Richer type theories for arithmetic universes

Peter LeFanu Lumsdaine
Stockholm University
jww Sina Hazratpour, Steve Vickers

Synthetic mathematics, logic-affine computation and efficient proof systems

CIRM Luminy, September 8 2025

Locoses, arithmetic universes

Definition

- A locos is a lextensive category with parametrised list objects.
- An arithmetic universe is an exact locos
  = a pretopos with parametrised list objects.

Locoses, arithmetic universes

Definition

- A locos is a lextensive category with parametrised list objects.
- An arithmetic universe is an exact locos
  = a pretopos with parametrised list objects.

What? Why?

AU's: motivation and history

Idea of AU's: Proposed by Joyal 1973, for categorical account of Gödel incompleteness — more generally, finitistic mathematics.

AU’s: motivation and history

Idea of AU’s: Proposed by Joyal 1973, for categorical account of Gödel incompleteness — more generally, finitistic mathematics.

Technical crux: Category with minimal structure permitting internal construction of syntax, abstracting Gödel-numbering.

Initially unpublished; taken up in 90’s by Vickers, Maietti, Taylor and others.

AU's: motivation and history

Idea of AU's: Proposed by Joyal 1973, for categorical account of Gödel incompleteness — more generally, finitistic mathematics.

Technical crux: Category with minimal structure permitting internal construction of syntax, abstracting Gödel-numbering.

Initially unpublished; taken up in 90's by Vickers, Maietti, Taylor and others.

Modern definition:

- Locos: list-arithmetic lextensive category
  setting for (unquotiented) syntax
- Arithmetic universe: list-arithmetic pretopos = exact locos
  setting for quotiented syntax = free algebraic structures

Locoses, AU's

Definition (Maietti 2010)

A locos is a category that is:

- lex: all finite limits
- extensive: 0, +, with descent $\mathcal{E}/(A+B) \cong \mathcal{E}/A \times \mathcal{E}/B$
- list-arithemtic: parametrised list objects (including NNO)

An arithmetic universe is moreover:

- exact: regular, & every congruence a kernel pair

Locoses, AU's, and their type theory

Definition (Maietti 2010, 1999, 2005)

A locos is a category that is:

- lex: all finite limits

  ...interprets DTT with 1, Σ, =

- extensive: 0, +, with descent $\mathcal{E}/(A+B) \cong \mathcal{E}/A \times \mathcal{E}/B$

  ...0, +, with large eliminators

- list-arithemtic: parametrised list objects (including NNO)

  ...list types, naturals

An arithmetic universe is moreover:

- exact: regular, & every congruence a kernel pair

  ...effective quotient types

Motivating directions

Two previous projects that brought me to this:

- Rocq formalisation of syntax for general type theories (Bauer, Haselwarter and Lumsdaine 2020)

  Want to base it on natural/minimal type-theoretic primitives for formalising syntax

Motivating directions

Two previous projects that brought me to this:

- Rocq formalisation of syntax for general type theories (Bauer, Haselwarter and Lumsdaine 2020)

  Want to base it on natural/minimal type-theoretic primitives for formalising syntax

- Makkai's categorical proof of projectivity of N in free topos = rule of countable choice for IHOL (jww Forssell, Swan)

  Want minimal categorical setting for syntax / free models of structured categories

Formalisation of syntax

Syntax with contexts, binding:

Formalisation of syntax

Syntax with contexts, binding:

- Finitary inductive family…
- …indexed over scopes/contexts

$$\mathrm{var} : \{n : \mathbb{N}\} \longrightarrow \mathrm{Fin}(n) \longrightarrow \mathrm{Tm}(n)$$
$$\mathrm{lam} : \{n : \mathbb{N}\} \longrightarrow \mathrm{Tm}(n+1) \longrightarrow \mathrm{Tm}(n)$$

Formalisation of syntax

Syntax with contexts, binding:

- Finitary inductive family…
- …indexed over scopes/contexts

$$\begin{aligned} \mathrm{var} &: \{n : \mathrm{N}\} \longrightarrow \mathrm{Fin}(n) \longrightarrow \mathrm{Tm}(n) \\ \mathrm{lam} &: \{n : \mathrm{N}\} \longrightarrow \mathrm{Tm}(n+1) \longrightarrow \mathrm{Tm}(n) \end{aligned}$$

- N used just as a universe: never depend on equality on it, etc.
- Replacing with a larger (suitably-closed) universe gives infinitary syntax.

Free models in AU's

What should "setting for syntax" mean?

Goal

AU's should admit free models of finitely presented essentially algebraic theories

Free models in AU's

What should "setting for syntax" mean?

Goal

AU's should admit free models of finitely presented essentially algebraic theories

EAT's: theories like the theory of categories,
i.e. multi-sorted algebraic; domains of operations defined equationally from earlier operations.

Why EAT's? Sufficient to

- build free internal AU (for Gödel incompleteness)
- present + handle syntax of most other logics (FOL, DTT, ...)

Free models of EAT's in AU's?

- Stated in Morrison 1996, but part of proof conjectural
- Many non-trivial special cases, in e.g. Maietti 1999
- …showing sufficient techniques to cover arbitrary examples
- Generally known/believed in folklore
- …but no full treatment in literature, to my knowledge

Free models of EAT's in AU's?

- Stated in Morrison 1996, but part of proof conjectural
- Many non-trivial special cases, in e.g. Maietti 1999
- …showing sufficient techniques to cover arbitrary examples
- Generally known/believed in folklore
- …but no full treatment in literature, to my knowledge

Why not?

Presentations of EAT's

- Algebraic: domains of operations defined equationally from theory so far

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does “finitely presented” mean?

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

  Syntax/free model construction many-staged.

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

  Syntax/free model construction many-staged.

- FOL fragment $=, \wedge, \exists(!)$

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

  Syntax/free model construction many-staged.

- FOL fragment $=, \wedge, \exists(!)$

  Stratification, again.

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

  Syntax/free model construction many-staged.

- FOL fragment $=, \wedge, \exists(!)$

  Stratification, again.

- Partial Horn logic/quasi-equational theories

Presentations of EAT's and their problems

- Algebraic: domains of operations defined equationally from theory so far

  Problem: requires stratification!

- Dependently algebraic (GAT): sorts dependent over earlier contexts

  Again, stratification.

- Finite limit categories

  What does "finitely presented" mean?

- Finite limit sketches

  Syntax/free model construction many-staged.

- FOL fragment $=, \wedge, \exists(!)$

  Stratification, again.

- Partial Horn logic/quasi-equational theories

  Rather more tractable…

Partial Horn logic

Partial Horn logic (Palmgren and Vickers 2007): a logic of partial algebraic theories.

Term syntax: ordinary multi-sorted finitary algebraic,
but operations understood as partial

Can include atomic predicates; for now omit these, quasi-equational

Axioms: finite conjunction of atomics entail an atomic

$$t_1 = s_1, \ldots, t_n = s_n \vdash_{\vec{x}} t = s$$

Reflexivity represents definedness: $t = t$, "t defined".

Partial Horn logic

Partial Horn logic (Palmgren and Vickers 2007): a logic of partial algebraic theories.

Term syntax: ordinary multi-sorted finitary algebraic,
but operations understood as partial

Can include atomic predicates; for now omit these, quasi-equational

Axioms: finite conjunction of atomics entail an atomic

$$t_1 = s_1, \ldots, t_n = s_n \vdash_{\vec{x}} t = s$$

Reflexivity represents definedness: $t = t$, "t defined".

Partial Horn logic

Partial Horn logic (Palmgren and Vickers 2007): a logic of partial algebraic theories.

Term syntax: ordinary multi-sorted finitary algebraic,
but operations understood as partial

Can include atomic predicates; for now omit these, quasi-equational

Axioms: finite conjunction of atomics entail an atomic

$$t_1 = s_1, \ldots, t_n = s_n \vdash_{\vec{x}} t = s$$

Reflexivity represents definedness: $t = t$, "t defined".

Sufficent to express EAT's, finitely presented if they are in any other sense.

Syntax/free models of quasi-equational theories

(Palmgren and Vickers 2007) construct free models, adjunctions, etc.,

Syntax/free models of quasi-equational theories

(Palmgren and Vickers 2007) construct free models, adjunctions, etc.,
working in non-formalised constructive, predicative metatheory.

Syntax/free models of quasi-equational theories

(Palmgren and Vickers 2007) construct free models, adjunctions, etc., working in non-formalised constructive, predicative metatheory.

They note: should internalise to predicative toposes, i.e. stratified pseudotoposes (Moerdijk and Palmgren 2002).

In fact, entirely finitary; so should work in AU's too.

Syntax/free models of quasi-equational theories

(Palmgren and Vickers 2007) construct free models, adjunctions, etc.,
working in non-formalised constructive, predicative metatheory.

They note: should internalise to predicative toposes, i.e. stratified pseudotoposes (Moerdijk and Palmgren 2002).

In fact, entirely finitary; so should work in AU's too.

For a thorough treatment, simpler than other presentations:
syntax 2-staged only,

- simple algebraic term syntax;
- derivations in equational logic over this

Inductive types

W-types: types of trees; (stably) initial algebras for polynomial endofuntors.

Inductive types

W-types: types of trees; (stably) initial algebras for polynomial endofuntors.

Inductive types in DTT: general scheme of declarations, convenient for programming. Typical example: simply-algebraic syntax.

Inductive types

W-types: types of trees; (stably) initial algebras for polynomial endofuntors.

Inductive types in DTT: general scheme of declarations, convenient for programming. Typical example: simply-algebraic syntax.

Indexed inductive types/W-types: mild generalisation with more dependency. Typical example: derivations in a logic.

Inductive types

W-types: types of trees; (stably) initial algebras for polynomial endofuntors.

Inductive types in DTT: general scheme of declarations, convenient for programming. Typical example: simply-algebraic syntax.

Indexed inductive types/W-types: mild generalisation with more dependency. Typical example: derivations in a logic.

Theorem (Dybjer 1997; Maietti and Sabelli 2023)

(Assuming standard basic types, some extensionality.)
CIC-style indexed-inductive types can be constructed from W-types.

Inductive types

W-types: types of trees; (stably) initial algebras for polynomial endofuntors.

Inductive types in DTT: general scheme of declarations, convenient for programming. Typical example: simply-algebraic syntax.

Indexed inductive types/W-types: mild generalisation with more dependency. Typical example: derivations in a logic.

Theorem (Dybjer 1997; Maietti and Sabelli 2023)

(Assuming standard basic types, some extensionality.) CIC-style indexed-inductive types can be constructed from W-types.

Building up in locos/AU

Maietti: Locos interprets 1, Σ, 0, + (with large elims), =, N, List. [AU: also truncation, effective quotients.]

Building up in locos/AU

Maietti: Locos interprets 1, Σ, 0, $+$ (with large elims), $=$, N, List. [AU: also truncation, effective quotients.]

$\rightsquigarrow$ N as universe of finite types, exponentiable, with funext [AU: admitting finite choice, preserving quotients]

Building up in locos/AU

Maietti: Locos interprets 1, Σ, 0, + (with large elims), =, N, List. [AU: also truncation, effective quotients.]

$\rightsquigarrow$ N as universe of finite types, exponentiable, with funext [AU: admitting finite choice, preserving quotients]

$\rightsquigarrow$ (internally) finitary W-types

Building up in locos/AU

Maietti: Locos interprets $1, \Sigma, 0, +$ (with large elims), $=$, N, List. [AU: also truncation, effective quotients.]

$\rightsquigarrow$ N as universe of finite types, exponentiable, with funext [AU: admitting finite choice, preserving quotients]

$\rightsquigarrow$ (internally) finitary W-types

$\rightsquigarrow$ finitary inductive, indexed-inductive types

Building up in locos/AU

Maietti: Locos interprets 1, Σ, 0, $+$ (with large elims), $=$, N, List. [AU: also truncation, effective quotients.]

$\rightsquigarrow$ N as universe of finite types, exponentiable, with funext [AU: admitting finite choice, preserving quotients]

$\rightsquigarrow$ (internally) finitary W-types

$\rightsquigarrow$ finitary inductive, indexed-inductive types

$\rightsquigarrow$ syntax of partial Horn logic

Building up in locos/AU

Maietti: Locos interprets 1, Σ, 0, + (with large elims), =, N, List. [AU: also truncation, effective quotients.]

$\rightsquigarrow$ N as universe of finite types, exponentiable, with funext [AU: admitting finite choice, preserving quotients]

$\rightsquigarrow$ (internally) finitary W-types

$\rightsquigarrow$ finitary inductive, indexed-inductive types

$\rightsquigarrow$ syntax of partial Horn logic

$\rightsquigarrow$ [AU: free models of EAT's]

Type theory for finitistic mathematics

- 1, Σ; $=$ with reflection/K
- 0, $+$, with large eliminators; N, List
- [optional: effective quotients]
- universe F, closed under 1, Σ, 0, $+$, $=$
- dependent products over F-types, extensional, [commuting w. quotients]
- F-ary inductive types (W-types, or CIC-style scheme)

---

[1] Thanks to Ulrik Buchholtz for bringing this to my attention after the talk!

Type theory for finitistic mathematics

- $1, \Sigma; =$ with reflection/K
- $0, +,$ with large eliminators; N, List
- [optional: effective quotients]
- universe F, closed under $1, \Sigma, 0, +, =$
- dependent products over F-types, extensional, [commuting w. quotients]
- F-ary inductive types (W-types, or CIC-style scheme)

Theorem

This is conservative over Maietti's type theory for AU's. □

[1] Thanks to Ulrik Buchholtz for bringing this to my attention after the talk!

Type theory for finitistic mathematics

- $1, \Sigma; =$ with reflection/K
- $0, +,$ with large eliminators; N, List
- [optional: effective quotients]
- universe F, closed under $1, \Sigma, 0, +, =$
- dependent products over F-types, extensional, [commuting w. quotients]
- F-ary inductive types (W-types, or CIC-style scheme)

Theorem

This is conservative over Maietti's type theory for AU's. □

Closely comparable: Calculus of Primitive Recursive Constructions (Patey and Herbelin 2014)[1], Primitive Recursive Type Theory (Buchholtz and Branitz 2024).

[1] Thanks to Ulrik Buchholtz for bringing this to my attention after the talk!

Type theory for finitistic mathematics

- $1, \Sigma; =$ with reflection/K
- $0, +,$ with large eliminators; N, List
- [optional: effective quotients]
- universe F, closed under $1, \Sigma, 0, +, =$
- dependent products over F-types, extensional, [commuting w. quotients]
- F-ary inductive types (W-types, or CIC-style scheme)

Theorem

This is conservative over Maietti's type theory for AU's. $\square$

Closely comparable: Calculus of Primitive Recursive Constructions (Patey and Herbelin 2014)[1], Primitive Recursive Type Theory (Buchholtz and Branitz 2024).

[1] Thanks to Ulrik Buchholtz for bringing this to my attention after the talk!

Formalisation: logical framework

Formalisation in Rocq (WIP with Sina Hazratpour): How to avoid over-powered type theory?

Formalisation: logical framework

Formalisation in Rocq (WIP with Sina Hazratpour): How to avoid over-powered type theory?

- Use Rocq's type theory as logical framework
- Posit type-class of *small/internal* types for types of finitistic type theory
- Eliminators of internal inductives have type-class constraints

Formalisation: logical framework

Formalisation in Rocq (WIP with Sina Hazratpour): How to avoid over-powered type theory?

- Use Rocq's type theory as logical framework
- Posit type-class of *small/internal* types for types of finitistic type theory
- Eliminators of internal inductives have type-class constraints
- Should be conservative over finitistic type theory, by established LF (pre)sheaf methods (WIP)

Formalisation: logical framework

Formalisation in Rocq (WIP with Sina Hazratpour): How to avoid over-powered type theory?

- Use Rocq's type theory as logical framework
- Posit type-class of *small/internal* types for types of finitistic type theory
- Eliminators of internal inductives have type-class constraints
- Should be conservative over finitistic type theory, by established LF (pre)sheaf methods (WIP)

Is this synthetic mathematics now?

Formalisation: logical framework

Formalisation in Rocq (WIP with Sina Hazratpour): How to avoid over-powered type theory?

- Use Rocq's type theory as logical framework
- Posit type-class of *small/internal* types for types of finitistic type theory
- Eliminators of internal inductives have type-class constraints
- Should be conservative over finitistic type theory, by established LF (pre)sheaf methods (WIP)

Is this synthetic mathematics now?

Would Hilbert like it?

Summary

- Locoses: home of syntax
- Arithmetic universes: home of quotiented syntax = free models for EAT's
- Quasi-equational presentation of EAT's: requires just two stages of indexed-inductives for syntax
- Can build up to finitary indexed-inductives from locos/AU primitives, via finitary W-types (Rocq formalisation in progress)
- Abstract the resulting language as a rich dependent type theory for finitistic mathematics

Literature

- Alan Morrison (1996). 'Reasoning in Arithmetic Universes'. MA thesis. URL: https://sjvickers.github.io/MorrisonAUs.pdf
- Maria Emilia Maietti (1999). *The typed calculus of arithmetic universes.* Tech. report, U. Birmingham, CSR-99-14
- Maria Emilia Maietti (2010). 'Joyal's arithmetic universe as list-arithmetic pretopos'. URL: http://www.tac.mta.ca/tac/volumes/24/3/24-03abs.html
- Erik Palmgren and Steven J. Vickers (2007). 'Partial horn logic and Cartesian categories'. DOI: 10.1016/j.apal.2006.10.001
- Peter Dybjer (1997). 'Representing inductively defined sets by wellorderings in Martin-Löf's type theory'.
- Maria Emilia Maietti and Pietro Sabelli (Nov. 2023). 'A topological counterpart of well-founded trees in dependent type theory'. DOI: 10.46298/entics.11755. arXiv: 2308.08404
- Ludovic Patey (2014). 'A calculus of primitive recursive constructions'. Slides from talk at IHP 2014; joint work with Hugo Herbelin. URL: https://ludovicpatey.com/media/talks/cprc.pdf
- Ulrik Buchholtz and Johannes Schipp von Branitz (2024). *Primitive Recursive Dependent Type Theory.* Preprint. arXiv: 2404.01011