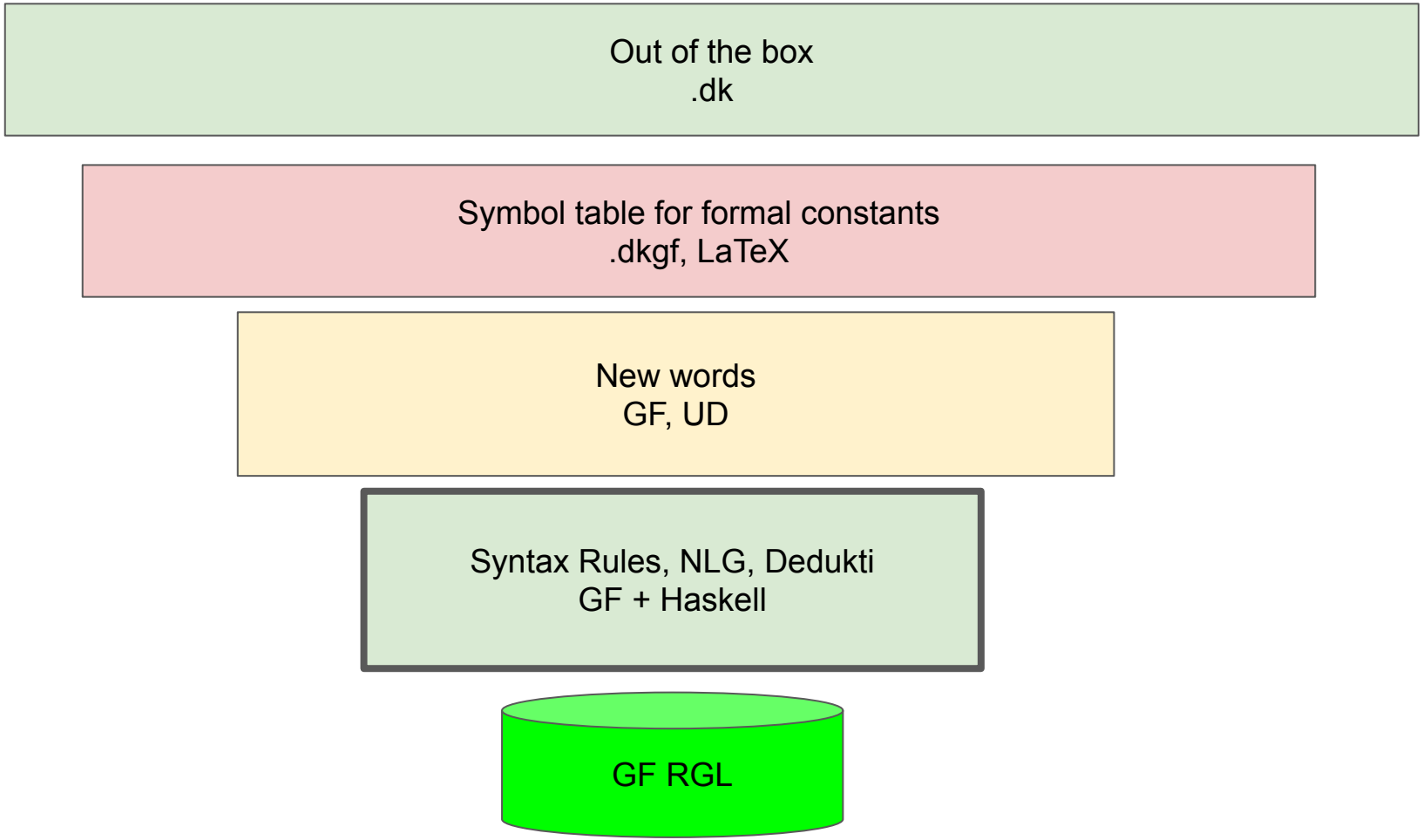
```
# file godement.conllu

# text = Let LATEXMATH be elements of a principal ideal domain K .
1       Let     let     VERB    VB      Mood=Imp|VerbForm=Fin   0       root    _       SpacesAfter=\s\s
2       LATEXMATH       Latexmath       PROPN   NNP     Number=Sing     1       obj     _       SpacesAfter=\s\s
3       be      be      AUX     VB      VerbForm=Inf    4       cop     _       _
4       elements        element NOUN    NNS     Number=Plur     1       xcomp   _       _
5       of      of      ADP     IN      _       10      case    _       _
6       a       a       DET     DT      Definite=Ind|PronType=Art       10      det     _       _
7       principal       principal       ADJ     JJ      Degree=Pos      9       amod    _       _
8       ideal   ideal   ADJ     JJ      Degree=Pos      9       amod    _       _
9       domain  domain  NOUN    NN      Number=Sing     10      compound        _       _
10      K       K       PROPN   NNP     Number=Sing     4       nmod    _       _
11      .       .       PUNCT   .       _       1       punct   _       SpacesAfter=\r\n\r\n
```

```
$ cut -f3,4 godement.conllu | grep ADJ
```

Probing the existing lexicon

```
# file adjs.txt
explicit
implicit
elegant
hard
connected
```

```
$ RunInformath -unknown-words adjs.txt >unadjs.txt
```


Out of the box
.dk
Symbol table for formal constants
.dkgf, LaTeX
New words
GF, UD
Syntax Rules, NLG, Dedukti
GF + Haskell
GF RGL

GF + dkgf

```
abstract MySyntax = Categories ** {

fun InverseIf : Prop -> Prop -> Prop ;

}
```

```
concrete MySyntaxEng = CategoriesEng **
  open GrammarEng, SyntaxEng in {

lin InverseIf A B = SSubjS A if_Subj B ;

}
```

```
# file mysyntax.dkgf

#SEMANTICS InverseIf A B = IfProp B A

#NLG IfProp A B = InverseIf B A
```

GF + dkgf (special case)

```
abstract MySyntax = Categories ** {

fun InverseIf : Prop -> Prop -> Prop ;

}
```

```
concrete MySyntaxEng = CategoriesEng **
  open GrammarEng, SyntaxEng in {

lin InverseIf A B = SSubjS A if_Subj B ;

}
```

```
# file mysyntax.dkgf

#SEMANTICS InverseIf A B = IfProp B A

#NLG IfProp A B = InverseIf B A
```

Restriction: flat linear patterns on the left hand side (compositionality)

GF + Haskell (general case)

```
abstract MySyntax = Categories ** {


fun AndAdj : Adj -> Adj -> Adj ;
}
```

```
concrete MySyntaxEng = CategoriesEng **
  open GrammarEng, SyntaxEng in {

lin AndAdj A B = mkAP and_Conj A B ;
}
```

```
# file Informath2MathCore.hs

sem (AdjProp (AndAdj A B) x) = AndProp (AdjProp A x) (AdjProp B x)

# file MathCore2Informath.hs

aggregate (AndProp (AdjProp A x) (AdjProp B y)) | x == y = AdjProp (AndAdj A B) x
```

After this: compile both Informath.pgf and RunInformath.hs

Conclusion

Knowledge and tools needed at each level

dkgf:

- how to express concepts in natural language
- RunInformath binary, unix-like shell

gf:

- how to define words in GF by using its libraries
- the GF grammar compiler and Resource Grammar Library

hs:

- hs: how to define recursive traversals in right order and right conditions
- a Haskell compiler (ghc)

Moving things higher up the stack

Goal: to make Informath more accessible, more adaptable, and easier to deploy

Method: by extending the expressivity of dkgf

- first, just GF lexical functions
- second, arbitrary GF functions
- third: natural language examples
- fourth: latex macros
- fifth: #SEMANTICS definitions
- sixth: #NLG definitions
- seventh: $ notation for automatically generated macros
- TODO: more powerful #SEMANTICS and #NLG

More automation

Goal: to speed up deployment without loss of quality

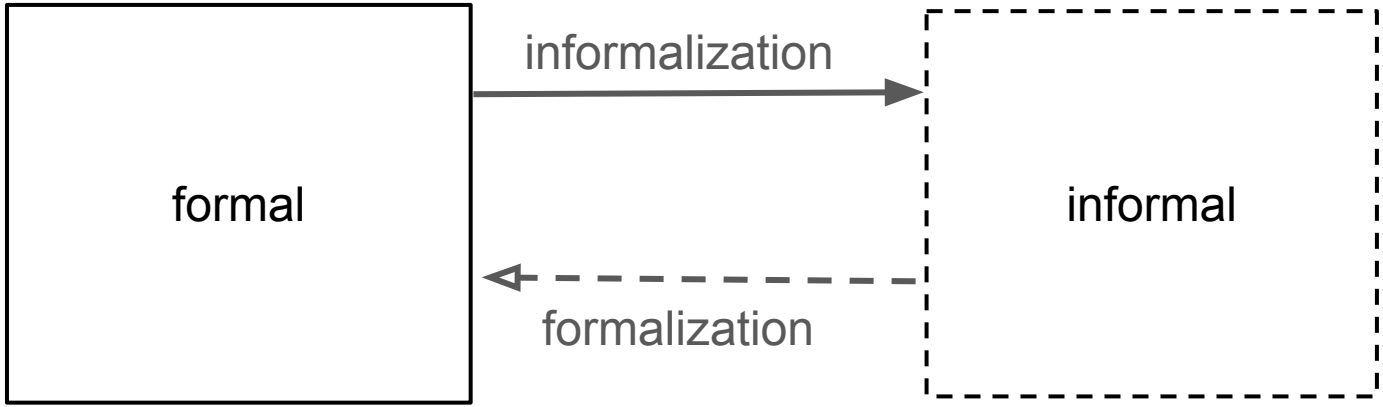
Ideas:

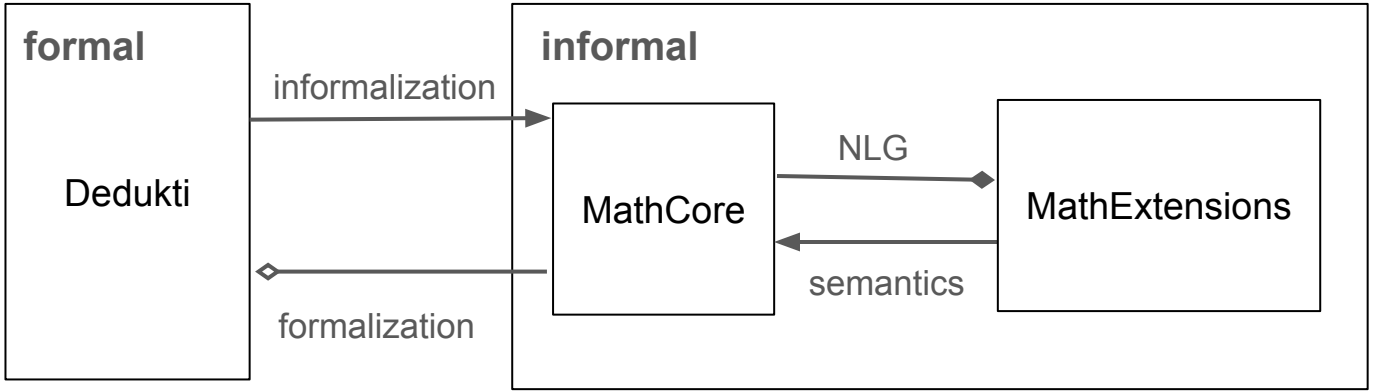
- Symbol table and grammar building as a by-product of LLM autoformalization
- LLM-based ranking of NLG results

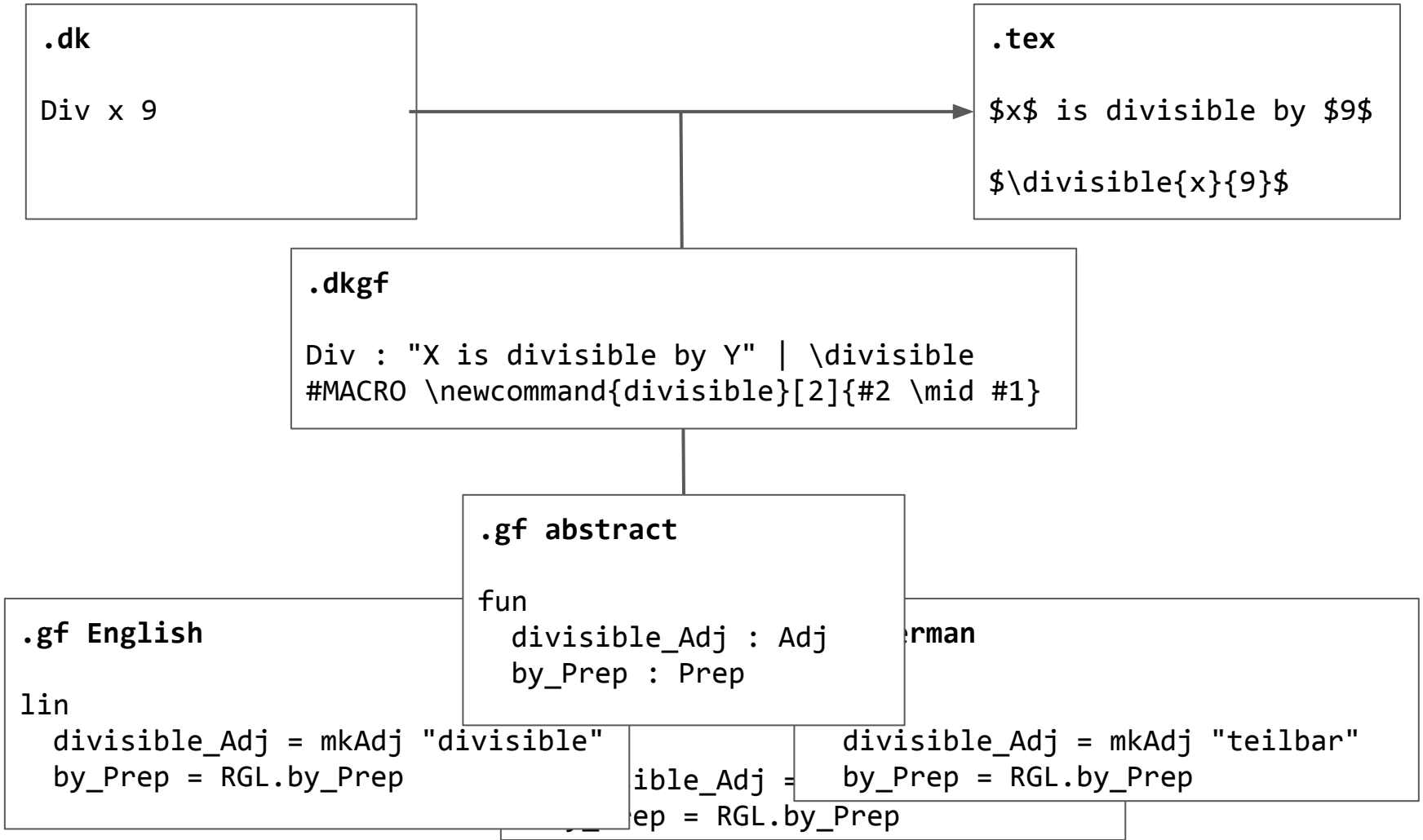
Idea from Paul-André Melliès at the MALINCA workshop: when informalizing the result of autoformalizing, choose the informalization that is as close to the original text as possible. In this way, you will get a machine-checked minimally revised version of the text.

This Is Not

The End


formal
informalization
informal
formalization


formal
Dedukti
informalization
formalization
informal
MathCore
NLG
semantics
MathExtensions


.dk
Div x 9
.tex
x is divisible by 9
$\divisible{x}{9}$
.dkgf
Div : "X is divisible by Y" | \divisible
#MACRO \newcommand{divisible}[2]{#2 \mid #1}
.gf abstract
fun
divisible_Adj : Adj
by_Prep : Prep
.gf English
lin
divisible_Adj = mkAdj "divisible"
by_Prep = RGL.by_Prep
rman
divisible_Adj = mkAdj "teilbar"
by_Prep = RGL.by_Prep