Modeling Formal Languages in Grammatical Framework

On the Grammar of Proof

Warrick Macmillan

7^{th} August 2021

One Remark

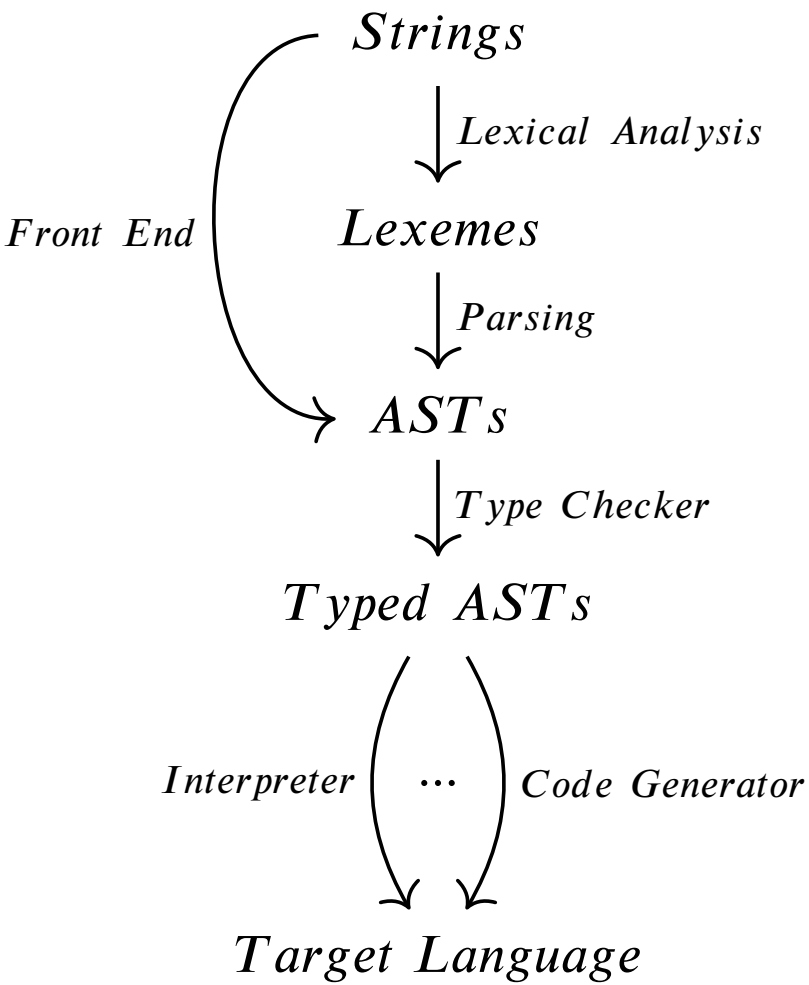
I want feedback on this talk, please

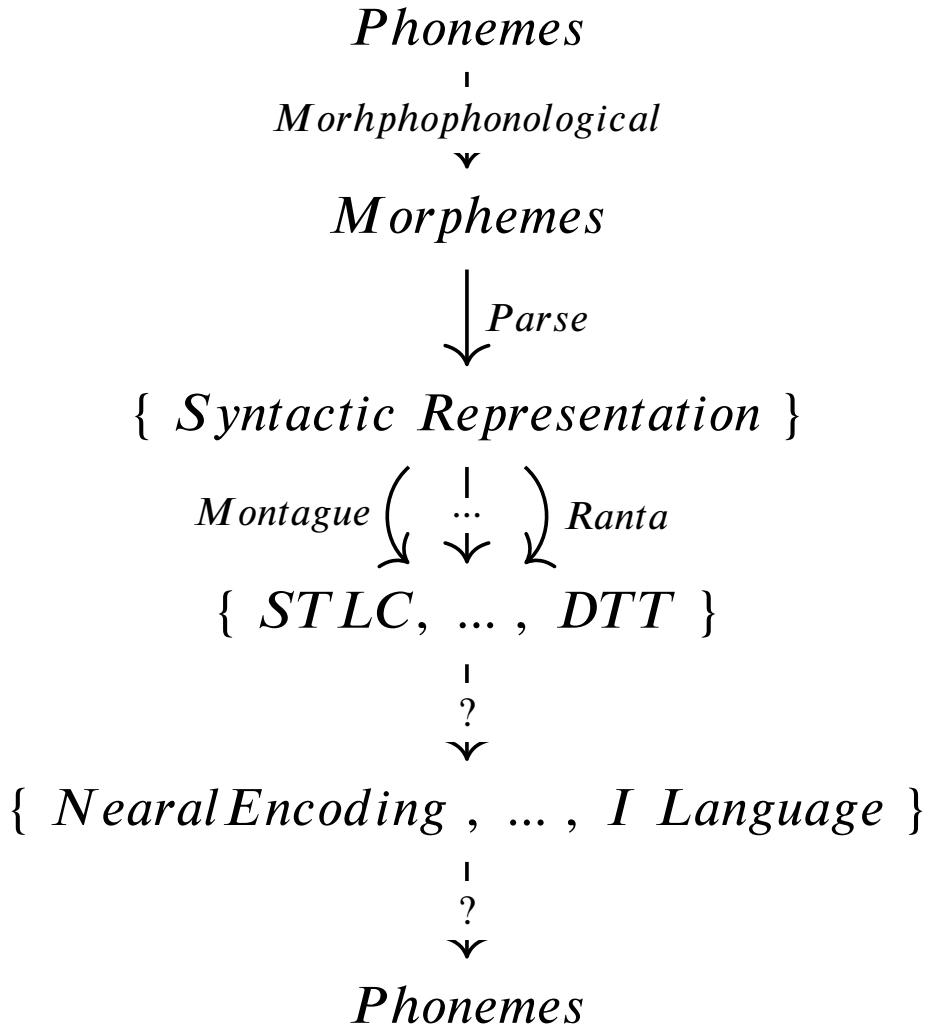
Table of Contents

Abbreviations

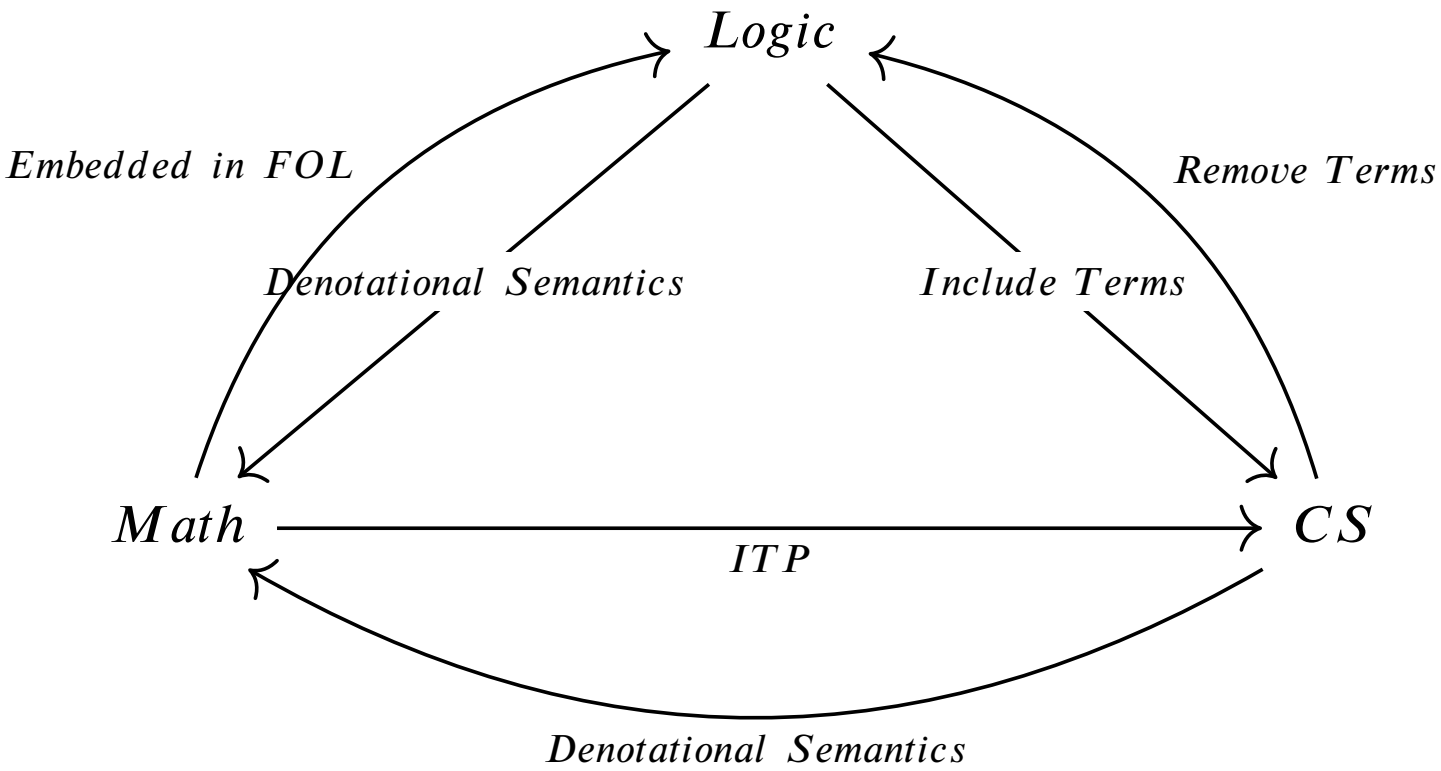
- **TT** : Type Theory
- **MLTT** : Martin-Löf Type Theory
- **MLTT** : Homotopy Type Theory
- **CTT** : Cubical Type Theory
- **NL** : Natural Language
- **PL** : Programming Language
- **GF** : Grammatical Framework
- **PGF** : Portable Grammar Format
- **ITP** : Interactive Theorem Prover
- **FOL** : First Order Logic
- **BNF** : Backus-Naur form
- **CADE** : Conference on Automated Deduction
- **HOL** : Higher Order Logic
- **HIT** : Higher Inductive Type
- **GADT** : Generalized Algebraic Data Type

Abstraction Ladders





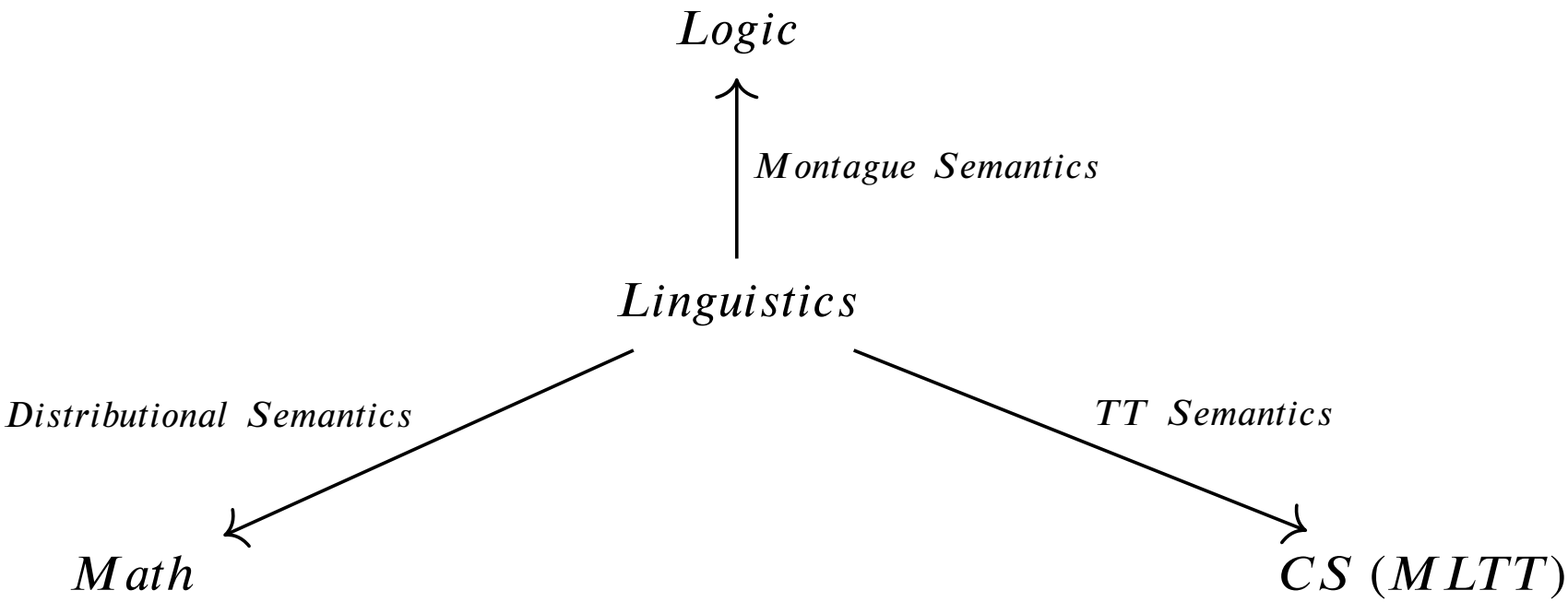
Computational Trinitarianism



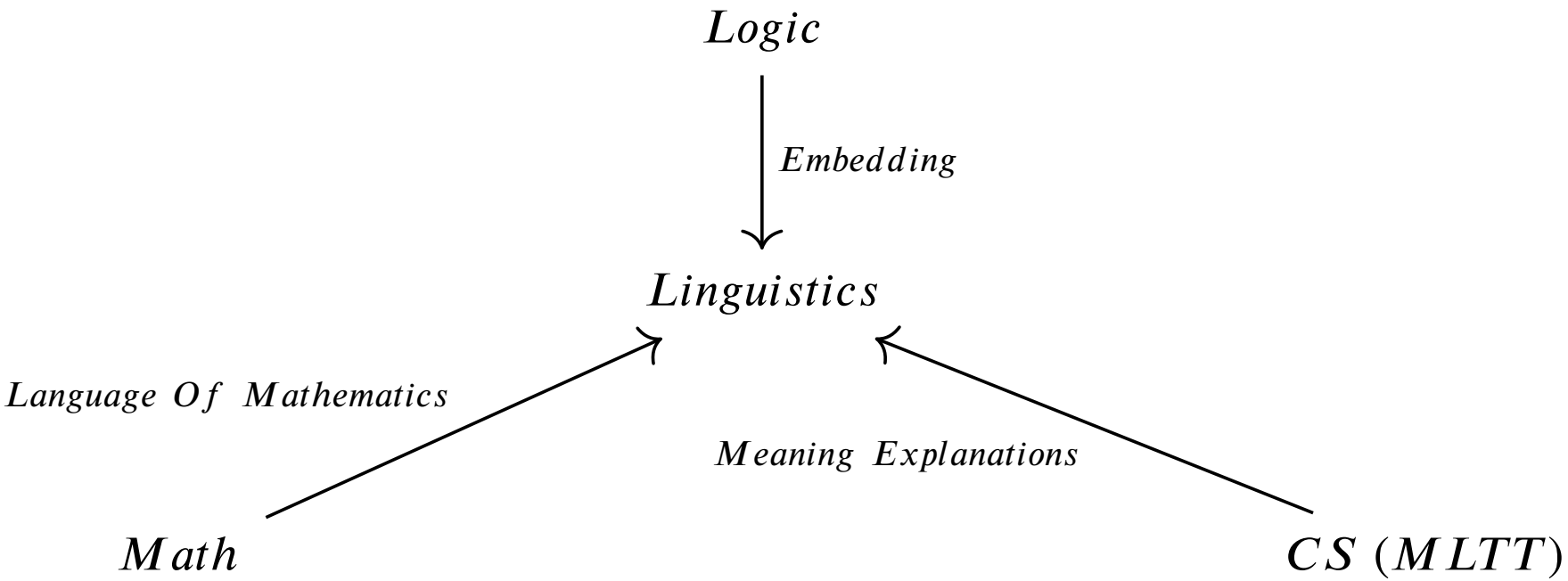
Interpretation of Language

Observation

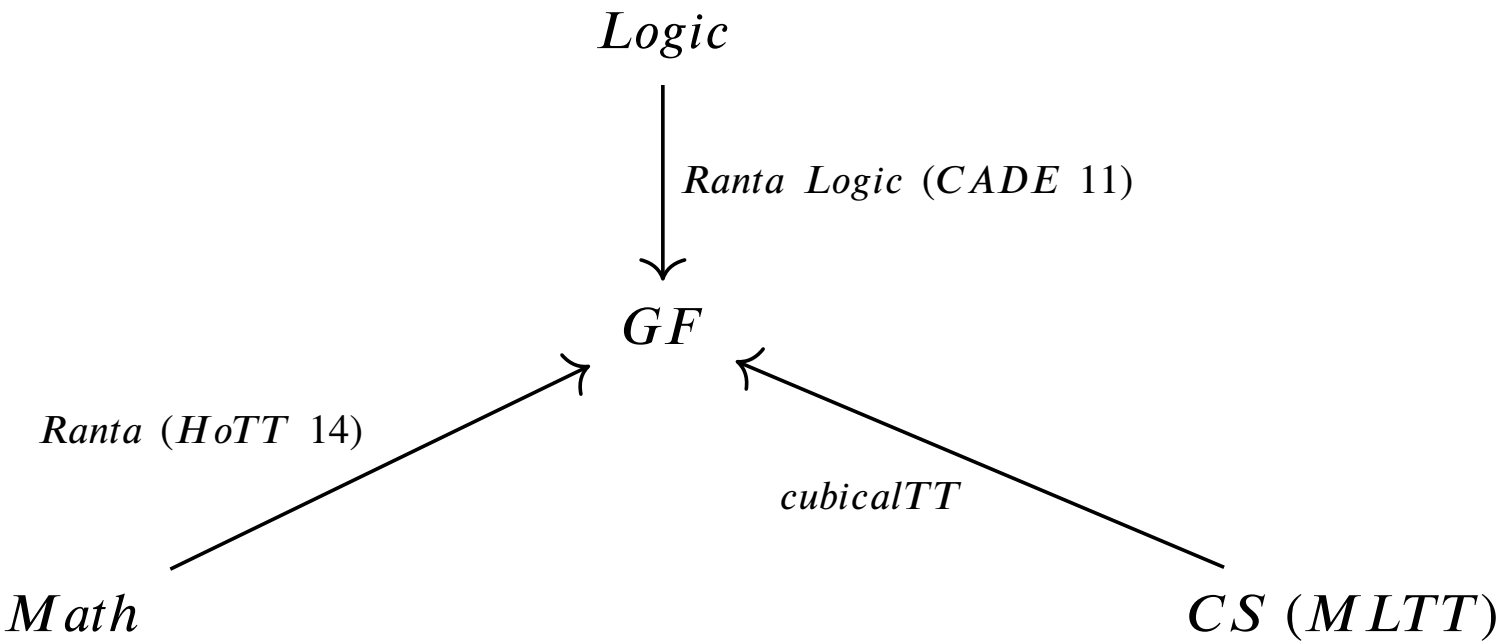
Only semantic interpretations in these domains. One may decide on syntactic, pragmatic, or other paradigms to view this through.



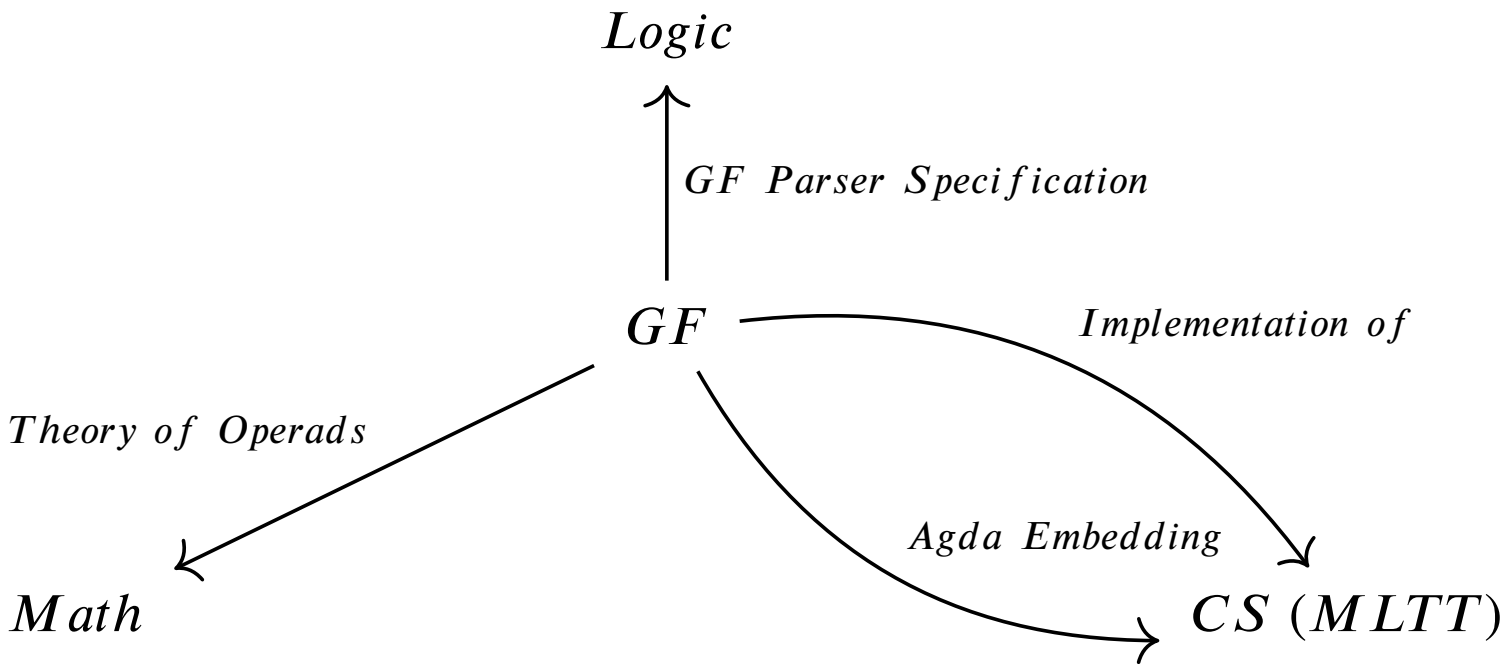
Trinitarian Linguistics



Trinitarian Grammars



Models of GF


Logic
GF Parser Specification
GF
Implementation of
Theory of Operads
Agda Embedding
Math
CS (MLTT)

Remarks

Concrete syntax is, in some sense, where programming meets psychology.

(Robert Harper, OPLSS)

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

Predecessors

- Frege : Formal proof, predicate logic
- Russel : Invents TT to resolve his paradox
- Brouwer : Constructivism
- Heyting, Kolmogorov, Church, Gödel, Kleene, Bishop, ...

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)

Props vs Types

First Order Logic

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

Dependent Type Theory

- Π
- Σ
- $\rightarrow$
- $\times$
- $+$
- $\neg$
- $\top$
- $\bot$
- $\equiv$

Sets vs Types

Sets

- $\mathbb{N}$
- $\mathbb{N} \times \mathbb{N}$
- $\mathbb{N} \to \mathbb{N}$
- $\{x | P(x)\}$
- $\varnothing$
- ?
- $\cup$
- ?

Types

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

Sets

- 1
- $(1, 0)$

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.

TT vs classical FOL

- 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
- perhaps this is possible for other TTs, like those based of HOL

Other issues

- One doesn't define logics and type systems in mathematics (e.g. metamathematics)
- Encoding concepts like rational and real numbers in TT are difficult
- Sets are just one way of encoding mathematics
- Already category theorists and set theorists are at odds. Think small and large categories, higher categories, simplicial, cubical, globular, ... enrichment, etc.
- Intensional TT comes with two distinct notions of equality : judgmental (definitional, computational) and propositional

Donkey Anaphora

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

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

- We judge $\vdash man$:type and $\vdash donkey$:type.
- type really denotes a universe

HoTT

- Homotopy Type Theory is an all out coming to terms with what equality is in type theory
- Equality is perhaps the most confusing detail for mathematicians learning TT
- Propositional equality can be iterated
- given a type A, we can form the homotopy $p =_{x=_A y} q$ with endpoints p and q inhabiting the path space $x =_A y$.
- Univalence axiom : Equivalence is equivalent to equality
- Allows one to admit a topological interpretation of types
- Has led to HITs, where constructors can include equality types.

Interpretations of $t : \tau$

- Set theory : n is an element of $\mathbb{N}$
- Type theory : n is a term of type $\mathbb{N}$
- Homotopy theory : n is a point in the space $\mathbb{N}$
- Category theory : n is an arrow between the object $\mathbb{N}$ and itself
- Logic : n is a proof of the proposition $\mathbb{N}$ –broken

What is Agda?

- Implementation of MLTT
- Logical Framework
- Think a kernel with Π and Σ (think $\forall$ and $\exists$, respectively)
- Interactive proof development environment
- Inductive types, modules, pattern matching, & more
- Large standard library, many other libraries
- Universe Hierarchy (not present classically)
- Tons of experimental stuff
  sized types, coinduction, tactics, etc.

Math Keywords in Agda

Mathematical Declarations

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

```
postulate -- Axiom
  axiom : A

definition : stuff → Set
definition s = definition-body

theorem : T -- Theorem Statement
theorem = proof -- Proof

lemma : L -- Lemma Statement
lemma = proof

example : E -- Example Statement
example = proof
```

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

A proof is what makes a judgment evident

(Per Martin-Löf, On the Meanings of the Logical Constants And the Justifications of the Logical Laws, 1983)

...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)

Comparing Formal and Informal Proofs

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

Specification and Implementation

My Takeaway

An informal proof is a specification and a formal proof is an implementation.

- Historically, we think of semantics preceding
  i.e. the abstract notion of a circle preceded its “algebraic” understanding
- Syntax oriented thinking is very popular in the CS tradition
- Sometimes given an expressive enough type, the programs seem to *write themselves*
- Both syntactic and semantic thinking are necessary in the end, especially for big proofs

Formal Abstracts vision

The Formal Abstracts (FAbstracts) project will establish a formal abstract service that will express the results of mathematical publications in a computer-readable form that captures the semantic content of publications

(The Formal Abstracts Project)

A Smaller Problem

Propositions or theorem statements in natural language 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, Mathematics and language, 2015)

Syntactic Completeness

Given an natural language expression that a mathematician understands, does the GF grammar emit a well-formed and well-typed expression in the target logic or programming language?

- The intended meaning manifests differently for different people - there may be no absolute intention
- Many possible syntaxes beyond just alpha conversion
- The details will have to change depending on the tools and machines available
- Begin with math as developed in an ITP

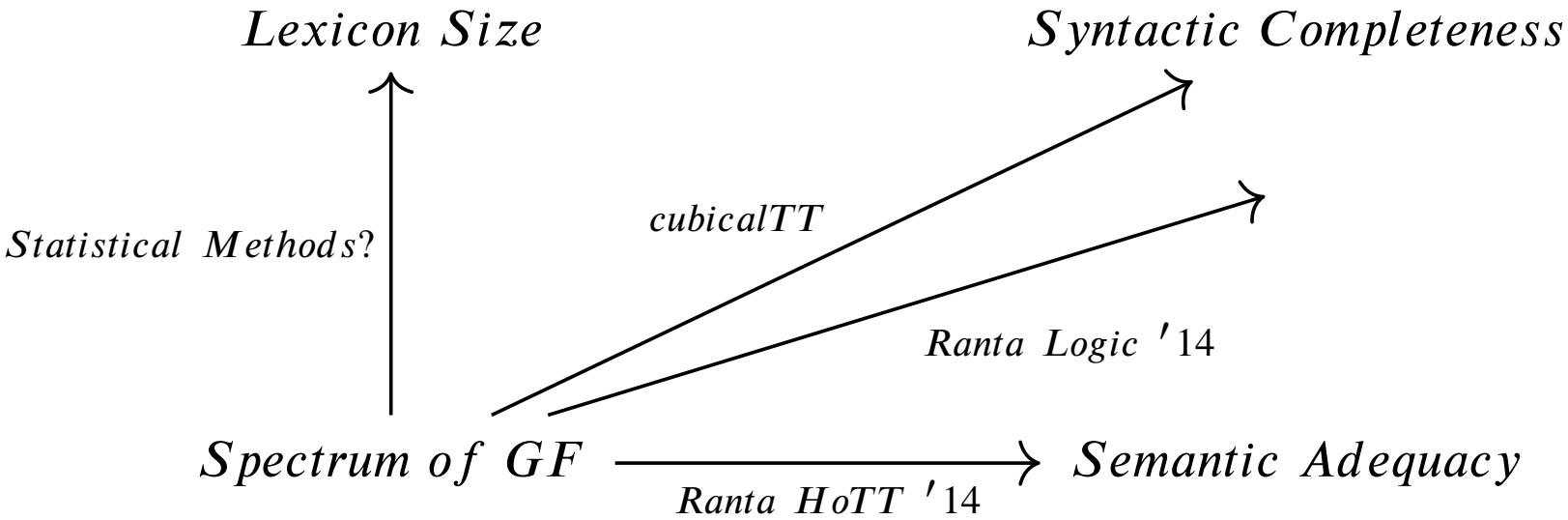
Semantic Adequacy

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 machine proof translation
- Depends on historical developments as our language develops, both internally and more interestingly, as a society
- Ganesalingam calls a related notion *adaptivity*, the phenomenon whereby “the grammar of an individual mathematician changes as definitions are encountered”
- Grammar should just be a starting point for mathematical literacy generally
- Begin with *real mathematics* empirically

The Dimensions of Grammar



Choosing Sides: Concrete vs Abstract

Abstract

- Capture more semantic content from a NL perspective and more "freedom of expression"
- Simpler makes easier to work with in Haskell - really large GADTs after PGF embedding

Concrete

- Simpler semantic space, easier for designing PL for it eliminates ambiguity
- Focus on this may be more feasible for real implementation

As will be seen, a smaller abstract syntax leans towards syntactic completeness, but it comes at the cost of larger, more complex *lincats*.

A Grammar for Propositions

- Ranta, *Translating between Language and Logic : What Is Easy and What Is Difficult*
- Application grammar for logical propositions defined over some mathematical domain (integral arithmetic, euclidean geometry)
- Doesn't cover proofs
- Uses a core and extended set of categories
- Translates between them via PGF extension

Ideas

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

Logic Grammar Properties

Core

- Minimal necessary expressivity
- Syntactically complete
- Ambiguous parses : Catalan explosion with n conjunctions
- Needs to be evaluated to remain legible

Core Example

- $\forall x(Nat(x) \supset Even(x) \vee Odd(x))$
- "for all x, if x is a natural number then x is even or x is odd"

Semantically Inadequate Example

"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	**symbolic**	**verbal**
negation	$\sim P$	*it is not the case that P*
conjunction	$P \,\&\, Q$	*P and Q*
disjunction	$P \vee Q$	*P or Q*
implication	$P \supset Q$	*if P then Q*
universal quantification	$(\forall x)P$	*for all x, P*
existential quantification	$(\exists x)P$	*there exists an x such that P*

Extended Properties

Extended

- Much more expressive
- Semantically adequate
- 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 Example

- $\forall x(Nat(x) \supset Even(x) \vee Odd(x))$
- "every natural number is even or odd"

Extended Syntax

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

Translating to Core

$[\![-]\!] : Extended \to Core$

- Core syntax as a model for extended
- relatively simple in the sense that it is be deterministic
- More or less uses the same logical structure from the "standard view"

Translating to Extended

$[\![-]\!] : Core \to Extended$

- Difficult problem, infeasible at scale
- Obscures formal detail, but is more intuitive

Conversion

- Flattening a list
  $x\ and\ y\ and\ z \mapsto x,\ y\ and\ z$
- Aggregation
  $x\ is\ even\ or\ x\ is\ odd \mapsto x\ is\ even\ or\ odd$
- In-situ quantification
  $\forall\ n \in Nat,\ x\ is\ even\ or\ x\ is\ odd \mapsto every\ Nat\ is\ even\ or$
- Negation
  $it\ is\ not\ that\ case\ that\ x\ is\ even \mapsto is\ not\ even$
- Reflexivitazion
  $x\ is\ equal\ to\ x \mapsto x\ is\ equal\ to\ itself$
- Modification
  $x\ is\ a\ number\ and\ x\ is\ even \mapsto x\ is\ an\ even\ number$