

# Modeling Formal Languages in Grammatical Framework

On the Grammar of Proof

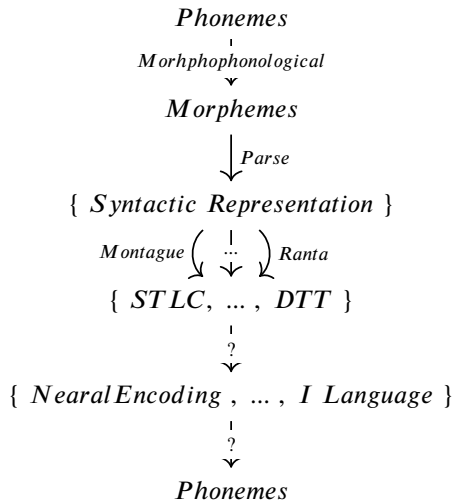
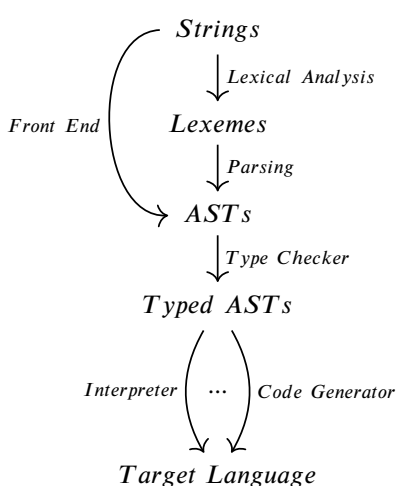
Warrick Macmillan

7<sup>th</sup> August 2021

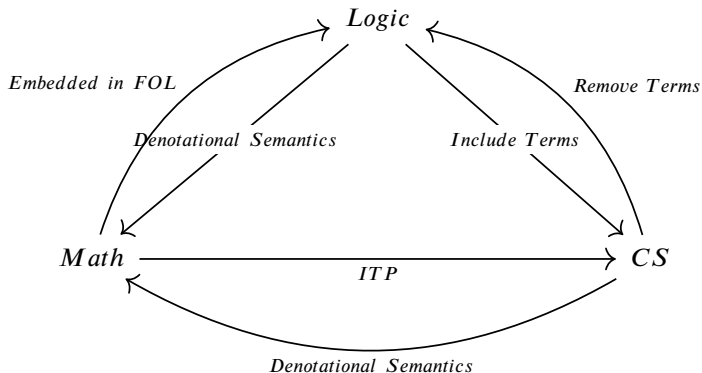
# Table of Contents

- 1 Explore abstract relationships between math, CS, Type Theory, and Linguistics
- 2 Practical and brief intro to MLTT and Agda
- 3 Grammars elaborating the abstractions above

# Abstraction Ladders



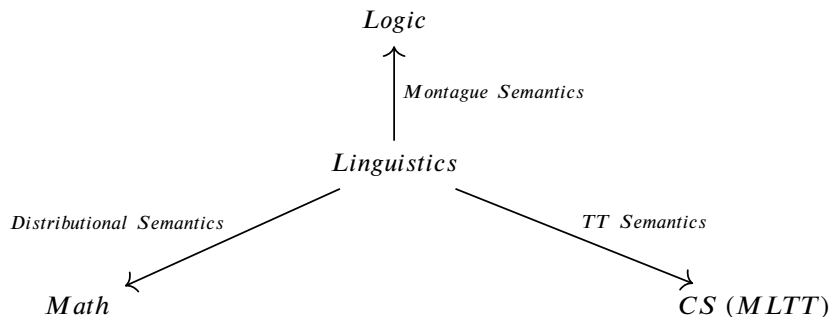
# Computational Trinitarianism



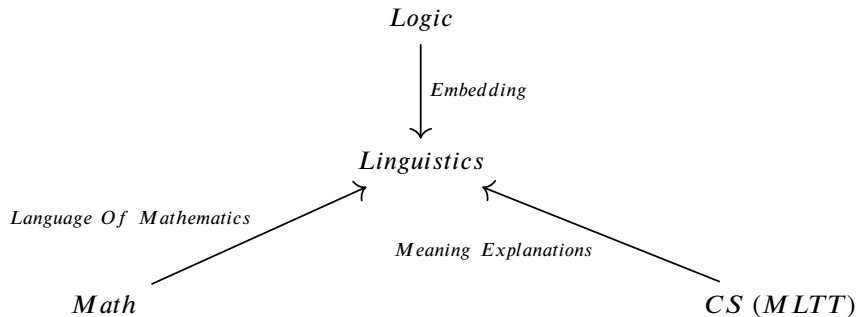
# Interpretation Language

## Observation 1.1

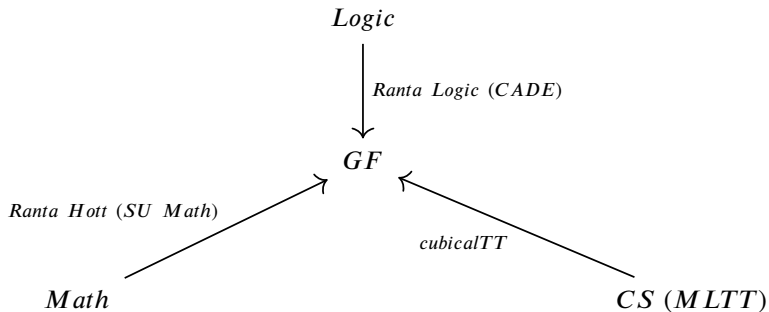
We acknowledge this is only semantic interpretations in these domains. One may decide on syntactic, pragmatic, or other ways in which to treat linguistics via these fields



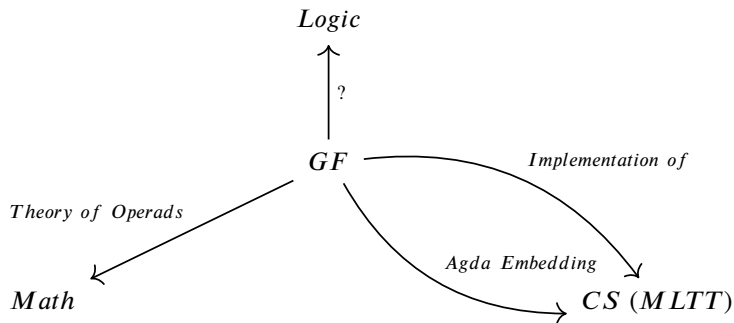
# Trinitarian Linguistics



# Trinitarian Grammars



# Models of GF





# Remarks

- Trinitarian doctrine is in the “formal” space
- Trinitarian + Linguistics is partially formal, and very underexplored
- Introduces many philosophical concerns, perhaps a rereading of Wittgenstein should take place in this context

- Frege : Formal Proof, Predicate Logic
- Russel : Type Theory to resolve his paradox
- Brouwer : Constructivism

*Mathematical logic and the relation between logic and mathematics have been interpreted in at least three different ways:*

- i. mathematical logic as symbolic logic, or logic using mathematical symbolism;*
- ii. mathematical logic as foundations (or philosophy) of mathematics;*
- iii. mathematical logic as logic studied by mathematical methods, as a branch of mathematics.*

*We shall here mainly be interested in mathematical logic in the second sense. What we shall do is also mathematical logic in the first sense, but certainly not in the third.*

*(Per Martin-Löf, Padua Italy, June 1980)*

# Syntactic Comparisons

## First Order Logic

- $\forall$
- $\exists$
- $\supset$
- $\wedge$
- $\vee$
- $\neg$
- $\top$
- $\perp$
- $=$

## Dependent Type Theory

- $\Pi$
- $\Sigma$
- $\rightarrow$
- $\times$
- $+$
- $\neg$
- $\top$
- $\perp$
- $\equiv$

## Sets

- $\mathbb{N}$
- $\mathbb{N} \times \mathbb{N}$
- $\mathbb{N} \rightarrow \mathbb{N}$
- $\{x \mid P(x)\}$
- $\emptyset$
- $?$
- $\cup$
- $?$

## More Sets

- 1
- (1, 0)

## Types

- $Nat$
- $Nat \times Nat$
- $Nat \rightarrow Nat$
- $\Sigma x : \_ . P(x)$
- $\perp$
- $\top$
- $?$
- $U_1$

## Programs

- *suc zero*
- *(suc zero, zero)*

# Judgments

## Type Theoretic Judgments

- $T$  is a type
- $T$  and  $T'$  are equal types
- $t$  is a term of type  $T$
- $t$  and  $t'$  are equal terms of type  $T$

## Mathematical Judgments

- $P$  is a proposition
- $P$  is true

- Notice that judgmental equality is uniquely type theoretic
- Judgments in type theory are decidable
- Truth (inhabitation) is not decidable
- More exotic judgments are available in TT, i.e.  $P$  is possible.

# Important Differences

- The rules of the types make explicit that they are not equivalent to those of classical FOL
- An existential assertion in type theory requires data
- Excluded middle and double negation are not admitted in MLTT
- To be *not unhappy* is clearly of a different meaning than to be *happy*.
- This makes our approach to general translation of non-constructive mathematics *impossible* (at least such that it type-checks)

- One doesn't define logics, type systems in mathematics (e.g. metamathematics)
- Encoding things like rational and real numbers in type theory are already, category theorists and set theorists are at odds, (small and large categories), higher categories, which skeletons of categories are canonical, etc. incredibly difficult
- Additionally, intensional type theory comes with two distinct notions of equality, judgmental/definitional/computational and propositional equality



# Example Donkey Anaphora

Interpret the sentence “every man who owns a donkey beats it” in MLTT via the following judgment :

$$\Pi z : (\Sigma x : \textit{man}. \Sigma y : \textit{donkey}. \textit{owns}(x, y)). \textit{beats}(\pi_1 z, \pi_1(\pi_2 z))$$

We judge  $\vdash \textit{man} : \textit{type}$  and  $\vdash \textit{donkey} : \textit{type}$ .  $\textit{type}$  really denotes a universe

# What is Agda?

- Implementation of MLTT
- Logical Framework
- Interactive proof development environment
- Inductive Types, Modules, Pattern Matching, more

## Mathematical Declarations

- Theorem
- Proof
- Lemma
- Axiom
- Definition
- Example

# Twin Prime Conjecture

## Definition

A *twin prime* is a prime number that is either 2 less or 2 more than another prime number

Alternatively, we may state it as follows :

## Definition

A *twin prime* is a prime that has a prime gap of two.

## Definition

A *prime gap* is the difference between two successive prime numbers.

## Theorem

*There are infinitely many twin primes.*

# Twin Prime Conjecture in Agda

# What is a Proof?

*A proof is what makes a judgment evident*

*(Per Martin-Löf)*

*...there is a considerable gap between what mathematicians claim is true and what they believe, and this mismatch causes a number of serious linguistic problems.*

*(Mohan Ganesalingam)*

Comparsion	Formal Proof	Informal Proof
Audience	Agda (and Human)	Human
Translation	Compiler	Human
Objectivity	Objective	Subjective
Historical	20th Century	$\leq$ Euclid
Orientation	Syntax	Semantics
Inferability	Complete	Domain Expertise Necessary
Verification	PL Designer	Human
Ambiguity	Unambiguous	Ambiguous



- One missed comparison from above : formal proof is an implementation, an informal proof is a specification
- Why? Historically, we think of semantics preceding (the abstract notion of a circle preceded its geometric understanding)
- although syntax oriented thinking may now be dominating the CS tradition)
- syntax oriented approach in Agda program,
- Both are necessary in the end, especially for big proofs.
- Propositions or theorem statements are *intentionally unambiguous*

*...when it comes to understanding the power of mathematical language to guide our thought and help us reason well, formal mathematical languages like the ones used by interactive proof assistants provide informative models of informal mathematical language. The formal languages underlying foundational frameworks such as set theory and type theory were designed to provide an account of the correct rules of mathematical reasoning, and, as Gödel observed, they do a remarkably good job. But correctness isn't everything: we want our mathematical languages to enable us to reason efficiently and effectively as well. To that end, we need not just accounts as to what makes a mathematical argument correct, but also accounts of the structural features of our theorizing that help us manage mathematical complexity.*

*(Avigad)*

## Syntactic Completeness

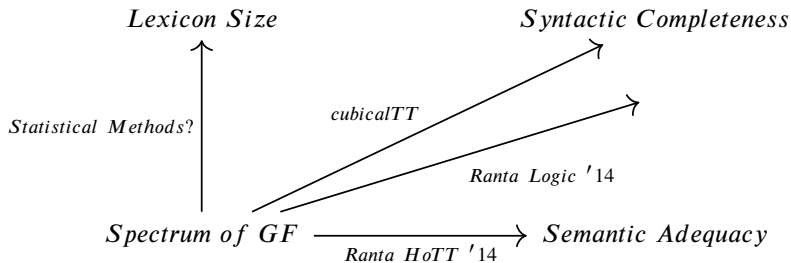
Given an utterance or natural language expression that a mathematician might understand, does the GF grammar emit a well-formed, syntactically complete expression in the target logic or programming language?

- The intended meaning manifests differently for different people
  - there may be no absolute intention
- Many possible syntax's - beyond just alpha conversion (think Coq) Perhaps the details will have to change depending on the tools available

## Semantic Adequacy

Given a well formed syntactic expression in, for instance, Agda, one can ask if the resulting NL expression generated by GF is *semantically adequate*, and coherent to a “fluent speaker”

- Think expressive adequacy in logic
- In dispute among mathematicians
- Kind of like a “Turing test for our approach”
- depends historical developments as our language develops, both internally (cite gangaselem) and more interestingly, as a society. so the things all mathematicians take for granted (and therefore, have implicitly more intuition for). think of technology as well



# Concrete vs Abstract

Comparison of concrete vs. abstract perspective when designing a GF Grammar

Abstract : Capture more semantic content (from a NL perspective) more “freedom of expression” simpler makes easier to work with in Haskell (where tactics would come into play in Coq)

Concrete : Simpler semantic space, easier for designing PL for it eliminates ambiguity. focus on this may mean more feasibility for real implementation

As will be seen, a smaller abstract syntax leans towards syntactic completeness with a larger linearization syntax (in terms of the complexity of the lincats effects the complexity of the entire linearization space)

# Logic '11

- Ranta Cade 2011
- Application grammar for logical propositions defined over some mathematical domain (integral arithmetic, euclidean geometry)
- Doesn't cover proofs
- Uses a core grammar and extended, translate between them via PGF extension

## Ideas

- One can use theorems from FOL, to simplify the NL expression (semantics preserving normalizer)
- Serves as a basis for other logics

## Core

- minimal necessary expressivity
- syntactically complete
- $\forall x(Nat(x) \supset Even(x) \vee Odd(x))$
- “for all x, if x is a natural number then x is even or x is odd”
- ambiguous parses : Catalan explosion with n conjunctions
- need normal form

## Semantically Inadequate

“is it the case that the sum of 3 and the sum of 4 and 10 is prime and 9999 is odd”



# Core Syntax

## construction

negation

conjunction

disjunction

implication

universal quantification

existential quantification

## symbolic verbal

$\sim P$      *it is not the case that  $P$*

$P \ \& \ Q$       *$P$  and  $Q$*

$P \ \vee \ Q$       *$P$  or  $Q$*

$P \ \supset \ Q$      *if  $P$  then  $Q$*

$(\forall x)P$      *for all  $x$ ,  $P$*

$(\exists x)P$      *there exists an  $x$  such that  $P$*

## Extended

- much more expressive
- semantically adequate
- “every natural number is even or odd”
- increases both number of categories and functions
- also need for more complicated linearization categories
- complex PGF backend to keep this syntactically complete
- questions about scalability

# Extended Syntax

construction	symbolic	verbal (example)
atom negation	$\overline{A}$	<i>x is not even</i>
conjunction of proposition list	$\&[P_1, \dots, P_n]$	<i>P, Q and R</i>
conjunction of predicate list	$\&[F_1, \dots, F_n]$	<i>even and odd</i>
conjunction of term list	$\&[a_1, \dots, a_n]$	<i>x and y</i>
bounded quantification	$(\forall x_1, \dots, x_n : K)P$	<i>for all numbers x and y, P</i>
in-situ quantification	$F(\forall K)$	<i>every number is even</i>
one-place predication	$F^1(x)$	<i>x is even</i>
two-place predication	$F^2(x, y)$	<i>x is equal to y</i>
reflexive predication	$\text{Refl}(F^2)(x)$	<i>x is equal to itself</i>
modified predicate	$\text{Mod}(K, F)(x)$	<i>x is an even number</i>

# Translation

$\llbracket - \rrbracket : \textit{Extended} \rightarrow \textit{Core}$

- relatively simple (in the sense that it should be deterministic)
- More or less uses the same logical structure from the “standard view”
- Core syntax as a model for extended

$\llbracket - \rrbracket : \text{Core} \rightarrow \text{Extended}$

- Flattening a list

$x \text{ and } y \text{ and } z \mapsto x, y \text{ and } z$

- Aggregation

$x \text{ is even or } x \text{ is odd} \mapsto x \text{ is even or odd}$

- In-situ quantification

$\forall n \in \text{Nat}, x \text{ is even or } x \text{ is odd} \mapsto \text{every Nat is even or}$

- Negation

$it \text{ is not that case that } x \text{ is even} \mapsto \text{is not even}$

- Reflexivitazion

$x \text{ is equal to } x \mapsto x \text{ is equal to itself}$

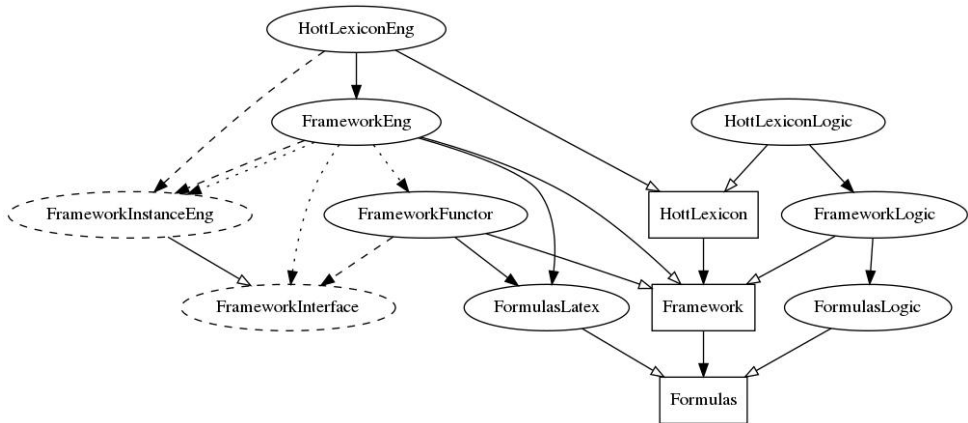
- Modification

$x \text{ is a number and } x \text{ is even} \mapsto x \text{ is an even number}$

# Ranta's (unpublished) talk at Stockholm Math Seminar

- Case study for text of *real* mathematics writing
- Purely GF translation
- Complex abstract structure including latex, metadocument structure, etc.
- Homotopy Type Theory (HoTT) specific lexicon
- Includes proofs, but AST not similar to PL
- Includes both expressions and propositions (breaks curry howard)
- Semantically Adequate but syntactically incomplete

# HoTT Grammar Modules



# Framework.gf

cat

Paragraph ; -- definition, theorem, etc  
Definition ; -- definition of a new concept  
Assumption ; -- assumption in a proof -- let ...  
[Assumption]{1} ; -- list of assumptions in one sentence  
Conclusion ; -- conclusion in a proof -- thus P  
Prop ; -- proposition, sentence or formula, A is contractible  
Sort ; -- set, type, etc corresponding to a common noun  
Ind ; -- individual element corresponding to a singular term  
Fun ; -- function with individual value  
Pred ; -- predicate: function with proposition value  
[Ind] ; -- list of individual expressions -- 1, 2 and 3  
UnivPhrase ; -- universal noun phrase -- for all x,y : A  
ConclusionPhrase ; -- conclusion word -- hence  
Label ; -- name/number of definition, theorem, etc  
Title ; -- title for theorem, definition, etc



# Formulas.gf

```
cat
  Exp ;          -- formal expression
                  --  $x + y = z$ 
  Var ;          -- variable
                  --  $x$ 
  [Var]{1} ;     -- list of variables
                  --  $x, y, z$ 
  [Exp]{1} ;     -- list of expressions
                  --  $1_{\{A\}}, A$ 
  Format ;       -- line other than content
                  --  $\backslash\text{begin}\{\text{document}\}$ 
  MathEnv ;     -- math environment
                  --  $\$ \dots \$$ 
```

# Comparative Syntax

- We work with the definition of contractability, the notion that a type (or space) is actually a point, i.e. up to equality, there is only object.
- We show the rendered latex, a pidgin agda syntax (after some concrete modifications), and the Agda code

**Definition:** A type  $A$  is contractible, if there is  $a : A$ , called the center of contraction, such that for all  $x : A$ ,  $a = x$ .

```
isContr ( A : Set ) : Set =
  ( a : A ) ( * ) ( ( x : A ) -> Id ( a ) ( x ) )
```

isContr : (A : Set) → Set  
 isContr A =  $\sum A \lambda a \rightarrow (x : A) \rightarrow (a \equiv x)$

- We also show that a notion of a map being an equivalence, that of the every element in the codomain having a contractible image (think bijection)
- notice the error in the Pidgin case

**Definition:** A map  $f : A \rightarrow B$  is an equivalence, if for all  $y : B$ , its fiber,  $\{x : A \mid fx = y\}$ , is contractible. We write  $A \simeq B$ , if there is an equivalence  $A \rightarrow B$ .

```
Equivalence ( f : A -> B ) : Set =
  ( y : B ) -> ( isContr ( fiber it ) ) ; ; ;
  fiber it : Set = ( x : A ) ( * ) ( Id ( f ( x ) ) ( y ) )
```

```
Equivalence : (A B : Set) → (f : A → B) → Set
Equivalence A B f = ∀ (y : B) → isContr (fiber' y)
  where
    fiber' : (y : B) → Set
    fiber' y = Σ A (λ x → y ≡ f x)
```

# This project

# Future Work

Convert mathematicians. More interaction between PL communities - Lean, Agda, Coq, Prl - These languages each represent a huge field of interrelated and independent research.

Linguistic analysis. Gangaselem's work is both practical and philosophical, both directions need a lot more research

A mathematician objected that it can't be difficult to prove 2 is prime in Agda. This is because they in some sense are mixing definitional and propositional equality.

Prop equality  $\leftrightarrow$  isomorphism via univalence, but even more so after cubical

HITs  $\leftrightarrow$  new proofs, new discoveries more akin to semantic feeling

A FORMAL SYSTEM FOR EUCLID'S ELEMENTS

<https://www.cambridge.org/core/journals/review-of-symbolic-logic/article/abs/formal-system-for-euclids-elements/07CA7E5F8E1C5C2EB632E1005CBE7BEF>

Different intersecting ideas of models, syntax vs semantic

Krasimir : bigger grammar, begins resembling RGL (but we also need to investigate where compile time optimizations are necessary for this) We need an RGL to take care of things that satisfy both the