NatFoM 2023 and LFPNML 2023

Little Theories

A Method for Organizing Mathematical Knowledge Illustrated Using Alonzo, a Practice-Oriented Version of Simple Type Theory

William M. Farmer

wmfarmer@mcmaster.ca

Department of Computing and Software
McMaster University

7 September 2023

Outline

1. Background
   a. Philosophy.
   b. Alonzo.
   c. Little theories method.
2. A formalization of monoid theory in Alonzo.
3. Final remarks.

The full formalization is presented in the forthcoming paper Monoid Theory in Alonzo [FZ23].

Part 1

Background: Philosophy

What is Mathematics?

- Mathematics is a process for understanding the mathematical aspects of the world.
- Here is how it works:
  1. A mathematical model consisting of objects, concepts, and facts is created to describe a mathematical phenomenon exhibited in the world.
  2. The model is explored in various ways to discover new objects, concepts, and facts related to the model.
  3. This enriched model then provides a deeper understanding of the mathematical phenomenon being modeled.
- Example: A computer internet modeled as a bipartite graph of hosts and physical networks.
- The building blocks for mathematical models are mathematical structures.

What is a Mathematical Structure?

- A first-order mathematical structure is a nonempty set D of values plus a set of distinguished elements, functions, and relations over D. Example: $(\mathbb{N}, 0, +, \leq)$.
- A general mathematical structure (structure for short) is a pair $S = (\mathcal{D}, \mathcal{A})$ where:
  1. $\mathcal{D}$ is a nonempty finite set of base domains that are nonempty sets of values.
  2. $\mathcal{A}$ is a set of distinguished values that are members of the domains in $\{\mathbb{B}\} \cup \mathcal{D}$ or domains constructed from these domains by the function space, power set, Cartesian product, and Kleene star operations. $\mathbb{B} = \{\mathrm{T}, \mathrm{F}\}$.
- Example: A monoid $(\{m\}, \{\cdot, e\})$, where m is a nonempty set, $\cdot : (m \times m) \rightarrow m$ is an associative function, and $e \in m$ is an identity element with respect to $\cdot$, is a structure.

What is Mathematical Knowledge?

- Mathematical knowledge is knowledge about the mathematics process, that is, knowledge about the creation and exploration of mathematical models.
- Since structures are the building blocks of mathematical models, the core of mathematical knowledge is knowledge about structures, their components, and their relationships with each other.
- Formal mathematical knowledge is mathematical knowledge expressed in a formal logic.

What is a Formal Logic?

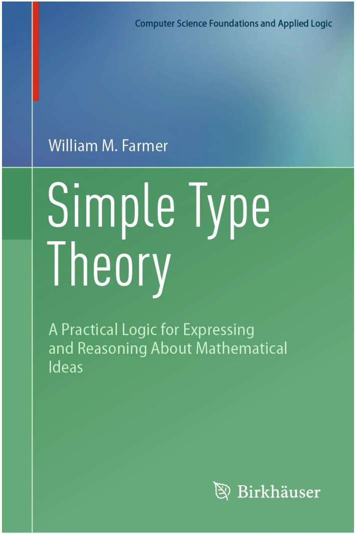
- We define a formal logic as a family of formal languages with:
  1. A precise common syntax.
  2. A precise common semantics with a notion of logical consequence.
  3. A formal proof system for proving that a statement is a logical consequence of a set of statements.
- Examples: first-order logic, set theory, simple type theory, dependent type theory.

Part 1

Background: Alonzo

Formal Mathematics for the Masses

- The 50-year-old campaign to transform traditional mathematical practice into a formal discipline has been both a great success and a great failure.
- At NatFoM 2021 I have proposed an alternative approach to formal mathematics that:
  1. Is fully formal except proofs are written in a traditional (informal) style.
  2. Emphasizes the communication of mathematical ideas instead of the formal certification of mathematical results.
- First step: Develop a formal logic suitable for practical use with or without software support.
- First step is done: We have developed a logic called Alonzo and presented it in a textbook Simple Type Theory [Fa23].
- Next steps: Demonstrate this alternative approach using Alonzo and develop software to support it.


Computer Science Foundations and Applied Logic
William M. Farmer
Simple Type Theory
A Practical Logic for Expressing and Reasoning About Mathematical Ideas
Birkhäuser

Alonzo: Overview

- Alonzo is a version of simple type theory [Fa08] that is based on Church's type theory (CTT) [Ch40].
- Has a simple syntax with two kinds of notation.
- Admits partial functions and undefined expressions.
- Employs two semantics, one for math and one for logic.
- Has a simple and elegant proof system derived from Peter Andrews' proof system for $\mathcal{Q}_0$ [An02].
- Equipped with theories and theory morphisms, the tools needed for building libraries of mathematical knowledge.

Alonzo: Syntax

- Alonzo has two kinds of syntactic entities:
  1. Types that denote nonempty sets of values.
  2. Expressions that either denote values (when they are defined) or denote nothing (when they are undefined).
- There are two notations for types and expressions:
  1. A formal notation, an "internal" syntax for machines.
  2. A compact notation, an "external" syntax for humans that resembles the notation found in mathematical practice.

Alonzo: Types

- A type of Alonzo (denoted by $\alpha, \beta, \ldots$) is a string of symbols defined inductively by:

  T1. Type of truth values: BoolTy is a type.
  T2. Base type: BaseTy(**a**) is a type where **a** is any base type symbol.
  T3. Function type: FunTy(α, β) is a type.
  T4. Product type: ProdTy(α, β) is a type.

- A type is presented in the formal notation when it is written as a string according to this definition.

Alonzo: Expressions

- An expression of type α of Alonzo (denoted by $\mathbf{A}_\alpha, \mathbf{B}_\alpha, \ldots$) is a string of symbols defined inductively by:
  - E1. Variable: $\mathsf{Var}(\mathbf{x}, \alpha)$ is an expression of type α where $\mathbf{x}$ is any variable symbol.
  - E2. Constant: $\mathsf{Con}(\mathbf{c}, \alpha)$ is an expression of type α where $\mathbf{c}$ is any constant symbol.
  - E3. Equality: $\mathsf{Eq}(\mathbf{A}_\alpha, \mathbf{B}_\alpha)$ is an expression of type BoolTy.
  - E4. Function application: $\mathsf{FunApp}(\mathbf{F}_{\alpha \rightarrow \beta}, \mathbf{A}_\alpha)$ is an expression of type β.
  - E5. Function abstraction: $\mathsf{FunAbs}(\mathsf{Var}(\mathbf{x}, \alpha), \mathbf{B}_\beta)$ is an expression of type $\mathsf{FunTy}(\alpha, \beta)$.
  - E6. Definite description: $\mathsf{DefDes}(\mathsf{Var}(\mathbf{x}, \alpha), \mathbf{A}_{\mathsf{BoolTy}})$ is an expression of type α where $\alpha \neq \mathsf{BoolTy}$.
  - E7. Ordered pair: $\mathsf{OrdPair}(\mathbf{A}_\alpha, \mathbf{B}_\beta)$ is an expression of type $\mathsf{ProdTy}(\alpha, \beta)$.
- An expression is presented in the formal notation when it is written as a string according to this definition.

Alonzo: Compact Notation

- The compact notation for types and expressions is introduced in Simple Type Theory [Fa23] by:
  - 131 notational definitions.
  - 13 notational conventions.
- A notational definition, with the form *A* stands for *B*, is for:
  - Introducing a standard mathematical notation.
  - Defining a useful operator, binder, or abbreviation.
  - Defining a notation in which a variable symbol is bound to a set-valued expression (called quasitype) instead of to a type.
- A notational convention is for simplifying notation, e.g., by:
  - Dropping matching parentheses when meaning is not lost.
  - Dropping types from variables and constants when meaning is not lost.
  - Condensing blocks of quantifiers.

Compact Notation for Types and Expressions

- Notational definitions for types:

o	stands for	BoolTy.
$\mathbf{a}$	stands for	BaseTy($\mathbf{a}$).
$(\alpha \rightarrow \beta)$	stands for	FunTy(α, β).
$(\alpha \times \beta)$	stands for	ProdTy(α, β).

- Notational definitions for expressions:

$(\mathbf{x} : \alpha)$	stands for	Var($\mathbf{x}, \alpha$).
$\mathbf{c}_\alpha$	stands for	Con($\mathbf{c}, \alpha$).
$(\mathbf{A}_\alpha = \mathbf{B}_\alpha)$	stands for	Eq($\mathbf{A}_\alpha, \mathbf{B}_\alpha$).
$(\mathbf{F}_{\alpha \rightarrow \beta}\, \mathbf{A}_\alpha)$	stands for	FunApp($\mathbf{F}_{\alpha \rightarrow \beta}, \mathbf{A}_\alpha$).
$(\lambda\, \mathbf{x} : \alpha \,.\, \mathbf{B}_\beta)$	stands for	FunAbs(Var($\mathbf{x}, \alpha$), $\mathbf{B}_\beta$).
$(\mathrm{I}\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$	stands for	DefDes(Var($\mathbf{x}, \alpha$), $\mathbf{A}_o$).
$(\mathbf{A}_\alpha, \mathbf{B}_\beta)$	stands for	OrdPair($\mathbf{A}_\alpha, \mathbf{B}_\beta$).

Parametric Polymorphism

- Alonzo has no parametric polymorphism at the object level.
  - Alonzo does not have type variables.
  - All constants have a fixed type.
- Alonzo has parametric polymorphism at the meta level.
  - Parametric pseudoconstants are defined by notational definitions.
  - Facts about parametric pseudoconstants are proved in "little theories" and then transported to other contexts as needed.
- Alonzo is not a polymorphic logic, but it supports polymorphic reasoning!

Notational Definitions for Boolean Operators

T_o	stands for	$(\lambda\, x : o\,.\, x) = (\lambda\, x : o\,.\, x)$.
F_o	stands for	$(\lambda\, x : o\,.\, T_o) = (\lambda\, x : o\,.\, x)$.
$\wedge_{o \to o \to o}$	stands for	$\lambda\, x : o\,.\, \lambda\, y : o\,.$ $(\lambda\, g : o \to o \to o\,.\, g\, T_o\, T_o) =$ $(\lambda\, g : o \to o \to o\,.\, g\, x\, y)$.
$(\mathbf{A}_o \wedge \mathbf{B}_o)$	stands for	$\wedge_{o \to o \to o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$\Rightarrow_{o \to o \to o}$	stands for	$\lambda\, x : o\,.\, \lambda\, y : o\,.\, x = (x \wedge y)$.
$(\mathbf{A}_o \Rightarrow \mathbf{B}_o)$	stands for	$\Rightarrow_{o \to o \to o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$\neg_{o \to o}$	stands for	$\lambda\, x : o\,.\, x = F_o$.
$(\neg \mathbf{A}_o)$	stands for	$\neg_{o \to o}\, \mathbf{A}_o$.
$\vee_{o \to o \to o}$	stands for	$\lambda\, x : o\,.\, \lambda\, y : o\,.\, \neg(\neg x \wedge \neg y)$.
$(\mathbf{A}_o \vee \mathbf{B}_o)$	stands for	$\vee_{o \to o \to o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$\mathrm{if}_{o \to \alpha \to \alpha \to \alpha}$	stands for	$\lambda\, b : o\,.\, \lambda\, x : \alpha\,.\, \lambda\, y : \alpha\,.$ $\mathrm{I}\, z : \alpha\,.\, (b \Rightarrow z = x) \wedge (\neg b \Rightarrow z = y)$.
$(\mathbf{A}_o \mapsto \mathbf{B}_\alpha \mid \mathbf{C}_\alpha)$	stands for	$\mathrm{if}_{o \to \alpha \to \alpha \to \alpha}\, \mathbf{A}_o\, \mathbf{B}_\alpha\, \mathbf{C}_\alpha$.

Notational Definitions for Binary Operators

$(\mathbf{A}_\alpha \ \mathbf{c} \ \mathbf{B}_\alpha)$	stands for	$\mathbf{c}_{\alpha \rightarrow \alpha \rightarrow \beta} \, \mathbf{A}_\alpha \, \mathbf{B}_\alpha$ or $\mathbf{c}_{(\alpha \times \alpha) \rightarrow \beta} \, (\mathbf{A}_\alpha, \mathbf{B}_\alpha)$.
$(\mathbf{A}_o \Leftrightarrow \mathbf{B}_o)$	stands for	$\mathbf{A}_o = \mathbf{B}_o$.
$(\mathbf{A}_\alpha \neq \mathbf{B}_\alpha)$	stands for	$\neg(\mathbf{A}_\alpha = \mathbf{B}_\alpha)$.
$(\mathbf{A}_\alpha < \mathbf{B}_\alpha)$	stands for	$(\leq_{\alpha \rightarrow \alpha \rightarrow o} \, \mathbf{A}_\alpha \, \mathbf{B}_\alpha) \wedge (\mathbf{A}_\alpha \neq \mathbf{B}_\alpha)$.
$(\mathbf{A}_\alpha > \mathbf{B}_\alpha)$	stands for	$\mathbf{B}_\alpha < \mathbf{A}_\alpha$.
$(\mathbf{A}_\alpha \geq \mathbf{B}_\alpha)$	stands for	$\mathbf{B}_\alpha \leq \mathbf{A}_\alpha$.
$(\mathbf{A}_\alpha = \mathbf{B}_\alpha = \mathbf{C}_\alpha)$	stands for	$(\mathbf{A}_\alpha = \mathbf{B}_\alpha) \wedge (\mathbf{B}_\alpha = \mathbf{C}_\alpha)$.
$(\mathbf{A}_\alpha \ \mathbf{c} \ \mathbf{B}_\alpha \ \mathbf{d} \ \mathbf{C}_\alpha)$	stands for	$(\mathbf{A}_\alpha \ \mathbf{c} \ \mathbf{B}_\alpha) \wedge (\mathbf{B}_\alpha \ \mathbf{d} \ \mathbf{C}_\alpha)$.

Notational Definitions for Quantifiers

$(\forall\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$	stands for	$(\lambda\, x : \alpha \,.\, T_o) = (\lambda\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$.
$(\exists\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$	stands for	$\neg(\forall\, \mathbf{x} : \alpha \,.\, \neg \mathbf{A}_o)$.
$(\exists!\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$	stands for	$\exists\, y : \alpha \,.\, (\lambda\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o) = (\lambda\, \mathbf{x} : \alpha \,.\, \mathbf{x} = y)$
		where y is not free in $(\lambda\, \mathbf{x} : \alpha \,.\, \mathbf{A}_o)$.

Notational Definitions for Definedness

$\bot_o$	stands for	F_o.
$\bot_\alpha$	stands for	$\mathrm{I}\, x : \alpha \,.\, x \neq x$ where $\alpha \neq o$.
$\Delta_{\alpha \rightarrow \beta}$	stands for	$\lambda\, x : \alpha \,.\, \bot_\beta$ where $\beta \neq o$.
$(\mathbf{A}_\alpha\downarrow)$	stands for	$\mathbf{A}_\alpha = \mathbf{A}_\alpha$.
$(\mathbf{A}_\alpha\uparrow)$	stands for	$\neg(\mathbf{A}_\alpha\downarrow)$.
$(\mathbf{A}_\alpha \simeq \mathbf{B}_\alpha)$	stands for	$(\mathbf{A}_\alpha\downarrow \vee \mathbf{B}_\alpha\downarrow) \Rightarrow \mathbf{A}_\alpha = \mathbf{B}_\alpha$.
$(\mathbf{A}_\alpha \not\simeq \mathbf{B}_\alpha)$	stands for	$\neg(\mathbf{A}_\alpha \simeq \mathbf{B}_\alpha)$.

Alonzo: Semantics

- In Henkin's general models semantics [He50] for CTT, function domains do not need to contain all possible functions.
- Alonzo's semantics is a modified form of the general models semantics that admits undefined expressions in accordance with the traditional approach to undefinedness [Fa04].
  - An undefined expression denotes no value at all.
  - A function domain contains both partial and total functions.
- Alonzo has effectively two semantics:
  1. Semantics of math. practice based on standard models for which there is no sound and complete proof system.
  2. Semantics of logical practice based on general models for which there is a sound and complete proof system.

Traditional Approach to Undefinedness

- The traditional approach to undefinedness, which is widely practiced in mathematics, is based on three principles:
  1. Atomic expressions (i.e., variables and constants) are always defined.
  2. Compound expressions may be undefined.
     - A function application $f(a)$ is undefined if f is undefined, a is undefined, or $a \not\in \mathrm{dom}(f)$.
     - A definite description $\mathrm{I}\, x \in S \,.\, E$ is undefined if there is not exactly one $a \in S$ for which E is true.
  3. Formulas are always true or false and hence defined.
     - So, by convention, a predicate application $p(a) = \mathrm{F}$ if p is undefined, a is undefined, or $a \not\in \mathrm{dom}(p)$.
- There are two kinds of equality:
  1. Equality: $a = b$ if a and b are defined and equal.
  2. Quasi-equality: $a \simeq b$ if $a = b$ or a and b are undefined.

Benefits of the Traditional Approach

- Meaningful statements can involve undefined expressions.

  $\forall\, x \in \mathbb{R} \,.\, 0 \leq x \Rightarrow (\sqrt{x})^2 = x.$

  $0 \leq -2 \Rightarrow (\sqrt{-2})^2 = -2.$

- Function domains can be implicit.

  $k(x) \simeq \frac{1}{x} + \frac{1}{x-1}.$

  $\left(\frac{f}{g}\right)(x) \simeq \frac{f(x)}{g(x)}.$

- Definedness assumptions can be implicit, and as a result, expressions involving undefinedness can be very concise.

  $\forall\, x, y, z \in \mathbb{R} \,.\, \frac{x}{y} = z \Rightarrow x = y * z.$

- Values can be defined implicitly using definite description.

  $\sqrt{x} \simeq \mathrm{I}\, y \in \mathbb{R} \,.\, 0 \leq y \wedge y^2 = x.$

Calculus Examples from [Fa23]

- Def10: $\mathrm{lim}_{(R \rightarrow R) \rightarrow R \rightarrow R} =$
  $\lambda\, f : R \rightarrow R \,.\, \lambda\, a : R \,.\, \mathrm{I}\, b : R \,.$
  $(\forall\, e : R \,.\, 0 < e \Rightarrow$
  $(\exists\, d : R \,.\, 0 < d \wedge$
  $(\forall\, x : R \,.\, 0 < |x - a| < d \Rightarrow$
  $|f\, x - b| < e)))$ (limit of a function).
- Def14: $\mathrm{cont\text{-}at}_{(R \rightarrow R) \rightarrow R \rightarrow o} =$
  $\lambda\, f : R \rightarrow R \,.\, \lambda\, a : R \,.\, \lim_{x \rightarrow a} f\, x = f\, a$
  (continuous at a point).
- Thm27: $\forall\, f, g : R \rightarrow R,\ a, b : R \,.$
  $(\mathrm{cont\text{-}on\text{-}closed\text{-}int}\, f\, a\, b \wedge g = \lambda\, x : R \,.\, \int_a^x (f\, s)\, ds) \Rightarrow$
  $((\forall\, x : R \,.\, a < x < b \Rightarrow \mathrm{deriv\text{-}at}\, g\, x = f\, x) \wedge$
  $\mathrm{right\text{-}deriv\text{-}at}\, g\, a = f\, a \wedge$
  $\mathrm{left\text{-}deriv\text{-}at}\, g\, b = f\, b)$
  (fundamental theorem of calculus).

Part 1

Background: Little Theories Method

Languages

- A language (or signature) of Alonzo is a pair $L = (\mathcal{B}, \mathcal{C})$ where:
  1. $\mathcal{B}$ is a finite set of base types.
  2. $\mathcal{C}$ is a set of constants.
- The base types and constants represent the base domains and distinguished values of a structure, respectively.
- A language specifies a set of expressions.

Theories

- A theory of Alonzo is a pair $T = (L, \Gamma)$ where L is a language and Γ is a set of sentences of L (called the axioms of T).
- A theory specifies the set of structures defined by:
  1. The standard models of T.
  2. The general models of T.

Developments

- A development of Alonzo is a pair $D = (T, \Xi)$ where T is a theory and Ξ is a sequence of definitions and theorems in T.
  - T is the bottom theory of D.
  - T plus the definitions in Ξ is the top theory of D.
- D is said to be a development of T.
  - $(T, [\,])$ is the trivial development of T.
  - We identity T with its trivial development.

Theory Morphisms

- Let T_1 and T_2 be theories of Alonzo.
- A theory morphism Φ from T_1 to T_2 is a mapping of the expressions of T_1 to the expressions of T_2 such that:
  1. Base types are mapped to types and quasitypes.
  2. Constants are mapped to expressions.
  3. Valid sentences are mapped to valid sentences.
- The image of T_1 in T_2 under Φ is an instance of T_1.

Development Morphisms

- Let D_1 and D_2 be developments of Alonzo.
- A development morphism Φ from D_1 to D_2 is, roughly speaking, a theory morphism from the top theory of D_1 to the top theory of D_2.
- A defined constant of D_1 can be mapped in three ways:
  1. Explicitly to a constant of D_2.
  2. Explicitly to a nonconstant of D_2.
  3. Implicitly to an expression of D_2.
- The image of D_1 under Φ in D_2 is an instance of D_1.
- The definitions and theorems of D_1 can be transported to D_2 via a development morphism from D_1 to D_2.
  - That is, the definitions and theorems of D_1 can be transported to all instances of D_1.

Theory and Development Graphs

- A theory graph [KRZ10] is a directed graph whose nodes are theories and whose edges are theory morphisms.
- A development graph is a directed graph whose nodes are developments and whose edges are development morphisms.

Little Theories Method

- The little theories method [FGT92] is an attractive and powerful method for organizing mathematical knowledge:
  1. A body of mathematical knowledge is represented as a development graph G.
  2. Each mathematical topic is developed in a development D of the "little theory" T in G that has the most convenient level of abstraction and the most convenient vocabulary.
  3. The definitions and theorems produced in D are transported, as needed, from D to other, usually more concrete, developments in G via the development morphisms in G.
- Provides a strong form of polymorphism since the theorems of a development hold in all instances of the development.
- The little theories method unleashes the power of the axiomatic method!

Alonzo is Well Suited for the Little Theories Method

1. Alonzo is designed for reasoning about structures.
   - Base types represent the base domains of a structure.
   - Constants represent the distinguished values of a structure.
   - Alonzo has types for function spaces $\alpha \rightarrow \beta$, Cartesian products $\alpha \times \beta$, and power sets $\{\alpha\}$.
   - Alonzo has quasitypes for infinite lists $\langle\alpha\rangle$ and finite lists $[\alpha]$.
2. Alonzo admits categorical theories (in the standard sense).
   - Alonzo has higher-order quantification.
3. Development morphisms can map base types to quasitypes.
   - Alonzo has notational definitions and conventions to enable quasitypes to be treated like types.
   - Alonzo can directly represent the partial functions that arise.
4. Alonzo is equipped with mathematical knowledge modules for constructing developments and development morphisms.

Part 2

A Formalization of Monoid Theory in Alonzo

Monoid Theory

- A monoid is a structure $(\{m\}, \{\cdot, e\})$ or $(m, \cdot, e)$ where:
  1. m is a nonempty set of values.
  2. $\cdot : (m \times m) \to m$ is an associative function.
  3. $e \in m$ is an identity element with respect to $\cdot$.
- Mathematics and computing are replete with examples of monoids such as $(\mathbb{N}, +, 0)$, $(\mathbb{N}, *, 1)$, and $(\Sigma^*, ++, \epsilon)$.
- Monoid theory is the set of the concepts, properties, and facts about monoids.
  - Lacks the rich structure of group theory.
  - But has enough structure to adequately illustrate the little theories method.
- We have formalized monoid theory in Alonzo as a development graph using the little theories method.

Definition of a Monoid in Alonzo

- We need a way to say in Alonzo that a triple $(\mathbf{M}_{\{\alpha\}}, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}, \mathbf{E}_{\alpha})$ denotes a monoid.
- So we introduce an abbreviation via a notational definition:

$$
\boxed{
\begin{array}{l}
\mathrm{MONOID}(\mathbf{M}_{\{\alpha\}}, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}, \mathbf{E}_{\alpha}) \\[1em]
\quad \text{stands for} \\[1em]
\mathbf{M}_{\{\alpha\}}\downarrow \wedge \\
\mathbf{M}_{\{\alpha\}} \neq \emptyset_{\{\alpha\}} \wedge \\
\mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha} \downarrow (\mathbf{M}_{\{\alpha\}} \times \mathbf{M}_{\{\alpha\}}) \rightarrow \mathbf{M}_{\{\alpha\}} \wedge \\
\mathbf{E}_{\alpha} \downarrow \mathbf{M}_{\{\alpha\}} \wedge \\
\forall\, x, y, z : \mathbf{M}_{\{\alpha\}} \,. \\
\quad \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (x, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (y, z)) = \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (\mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (x, y), z) \wedge \\
\forall\, x : \mathbf{M}_{\{\alpha\}} \,.\, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (\mathbf{E}_{\alpha}, x) = \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}\, (x, \mathbf{E}_{\alpha}) = x.
\end{array}
}
$$

- Notice that $\mathbf{M}_{\{\alpha\}}$ is a quasitype within α.

Application of the Little Theories Method

- Let $T = (L, \Gamma)$ be a theory of Alonzo. We can show that $(\mathbf{M}_{\{\alpha\}}, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}, \mathbf{E}_{\alpha})$ denotes a monoid in T by proving

$$T \models \mathsf{MONOID}(\mathbf{M}_{\{\alpha\}}, \mathbf{F}_{(\alpha \times \alpha) \rightarrow \alpha}, \mathbf{E}_{\alpha}).$$

- We may need general definitions and theorems about monoids to prove properties in T about this triple.
- It would be extremely inefficient to state these definitions and prove these theorems in T.
- Instead we should develop a little theory $T_{\rm mon}$ of monoids.
- The definitions and theorems of monoids can be introduced in a development $D_{\rm mon}$ of $T_{\rm mon}$ in a universal abstract form.
- Then these definitions and theorems can be transported to a development D via a development morphism from $D_{\rm mon}$ to D.

Theory Definition (Monoids)

Name: MON.

Base types: M.

Constants: $\cdot_{(M \times M) \to M}$, e_M.

Axioms:

1. $\forall x, y, z : M \,.\, x \cdot (y \cdot z) = (x \cdot y) \cdot z$ (· is associative).
2. $\forall x : M \,.\, \mathsf{e} \cdot x = x \cdot \mathsf{e} = x$
   (e is an identity element with respect to ·).

Development Definition (Monoids 1)

Name: MON-1.

Bottom theory: MON.

Definitions and theorems:

Thm1: $\mathsf{MONOID}(U_{\{M\}}, \cdot_{(M \times M) \to M}, \mathsf{e}_M)$ (models of MON define monoids).

Thm2: $\mathsf{TOTAL}(\cdot)$ ($\cdot$ is total).

Thm3: $\forall\, x : M \,.\, (\forall\, y : M \,.\, x \cdot y = y \cdot x = y) \Rightarrow x = \mathsf{e}$ (uniqueness of identity element).

Def1: $\mathsf{submonoid}_{\{M\} \to o} =$
$\lambda\, s : \{M\} \,.\, s \neq \emptyset_{\{M\}} \wedge (\cdot\restriction_{s \times s} \downarrow (s \times s) \to s) \wedge \mathsf{e} \in s$ (submonoid).

Thm4: $\forall\, s : \{M\} \,.\, \mathsf{submonoid}\, s \Rightarrow \mathsf{MONOID}(s, \cdot\restriction_{s \times s}, \mathsf{e})$ (submonoids are monoids).

Thm5: $\mathsf{submonoid}\, \{\mathsf{e}\}$ (minimum submonoid).

Thm6: $\mathsf{submonoid}\, U_{\{M\}}$ (maximum submonoid).

Def2: $\cdot^{\mathrm{op}}_{(M \times M) \to M} = \lambda\, p : M \times M \,.\, (\mathsf{snd}\, p) \cdot (\mathsf{fst}\, p)$

(opposite of $\cdot$).

Thm7: $\forall\, x, y, z : M \,.\, x \cdot^{\mathrm{op}} (y \cdot^{\mathrm{op}} z) = (x \cdot^{\mathrm{op}} y) \cdot^{\mathrm{op}} z$

($\cdot^{\mathrm{op}}$ is associative).

Thm8: $\forall\, x : M \,.\, \mathsf{e} \cdot^{\mathrm{op}} x = x \cdot^{\mathrm{op}} \mathsf{e} = x$

(e is an identity element with respect to $\cdot^{\mathrm{op}}$).

Def3: $\odot_{(\{M\} \times \{M\}) \to \{M\}} =$
$\mathsf{set\text{-}op}_{((M \times M) \to M) \to ((\{M\} \times \{M\}) \to \{M\})}\, \cdot$ (set product).

Def4: $\mathsf{E}_{\{M\}} = \{\mathsf{e}_M\}$ (set identity element).

Thm9: $\forall\, x, y, z : \{M\} \,.\, x \odot (y \odot z) = (x \odot y) \odot z$

($\odot$ is associative).

Thm10: $\forall\, x : \{M\} \,.\, \mathsf{E} \odot x = x \odot \mathsf{E} = x$

(E is an identity element with respect to $\odot$).

- $\mathsf{set\text{-}op}_{((\alpha \times \beta) \to \gamma) \to ((\{\alpha\} \times \{\beta\}) \to \{\gamma\})}$

  stands for

$$\lambda\, f : (\alpha \times \beta) \to \gamma \,.\, \lambda\, p : \{\alpha\} \times \{\beta\} \,.$$
$$\{z : \gamma \mid \exists\, x : \mathsf{fst}\, p,\ y : \mathsf{snd}\, p \,.\, z = f\,(x, y)\}.$$

Opposite Monoids

- $(M, \cdot^{\mathrm{op}}_{(M \times M) \rightarrow M}, \mathsf{e})$ is a monoid in MON-1 called the opposite monoid of $(M, \cdot_{(M \times M) \rightarrow M}, \mathsf{e}_M)$.
- We can prove this directly using Thm7 and Thm8.
- A better approach is to construct a development morphism from MON to MON-1 that maps

$$(M, \cdot_{(M \times M) \rightarrow M}, \mathsf{e}_M)$$

to

$$(M, \cdot^{\mathrm{op}}_{(M \times M) \rightarrow M}, \mathsf{e}).$$

- This establishes an information conduit for transporting information about monoids to corresponding information about their opposite monoids.

Development Translation (MON to Opposite Monoid)

Name: MON-to-opposite-monoid.

Source development: MON.

Target development: MON-1.

Base type mapping:

1. $M \mapsto M$.

Constant mapping:

1. $\cdot_{(M \times M) \to M} \mapsto \cdot^{\mathrm{op}}_{(M \times M) \to M}$.
2. $\mathsf{e}_M \mapsto \mathsf{e}_M$.

- This translation is normal and thus is a development morphism by Thm7 and Thm8 of MON-1.
- It can be used to transport a theorem to its dual form.
- Example: $x = y \Rightarrow x \cdot z = y \cdot z \quad \mapsto \quad x = y \Rightarrow z \cdot x = z \cdot y$.

Theorem Transportation (Transport of Thm1 to MON-1)

Name: monoid-via-MON-to-opposite-monoid.

Source development: MON.

Target development: MON-1.

Development morphism: MON-to-opposite-monoid.

Theorem:

Thm1: $\mathsf{MONOID}(U_{\{M\}}, \cdot_{(M \times M) \rightarrow M}, \mathsf{e}_M)$
(models of MON define monoids).

Transported theorem:

Thm11 (Thm1-via-MON-to-opposite-monoid):
$\mathsf{MONOID}(U_{\{M\}}, \cdot^{\mathrm{op}}_{(M \times M) \rightarrow M}, \mathsf{e}_M)$
(opposite monoids are monoids).

New target development: MON-2.

Set Monoids

- $(\{M\}, \odot_{(\{M\}\times\{M\})\to\{M\}}, \mathsf{E}_{\{M\}})$ is a monoid in MON-1 called the set monoid of $(M, \cdot_{(M\times M)\to M}, \mathsf{e}_M)$.
- We will prove this by constructing a development morphism from MON to MON-2 that maps

  $$(M, \cdot_{(M\times M)\to M}, \mathsf{e}_M)$$

  to

  $$(\{M\}, \odot_{(\{M\}\times\{M\})\to\{M\}}, \mathsf{E}_{\{M\}}).$$

Development Translation (MON to Set Monoid)

Name: MON-to-set-monoid.

Source development: MON.

Target development: MON-2.

Base type mapping:

1. $M \mapsto \{M\}$.

Constant mapping:

1. $\cdot_{(M \times M) \to M} \mapsto \odot_{(\{M\} \times \{M\}) \to \{M\}}$.
2. $\mathsf{e}_M \mapsto \mathsf{E}_{\{M\}}$.

- This translation is normal and thus is a morphism by Thm9 and Thm10 of MON-2.

Theorem Transportation (Transport of Thm1 to MON-2)

Name: monoid-via-MON-to-set-monoid.

Source development: MON.

Target development: MON-2.

Development morphism: MON-to-set-monoid.

Theorem:

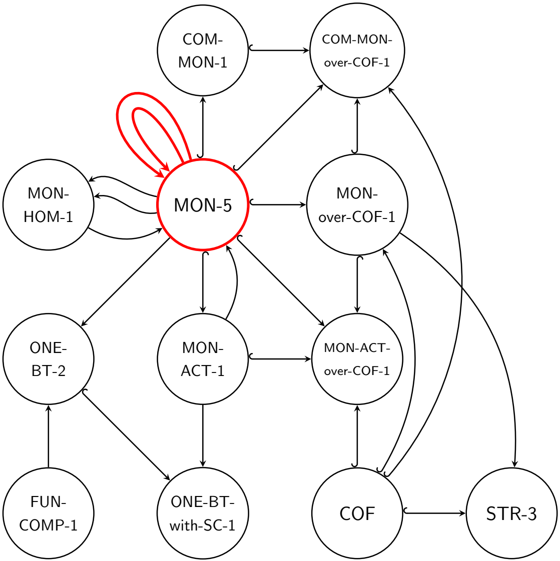
Thm1: $\mathsf{MONOID}(U_{\{M\}}, \cdot_{(M \times M) \to M}, \mathsf{e}_M)$
(models of MON define monoids).

Transported theorem:

Thm12 (Thm1-via-MON-to-set-monoid):
$\mathsf{MONOID}(U_{\{\{M\}\}}, \odot_{(\{M\} \times \{M\}) \to \{M\}}, \mathsf{E}_{\{M\}})$
(set monoids are monoids).

New target development: MON-3.

Development Graph for Monoid Theory



Transformation Monoids

- Let s be a nonempty set and $(f, \circ, \mathrm{id})$ be a triple where:
  1. f is a set of (partial or total) functions from s to s.
  2. $\circ : ((s \rightarrow s) \times (s \rightarrow s)) \rightarrow (s \rightarrow s)$ is function composition.
  3. $\mathrm{id} : s \rightarrow s$ is the identity function.
- $(f, \circ, \mathrm{id})$ a transformation monoid on s if both:
  1. f is closed under $\circ$.
  2. $\mathrm{id} \in f$.
- Theorem. Every transformation monoid is a monoid.
- How should we formalize this theorem?
- First, we define a theory T with one base type representing s.
- Second, we develop T so that we can state and prove the theorem.

Theory Definition (One Base Type)

Name: ONE-BT.

Base types: S.

Constants:

Axioms:

$\mathsf{id}_{\alpha \to \alpha}$

stands for

$\lambda\, x : \alpha \,.\, x$

$\circ_{((\alpha \to \beta) \times (\beta \to \gamma)) \to (\alpha \to \gamma)}$

stands for

$\lambda\, p : (\alpha \to \beta) \times (\beta \to \gamma) \,.\, \lambda\, x : \alpha \,.\, (\mathsf{snd}\, p)\,((\mathsf{fst}\, p)\, x).$

Theory Development (One Base Type 1)

Name: ONE-BT-1

Bottom theory: ONE-BT

Definitions and theorems

Thm13: $\forall f, g, h : S \rightarrow S \,.\, f \circ (g \circ h) = (f \circ g) \circ h$
($\circ$ is associative).

Thm14: $\forall f : S \rightarrow S \,.\, \mathrm{id}_{S \rightarrow S} \circ f = f \circ \mathrm{id}_{S \rightarrow S} = f$
($\mathrm{id}_{S \rightarrow S}$ is an identity element with respect to $\circ$).

Def5: $\textrm{trans-monoid}_{\{S \rightarrow S\} \rightarrow o} =$
$\lambda\, s : \{S \rightarrow S\} \,.$
$s \neq \emptyset_{\{S \rightarrow S\}} \wedge (\circ\restriction_{s \times s} \downarrow (s \times s) \rightarrow s) \wedge \mathrm{id}_{S \rightarrow S} \in s$
(transformation monoid).

Thm15: $\forall\, s : \{S \rightarrow S\} \,.$
$\textrm{trans-monoid}\, s \Rightarrow \mathrm{MONOID}(s, \circ\restriction_{s \times s}, \mathrm{id}_{S \rightarrow S})$
(transformation monoids are monoids).

This is the wrong approach!

Theory Definition (Function Composition)

Name: FUN-COMP.

Base types: A, B, C, D.

Constants:

Axioms:

Development Definition (Function Composition 1)

Name: FUN-COMP-1.

Bottom theory: FUN-COMP.

Definitions and theorem:

Thm13: $\forall f : A \rightarrow B, g : B \rightarrow C, h : C \rightarrow D$.
$f \circ (g \circ h) = (f \circ g) \circ h$ ($\circ$ is associative).

Thm14: $\forall f : A \rightarrow B$. $\mathrm{id}_{A \rightarrow A} \circ f = f \circ \mathrm{id}_{B \rightarrow B} = f$
(identity functions are left and right identity elements).

Theory Translation (FUN-COMP to ONE-BT)

Name: FUN-COMP-to-ONE-BT.

Source development: FUN-COMP.

Target development: ONE-BT.

Base type mapping:

1. $A \mapsto S$.
2. $B \mapsto S$.
3. $C \mapsto S$.
4. $D \mapsto S$.

Constant mapping:

- This translation is a morphism since it is normal and FUN-COMP contains no constants or axioms.

Group Transportation (Thm13 and Thm14 to ONE-BT)

Name:

function-composition-theorems-via-FUN-COMP-to-ONE-BT.

Source development: FUN-COMP-1.

Target development: ONE-BT.

Development morphism: FUN-COMP-to-ONE-BT.

Definitions and theorems:

Thm13: $\forall\, f : A \rightarrow B,\, g : B \rightarrow C,\, h : C \rightarrow D\ .$
$f \circ (g \circ h) = (f \circ g) \circ h$ ($\circ$ is associative).

Thm14: $\forall\, f : A \rightarrow B\ .\ \mathrm{id}_{A \rightarrow A} \circ f = f \circ \mathrm{id}_{B \rightarrow B} = f$
(identity functions are left and right identity elements).

Transported definitions and theorems:

Thm15 (Thm13-via-FUN-COMP-to-ONE-BT):

$\forall f, g, h : S \to S \,.\, f \circ (g \circ h) = (f \circ g) \circ h$

($\circ$ is associative).

Thm16 (Thm14-via-FUN-COMP-to-ONE-BT):

$\forall f : S \to S \,.\, \mathrm{id}_{S \to S} \circ f = f \circ \mathrm{id}_{S \to S} = f$

($\mathrm{id}_{S \to S}$ is an identity element with respect to $\circ$).

New target development: ONE-BT-1.

New development morphism: FUN-COMP-to-ONE-BT.

Theory Translation (MON to ONE-BT)

Name: MON-to-ONE-BT.

Source development: MON.

Target development: ONE-BT.

Base type mapping:

1. $M \mapsto S \rightarrow S$.

Constant mapping:

1. $\cdot_{(M \times M) \rightarrow M} \mapsto \circ_{((S \rightarrow S) \times (S \rightarrow S)) \rightarrow (S \rightarrow S)}$.
2. $\mathsf{e}_M \mapsto \mathsf{id}_{S \rightarrow S}$.

- This translation is normal and thus is a development morphism, in part, by Thm15 and Thm16 of ONE-BT-1.

Group Transportation (Transport of Def1 to ONE-BT-1)

Name: submonoids-via-MON-to-ONE-BT.

Source development: MON-1.

Target development: ONE-BT-1.

Development morphism: MON-to-ONE-BT.

Definitions and theorems:

Def1: $\mathsf{submonoid}_{\{M\} \to o} =$
$\lambda\, s : \{M\} \,.\, s \neq \emptyset_{\{M\}} \wedge (\cdot\restriction_{s \times s} \downarrow (s \times s) \to s) \wedge \mathsf{e} \in s$
(submonoid).

Thm4: $\forall\, s : \{M\} \,.\, \mathsf{submonoid}\, s \Rightarrow \mathsf{MONOID}(s, \cdot\restriction_{s \times s}, \mathsf{e})$
(submonoids are monoids).

Transported definitions and theorems:

Def5 (Def1-via-MON-to-ONE-BT):

$$\begin{array}{l} \mathsf{trans\text{-}monoid}_{\{S \to S\} \to o} = \\ \quad \lambda\, s : \{S \to S\} \,. \\ \quad s \neq \emptyset_{\{S \to S\}} \wedge (\circ\restriction_{s \times s} \downarrow (s \times s) \to s) \wedge \mathsf{id}_{S \to S} \in s \end{array}$$

(transformation monoid).

Thm17 (Thm4-via-MON-to-ONE-BT):

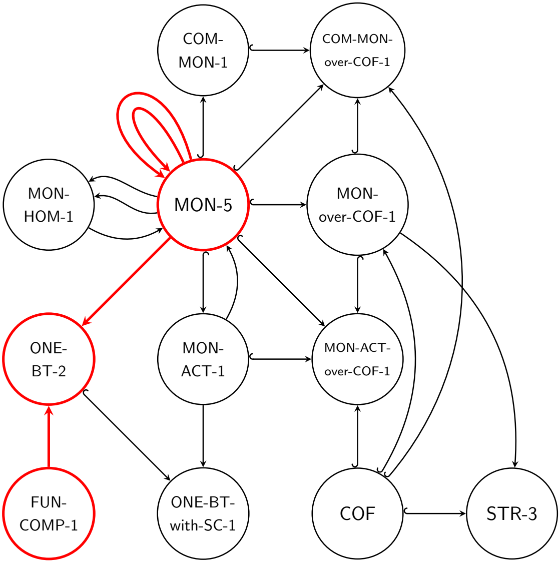
$$\begin{array}{l} \forall\, s : \{S \to S\} \,. \\ \quad \mathsf{trans\text{-}monoid}\, s \Rightarrow \mathsf{MONOID}(s, \circ\restriction_{s \times s}, \mathsf{id}_{S \to S}) \end{array}$$

(transformation monoids are monoids).

New target development: ONE-BT-2.

New development morphism: MON-1-to-ONE-BT-2.

Development Graph for Monoid Theory



Other Topics

Here are the other topics that are components of the development graph for monoid theory:

1. Commutative monoids.
2. Monoid actions.
3. Monoid homomorphisms.
4. Monoids over real number arithmetic.
5. Monoid theory applied to strings.

Part 3

Final Remarks

Related Work: IMPS

- IMPS proof assistant [FGT93] was developed by W. M. Farmer, J. D. Guttman, and F. J. Thayer at MITRE 1990–1993.
- Introduced three major innovations:
  1. Proofs are supported by various kinds of computation.
  2. The IMPS logic, LUTINS, is a version of Church's type theory that admits undefined expressions.
  3. Mathematical knowledge is organized using the little theories method.
- The design of Alonzo is heavily influenced by IMPS.
- Alonzo is simpler than LUTINS.
- Alonzo employs many more notational definitions than IMPS.
- IMPS uses sorts (types and subtypes) instead of quasitypes.
  - But AlonzoS does use sorts.

Other Related Work

- Proof assistants and logical frameworks using theory graphs: Ergo, Isabelle, LF, MMT, PVS.
- Software specification and development systems using theory graphs: ASL, CASL, EHDM, Hets, IOTA, KIDS, OBJ, Specware.
- Proof assistants based on Church's type theory: HOL, HOL Light, Isabelle/HOL, ProofPower, PVS, TPS.
- Proof assistants and programming languages based on dependent type theory: Agda, Automath, Coq, Epigram, F*, Idris, Lean, Nuprl.

Conclusion

1. The little theories method is an effective method for organizing mathematics so that:
   - Similar structures can be formally connected.
   - Redundancy can be systematically reduced.
   - Information can flow between related contexts.
2. Alonzo is a logic well-suited for expressing and reasoning about mathematical ideas because it:
   - Has sufficient expressivity, both theoretical and practical.
   - Is designed for reasoning about mathematical structures.
   - Employs many notational definitions and conventions.
   - Admits partial functions and undefined expressions.
   - Is equipped with mathematical knowledge modules.
3. The little theories method using Alonzo requires minimal software support — just LaTeX macros and environments.

Thank you!

References

[An02] P. B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition*, Kluwer, 2002.

[Ch40] A. Church, "A Formulation of the Simple Theory of Types", *Journal of Symbolic Logic*, 5:56—68, 1940.

[Fa04] W. M. Farmer, "Formalizing Undefinedness Arising in Calculus", in: D. Basin et al., eds, *Automated Reasoning*, *LNCS*, 3097:475–489, 2004.

[Fa08] W. M. Farmer, "The Seven Virtues of Simple Type Theory", *Journal of Applied Logic*, 6:267—286, 2008.

[Fa23] W. M. Farmer, *Simple Type Theory: A Practical Logic for Expressing and Reasoning about Mathematical Ideas*, Birkhäuser/Springer, 2023.

[FGT92] W. M. Farmer, J. D. Guttman, and F. J. Thayer, "Little theories", in: D. Kapur, ed., *Automated Deduction — CADE-11*, *LNCS*, 607:567–581, 1992.

[FGT93] W. M. Farmer, J. D. Guttman, and F. J. Thayer, "IMPS: An Interactive Mathematical Proof System", *Journal of Automated Reasoning*, 11:213–248, 1993.

[FZ23] W. M. Farmer and D. Y. Zvigelsky, "Monoid Theory in Alonzo: A Little Theories Formalization in Simple Type Theory", 2023.

[KRZ10] M. Kohlhase, F. Rabe, and V. Zholudev, "Towards MKM in the Large: Modular Representation and Scalable Software Architecture. In S. Autexier et al., eds. *Intelligent Computer Mathematics*, *LNCS*, 6167:370–384, 2010.

[He50] L. Henkin, "Completeness in the theory of types", *Journal of Symbolic Logic*, 15:81–91, 1950.