Modeling Formal Languages in Grammatical Framework

On the Grammar of Proof

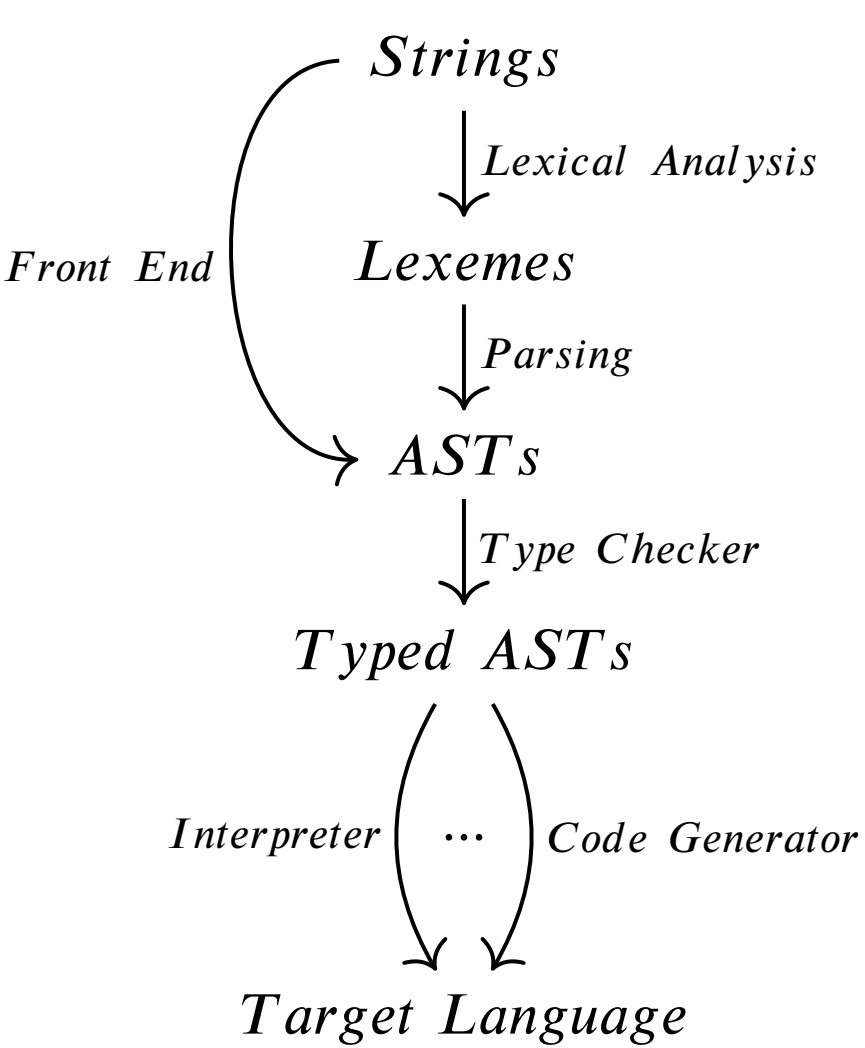
Warrick Macmillan

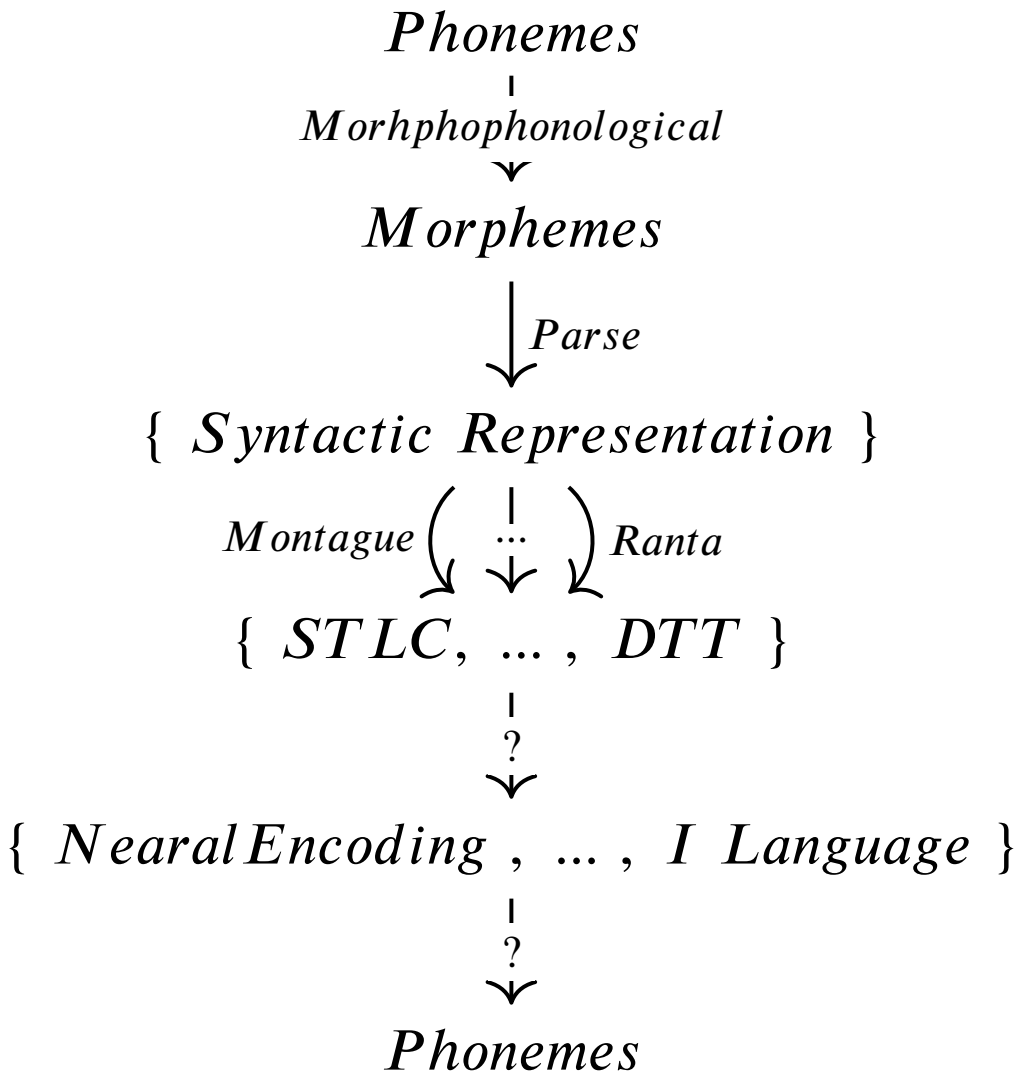
7^{th} August 2021

Table of Contents

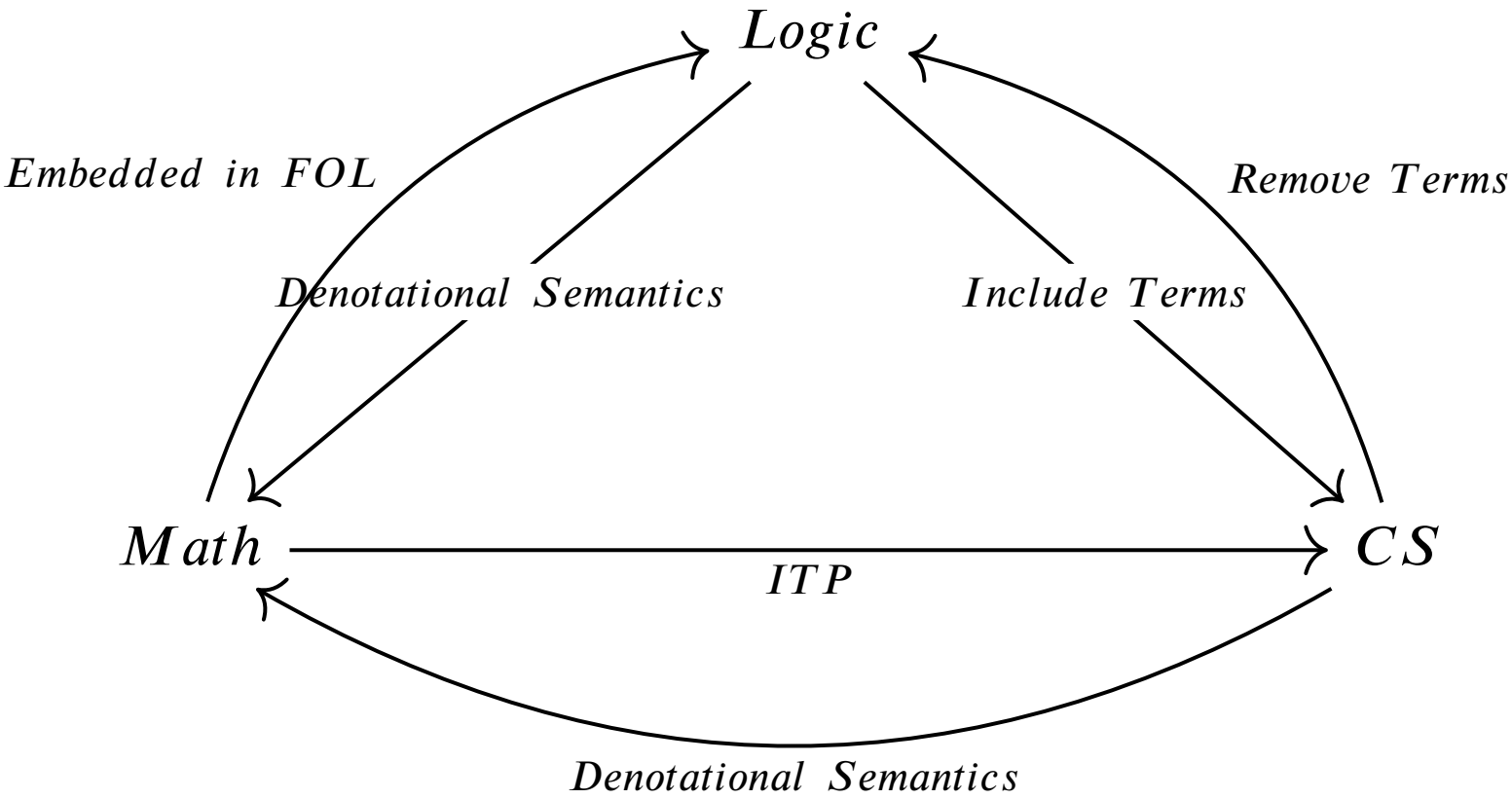
1. Explore abstract relationships between math, CS, Type Theory, and Linguistics
2. Practical and brief intro to MLTT and Agda
3. Grammars elaborating the abstractions above

Abstraction Ladders





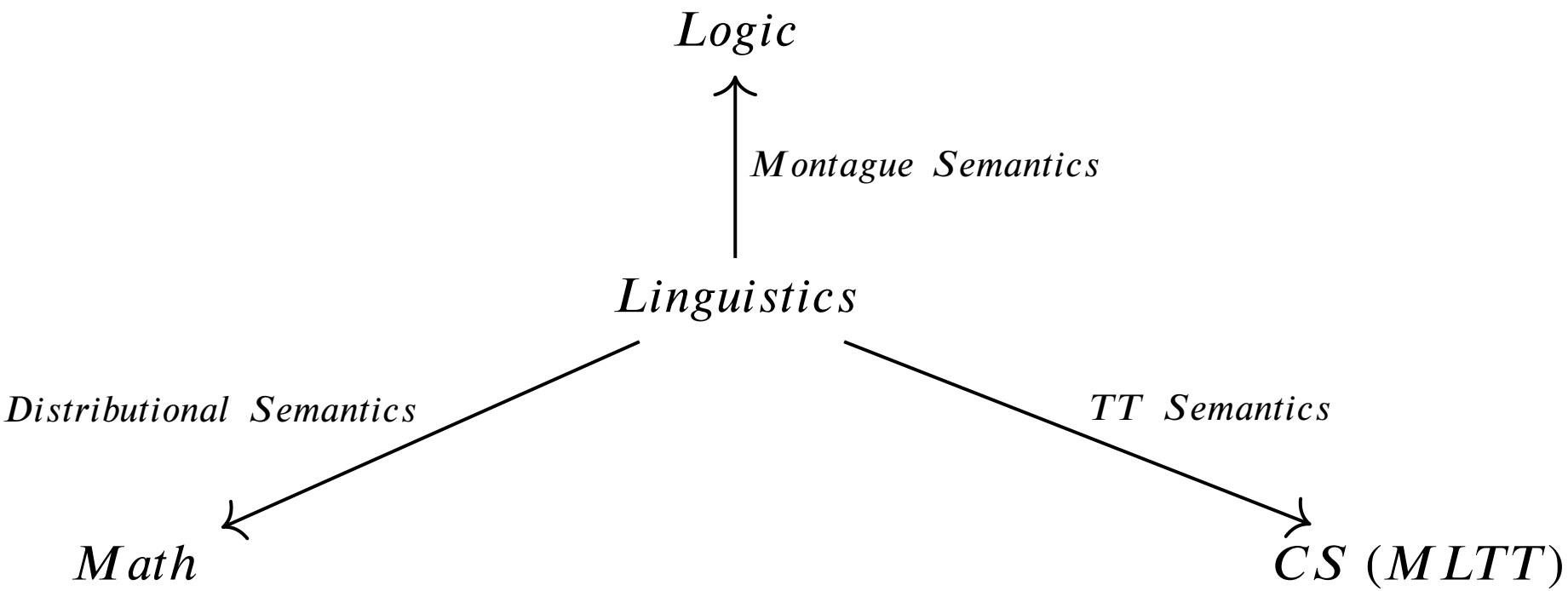
Computational Trinitarianism



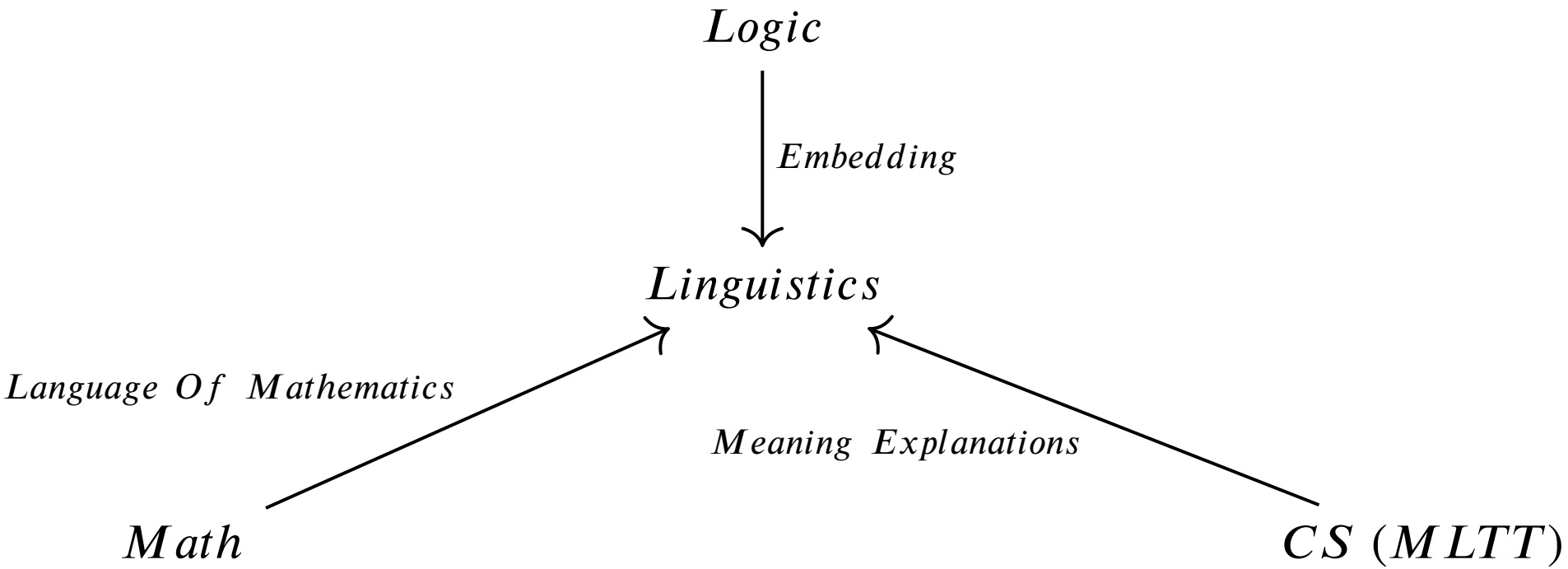
Interpretation Language

Observation 1.1

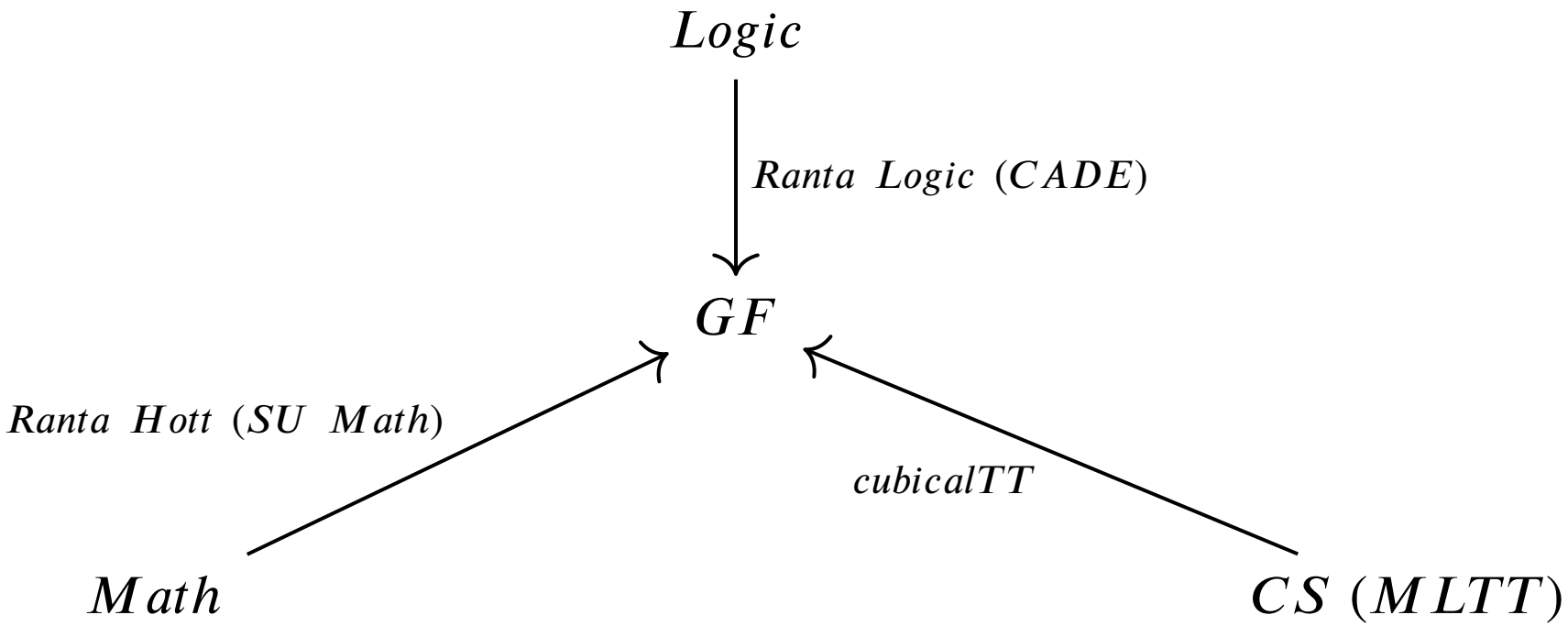
We acknowledge this is only semantic interpretations in these domains. One may decide on syntactic, pragmatic, or other ways in which to treat linguistics via these fields



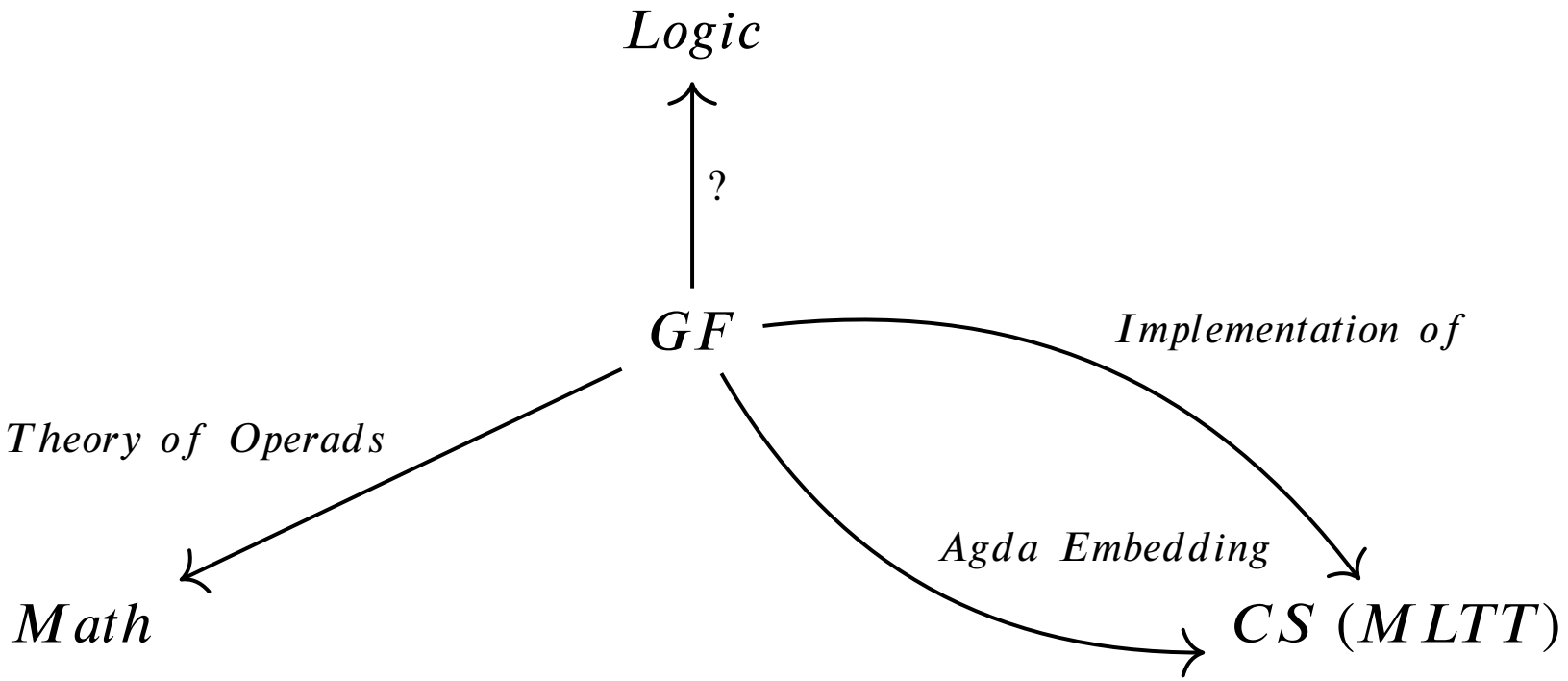
Trinitarian Linguistics



Trinitarian Grammars



Models of GF


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

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)

First Order Logic

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

Dependent Type Theory

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

Sets

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

More Sets

- 1
- $(1, 0)$

Types

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

Programs

- $suc\ \ zero$
- $(suc\ \ zero, zero)$

- already, category theorists and set theorists are at odds, (small and large categories), higher categories, which skeletons of categories are canonical, etc.
- The rules of the types make explicit that they are not equivalent to those of classical FOL (e.g. an existential assertions in type theory require 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 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. metamathemeatics)
- encoding things like rational and real numbers in type theory are incredibly difficult
- Additionally, intensional type theory comes with two distinct notions of equality, definitional (computational) and propositional
- judgments are separate [separate slides?]

Donkey Anaphora

interpret the sentence "every man who owns a donkey beats it" in MLTT via the following judgment :

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

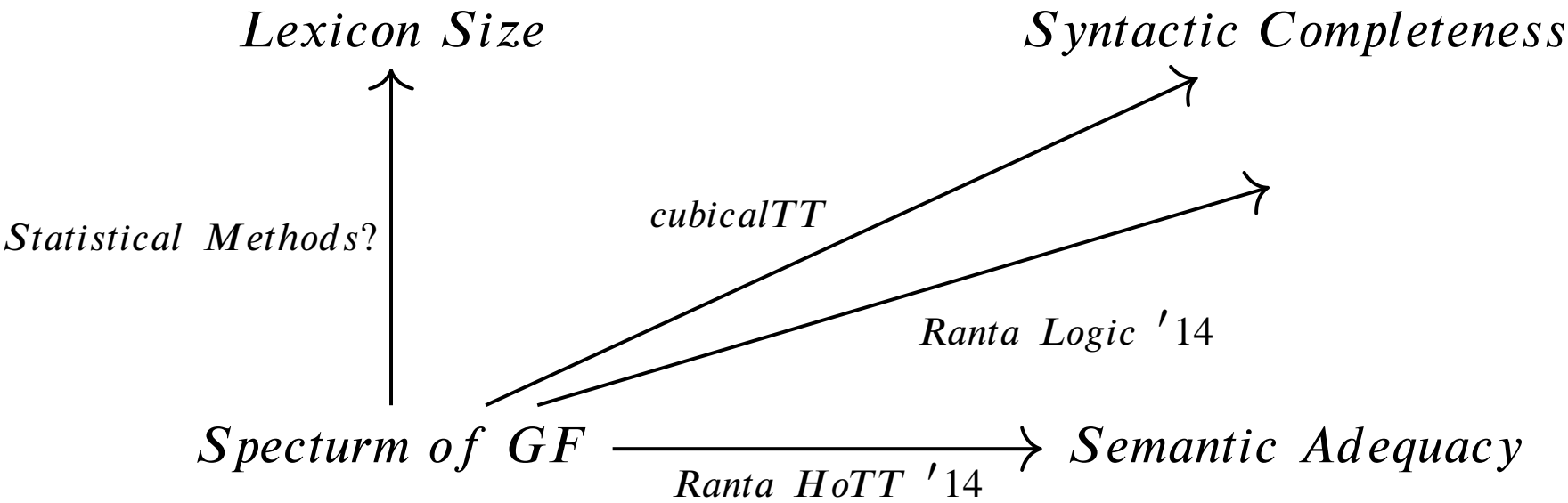
we judge $\vdash man : \mathrm{type}$ and $\vdash donkey : \mathrm{type}$ where type really denotes a universe

What is Agda - Logical Framework - Interactive proof development environment - Inductive Types, Modules, - Implementation of MLTT

Boolean example
(type theoretic syntax)

What is a Proof?

Defintions
Syntctic Completeness Semantic Adequacy


Lexicon Size
Syntactic Completeness
Statistical Methods?
cubicalTT
Ranta Logic '14
Specturm of GF
Semantic Adequacy
Ranta HoTT '14

Overview

1. *Negation* is a contentions notion, in a sense (but we'll not get into this here)
2. Buszkowski added axioms for a kind of negation in categorial grammars
3. Wansing presents a different kind, which leads to motivation of connexive logic

Negative Information

To allow for the expression that "'sleeps John' is an invalid sentence" (it's not just "not valid", it's *invalid*, one could assign it the type "$\neg(s\backslash n)$" There are many nice connections to algebra, and even category theory (Lambek calculus was inspired on that), but we won't be touching upon (no time).

The Negation Normal Form

Observation 3.4

For every type symbol x, x' is in NNF and $\vdash_S x \Leftrightarrow x'$, for $S \in \{\mathbf{NL}^{\neg}, \mathbf{L}^{\neg}\}$.

Definition 3.1: type symbol

1. atomic type symbols $x, y, w, \ldots$ are type symbols;
2. if X and Y are type symbols, also $(X * Y)$ is a type symbol, for $* \in \{\backslash, /, \times\}$
3. if X is a type symbol, also $\neg X$ is a type symbol $(X \neq Y \times Z)$;

Negation Normal Form

Define a function $'$ such that:

$$
\begin{aligned}
x' &= x \quad \text{(x atomic)} \\
(\neg x)' &= \neg x \quad \text{(x atomic)} \\
(\neg\neg X)' &= X' \\
(X * Y)' &= (X' * Y') \\
(\neg(Y/X))' &= ((\neg Y)'/X') \\
(\neg(X\backslash Y))' &= (X'\backslash(\neg Y)')
\end{aligned}
$$

Proof:

A. x is a type symbol $\rightarrow x'$ is its negation normal form;

B. x is a type symbol $\rightarrow \vdash_S x \Leftrightarrow x'$

Proof:

A. x is a type symbol $\rightarrow x'$ is its negation normal form;

B. x is a type symbol $\rightarrow \vdash_S x \Leftrightarrow x'$

A.

Proof is straightforward by induction on the complexity of type symbols.

(note: $\neg(X * Y)$ is not a valid type symbol here).

B. ($\Rightarrow$ *case*)

By induction on the complexity of x:

1. x is atomic: $x' = x$ and by $(id) \vdash x \Rightarrow x'$;

B. ($\Rightarrow$ *case*)

By induction on the complexity of x:

1. x is atomic: $x' = x$ and by $(id) \vdash x \Rightarrow x'$;
2. $x = (y \times w)$: by IH $\vdash y \Leftrightarrow y'$ and $\vdash w \Leftrightarrow w'$

$$\dfrac{\dfrac{y \Rightarrow y' \qquad w \Rightarrow w'}{y, w \Rightarrow (y' \times x')}\,(\rightarrow \times)}{(y \times w) \Rightarrow (y' \times x')}\,(\times \rightarrow)$$

3. $x = (y/w)$: same IH

$$\dfrac{\dfrac{w' \Rightarrow w \qquad y \Rightarrow y'}{(y/w), w' \Rightarrow y'}\,(/ \rightarrow)}{y/w \Rightarrow y'/w'}\,(\rightarrow /)$$

4. $x = (y\backslash w)$: dual.

5 $x = \neg y$: by IH $\vdash y \Leftrightarrow y'$
We can't deduce $\neg y \Leftrightarrow (\neg y)'$ directly. We need to look at y:

5 $x = \neg y$: by IH $\vdash y \Leftrightarrow y'$
We can't deduce $\neg y \Leftrightarrow (\neg y)'$ directly. We need to look at y:

- y atomic then $(\neg y)' = \neg y$ and $\vdash \neg y \Rightarrow \neg y$ by (id);
- $y = \neg w$. We want a proof of $\neg\neg w \Leftrightarrow (\neg\neg w)'$. By IH we know that $\vdash w \Leftrightarrow w'$

$$\dfrac{\dfrac{w \Rightarrow w'}{\neg\neg w \Rightarrow w'}\,(\neg\neg \rightarrow)}{\neg\neg w \Rightarrow \neg\neg w'}\,(\rightarrow \neg\neg)$$

5 $x = \neg y$: by IH $\vdash y \Leftrightarrow y'$
We can't deduce $\neg y \Leftrightarrow (\neg y)'$ directly. We need to look at y:

- y atomic then $(\neg y)' = \neg y$ and $\vdash \neg y \Rightarrow \neg y$ by (id);
- $y = \neg w$. We want a proof of $\neg\neg w \Leftrightarrow (\neg\neg w)'$. By IH we know that $\vdash w \Leftrightarrow w'$

$$\dfrac{\dfrac{w \Rightarrow w'}{\neg\neg w \Rightarrow w'}\,(\neg\neg \rightarrow)}{\neg\neg w \Rightarrow \neg\neg w'}\,(\rightarrow \neg\neg)$$

- $y = w/z$. We want a proof of $\neg(w/z) \Rightarrow ((\neg w')/z')$ since $(\neg(w/z))' = ((\neg w')/z')$ By IH we can assume

$$z \Leftrightarrow z' \qquad \neg w \Leftrightarrow (\neg w)'$$

$$\dfrac{\dfrac{z' \Rightarrow z \qquad \neg w \Rightarrow (\neg w)'}{\neg(w/z), z' \Rightarrow (\neg w)'}\,(\neg/ \rightarrow)}{\neg(w/z) \Rightarrow ((\neg w)'/z')}\,(\rightarrow /)$$

Definition 3.1: type symbol

1. Atomic type symbols $x, y, w, \ldots$ are type symbols;
2. if X and Y are type symbols, also $(X * Y)$ is a type symbol, for $* \in \{\backslash, /, x\}$
3. if X is a type symbol, also $\neg X$ is a type symbol $(X \neq Y \times Z)$;
4. nothing else is a type symbol.

Definition 3.2: categorial entailment

$$\frac{}{x \Rightarrow x}\ (id)$$

$$\frac{x, X \Rightarrow y}{X \Rightarrow (x \backslash y)}\ (\rightarrow \backslash)$$

$$\frac{X \Rightarrow x \qquad Y, y, Y' \Rightarrow z}{Y, X, (x \backslash y), Y' \Rightarrow z}\ (\backslash \rightarrow)$$