New Foundations is consistent

Sky Wilshaw

July 2023

Abstract

We give a self-contained account of a version of Holmes' proof [2] that Quine's set theory *New Foundations* [4] is consistent relative to the metatheory ZFC. This is a 'deformalisation' of the formal proof written in Lean at [7].

Contents

Overview

In §1, we outline the context for the proof we will present. The mathematical background expected in subsequent sections will be limited to basic familiarity with cardinals and ordinals. We will then give an outline of the proof in §2. In §3, we introduce some basic preliminaries, and explicitly describe the structure within which our model will reside. The objects of our model will be constructed in §4.

All proofs given in §§3 and 4 are verified by the theorem prover Lean.

Note: At the present time, the formal proof [7] is incomplete, and this paper reflects the unfinished state of that proof. We aim to keep this paper proof in line with the formal proof, although as the project is ongoing, some variance is to be expected. The current version of the paper is available at https://zeramorphic.github.io/con-nf-paper/main.pdf.

1 The theories at issue

In 1937, Quine introduced *New Foundations* (NF) [4], 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 [2]. We will then describe the theory TTT, which we will use to prove our theorem.

1.1 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 $\texttt{type}(\text{`}x\text{'})$ 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 $\texttt{type}(\text{`}x\text{'}) = \texttt{type}(\text{`}y\text{'})$, and similarly a membership formula '$x \in y$' is well-formed precisely when $\texttt{type}(\text{`}x\text{'}) + 1 = \texttt{type}(\text{`}y\text{'})$.

The axioms of this theory are extensionality

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

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 1.1. (i) These are both axiom schemes, quantifying over all type levels n, and (in the latter case) over 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 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.

1.2 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

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

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 1.2. (i) It is important to emphasise that while the axioms come from a many-sorted theory, NF is not one; it 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) The infinite set of stratified comprehension axioms can be described with a finite set; this is a result of Hailperin [1].

(iv) Specker showed in [5] 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.

1.3 Tangled type theory

Introduced by Holmes in [3], *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. As in TST, each variable x has a type that it belongs to, denoted $\texttt{type}(\text{`}x\text{'})$. 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 $\texttt{type}(\text{`}x\text{'}) = \texttt{type}(\text{`}y\text{'})$. A membership formula '$x \in y$' is well-formed when $\texttt{type}(\text{`}x\text{'}) < \texttt{type}(\text{`}y\text{'})$.

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 a type variable α with $s(\alpha)$.

Remarks 1.3. (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 1.4 (Holmes)**.** NF is consistent if and only if TTT is consistent.

The proof is not long, but is outside the scope of this paper; it requires more model theory than the rest of this paper expects a reader to be familiar with, and relies on additional results such as those proven by Specker in [6].

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. 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.

2 Outline

To construct a model of tangled type theory, we build each type individually, and then prove that the resulting structure satisfies the required axioms. The process for building each type is complicated, and depends on some knowledge about the construction of the previous types. In the following subsections, we outline the construction the types, as well as the precise facts we need to carry through the inductive hypothesis at each stage.

2.1 Model parameters

As described in §1.3, the types of a given model of tangled type theory are indexed by a limit ordinal λ. Our model will also have two more cardinal parameters, denoted κ and μ, satisfying $\lambda < \kappa < \mu$.

Sets smaller than size κ will be called *small*. We require that κ is a regular cardinal; this ensures that small-indexed unions of small sets are small.

Each type in our model will have size μ. We require μ to be a strong limit cardinal; power sets of sets smaller than μ must also be smaller than μ. We stipulate that the cofinality of μ is at least κ. This assumption will become important whenever we consider objects indexed by small ordinals.

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

2.2 Atoms and permutations

To aid our construction, 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 collection 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.

2.3 Construction of each type

Objects in our model are defined by their elements at all lower type indices. However, not all collections of extensions may become model elements; for example, they may fail to satisfy extensionality at all levels simultaneously. We impose two restrictions on what kind of extensions an object may have.

The first restriction is that one of the extensions of a given object must be 'preferred', and every other extension must be easily derivable from that particular extension. This will help us to establish extensionality, as model elements will be the same if and only if their preferred extensions are the same. The system to compute other extensions uses a construction called the *fuzz* map. This map turns information about one extension into 'ordered junk' in another extension, in such a way that the

model cannot learn anything useful about the non-preferred extensions. Our allowable permutations will be defined as a set of permutations that respect the fuzz map.

The second restriction is that the object must have a small *support* comprised of *addresses*. That is, the behaviour of the object under the action of allowable permutations must be fully characterisable by the behaviour of a small set of addresses under allowable permutations. This will ensure that the objects of our model are not too complex. Because the cofinality of μ is at least κ, there are only μ small sets of elements taken from a collection of size μ; this observation will play a key role in establishing the sizes of our types.

2.4 Constraining the size of each type

The construction of a given type can only be done under the assumption that each smaller type was of size exactly μ. This means that we need to prove that each type has size μ in the inductive step. In order to do this, we will need to show that there are a lot of allowable permutations. The main theorem establishing this, called the *freedom of action theorem*, roughly states that under certain assumptions, a permutation defined on a small set of addresses can be extended to an allowable permutation. The majority of this paper will be allocated to proving the freedom of action theorem, and it will be outlined in more detail when we are in a position to prove it. One this is established, we can prove that the size of each type is precisely μ by carefully counting the possible ways to describe a model element.

2.5 Finishing the induction

We can then finish the inductive step and build the entire model. It remains to show that this is a model of TTT as desired. This part of the proof is quite direct, and also uses the freedom of action theorem.

3 The underlying structure

3.1 Atoms, litters, and near-litters

As described in §2.2, we have 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 3.1. 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 set 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*.

Definition 3.2. 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.

This somewhat arcane definition will be used to great effect 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 3.3. There are precisely μ litters.

Definition 3.4. 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 $\mathrm{pr}_1(a)$, 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 set of atoms into μ sets of κ atoms, and there are μ atoms in total.

Remark 3.5. Many of our constructions 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 are unconstrained. Motivated by smallness concerns, we make the following definition.

Definition 3.6. 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 $N^\circ = \mathrm{pr}_1(N)$, or that N is *near* L.

Remarks 3.7. (i) A set of atoms can be near at most one litter. For brevity, we will frequently identify a near-litter with its underlying set.

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

(iii) Each near-litter has size exactly κ, and there are μ near-litters in total; the latter follows from the fact that the cofinality of μ is at least κ.

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 3.8. A *near-litter permutation* π is a permutation of atoms that sends near-litters to near-litters.

Remarks 3.9. (i) A near-litter permutation π induces a permutation of litters, which we will also call π. This is defined by mapping L to the associated litter of $\pi'' \mathsf{LS}(L)$, where the double apostrophe denotes pointwise function application ($f''s$ denotes the set $\{f(x) \mid x \in s\}$). Thus, a near-litter permutation is simultaneously a permutation of atoms, litters, and near-litters.

(ii) The set of near-litter permutations forms a group under composition.

3.2 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 3.10. 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 $A \gg B$ is a path $\alpha \rightsquigarrow \gamma$ obtained by concatenation of the sequences but removing the duplicated index β.

Remark 3.11. For any $\alpha \in \lambda^\bot$, the set of α-extended type indices has size at most λ.

In our model, the iterated extensions of objects of type α are indexed by the α-extended type indices. We can apply operations to an object along each path $\alpha \rightsquigarrow \bot$. One important notion defined in this way will be *structural permutations*.

Definition 3.12. An *α-structural permutation* π assigns a near-litter permutation to each α-extended index. If A is an α-extended index, the assigned near-litter permutation is called the *derivative* of π

along A, and is denoted π_A. We can extend this notion to arbitrary paths $\alpha \rightsquigarrow \beta$; in this case, the derivative π_A is a β-structural permutation. The group of α-structural permutations is denoted S_α.

Remark 3.13. Near-litter permutations may be identified with $\bot$-structural permutations.

To apply an α-structural permutation π to some model element x of type α, we first consider its elements of some type $\beta < \alpha$. By recursion, we apply $\pi_{(\alpha>\beta)}$ to each such element, where $\pi_{(\alpha>\beta)}$ is the derivative of π along the one-step path from α to β. These images are then assembled to form a new object of type α; note however that this new object need not in general satisfy the constraints outlined in §2.3, so structural permutations need not map elements of the model to other elements of the model.

3.3 Addresses and supports

We are interested in objects that can be characterised by a small amount of data; this data will take the form of *addresses*.

Definition 3.14. Let α be a type index. An *α-address* is a pair (A, x) where A is an α-extended type index and x is either an atom or a near-litter.

An α-address encodes a set of atoms, and a path to get there from a type-α object by descending through iterated extensions.

Note. In [2], addresses are called *support conditions*.

Remark 3.15. The set of α-addresses has cardinality μ. This follows from remarks 3.7 and remark 3.11.

The group of α-structural permutations acts on the set of α-addresses by

$$\pi(A, x) = (A, \pi_A(x))$$

We would like to say that a *support* is a small set of α-addresses; for technical reasons that will be discussed later it will actually be better to make the following definition instead.

Definition 3.16. An α-support is a function S whose domain is a small ordinal and whose values are α-addresses, such that whenever N_1, N_2 are near-litters in the range of S, the atoms in their symmetric difference $N_1 \triangle N_2$ are also in the range of S. The *underlying set* of a support is its range.

Remarks 3.17.

(i) An address may occur multiple times in the range of a support, but all we usually care about is the presence or absence of an address. The fact that these addresses have indices in a given support will be useful for some constructions later. The symmetric difference condition is also a technical condition that we will use later, it avoids a problem where initial conditions for a construction could be overspecified.

(ii) For any small set of addresses s, there is a support S whose range is s together with the small set of atom addresses required to satisfy the symmetric difference constraint.

As with many parts of our construction, we will need to count precisely how many supports there are. We will first need to establish the following lemma.

Lemma 3.18. Let μ be a strong limit cardinal. Then for any $\kappa > 0$ strictly smaller than the cofinality of μ, we have

$$\mu^\kappa = \mu$$

Recall that *König's theorem* states that for an infinite cardinal μ, we have $\mu^{\mathrm{cf}(\kappa)} > \mu$; this lemma is a partial converse to this theorem in the case where μ is a strong limit.

Proof. Let A be a set of size κ, and let B be a set of size μ. We assume that B has a well-ordering, perhaps obtained from the order on μ. Given any function $f : A \to B$, we define a relation $\prec_f$ on A by the inverse image of the ordering on B under f:

$$a \prec_f b \iff f(a) < f(b)$$

One can show by induction on B that any for two functions $f, g : A \to B$, if $\operatorname{im} f = \operatorname{im} g$ and $\prec_f = \prec_g$, then $f = g$. The amount of pairs $(\operatorname{im} f, \prec_f)$ is bounded by μ. Indeed, the amount of possible images of a function $f : A \to B$ is bounded by μ as κ is less than the cofinality of μ and μ is a strong limit, and the amount of relations $\prec_f$ is bounded by $(2^\kappa)^\kappa$, which is bounded by μ as it is a strong limit. Thus, $\mu^\kappa \leq \mu$, and the converse is clear. □

We can now prove that there are exactly μ supports for any type index α.

Proposition 3.19. Let α be a type index. The set of α-supports has cardinality μ.

Proof. A support is a function from a small ordinal to the set of α-addresses, which has cardinality μ by remark 3.15. Thus, the amount of supports is bounded by

$$\sum_{i<\kappa} \mu^i$$

By lemma 3.18, each summand is precisely μ, and so the sum is precisely $\kappa \cdot \mu = \mu$. □

The α-structural permutations act on α-supports pointwise:

$$\pi(S)(i) = \pi(S(i))$$

We will need to concatenate supports together. The naïve concatenation of supports S_1, S_2 given by

$$S(i) = \begin{cases} S_1(i) & \text{if } i \in \operatorname{dom} S_1 \\ S_2(i - \operatorname{dom} S_1) & \text{otherwise} \end{cases}$$

need not satisfy the symmetric difference condition, so S may not be a support. There is not a unique way to expand this concatenation into a support, so we will instead make the following definition.

Definition 3.20. Let f be a function from a small ordinal whose values are α-support conditions. We say that a support S is a *completion* of f if $S|_{\operatorname{dom} f} = f$ and every address in the extension $S|_{\operatorname{dom} S \setminus \operatorname{dom} f}$ is an atom address (A, x) where $x \in N_1 \triangle N_2$ and $(A, N_1), (A, N_2) \in \operatorname{ran} f$. Every such function has a completion, as the amount of atom addresses in question is small. A support S is a *sum* of supports S_1, S_2 if it is a completion of the concatenation of S_1 and S_2. Every pair of supports has a sum.

4 Constructing the types

In this section, we describe the way in which we construct each type in the model, under the assumption that all previous levels were constructed properly.

4.1 Hypotheses

We explicitly describe the hypotheses required for the inductive step to succeed. The data constructed at each level is called *tangle data*; for uniformity we will also define tangle data at level $\bot$.

Definition 4.1. Let α be a type index. Then *tangle data* at level α consists of

(i) A set τ_α, called the set of *α-tangles*. Tangles encapsulate our model elements.

(ii) A group A_α, called the set of *α-allowable permutations*, which acts on τ_α. Allowable permutations will be denoted using the symbol ρ.

(iii) A group homomorphism $A_\alpha \to S_\alpha$, where S_α is the group of α-structural permutations as defined in definition 3.12. We will notationally suppress this homomorphism, and treat A_α as a subgroup of S_α, so allowable permutations act on anything that structural permutations do.

(iv) A function supp that assigns an α-support to each α-tangle. This support must actually support the input tangle: for any α-allowable permutation ρ, if ρ fixes $\mathsf{supp}\, t$ pointwise, it fixes t.

$$(\forall i.\, \rho((\mathsf{supp}\, t)(i)) = (\mathsf{supp}\, t)(i)) \to \rho(t) = t$$

We require that supp commutes with α-allowable permutations ρ in the sense that

$$\mathsf{supp}(\rho(t)) = \rho(\mathsf{supp}\, t)$$

The tangle data at level $\bot$ is constructed by taking the tangles to be the atoms, the allowable permutations to be the near-litter permutations, and the support function to be given by

$$\mathrm{dom}(\mathsf{supp}\, a) = 1; \quad (\mathsf{supp}\, a)(0) = (\varnothing, a)$$

where $\varnothing$ is the empty path $\bot \nrightarrow \bot$.

At higher levels $\alpha \in \lambda$, the tangles will be pairs (x, S) where x is a model element of type α and S is a support that supports x under the action of α-allowable permutations. There are μ tangles that encapsulate any given model element. Allowable permutations act on these pairs pointwise.

$$\rho((x, S)) = (\rho(x), \rho(S))$$

It is clear to see that the second projection pr_2 satisfies the requirements of supp defined in (iv) above are satisfied.

Definition 4.2. Let α be a type index with tangle data. Then a *position function* at level α is an injection $\iota_\alpha : \tau_\alpha \to \mu$. For convenience, we will simply write ι for all position functions.

At level $\bot$, the position function is chosen arbitrarily; one exists as there are exactly μ atoms. At higher levels $\alpha \in \lambda$, the existence of a position function depends on the fact that τ_α has size at most μ. We will typically assume that all type levels previously constructed have a position function.

Let $\alpha \in \lambda$, and let N be a near-litter. We will construct our model in such a way that there is an element x of type α with a $\bot$-extension that is exactly (the atoms inside) N. This will imply, in particular, that each type must have size at least μ by remarks 3.7. Together with the existence of a position function, the size of each type can be seen to be exactly μ. We cannot express the notion of a $\bot$-extension directly with the language available to us, but we do not need it in order to capture the information relevant for this section.

Definition 4.3. Let α be a proper type index with tangle data. We say that α has *typed near-litters* if there is an injection typed_α from the set of near-litters to τ_α, commuting with allowable permutations in the following sense.

$$\rho(\mathsf{typed}_\alpha N) = \mathsf{typed}_\alpha(\rho_{(\alpha > \bot)}(N))$$

4.2 The fuzz map

We will now perform a construction that underpins the rest of the model, and helps us enforce extensionality. In tangled type theory, a given model element has extensions at each level below it. In our model, each object has a 'preferred' extension, and must find a way to compute the other extensions from that information. In order to do this, we need to be able to convert arbitrary model elements into 'junk' at other levels, which can then be interpreted as a 'non-preferred' extension.

The *fuzz maps* perform this task. They are parametrised by type indices $\beta \neq \gamma$ where $\gamma \neq \bot$, representing the source type level and the target type level. For each such pair, the fuzz map is an injection from β-tangles to litters. An arbitrary litter can only be the image of a fuzz map defined at a single pair of type levels.

Treating the output of the fuzz map as a typed near-litter, its position is always greater than the position of the input to the function. A similar property holds for atoms. This ensures a well-foundedness condition that we can use in many places later.

Definition 4.4. Let β be a type index, and let $\gamma \neq \beta$ be a proper type index. Suppose that β, γ have tangle data and a position function, and that γ has typed near-litters. A *fuzz map* $f_{\beta,\gamma}$ is an injection from τ_β to the set of litters, such that

(i) For any $t \in \tau_\beta$, $f_{\beta,\gamma}(t) = (\nu, \beta, \gamma)$ for some $\nu \in \mu$, so all of the fuzz maps have pairwise disjoint ranges.

(ii) For any $t \in \tau_\beta$ and any near-litter N near $f_{\beta,\gamma}(N)$,

$$\iota_\beta(t) < \iota_\gamma(\mathsf{typed}_\gamma N)$$

(iii) For any $t \in \tau_\beta$ and atom a in the litter set corresponding to $f_{\beta,\gamma}(t)$,

$$\iota_\beta(t) < \iota_\bot(a)$$

We will now show that such a function exists, and will thenceforth refer to it as *the* fuzz map.

Proposition 4.5. Let β be a type index, and let $\gamma \neq \beta$ be a proper type index. Suppose that β, γ have tangle data and a position function, and that γ has typed near-litters. Then a fuzz map $f_{\beta,\gamma}$ exists.

Proof. We construct the map by recursion along τ_β, with the order induced by ι_β. Suppose we have already constructed $f_{\beta,\gamma}$ for $s \in \tau_\beta$ with $\iota(s) < \iota(t)$. We must choose an index $\nu \in \mu$ subject to three constraints:

(i) ν was not picked for any previous s;

(ii) $\iota(t)$ is greater than $\iota_\gamma(\mathsf{typed}_\gamma N)$ for any near-litter near (ν, β, γ);

(iii) $\iota(t)$ is greater than $\iota_\bot(a)$ for any atom a with $a^\circ = (\nu, \beta, \gamma)$.

Each constraint denies us less than μ choices, and so there is always an available index ν to choose. □

4.3 Codes and clouds

From now, we will assume that $\alpha \in \lambda$ is a fixed proper type index at which we intend to construct tangle data. In this section, all other (proper) type indices will be assumed to be strictly less than

α. We assume that we have tangle data, position functions, and typed near-litters for all type indices below α.

We will now begin the process of stitching together the fuzz maps to form the function that deduces an alternative extension from a preferred extension.

Definition 4.6. Let $\beta < \alpha$ be a type index, and let $\gamma < \alpha$ be a proper type index not equal to β. The *cloud map* $c_{\beta,\gamma} : \mathcal{P}(\tau_\beta) \to \mathcal{P}(\tau_\gamma)$ is given by

$$c_{\beta,\gamma}(s) = \mathsf{typed}_\gamma{}'' \bigcup_{t \in s} \{N \mid N^\circ = f_{\beta,\gamma}(t)\}$$

For each $t \in s$, the map $c_{\beta,\gamma}$ produces the 'cloud' of near-litters near to $f_{\beta,\gamma}(t)$, and turns them into tangles through the typed near-litter map. This will be used to construct the alternative extension map.

Note. In [2], the cloud map is called $A_{\beta,\gamma}$, but in this paper the notation $A_{\beta,\gamma}$ is reserved for the alternative extension map.

Remark 4.7. The cloud map is injective, and further, if $c_{\beta,\gamma}(s) = c_{\beta',\gamma}(s')$ and s is nonempty, then $\beta = \beta'$. In particular, a nonempty set of γ-tangles has at most one inverse image under any cloud map $c_{\beta,\gamma}$. We will show that there are only finitely many iterated images under an inverse cloud map, and we will use this to define preferred extensions. To prove this, we will make the following definition that encapsulates the notion of a particular choice of $\beta < \alpha$ and an extension at that level.

Definition 4.8. Let $\alpha \in \lambda$. An *α-code* is a pair (β, s) where $\beta < \alpha$ is a type index, and $s \subseteq \tau_\beta$.

Definition 4.9. Let $\beta < \alpha$ be a proper type index. The *code cloud map* C_β is a map from α-codes to α-codes, given by

$$C_\beta((\gamma, s)) = \begin{cases} (\beta, s) & \text{if } \gamma = \beta \\ (\beta, c_{\gamma,\beta}(s)) & \text{if } \gamma \neq \beta \end{cases}$$

That is, C_β applies the relevant cloud map to obtain a code describing a β-extension, or does nothing if the first projection of the code is already β.

Observe that C_β is injective on codes with first projection not equal to β. We will show that there are only finitely many iterated images under the inverse of C_β.

Definition 4.10. Define a relation $\rightsquigarrow$ on α-codes by letting $x \rightsquigarrow C_\beta(x)$ whenever $\mathsf{pr}_1(x) \neq \beta$.

Remark 4.11. A code x has at most one predecessor under $\rightsquigarrow$. If $x \rightsquigarrow y$, then x is empty (more precisely, its second projection is empty) if and only if y is. Moreover, all empty codes are related to each other under $\rightsquigarrow$.

Lemma 4.12. The relation $\rightsquigarrow$ is well-founded on nonempty codes.

Proof. For each nonempty α-code x, we define an ordinal $\iota(x) \in \mu$ given by the smallest position of any tangle in $\mathsf{pr}_2(x)$.

$$\iota((\beta, s)) = \min_{t \in s} \iota_\beta(t)$$

We show that if $x \rightsquigarrow y$, then $\iota(x) < \iota(y)$. Let $x = (\beta, s)$ and $y = (\gamma, c_{\beta,\gamma}(s))$. Suppose $t \in \tau_\gamma$ is the tangle with smallest position in $c_{\beta,\gamma}(s)$. Then $t = \mathsf{typed}_\gamma N$ where $N = f_{\beta,\gamma}(t')$ for some $t' \in s$. It suffices to show $\iota_\beta(t') < \iota_\gamma(t)$, but this holds by the construction of the fuzz map in definition 4.4. □

An object in our model will correspond to a collection of codes, representing its extensions. A given object at level α has exactly one extension at every proper type index $\beta < \alpha$, and may optionally have a $\perp$-extension.

By lemma 4.12, the set of nonempty α-codes forms a tree. Each code has a finite height; a code with no $\rightsquigarrow$-predecessor is of height zero, and the height of any other code can be computed recursively. We will partition the tree of nonempty α-codes into smaller trees of height 1, and use those as equivalence classes of codes representing a single model element.

Definition 4.13. We define an equivalence relation $\equiv$ on nonempty α-codes, generated by the assertion that $x \equiv y$ whenever $x \rightsquigarrow y$ and x has even height.

Remarks 4.14. (i) If two codes x, y are equivalent, then either $x = y$, or x has even height and $x \rightsquigarrow y$ (or vice versa), or both x, y have odd height and there is some z with $z \rightsquigarrow x, y$.

(ii) Each equivalence class contains precisely one even code. This will be the 'preferred' extension of the corresponding model element, as outlined in §2.3.

(iii) Each equivalence class contains exactly one code for each proper type index $\beta < \alpha$. If the equivalence class contains a code at level $\perp$, it must be the even code, because such codes can never be produced by the code cloud map.

4.4 Model elements

We now collate equivalence classes of codes to create our model elements.

Definition 4.15. An *α-preobject* x assigns to each proper type index $\beta < \alpha$ a set of β-tangles x_β, in such a way that either

(i) there is a set s of atoms such that $x_\beta = c_{\perp,\beta}(s)$ for all $\beta < \alpha$; or

(ii) there is a proper type index $\beta < \alpha$ such that (β, x_β) is an even code and $x_\gamma = c_{\beta,\gamma}(x_\beta)$ for all $\gamma \neq \beta$.

Thus nonempty preobjects correspond to equivalence classes of nonempty codes, and there is precisely one empty α-preobject which corresponds to all of the empty codes. An even representative code can be easily extracted from any preobject.

These preobjects satisfy the required form of extensionality in tangled type theory.

Theorem 4.16. Let x, y be α-preobjects, and let $\beta < \alpha$ be a proper type index such that $x_\beta = y_\beta$. Then $x = y$.

Note that applying metatheoretic extensionality to x_β and y_β strengthens this assertion into the proper form.

Proof. First, note that if x and y have the same representative (even) code, then they are equal. This follows from the fact that the extensions of a preobject can be calculated by appling the code cloud map to the representative code. We have three cases.

(i) *The extensions of x and y can both be calculated from a set of atoms.* Let s_x, s_y be the set of atoms for x, y respectively. Then $c_{\perp,\beta}(s_x) = x_\beta = y_\beta = c_{\perp,\beta}(s_y)$, hence $s_x = s_y$, so x and y have the same representative code.

(ii) *The extensions of x can be calculated from a set of atoms s, and the extensions of y can be calculated from its γ-extension.* We must show that $(\perp, s) \equiv (\gamma, y_\gamma)$. Suppose $\beta = \gamma$, so $c_{\perp,\beta}(s) = x_\beta = y_\beta$.

In this case, $(\bot, s) \equiv (\beta, x_\beta)$ by assumption, giving the result. Instead, suppose $\beta \neq \gamma$. In this case,

$$(\gamma, y_\gamma) \equiv (\beta, c_{\gamma,\beta}(y_\gamma)) = (\beta, y_\beta) = (\beta, x_\beta) \equiv (\bot, s_x)$$

The case where x and y are swapped holds by symmetry.

(iii) *The extensions of x can be calculated from its γ-extension, and the extensions of y can be calculated from its δ-extension.* We must show $(\gamma, x_\gamma) \equiv (\delta, y_\delta)$. We have

$$(\gamma, x_\gamma) \equiv (\beta, x_\beta) = (\beta, y_\beta) \equiv (\delta, y_\delta)$$

□

We want to define the α-objects to be those α-preobjects with a small support under the action of α-allowable permutations, so we must first define this group. The allowable permutations will be built up from allowable permutations at lower levels. These will become the derivatives along the one-step paths $\alpha > \beta$. These permutations are constrained so as to commute with the fuzz map.

Definition 4.17. An *α-allowable permutation* ρ assigns to each type index $\beta < \alpha$ a β-allowable permutation $\rho_{(\alpha>\beta)}$, in such a way that

$$(\rho_{(\alpha>\gamma)})_{(\gamma>\bot)}(f_{\beta,\gamma}(t)) = f_{\beta,\gamma}(\rho_{(\alpha>\beta)}) \qquad (*)$$

for all type indices β, all proper type indices $\gamma \neq \beta$, and all β-tangles t. The homomorphism to the group of α-structural permutations is given by mapping each of the one-step derivatives $\rho_{(\alpha>\beta)}$ to structural permutations and collating the results.

Remark 4.18. (i) Note that the appearance of β-allowable permutations in this definition is not circular; these are assumed to exist in the tangle data assigned to level β. We will later stipulate that the allowable permutations obtained through tangle data at lower levels already satisfy this condition.

(ii) A small calculation reveals that condition $(*)$ enforces, for any allowable ρ, that for any set $s \subseteq \tau_\beta$ and $\gamma \neq \beta$ proper,

$$\rho_{(\alpha>\gamma)}{}''c_{\beta,\gamma}(s) = c_{\beta,\gamma}(\rho_{(\alpha>\beta)}{}''s)$$

Hence, for any code x, we have $\rho(C_\beta(x)) = C_\beta(\rho(x))$, and so allowable permutations preserve code equivalence. This suggests that they are a sensible choice for permutations that respect the structure of α-preobjects.

We are now in a position to define our model elements.

Definition 4.19. An *α-object* is an α-preobject that admits an α-support that supports it under the action of α-allowable permutations. That is, x is an object if there is some support S such that $\rho(S) = S$ implies $\rho(x) = x$. An *α-tangle* is a pair (x, S) where x is an α-object and S is an α-support that supports x under the action of allowable permutations.

We have thus defined tangle data at level α. We can also immediately define typed near-litters at level α, by considering the preobject x given by the even code $(\bot, N)$. This evidently has a small support S given by

$$\operatorname{dom} S = 1; \quad S(0) = ((\alpha > \bot), N)$$

and thus this is an α-object. The map typed_α thus sends N to the tangle (x, S).

References

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

[2] M. Randall Holmes. *NF is Consistent*. 2023. arXiv: 1503.01406 [math.LO].

[3] 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.

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

[5] 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.

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

[7] Sky Wilshaw et al. *The consistency of New Foundations*. 2022–2023. URL: https://leanprover-community.github.io/con-nf/.