Generating Custom Set Theories with Non-Set Structured Objects

CICM 2021

www.macs.hw.ac.uk/~cmd1/cicm2021/zf-plus-paper.pdf

Ciarán M. Dunne www.macs.hw.ac.uk/~cmd1/
Joint work with: J. B. Wells and Fairouz Kamareddine
Heriot-Watt University, Scotland

2021-07-27

Part I

Sets in Foundations of Mathematics

Foundations of Mathematics

- A foundation of mathematics is a small formal system that attempts to model the practice of mathematicians.
- Foundations provide explanations of common mathematical concepts, and a means for reasoning about them.
- Foundations can be implemented using computer software to help write and verify proofs.
- Some theorem provers emphasize the use of sets in their foundations:
  - Isabelle/ZF, Mizar, Metamath/ZFC
- Other (more popular) provers emphasize the use of types in their foundations:
  - Coq, Lean, Isabelle/HOL, Agda

Formalizing Mathematics in a Typical Set Theory

- In a formalism that emphasizes sets, we typically only have types for sets and truth values, and types for a few operators.
  - e.g. $\emptyset :: \text{set}$, $\mathcal{P} :: \text{set} \to \text{set}$, $\in :: \text{set} \to \text{set} \to \text{bool}$
- Most objects are encoded as things of type set.
  - natural numbers: $0^* := \emptyset$, $1^* := \{0^*\}$, $2^* := \{0^*, 1^*\}$, $\ldots$
  - ordered pairs: $\langle 0, 1 \rangle^* := \{\{0^*\}, \{0^*, 1^*\}\}$
    - the left object occurs in all members of the set, the right object occurs in exactly one
  - functions: addOne $:= \{\langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle, \ldots\}$

(+) Working within one big type gives freedom to state things however you want.

(-) User on their own when keeping track of intended meanings of sets.

(-) Representation overlap can cause formalization difficulties.

Representation Overlap in a Typical Set Theory

- The problem of representation overlap can be demonstrated by attempting to formalize $g : (\mathbb{N}^2 \cup \mathcal{P}(\mathbb{N})) \hookrightarrow \mathbb{N}$ such that:

$$g(x) = \begin{cases} 0 & \text{if } x = \{1, 2\} \\ 1 & \text{if } x = \langle 0, 1 \rangle \end{cases}$$

- We could try to formalize this function in a set theory as:

$$g := \{\langle \{1, 2\}, 0 \rangle, \langle \langle 0, 1 \rangle, 1 \rangle\}$$

- But due to our choice of encodings:

$$\{1, 2\} = \{\{0\}, \{0, 1\}\} = \langle 0, 1 \rangle$$

- Hence g cannot be a function because

$$\langle \{\{0\}, \{0, 1\}\}, 0 \rangle \in g \quad \text{and} \quad \langle \{\{0\}, \{0, 1\}\}, 1 \rangle \in g$$

Tagging Mechanisms

- Representation overlaps can sometimes be avoided by tagging objects to make them distinct, but this is generally painful.

Auxillary Definition

Choose two tagging functions:

$$\mathsf{tag}_1 : \mathcal{P}(\mathbb{N}) \to B_1 \qquad \mathsf{tag}_2 : \mathbb{N} \times \mathbb{N} \to B_2$$

such that $\mathsf{tag}_1, \mathsf{tag}_2$ are injective and $B_1 \cap B_2 = \emptyset$.

Definition

We redefine our example g as follows:

$$g : (B_1 \cup B_2) \to \mathbb{N} \qquad g(b) = \begin{cases} 0 & \text{if } b = \mathsf{tag}_1(\{1,2\}) \\ 1 & \text{if } b = \mathsf{tag}_2(\langle 0,1 \rangle) \end{cases}$$

- Then $g(\mathsf{tag}_1(\{1,2\})) = 0$ and $g(\mathsf{tag}_2(\langle 0,1 \rangle)) = 1$.

Formalizing Mathematics in a Type Theory

- In a formalism that emphasizes types, objects are organized into fine-grained types.

(+) Types can be used to encode information that can help automate tedious reasoning.

(+) Prevent users from forming mostly unwanted syntax.

(-) We don't always know how to define a type for a math object we have in mind.

(-) In some cases, obeying the typing rules can get in the way of reasoning.

Research Aims

- Is it possible to have more flexibility?
- Is there a foundation with all mathematical objects living in one big type (e.g. the size of a set theory), but some objects genuinely not sets?
- Our research aims to test whether this approach is workable.
- The paper I am presenting shows how to generate axiomatizations and build models for set theories with:
  - non-set ordered pairs
  - non-set natural numbers
  - a non-set exception object, that can't be inside another object
  - modular combinations of these features
- The next part of our talk will focus on the axiomatisations, the last part will give a brief overview of model building.

Part II

Generating Axiomatisations of Generalised Set Theories

Axiomatic Set Theory: A Recap

- Traditionally, axiomatic set theories are given by collections of (first-order or second-order) axioms which characterise a membership relation $\in$ over a domain of sets.
- The most popular systems are Zermelo-Fraenkel set theory (ZF), and its variants that add extra axioms (ZFC, TG, etc.)
- The axioms of ZF tell us:
  - when sets are equal (extensionality)
  - which sets must exist:
    - results of combinatorial operations (union, power set, replacement)
    - there is an infinite set (infinity)
  - no infinitely descending chains of sets (foundation)

Generalised Set Theories

- A Generalised Set Theory (GST) (Aczel, 1996) is a theory with:
  - pure sets, containing only pure sets 'all the way down' to $\emptyset$,
  - non-set objects which do not contain things via $\in$,
  - impure sets which may contain a mixture of pure sets, non-set objects, and other impure sets.
- There have been many GSTs with non-set objects called urelements or atoms.
- Urelements typically have no 'structure', so we use the terminology non-set object to avoid confusion.

GST Features

- We generate GSTs by combining features.
- Within a logical framework inspired by Isabelle/Pure.
- Each feature consists of:
  - a signature of symbols for predicates & operators
  - a theory of formulas which specify those symbols
  - a unary ownership predicate that identifies members of the feature
  - a binary child predicate that relates a structured object with its 'children'
- We have the features **Set**, **Pair**, **Nat**, **Exception**.
- Additional axioms generated in the feature combination procedure.
  - input: a set of features
  - output: a set of axioms for a GST with those features

The Set feature

- The signature for the **Set** feature consists of 9 symbols, including $\in, \mathsf{Set}, \mathcal{P}$.
- Ownership predicate: Set
- The theory has 10 axioms, including:
  - characterisation of sets: $\forall\, x \,.\, \mathsf{Set}\, x \leftrightarrow (x = \emptyset \vee \exists\, y \,.\, y \in x)$
  - power set axiom: (Zermelo 1908)
    $\forall[\mathsf{Set}]\ x \,.\, \forall\, z \,.\, z \in \mathcal{P}\, x \leftrightarrow z \subseteq x$
- Child predicate: $\in$
- Usual syntactic sugar for $\{X\}$, $\{X, Y\}$, ...
- Example: from the axioms, we can prove formulas like:
  - i.e. for any b, c, the set $\{b, c\}$ exists.

$$\forall\, b, c \,.\, \exists[\mathsf{Set}]\ x \,.\, b \in x \wedge c \in x$$

The Pair feature

- The signature for **Pair** has two symbols: pair and Pair.
- Syntactic sugar: (X, Y) for $(\mathsf{pair}\, X\, Y)$.
- Ownership predicate: Pair
- Theory with two axioms:
  - characterisation of pairs: $\forall\, p\,.\, \mathsf{Pair}\, p \leftrightarrow \exists\, x, y\,.\, p = (x, y)$
  - standard characteristic property of ordered pairs: (Peano 1897)
    $\forall\, a, b, x, y\,.\, (a, b) = (x, y) \leftrightarrow (a = x \wedge b = y)$
- Child predicate: $\lambda\, p, x\,.\, \exists\, y\,.\, p = (x, y) \vee p = (y, x)$
- Example: Prove formulas like:

$$\forall\, x, y\,.\, x \neq y \rightarrow (x, y) \neq (y, x)$$

The Nat features

- Signature for the $\mathbf{Nat}$ feature: $\mathbf{0}, \mathbf{S}, \mathsf{Nat}$.
- Ownership predicate: Nat
- Theory with 6 axioms, including:
  - ownership axioms: $\mathsf{Nat}\,\mathbf{0}$, $\forall[\mathsf{Nat}]\ n\,.\,\mathsf{Nat}\,(\mathbf{S}\,n)$
  - induction axiom: (Dedekind 1888)
    $\forall\, p\,.\,p\,\mathbf{0} \to (\forall[\mathsf{Nat}]\ x\,.\,p\,x \to p\,(\mathbf{S}\,x)) \to \forall[\mathsf{Nat}]\ y\,.\,p\,y$
    Notice this axiom does not mention the concept of set.
- No child predicate.
- Syntactic sugar: $\mathbf{1} := \mathbf{S}\,\mathbf{0}$, $\mathbf{2} := \mathbf{S}\,\mathbf{1}$, etc.

Combining Features

- To combine features:
  - take the union of theories,
  - add axioms stating: all objects are owned by exactly one feature,
  - add well-foundedness axiom using 'child' predicates.
    - no infinitely descending chains of sets, pairs, or mixtures
- Example: combining the $\mathbf{Set}$, $\mathbf{Pair}$ and $\mathbf{Nat}$ features:

$$\begin{aligned}
\mathsf{ZF}^\star := \mathsf{GZF} &\cup \mathsf{PTheory} \cup \mathsf{NTheory} \\
&\cup \{\, \forall\, x \,.\, \mathsf{Set}\, x \vee \mathsf{Pair}\, x \vee \mathsf{Nat}\, x \,\} \\
&\cup \{\, \forall x \,.\, \neg\, (\mathsf{Set}\, x \wedge \mathsf{Pair}\, x) \\
&\qquad \wedge\, \neg\, (\mathsf{Set}\, x \wedge \mathsf{Nat}\, x) \\
&\qquad \wedge\, \neg\, (\mathsf{Nat}\, x \wedge \mathsf{Pair}\, x \,\}) \\
&\cup \{\mathsf{WF}(\mathbf{Set}, \mathbf{Pair})\}
\end{aligned}$$

- $\mathbf{Exception}$ uses a more complicated combination process.

Example Function g in a GST

- Now that our sets, pairs, and naturals are distinct, our function g can be formulated simply as:
  $g := \{(\{\mathbf{1}, \mathbf{2}\}, \mathbf{0}), ((\mathbf{0}, \mathbf{1}), \mathbf{1})\}$
- No representation overlap: $\{\mathbf{1}, \mathbf{2}\} \neq (\mathbf{0}, \mathbf{1})$
- Hence $g(\{\mathbf{1}, \mathbf{2}\})$ and $g((\mathbf{0}, \mathbf{1}))$ are well-defined.

Conclusion of Part II

- Features allow users to specify mathematical objects independently of a representation.
- Combine features to generate axiomatisations of set theories in which the objects of distinct features are distinct.
- Provides a formal justification for informal mathematics that treats sets, pairs and numbers as distinct objects.
- Future work:
  - abstract number system such that: $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$
  - non-set functions, non-set ordinals, non-set structures, etc.

Part III

Building Models of GSTs for Confidence in Their Consistency

Safety of Generated GSTs

- ZF is believed by nearly everyone to be consistent.
  - i.e. there is no φ such that ZF $\vdash \varphi$ and ZF $\vdash \neg\varphi$
  - due to nearly a century of intense scrutiny
- Extra axioms could introduce contradictions!
- How do we know that our GSTs are safe to use?
- If we can build a structure in a consistent GST Γ_1 that satisfies the axioms of a GST Γ_2, then Γ_2 is consistent.
- Our work develops a model building kit which may be invoked in any GST with the $\mathbf{Set}$ feature.
- Hence, models are built at the object-level.

Modelling the Distinctness of GST Features

- We implement our requirement that each object is owned by exactly one feature.
- Hence objects in the model are tagged so that they can be distinguished.
- We currently build all model objects only using pure sets.
  Let $\langle \cdot, \cdot \rangle$ be the Kuratowski encoding of ordered pairs,
  and n^* be the von Neumann natural number of n.
  - sets: $\{X_1, \ldots, X_n\}^\star := \langle 0^*, \{X_1^\star, \ldots, X_n^\star\} \rangle$
  - pairs: $(X, Y)^\star := \langle 1^*, \langle X^\star, Y^\star \rangle \rangle$
  - natural numbers: $\mathbf{n}^\star := \langle 2^*, n^* \rangle$
- Example: $(\{\mathbf{1}, \mathbf{2}\}, \mathbf{0})^\star := \langle 1^*, \langle \langle 0^*, \{1^*, 2^*\} \rangle, \langle 2^*, 0^* \rangle \rangle \rangle$

Modelling the Size of a GST Domain

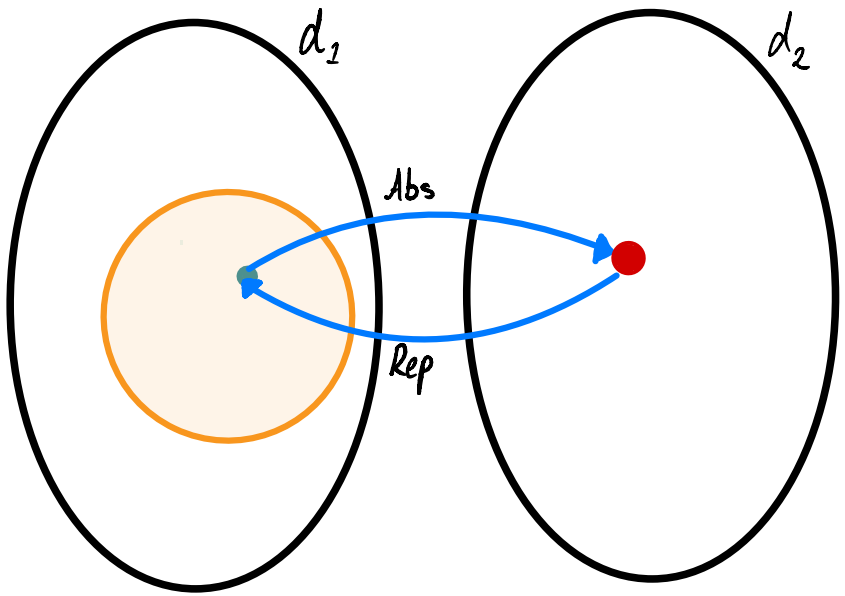
- We need to build a model of a set theory in another set theory of the same size.
- In general, building a structure of this size is problematic.
- Instead, models are built as cumulative hierarchies of sets defined using transfinite recursion over the ordinals.
- Our kit uses a transfinite recursion operator to define a model building operator with built-in tagging.
- We tweak parameters of the operator to model different features.

Domain Types

- We have two ways of presenting a GST:
  1. directly assuming a set of axioms
  2. defining a model, then proving a set of 'axioms'
- To support the second approach, our logical framework has many domain types $\mathsf{d}_0, \mathsf{d}_1, \mathsf{d}_2, \ldots$
- Constants for each domain: $\emptyset_0, \emptyset_1, \emptyset_2, \ldots$
- Formulas that speak about each domain: $\forall_2 \, x \, . \, x \notin_2 \emptyset_2$
- We provide a mechanism for connecting a model built in d_i to a domain d_j.

Connecting a Model to a Domain

- We use a technique used in HOL provers that allows presenting a type of abstract objects as a portion of an existing type.
- Operators Abs :: $d_1 \Rightarrow d_2$ and Rep :: $d_2 \Rightarrow d_1$ that form an isomorphism between the model in d_1 and the entirety of d_2.
- Formulas that speak about the model in d_1 have counterparts in d_2, truth preserved by isomorphism.



Direct and Indirect Axiomatisation of GSTs

- Each domain type has a deduction relation $\vdash_0, \vdash_1, \vdash_2, \ldots$
- Let Γ_1 be a set of formulas specifying a GST in d_1, Γ_2 be a set of formulas specifying a GST in d_2.
- Direct axiomatisation approach:
  - Take Γ_1 as axioms, reason about d_1 with $\vdash_1$.
  - Take Γ_2 as axioms, reason about d_2 with $\vdash_2$.
- Indirect axiomatisation approach:
  - Take Γ_1 as axioms, reason with $\vdash_1$.
  - Build a model of Γ_2 in d_1.
  - Connect the model in d_1 to d_2.
  - Show that Γ_2 can be proved using $\vdash_1$.
- If Γ_1 is consistent and connection is safe, then Γ_2 is consistent.

Summarizing Parts I, II and III:

- Sometimes its useful to do a lot of reasoning in one big type.
- Typical set theories provide this approach, but 'everything is a set' is unsatisfactory.
- The work presented shows that we can define mathematical concepts independently, and generate axiomatisations of GSTs that combine them all into one type.
- Our model building kit can be used in any GST, and we have used it to build models of GSTs with any of our features.
- The connection mechanism presents a GST in terms of a model, providing a method for showing consistency of GSTs.

Future Work

- User-friendly GST specification and use:
  - uniform feature specification & model building
  - users need not be aware they are working in a different GST
- Computer implementation to verify usability and formalization:
  - implement in Isabelle to verify current results
  - automate feature combination procedure
  - automate model building & connection
  - transfer theorems between GSTs with common features
  - subtypes for the portions of the domain owned by features