New Foundations is consistent: an exposition and formal verification

Sky Wilshaw

April 24, 2024

Abstract

We give a self-contained account of a version of Holmes' proof [6] that Quine's set theory *New Foundations* [8] is consistent relative to the metatheory ZFC. We have formalised this proof in the Lean interactive theorem prover [11], and this paper is a 'deformalisation' of that work. We discuss the challenges of formalising new and untested mathematics in an interactive theorem prover, and how the process of completing the formalisation has influenced our presentation of the proof.

Contents

1 Overview

In §2, we will briefly discuss Lean [7], the interactive theorem prover in which our result is formalised. We will also explain why our formalisation in [11] can be trusted as evidence that Holmes' proof in [6] is correct, without needing to understand the underlying details of the proof. We have made frequent use of the community-made repository `mathlib` [2], which encodes standard mathematical definitions and theorems in Lean; without this, we would have needed to write our own libraries for (for example) abstract algebra and cardinal and ordinal arithmetic.

Lean is based on a version of the *calculus of constructions*, which is a dependent type theory. In order to authentically present the formalised proof, the mathematics of this paper will take place in this type theory, or some suitable variant of it. We will rarely make note of this choice, and readers are not expected to be familiar with such type theories. However, this will be relevant for some discussion sections, as some parts of the proof were made significantly harder by the fact that we are working in a type theory.

In §3, we will establish the mathematical context for the proof we will present. In particular, our proof will not directly show the consistency of NF; instead, we will construct a model of a related theory known as *tangled type theory*, or TTT. This is the result which has been formally verified: there is a structure that satisfies a particular axiomatisation of TTT which we will discuss in §3. The expected conclusion that NF is consistent then follows from the fact that NF and TTT are equiconsistent [5].

We will now outline our general strategy for the construction of a model of tangled type theory. As we will outline in §3.3, TTT is a many-sorted theory with types indexed by a limit ordinal λ. In order to impose symmetry conditions on our structure, we will add an additional level of objects below type zero. These will not be a part of the final model we construct. This base type will be comprised of objects called *atoms* (although they are not atoms in the traditional model-theoretic sense). Alongside the construction of the types of our model, we will also construct a group of permutations of each type, called the *allowable permutations*. Such permutations will preserve the structure of the model in a strong sense; for instance, they preserve membership.

The construction of a given type can only be done under certain hypotheses on the construction of lower types. The most restrictive condition that we will need to satisfy is a bound on the size of each type. In order to do this, we will need to show that there are a lot of allowable permutations. The main technical lemma establishing this, called the *freedom of action theorem*, roughly states that a partial function that locally behaves like an allowable permutation can be extended to an allowable permutation. Much of this paper will be allocated to proving the freedom of action theorem and its various corollaries, and it will be outlined in more detail when we are in a position to prove it.

We can then finish the main induction to build the entire model out of the types we have constructed. This step, while invisible to a human reader in set theory, takes substantial effort to formally establish in a dependent type theory. It then remains to show that this is a model of TTT as desired, or more precisely, a model of a particular finite axiomatisation.

[Finish the introduction...]

2 The Lean interactive theorem prover

2.1 Lean and its type theory

Lean [7] is a functional programming language and interactive theorem prover. As indicated in §1, its underlying logic is a dependent type theory based on the calculus of constructions. Carneiro proved in [1] that Lean's type theory is consistent relative to

$$\mathsf{ZFC} + \{\text{there are } n \text{ inaccessible cardinals} \mid n < \omega\}$$

These inaccessible cardinals are needed to support Lean's hierarchy of type universes. Higher universes are commonly used whenever they are convenient, for example in definitions of cardinals and ordinals. However, these uses are not strictly necessary for our purposes, and the entire proof can be carried out in plain ZFC, as shown by [6] and this paper.

Proofs in Lean may be written in its *tactic mode*, which tracks hypotheses and goals, and enables the use of *tactics* to update these hypotheses and goals according to logical rules. There are a large variety of tactics to perform different tasks, such as simplification (`simp`), rewriting of subexpressions (`rw`), structural induction (`induction`), and so on. These tactics output a *proof term*, which is a term in Lean's underlying type theory. The type of this term corresponds to the proposition that we intend to prove under the Curry–Howard correspondence. The proof term is then passed to Lean's *kernel*, which contains a type-checking algorithm. If the proof term generated by a tactic has the correct type, the kernel accepts the proof.

2.2 Trusting Lean

Lean is a large project, but one need only trust its kernel to ensure that accepted proofs are correct. If a tactic were to output an incorrect proof term, then the kernel would have the opportunity to find this mistake before the proof were to be accepted.

It is important to note that the kernel has no way of knowing whether a formal definition written in Lean matches the familiar mathematical definition. Any definitions used in a theorem statement must be manually checked by a human reader; all that Lean guarantees is that the conclusion is correct as written in its own type theory. For example, if verifying a formalised proof of Fermat's last theorem, one should manually check the definitions of natural numbers, addition, exponentiation, and so on, but need not check (for example) definitions and results about elliptic curves.

All of the proofs in this paper (except in §3, upon which no other results depend) are verified by Lean. To help with the verification step, our main result can be found in the `Result.lean` file (source, documentation). Each definition and result is tagged with a hyperlink (such as ∃∀) to the documentation generated from the corresponding Lean code.

3 The theories at issue

In 1937, Quine introduced *New Foundations* (NF) [8], a set theory with a very small collection of axioms. To give a proper exposition of the theory that we intend to prove consistent, we will first make a digression to introduce the related theory TST, as explained by Holmes in [6]. We will then describe the theory TTT, which we will use to prove our theorem.

3.1 The simple theory of types

The *simple theory of types* (known as *théorie simple des types* or TST) is a first order set theory with several sorts, indexed by the nonnegative integers. Each sort, called a *type*, is comprised of *sets* of that type; each variable x has a nonnegative integer type('x') which denotes the type it belongs to. For convenience, we may write x^n to denote a variable x with type n.

The primitive predicates of this theory are equality and membership. An equality '$x = y$' is a well-formed formula precisely when type('x') = type('y'), and similarly a membership formula '$x \in y$' is well-formed precisely when type('x') + 1 = type('y').

The axioms of this theory are extensionality

$$\forall x^{n+1}. \forall y^{n+1}. (\forall z^n. z^n \in x^{n+1} \leftrightarrow z^n \in y^{n+1}) \rightarrow x^{n+1} = y^{n+1}$$

and comprehension

$$\exists x^{n+1}. \forall y^n. (y^n \in x^{n+1} \leftrightarrow \varphi(y^n))$$

where φ is any well-formed formula, possibly with parameters.

Remarks 3.1.

(i) These are both axiom schemes, instantiated for all type levels n, and (in the latter case) for all well-formed formulae φ.

(ii) The inhabitants of type 0, called *individuals*, cannot be examined using these axioms.

(iii) By comprehension, there is a set at each nonzero type that contains all sets of the previous type. Russell-style paradoxes are avoided as formulae of the form $x^n \in x^n$ are ill-formed.

3.2 New Foundations

New Foundations is a one-sorted first-order theory based on TST. Its primitive propositions are equality and membership. There are no well-formedness constraints on these primitive propositions.

Its axioms are precisely the axioms of TST with all type annotations erased. That is, it has an axiom of extensionality

$$\forall x.\, \forall y.\, (\forall z.\, z \in x \leftrightarrow z \in y) \rightarrow x = y$$

and an axiom scheme of comprehension

$$\exists x.\, \forall y.\, (y \in x \leftrightarrow \varphi(y))$$

the latter of which is defined for those formulae φ that can be obtained by erasing the type annotations of a well-formed formula of TST. Such formulae are called *stratified*. To avoid the explicit dependence on TST, we can equivalently characterise the stratified formulae as follows. A formula φ is said to be stratified when there is a function σ from the set of variables to the nonnegative integers, in such a way that for each subformula '$x = y$' of φ we have $\sigma(\text{`}x\text{'}) = \sigma(\text{`}y\text{'})$, and for each subformula '$x \in y$' we have $\sigma(\text{`}x\text{'}) + 1 = \sigma(\text{`}y\text{'})$.

Remarks 3.2.

(i) It is important to emphasise that while the axioms come from a many-sorted theory, NF is not one; it is well-formed to ask if any set is a member of, or equal to, any other.

(ii) Russell's paradox is avoided because the set $\{x \mid x \notin x\}$ cannot be formed; indeed, $x \notin x$ is an unstratified formula. Note, however, that the set $\{x \mid x = x\}$ is well-formed, and so we have a universe set.

(iii) Specker showed in [9] that NF disproves the Axiom of Choice.

While our main result is that New Foundations is consistent, we attack the problem by means of an indirection through a third theory.

3.3 Tangled type theory

Introduced by Holmes in [5], *tangled type theory* (TTT) is a multi-sorted first order theory based on TST. This theory is parametrised by a limit ordinal λ, the elements of which will index the sorts; ω works, but we prefer generality. As in TST, each variable x has a type that it belongs to, denoted $\mathsf{type}(\text{`}x\text{'})$. However, in TTT, this is not a positive integer, but an element of λ.

The primitive predicates of this theory are equality and membership. An equality '$x = y$' is a well-formed formula when $\mathsf{type}(\text{`}x\text{'}) = \mathsf{type}(\text{`}y\text{'})$. A membership formula '$x \in y$' is well-formed when $\mathsf{type}(\text{`}x\text{'}) < \mathsf{type}(\text{`}y\text{'})$.

The axioms of TTT are obtained by taking the axioms of TST and replacing all type indices in a consistent way with elements of λ. More precisely, for any order-embedding $s : \omega \to \lambda$, we can convert a well-formed formula φ of TST into a well-formed formula φ^s of TTT by replacing each type variable α with $s(\alpha)$.

Remarks 3.3.

(i) Membership across types in TTT behaves in some quite bizarre ways. Let $\alpha \in \lambda$, and let x be a set of type α. For any $\beta < \alpha$, the extensionality axiom implies that x is uniquely determined by its type-β elements. However, it is simultaneously determined by its type-γ elements for any $\gamma < \alpha$. In this way, one extension of a set controls all of the other extensions.

(ii) The comprehension axiom allows a set to be built which has a specified extension in a single type. The elements not of this type may be considered 'controlled junk'.

We now present the following striking theorem.

Theorem 3.4 (Holmes)**.** NF is consistent if and only if TTT is consistent. [5]

We will actually prove something slightly stronger.

Theorem 3.5. Let T be a theory in the language of TST. Let T_{NF} be the theory in the language of NF given by erasing the type annotations of T. Let T_{TTT} be the theory in the language of TTT given by instantiating the sentences of T at all possible combinations of type levels. Then T_{NF} is consistent if and only if T_{TTT} is consistent.

Proof. Suppose that T_{NF} has a model M. Let N be the structure in the language of TTT where each type α is interpreted as M, and where the membership relation is given by that on M. It is easy to see by induction that all sentences in T_{TTT} hold in N, as required.

Now suppose that T_{TTT} has some model M. This proof that T_{NF} is consistent proceeds in two stages. In the first stage, we show that $T + \mathsf{Amb}$ is consistent, where Amb is the *ambiguity scheme*

$$\mathsf{Amb} \equiv \{\varphi \leftrightarrow \varphi^+ \mid \varphi \text{ is a sentence in the language of TST}\}$$

This result is due to Holmes in [5]. We will then use this to show that T_{NF} is consistent, using a result due to Specker in [10].

Suppose that $T + \mathsf{Amb}$ is not consistent. By compactness, there is some finite set of sentences Σ in the language of TST such that $T + \mathsf{Amb}_\Sigma$ is inconsistent, where

$$\mathsf{Amb}_\Sigma \equiv \{\varphi \leftrightarrow \varphi^+ \mid \varphi \in \Sigma\}$$

Suppose that Σ uses only type indices $0, \dots, n-1$. Let $[\lambda]^n$ be the collection of n-element subsets of λ, and define a function $\sigma : [\lambda]^n \to \mathcal{P}(\Sigma)$ as follows. If

$$A = \{\alpha_0, \dots, \alpha_{n-1}\} \text{ with } \alpha_0 < \dots < \alpha_{n-1}$$

then $\varphi \in \sigma(A)$ if and only if the interpretation of φ in M at levels $\alpha_0, \dots, \alpha_{n-1}$ is true. This defines a partition of $[\lambda]^n$ into finitely many subsets. By Ramsey's theorem, there is an infinite homogeneous set $H \subseteq \lambda$ for this partition, that is, if $A, B \in [H]^n$, then $\sigma(A) = \sigma(B)$. Let $\alpha_0, \alpha_1, \dots$ be an increasing sequence in H, and define a structure N in the language of TST by interpreting type i as M_{α_i}. Then, N models $T + \mathsf{Amb}_\Sigma$ as required.

Now, we show that the consistency of $T + \mathsf{Amb}$ implies that of T_{NF}. This relies on a lemma of Specker in [10]. An *endomorphism* of a one-sorted language is an operation $(-)^*$ on the function and relation

symbols, mapping them to terms (respectively formulas) with the same free variables. This extends in a natural way to formulas in the language.

We can reformalise T into a theory T' over a one-sorted language by adding a unary relation symbol T_n for each type index n, and recursively replacing each instance of $\exists x^n.\, \varphi$ with $\exists x.\, T_n(x) \wedge \varphi$. This language has an endomorphism $(-)^+$ which maps T_n to T_{n+1}.

Specker's lemma can be phrased in the following way.

Lemma 3.6. Let U be a complete theory in a one-sorted language L with endomorphism $(-)^*$. Then if

$$U + \{\varphi \leftrightarrow \varphi^\star \mid \varphi \text{ is an } L\text{-sentence}\}$$

is consistent, then there is a model M of U that admits a function $f : M \to M$ such that for every relation symbol R of L,

$$M \vDash R(x_1, \dots, x_m) \text{ if and only if } M \vDash R(f(x_1), \dots, f(x_m))$$

In our case, $T + \mathsf{Amb}$ is consistent, so the corresponding one-sorted theory as required for the lemma is consistent (and has a complete extension). This requires choosing an interpretation of the membership relation for pairs of type indices that do not differ by one, but this does not interfere with anything that we need (for instance, the relation can always be interpreted as false). This yields a model of T' with a type-raising function f. This naturally gives rise to a model of T in the language of TST in which all type levels are isomorphic. Therefore, the carrier set of each type level of this model provides a model of T_{NF} as required. □

Thus, our task of proving NF consistent is reduced to the task of proving TTT consistent. We will do this by exhibiting an explicit model (albeit one that requires a great deal of Choice to construct). As TTT has types indexed by a limit ordinal, and sets can only contain sets of lower type, we can construct a model by recursion over λ. In particular, a model of TTT is a well-founded structure. This was not an option with NF directly, as the universe set $\{x \mid x = x\}$ would necessarily be constructed before many of its elements.

3.4 Finitely axiomatising tangled type theory

Hailperin showed in [4] that the comprehension scheme of NF is equivalent to a finite conjunction of its instances. These axioms are all stratified (as is extensionality), so NF is equivalent to a theory of the form T_{NF} where T is a particular finite theory in the language of TST. Then, by theorem 3.5, the consistency of NF can be established by witnessing a model of T_{TTT}. The same theorem shows that any model of T_{TTT} is a model of TTT, by executing Hailperin's proof in the language of NF and transporting the result back to the language of TTT.

We will exhibit one such theory T here, with a list of twelve axioms. We have formally verified the consistency of T_{TTT}, and the relevant Lean proof for each axiom is linked. Our choice of axioms for the comprehension scheme are inspired by those used in the Metamath implementation of Hailperin's algorithm in [3]. In the following table, the notation $\langle a, b\rangle$ denotes the Kuratowski pair $\{\{a\}, \{a, b\}\}$. The first column is Hailperin's name for the axiom, and the second is (usually) the name from [3].

–	extensionality	∃∀	$\forall x^1.\forall y^1.(\forall z^0.z \in x \leftrightarrow z \in y) \to x = y$
P1(a)	intersection	∃∀	$\forall x^1 y^1.\exists z^1.\forall w^0.w \in z \leftrightarrow (w \in x \wedge w \in y)$
P1(b)	complement	∃∀	$\forall x^1.\exists z^1.\forall w^0.w \in z \leftrightarrow w \notin x$
P2	singleton image	∃∀	$\forall x^3.\exists y^4.\forall z^0 w^0.\langle\{z\},\{w\}\rangle \in y \leftrightarrow \langle z,w\rangle \in x$
–	singleton	∃∀	$\forall x^0.\exists y^1.\forall z^0.z \in y \leftrightarrow z = x$
P3	insertion two	∃∀	$\forall x^3.\exists y^5.\forall z^0 w^0 t^0.\langle\{\{z\}\},\langle w,t\rangle\rangle \in y \leftrightarrow \langle z,t\rangle \in x$
P4	insertion three	∃∀	$\forall x^3.\exists y^5.\forall z^0 w^0 t^0.\langle\{\{z\}\},\langle w,t\rangle\rangle \in y \leftrightarrow \langle z,w\rangle \in x$
P5	cross product	∃∀	$\forall x^1.\exists y^3.\forall z^2.z \in y \leftrightarrow \exists w^0 t^0.z = \langle w,t\rangle \wedge t \in x$
P6	type lowering	∃∀	$\forall x^4.\exists y^1.\forall z^0.z \in y \leftrightarrow \forall w^1.\langle w,\{z\}\rangle \in x$
P7	converse	∃∀	$\forall x^2.\exists y^2.\forall z^0 w^0.\langle z,w\rangle \in y \leftrightarrow \langle w,z\rangle \in x$
P8	cardinal one	∃∀	$\exists x^2.\forall y^1.y \in x \leftrightarrow \exists z^0.\forall w.w \in y \leftrightarrow w = z$
P9	subset	∃∀	$\exists x^3.\forall y^1 z^1.\langle y,z\rangle \in x \leftrightarrow \forall w^0.w \in y \to w \in z$

Remark 3.7. Axioms P1–P9 except for P6 are *predicative*: they stipulate the existence of a set with type at least that of all of the parameters. It is relatively straightforward to prove the consistency of predicative TTT, and we will see later that the proof of P6 in our model takes a different form to the proofs of the other axioms.

4 The supertype structure

We will now begin our exposition of the proof of the consistency of tangled type theory. This section is based on the `BaseType` and `Structural` directories in the formalisation.

4.1 Model parameters

As described in §3.3, the types of a given model of tangled type theory are indexed by a limit ordinal λ.

Definition 4.1. ∃∀ A collection of *model parameters* consists of types λ, κ, μ satisfying the following requirements.

(i) λ, κ, μ have well-orderings. The order type of λ is a limit ordinal. The order types of κ and μ are initial ordinals.[1]

(ii) The cardinalities of λ, κ, μ satisfy

$$\aleph_0 \leq |\lambda| < |\kappa| < |\mu|$$

(iii) $|\kappa|$ is a regular cardinal. $|\mu|$ is a strong limit cardinal, and μ has cofinality at least $|\kappa|$.

Definition 4.2. ∃∀ Sets smaller than size κ are called *small*.

Remarks 4.3.

(i) ∃∀ κ has an additive monoid structure given from its well-ordered structure as follows. Write $f : \kappa \to \mathrm{Ord}$ for the function that maps each inhabitant i of κ to the order type of $\{j \mid j < i\}$. Then f is an injection with image equal to the order type of κ. We may now define

$$i + j = f^{-1}(f(i) + f(j))$$

[1] Set theorists would simply require that κ, μ be cardinals. In our type theory, a cardinal is an equivalence class of types in a given universe that biject.

We cannot define this operation constructively, so we directly include it in the model parameters in the Lean formalisation and prove its existence separately.

(ii) ∃∀ We remark that these constraints are satisfiable; $\lambda = \aleph_0, \kappa = \aleph_1, \mu = \beth_{\omega_1}$ suffice.

4.2 Atoms, litters, and near-litters

As described in §1, our model has an additional level of objects below type zero. To index the levels of the model, together with this new level, we make the following definition.

Definition 4.4. ∃∀ A *type index* is an element of λ or a distinguished symbol $\bot$. We impose an order on type indices by setting $\bot < \alpha$ for all $\alpha \in \lambda$. The collection of type indices is denoted $\lambda^\bot$.

Elements of λ may be called *proper type indices*.

Our base type is a set of *atoms*, organised into *litters*. The litters index as large amorphous sets of atoms that can be used as 'junk' data.

Definition 4.5. ∃∀ A *litter* is a triple $L = (\nu, \beta, \gamma)$ where $\nu \in \mu$, β is a type index, and $\gamma \neq \beta$ is a proper type index.[2]

This somewhat arcane definition will be used later when defining the fuzz map. A litter $L = (\nu, \beta, \gamma)$ encodes data coming from type β and going into type γ. Note that β may be $\bot$, but γ may not; this corresponds to the fact that we never construct data in type $\bot$ from data at higher levels. The first component ν is an index allowing us to have μ distinct litters with the same source and target types.

Remark 4.6. ∃∀ There are precisely $|\mu|$ litters.

Definition 4.7. ∃∀ An *atom* is a pair $a = (L, i)$ where L is a litter and $i \in \kappa$. The *associated litter* of an atom is its first projection $\mathsf{pr}_1(a)$, written a° for brevity. The *litter set* $\mathsf{LS}(L)$ of a given litter L is the set of atoms whose associated litter is L; that is, $\mathsf{LS}(L) = \{(L, i) \mid i \in \kappa\}$. The litter sets partition the type of atoms into $|\mu|$ sets of κ atoms, and there are $|\mu|$ atoms in total.

Remark 4.8. Many of our constructions for symmetry arguments rely on having only a small set of constraints. If our constraints take the form of atoms, the smallness assumption guarantees that most of the atoms in a given litter set are unconstrained. Motivated by smallness concerns, we make the following definition.

Definition 4.9. ∃∀ A *near-litter* is a pair $N = (L, s)$ where L is a litter and s is a set of atoms with small symmetric difference to the litter set of L. We say that the *associated litter* of N is $N^\circ = \mathsf{pr}_1(N)$, or that N is *near* L. For brevity, we will frequently identify a near-litter with its underlying set.

Remarks 4.10.

(i) ∃∀ A set of atoms can be near at most one litter.

(ii) ∃∀ The litter set of any litter L can be made into a near-litter: $\mathsf{NL}(L) = (L, \mathsf{LS}(L))$.

(iii) ∃∀ Each (second projection of a) near-litter has size exactly κ.

Lemma 4.11. ∃∀ There are precisely $|\mu|$ near-litters.

Proof. It suffices to show that if $|\alpha|$ is a strong limit cardinal and α is well-ordered with initial order type, then α has precisely $|\alpha|$ bounded subsets. Indeed, this would imply that there are only $|\mu|$ small

[2] Readers of [6] will note that Holmes defines a litter as a particular set of atoms. We instead define litters to be names for such sets, which will be called *litter sets*.

sets of atoms, since the cofinality of μ is at least $|\kappa|$, from which it follows that there are precisely $|\mu|$ near-litters near a given litter, and so precisely $|\mu|$ in total. Note that

$$|\{s \subseteq \alpha \mid s \text{ bounded}\}| = \left|\bigcup_{i:\alpha} \mathcal{P}(\{j \mid j < i\})\right| \leq |\alpha| \cdot \sup_{i:\alpha} |\mathcal{P}(\{j \mid j < i\})|$$

As α has initial order type, $\{j \mid j < i\}$ has cardinality strictly less than $|\alpha|$. As $|\alpha|$ is a strong limit, its power set also has cardinality strictly smaller than $|\alpha|$, so the supremum is bounded above by $|\alpha|$. □

We can now define the allowable permutations of type $\bot$, although we will give them a different name for now; they will be precisely those permutations of atoms that respect the structure of near-litters.

Definition 4.12. ∃∀ A *base permutation* π consists of a permutation of atoms π^A and a permutation of litters π^L such that if $N = (L, s)$ is a near-litter, then

$$(\pi^L(L), \pi^A[s])$$

is also a near-litter. The type of base permutations will occasionally be denoted `Base`.

We will often simply use π to denote the permutations π^A, π^L of atoms and litters. Such a base permutation π induces a permutation of near-litters π^N, given by

$$\pi^N(L, s) = (\pi^L(L), \pi^A[s])$$

We will also call this permutation π.

Remarks 4.13.

(i) ∃∀ A base permutation is determined by its action on atoms.

(ii) ∃∀ Base permutations form a group.

Implementation details 4.14.

(i) Lean's type theory does not allow us to directly write π for the two different functions π^A, π^L. We instead use the syntax $\pi \bullet a, \pi \bullet L$, which is the `mathlib` notation for group actions. In these expressions, the head symbol is $\bullet$, not π itself. This allows Lean to trigger typeclass resolution and dynamically choose the interpretation of $\bullet$ based on the type of the right-hand side.

(ii) We use an alternative formalisation of the pointwise image $\pi^A[s]$ under a permutation; instead, we write $((\pi^A)^{-1})^{-1}(s)$. This has the advantage that the bi-implication

$$a \in ((\pi^A)^{-1})^{-1}(s) \leftrightarrow (\pi^A)^{-1}(a) \in s$$

is a definitional equality.

4.3 Higher type structure

A type-α object has elements of any type $\beta < \alpha$, which have elements of any type $\gamma < \beta$, and so on; we must eventually reach $\bot$ in a finite number of steps by well-foundedness. We will now make a definition to deal with sequences of type indices obtained in this way.

Definition 4.15. ∃∀ A *path of type indices* $\alpha \rightsquigarrow \varepsilon$ is a finite sequence of type indices

$$\alpha > \beta > \gamma > \cdots > \varepsilon$$

If α is a type index, an *α-extended type index* is a path $\alpha \rightsquigarrow \bot$. If A is a path $\alpha \rightsquigarrow \beta$ and B is a path $\beta \rightsquigarrow \gamma$, their composition $A \cdot B$ is a path $\alpha \rightsquigarrow \gamma$ obtained by concatenation of the sequences but removing the duplicated index β.

Implementation detail 4.16. We define a quiver structure on $\lambda^\bot$ by setting the type of morphisms $\alpha \to \beta$ to be the type $\alpha > \beta$; that is, there is a single morphism $\alpha \to \beta$ whenever $\alpha > \beta$. Paths of type indices are then defined as paths in this quiver. In `mathlib`, paths in quivers are defined as an inductive data type, with a `cons` operation on the end. That is, it is easy to reason about paths of the form $\alpha \rightsquigarrow \beta > \gamma$, but difficult to reason about paths of the form $\alpha > \beta \rightsquigarrow \gamma$. In our formalisation, we frequently append segments to the end of paths, but infrequently prepend to the start of paths. This gives rise to technical hurdles whenever analysing the start of paths is required. An alternative implementation is to encode paths as finite sets, as is done in [6], but this representation has worse definitional equalities.

Remark 4.17. ∃∀ For any $\alpha \in \lambda^\bot$, the set of α-extended type indices has size at most $|\lambda|$.

In our model, the iterated extensions of objects of type α are indexed by the α-extended type indices. We frequently want to analyse or modify an object 'along paths': to easily package such data, we make the following definition.

Definition 4.18. ∃∀ Let τ be a type and α be a type index. Then the type of *α-trees of τ* is

$$\mathsf{Tree}(\tau)_\alpha = (\alpha \rightsquigarrow \bot) \to \tau$$

That is, an α-tree of τ assigns an inhabitant of τ to each α-extended type index.

Trees made out of base permutations are called structural permutations.

Definition 4.19. ∃∀ An *α-structural permutation* is an α-tree of base permutations. We write

$$\mathsf{Str}_\alpha = \mathsf{Tree}(\mathsf{Base})_\alpha$$

An α-structural permutation π assigns a base permutation to each α-extended index.

Remarks 4.20.

(i) ∃∀ Trees on τ are given the group structure of τ by acting 'along paths': for α-trees a, b, we define $(ab)_A = a_A b_A$.

(ii) ∃∀ We can extend the notation a_A to paths $A : \alpha \rightsquigarrow \beta$ where β is arbitrary. Given such a path $A : \alpha \rightsquigarrow \beta$, we obtain a *derivative* group homomorphism

$$\mathsf{Tree}(\tau)_\alpha \to \mathsf{Tree}(\tau)_\beta$$

written $a \mapsto a_A$ and given by

$$(a_A)_B = a_{A \cdot B}$$

where $A \cdot B$ is the concatenation of the paths A, B. This operates similarly to a restriction map.

(iii) ∃∀ There is a canonical group isomorphism $\mathsf{Tree}(\tau)_\bot \cong \tau$, as there is precisely one path $\bot \rightsquigarrow \bot$. In particular, we may identify $\bot$-structural permutations with base permutations.

At proper type indices α, we will define the set of α-allowable permutations to be a certain subgroup of the group of α-structural permutations. These permutations will be chosen in such a way that gives an action on the set of model elements at level α. Not every structural permutation will have such an action.

References

[1] Mario Carneiro. *The Type Theory of Lean*. 2019. URL: https://github.com/digama0/lean-type-theory/releases.

[2] The mathlib Community. "The Lean Mathematical Library". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 367–381. ISBN: 9781450370974. DOI: 10.1145/3372885.3373824. URL: https://github.com/leanprover-community/mathlib4.

[3] Scott Fenton. *New Foundations set theory developed in metamath*. 2015. URL: https://us.metamath.org/nfeuni/mmnf.html.

[4] Theodore Hailperin. "A set of axioms for logic". In: *Journal of Symbolic Logic* 9.1 (1944), pp. 1–19. DOI: 10.2307/2267307.

[5] M. Randall Holmes. "The Equivalence of NF-Style Set Theories with "Tangled" Theories; The Construction of ω-Models of Predicative NF (and more)". In: *The Journal of Symbolic Logic* 60.1 (1995), pp. 178–190. ISSN: 00224812. URL: http://www.jstor.org/stable/2275515.

[6] M. Randall Holmes and Sky Wilshaw. *NF is Consistent*. 2024. arXiv: 1503.01406 [math.LO].

[7] Leonardo de Moura and Sebastian Ullrich. "The Lean 4 Theorem Prover and Programming Language". In: *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 625–635. ISBN: 978-3-030-79875-8. DOI: 10.1007/978-3-030-79876-5_37. URL: https://doi.org/10.1007/978-3-030-79876-5_37.

[8] W. V. Quine. "New Foundations for Mathematical Logic". In: *American Mathematical Monthly* 44 (1937), pp. 70–80. URL: https://api.semanticscholar.org/CorpusID:123927264.

[9] Ernst P. Specker. "The Axiom of Choice in Quine's New Foundations for Mathematical Logic". In: *Proceedings of the National Academy of Sciences of the United States of America* 39.9 (1953), pp. 972–975. ISSN: 00278424. URL: http://www.jstor.org/stable/88561.

[10] Ernst P. Specker. "Typical Ambiguity". In: *Logic, Methodology and Philosophy of Science*. Ed. by Ernst Nagel. Stanford University Press, 1962, pp. 116–123.

[11] Sky Wilshaw, Yaël Dillies, Peter LeFanu Lumsdaine, et al. *New Foundations is consistent*. 2022–2024. URL: https://leanprover-community.github.io/con-nf/.