The construction

Randall Holmes

time stamped 1:25 PM 2/25/2016: corrected issues related to the error in definition of extension. Also corrected a nasty mental slip in the definition of relative type for set codes.

We are working in plain ZFC, with no considerations about atoms yet.

Parameters of the construction and related immediate definitions:

We recall that we have fixed a limit ordinal \( \lambda \).

If \( A \) is a finite subset of \( \lambda \), we define \( A_1 \) as \( A \setminus \{ \min(A) \} \). \( A_0 \) is defined as \( A \); \( A_{n+1} \) is defined as \( (A_n)_1 \). We write \( A << B \) and say that \( A \) downward extends \( B \) iff \( B \subseteq A \) and all elements of \( A \setminus B \) are less than all elements of \( B \).

We recall that we have fixed an uncountable regular cardinal \( \kappa \). We call sets of cardinality \( < \kappa \) small and all other sets large.

Basic classes and equivalences introduced:

We will recursively define classes \( C(A) \), \( N(A) \) and \( Q(n, A) \) for each finite subset \( A \) of \( \lambda \) and each natural number \( n \) with \( 0 \leq n \leq |A| + 1 \). These classes are in fact sets, which will be proved in due course. We will call these basic classes. We may talk about \( C \)-basic classes, \( N \)-basic classes, \( Q \)-basic classes.

We will define an equivalence relation \( \sim \) on each of these classes. The equivalence relations can all be denoted by the same symbol because the classes are disjoint. We may refer to these relations as “equivalence”.

Complexity of a \( Q \)-basic class: Each class \( Q(n, A) \) with \( n > 0 \) is assigned a complexity which is the minimum element of \( A_{n-1} \) if \( A_{n-1} \) is nonempty and otherwise \( \lambda \).
Description of C-basic classes: An element of \( C(A) \) will be of the form 
\((1, a, \alpha, A)\), where \( a \) is either an element of \( Q(|B| - |A| + 1, B) \) for some 
\( B << A \), or an element of \( C(A_1) \) if \( A \) is nonempty. or a small ordinal 
if \( A \) is empty.

\((1, a, \alpha, A) \sim (1, b, \beta, A)\) iff \( a \) and \( b \) are equal small ordinals or if \( a \) and 
\( b \) belong to the same basic class and \( a \sim b \).

Definitions and observations re C-basic classes: We say that \( a \) is the 
parent of \((1, a, \alpha, A)\).

We define \( L(a, A) \) as \( \{ (1, a, \alpha, A) : \alpha < \kappa \} \), where \((1, a, \alpha, A) \in C(A) \).

Notice that each \( C(A) \) contains a large collection of elements with 
iterated parent a small ordinal.

Description of N-basic classes: An element of \( N(A) \) is of the form \((2, a, D)\) 
where \((1, a, 0, A) \in C(A) \) and \( D \) is a small subset of \( C(A) \), and no two 
distinct elements of \( D \) stand in the relation \( \sim \) to each other.

\((2, a, D) \sim (2, b, E)\) iff \( a \sim b \) and each element of \( D \) stands in the 
relation \( \sim \) to some element of \( E \) and each element of \( E \) stands in the 
relation \( \sim \) to some element of \( D \).

Definitions re N-basic classes: We call \( a \) the parent of \((2, a, D)\).

An element of \( C(A) \) is a formal element of \((2, a, D)\) iff it has parent \( a \) 
and does not stand in the relation \( \sim \) to any element of \( D \), or if it is an 
element of \( D \) and does not stand in the relation \( \sim \) to any element of 
\( L(a, A) \).

Preliminary description of Q-basic classes: Each element of \( Q(n, A) \) is 
of the form \((3, f, L)\) where \( f \) is a function code (to be explained) and 
\( L \) is an argument list (to be explained).

Every argument list is a function from a set of small ordinals to the 
union of the classes \( C(A) \) and \( N(A) \) for all \( A \). Values at distinct ordinals 
of an argument list belonging to the same \( C(A) \) do not stand in the 
relation \( \sim \). If \( \alpha \neq \beta \) and \( L(\alpha) \) and \( L(\beta) \) both belong to the same \( N(A) \), 
then no formal element of \( L(\alpha) \) stands in the relation \( \sim \) to any formal 
element of \( L(\beta) \).

In addition, each argument list belongs to some argument list type (to 
be explained).
Definition (argument list types): We proceed to define argument list types.

An argument list type \( T \) is determined by three components, a domain \( D \), the common domain of all of its elements, a function \( A \) with domain \( D \) and a function \( R \) with domain \( D \). The triple \( (D, A, R) \) is used as a code for the argument list type (and such objects are called argument list type codes).

\( D \) will be a set of small ordinals. Each element of \( T \) has domain \( D \).

For each \( \alpha \in D \), \( A(\alpha) \) will be of the form \((1, A)\), in which case \( L(\alpha) \in C(A) \) for each \( \alpha \in D \), or the form \((2, A)\), in which case \( L(\alpha) \in N(A) \) for each \( L \in T \).

For each \( \alpha \in D \), \( R(\alpha) \) will be of one of the following forms.

1. \( A(\alpha) = (1, A) \) and \( R(\alpha) = \beta < \alpha \): if \( \beta \in D \), then \( A(\beta) = (2, A) \) and \( L(\alpha) \) is a formal element of \( L(\beta) \) for each \( L \in T \); if \( \beta \notin D \) \( L(\alpha) \) is not equivalent to any formal element of any \( L(\gamma) \) belonging to \( N(A) \), for each \( L \in T \).

2. \( A(\alpha) = (2, A) \) and \( A = \emptyset \) and \( R(\alpha) = (1, \beta), \beta < \kappa \), in which case \( L(\alpha) \) has parent \( \beta \) for each \( L \in T \).

3. \( A(\alpha) = (2, A) \) and \( R(\alpha) = (4, f, E) \), \( [4 \text{ is used strictly to avoid a } \text{pun with } (3, f, E)] \) where \( E \subseteq D \cap \alpha \) and \( f \) is a function code such that \( (3, f, L[E]) \) belongs to a suitable \( Q(m, B) \), and the parent of \( L(\alpha) \) is \( (3, f, L[E]) \in \text{some } Q(m, B) \) with \( m > 0 \), or, in the case where \( (3, f, L[E]) \in Q(0, A_1) \), the element of \( C(A_1) \) to which \( (3, f, L[E]) \) reduces [see below for definition], for each \( L \in T \).

4. \( A \) is nonempty, \( A(\alpha) = (2, A) \) and \( R(\alpha) = (4, (1, \beta, T, A_1), E) \), where \( (1, \beta, T, A_1) \) is an element of \( Q(0, A_1) \) as described below, \( \pi_1(T) = E \) is a subset of \( \alpha \), \( \beta \in E \), \( \beta \notin D \), and the value of \( A(\gamma) \) for each \( \gamma \in D \cap E \) is \( (1, A_1) \). In this case, the parent of \( L(\alpha) \) is an element of \( C(A_1) \) not equivalent to any element of the range of \( L \), for each \( L \in T \). [The reason for the form here can be seen in the definition of function codes with output type \( Q(0, A) \) below: \( (1, \beta, E, A_1) \) is a typical function code with output type \( Q(0, A_1) \)].

The argument lists in an argument list type are exactly the ones which satisfy these conditions for the given \( D, A, R \) determining the type.
The pointer information in cases 1 and 4 of the conditions on the function \( R \), which appears to give type information relative to positions not in the list under consideration, becomes relevant when one argument list is considered as a sublist of another.

**Description of typing of function codes:** A function code \( f \) has an input type, which is an argument list type as just defined, and an output type, which is a \( Q(n, A) \). \((3, f, L) \in Q(n, A)\) iff \( L \) belongs to the input type of \( f \). Further, if \( L \) belongs to the input type of \( f \), all range elements of \( L \) must belong to some \( C(B) \) or \( N(B) \) with \( B << A_{n-1} \) if \( n > 0 \), or simply to \( C(A) \) if \( n = 0 \). This imposes a strong restriction on what input types go with what output types.

**Definition (extension of an argument list):** We say that an argument list \( L \) extends an argument list \( M \), relative to specific assignments of types with codes \((D_L, A_L, A_R)\) and \((D_M, A_M, R_M)\) respectively to \( L \) and \( M \) respectively, iff \( L \) and \( M \) agree on the common part of their domain and all domain elements of \( M \) which are not domain elements of \( L \) exceed all domain elements of \( L \), and \( L \cup M \) is an argument list of type coded by \((D_L \cup D_M, A_L \cup A_M, R_L \cup R_M)\) [that this is a well-formed argument list code imposes consistency conditions on the argument list types assigned to \( L \) and \( M \)]. It is important to notice that the domain of \( M \) does not have to include the domain of \( L \).

**Description of basic classes \( Q(0, A) \):** If \( n = 0 \), all function codes with output type \( Q(0, A) \) are of the form \((1, \alpha, T, A)\), where \( T = (D, A, R) \) is an argument list type code and \( \alpha \in D \). The function \( A \) must have range \( \{(1, A)\} \). We say that \((3, (1, \alpha, T, A), L)\) is well formed iff \( L \in \delta(T) \) and reduces to \( L(\alpha) \) [this notion of reduction is the one used above] and \((3, (1, \alpha, T, A), L) \sim (3, (1, \beta, V, A), M)\) iff \( L(\alpha) \sim M(\beta) \).

We have thus defined all elements of the classes \( Q(0, A) \).

**Description of elements of basic classes \( Q(n + 1, A) \):** If \( n > 0 \), all function codes with output type \( Q(n, A) \) are of the form \((2, U, T)\), where \( T \) is the code for an argument list type and \( U \) is a nonempty set of function codes with output type \( Q(n - 1, A) \) [with some restrictions on input type of elements of \( U \) to be explained]. If \( L \) belongs to the input type coded by \( T \), all range elements of \( L \) must belong to some \( C(B) \) or \( N(B) \) with \( B << A_{n-1} \).
(3, (2, U, T), L) is an element of Q(n−1, A) iff L belongs to the argument list type coded by T.

(3, g, M) ∈ Q(n−1, A) is a formal element of (3, (2, U, T), L) iff g ∈ U and M extends L in the formal sense defined above, with the type associated with L being the type T internal to f and the type associated with g being the argument list type whose code appears as a component of g. The formal requirement on U is as follows: let T = (DT, AT, RT); for each g ∈ U, let the argument list type code appearing as a component of g be (Dg, Ag, Rg); all elements of Dg not belonging to DT must be larger than all elements of DT, and (DT ∪ Dg, AT ∪ Ag, RT ∪ Rg) must be an argument list type code (which imposes consistency requirements on the component functions, among other things).

Notice that range elements in L of excessively complex type cannot appear as range elements in M. This is part of what makes the recursion work (and enforces sethood of the various classes mentioned).

**Equivalence on basic classes Q(n + 1, A):** We now define the relation ∼ on Q(n, A)(n > 0).

**Definition (item in an argument list):** We define an item in an argument list L as an element of a C(A) which appears either as a range element of L, or as the second component a or an element of the third component D of a (2, a, D) in the range of L.

**Definition (item remote from an argument list):** We say that (1, a, α, A) ∈ C(A) is remote from an argument list L if it has an iterated parent which is a small ordinal and it has no iterated parent which is an item in L and it is not an item in L. Notice that computations of ∼ on such objects are trivial.

**Definition (novel item in an argument list):** We say that a notation (1, a, α, A) is a novel item in an argument list L iff it is a range element which is not a formal element of any element of N(A) in the range of L, or it is the parent of a range element in L and is not itself an element of the range of L, or it is an element of the third component D of an element (2, a, D) of N(A) in the range of L. We may briefly refer to a range element of an argument list in an N(A) as novel if its parent is novel.
Observations about novel items: Note that a non-novel element $L(\alpha)$ of the range of an argument list $L$, if in a $C(A)$, either has parent a small ordinal or is a formal element of an $L(\beta)$ for $\beta < \alpha$ and so either has parent the second component of $L(\beta)$ or is an element of the third component of $L(\beta)$, or, if in an $N(A)$, has parent either a small ordinal, or an $L(\beta) \in C(A_1)$ for $\beta < \alpha$, or parent $g(M)$ where $g$ is a function code appearing as a component of the code for the argument list type of $L$ and $M$ is a sublist of $L \cap \alpha$. The novel items are the only ones which introduce new material not found earlier in the argument list or in the code for its type.

Definition (bounded formal elements): We say that $(3, g, P)$ is an $M$-bounded formal element of $(3, (2, U, T), L)$ iff $M$ is an argument list, $(3, g, P)$ is a formal element of $(3, (2, U, T), L)$ and each novel item in $P$ (which we know extends $L$) is either an item in $L$, an item in $M$, or remote from both $L$ and $M$, or if it has parent equivalent to the parent of the element of $N(B)$ in whose third component it occurs, has exactly the same parent as the element of $N(B)$.

Observations about the definition of bounded formal elements: With notation as in the definition, each range element $P(\alpha)$ of $P$ is either an element of the third component of an element of an $N(B)$ appearing as $P(\beta)$ for $\beta < \alpha$, or an element of a $C(B)$ with iterated parent either an item in $L$, an item in $M$, or a small ordinal, or an element of an $N(B)$ with parent a small ordinal, or an element of an $N(B)$ with parent an element of $C(B_1)$ with an iterated parent either an item in $L$, an item in $M$, or a small ordinal, or a $(3, h, Q)$ with $h$ a function code in the transitive closure of $g$ and so of $(3, (2, U, T), L)$ and $Q$ a sublist of $P \cap \alpha$ – range elements of $Q$ thus being similarly bounded in complexity.

Definition (linkage of elements of $Q(n + 1, A)$): We say that $x = (3, (2, U, T), L) \in Q(n, A)$ is linked to $y = (3, (2, U', T'), L') \in Q(n, A)$ iff each $L$-bounded formal element of $y$ stands in the relation $\sim$ to some $L'$-bounded formal element of $x$ and each $L'$-bounded formal element of $x$ stands in the relation $\sim$ to some $L$-bounded formal element of $y$.

Observations re the definition of linkage: The device of bound-
edness is a method of avoiding having to consider equivalences between notations in $C(A)$’s with parents of unbounded complexity. Notice that formal elements of an element of $Q(n, A)$ are in $Q(n - 1, A)$, and this will eventually bottom out at $Q(0, A)$, reducing to $C(A)$.

The danger here is that formal elements involve extended argument lists: but the bounding ensures that nothing really new can be added. If we know how to compute $\sim$ on all simpler objects, we do know how to compute it on everything added in the extensions of argument lists here, but this has to be shown.

**Definition of equivalence on** $Q(n + 1, A)$: We say that $x \sim y$ for $x, y \in Q(n, A)$, $n > 0$ iff there is a finite sequence $z$ with $z(0) = x$, $z(m) = y$, $z(i)$ linked to $z(i + 1)$ for each $i$. [The linkage relation is actually transitive (I believe) but one cannot prove this before the $\sim$ relation is defined, so I state things this way to ensure that it is clear that $\sim$ is an equivalence relation].

I claim that I have fully defined the classes $C(A)$, $N(A)$, $Q(n, A)$, and the relations $\sim$ on each class. Moreover $\sim$ is an equivalence relation. All computations of $\sim$-equivalence involve further computations of $\sim$-equivalence on simpler objects, eventually bottoming out in equality of small ordinals or of hereditarily finite sets. This needs to be verified as the recursion in this definition is complicated.

**Definition (complexity of $Q$-basic classes and function codes):** We assign a complexity to each set $Q(A, n)(n > 0)$ and the same complexity to function codes with output type $Q(A, n)$, namely the minimum element of $A_{n-1}$, or $\lambda$ if $A_{n-1}$ is empty.

**Classification of range elements in an argument list by basic class:** We observe, using the fact that every argument list belongs to an argument list type, that any element $L(\alpha)$ of the argument list component of $(3, f, L)$ where $f$ has output type $Q(A, n)$ is either an element of a $C(B)$, which can be novel (not equivalent to anything in the range of $L$), or a formal element of an $L(\beta) = (2, a, D)$ for $\beta < \alpha$, either an element of $D$ (and so novel) or the result of affixing an ordinal index to $a$ (as it were), or on the other hand an element of $N(B)$, with parent either a small ordinal (if $B$ is empty), or an element of $C(B_1)$ not equivalent to
anything in the range of \( L \) (and so novel) or an element of \( C(B_1) \) appearing as an \( L(\beta) \), or a \((2,g,M) \in Q(m,D)\), where \( M \subseteq L \cap \alpha \). Now we know that \( B << A_{n-1} \), and we know that \( m = |D| - |B| + 1 \), so the complexity of the type of \((2,g,M)\), and so of \( g \), is the minimum of \( D_{|D|-|B|+1-1} \), that is, the minimum element of \( B \). Thus we see that items in the argument list \( L \) can be thought of as built from the novel items in the list (and the small ordinals) by iterated application of the operations of affixing an ordinal index to a suitable parent to get an element of a \( C(B) \) and the operation of “applying” a function code of complexity less than or equal to the minimum of \( A_{n-1} \) to a small list of simpler items of the same sort to get an element of a \( Q(m,D) \), and the operation of taking formal symmetric differences from small collections to handle arguments in \( N(B) \)’s.

**Observations on dependencies of equivalence computations:** Now observe that the computation of equivalence of two elements of a \( Q(n,A) \), say \((3,f,L) \) and \((3,g,M) \), reduces to the computation of equivalences between formal elements \((3,f',L') \) and \((3,g',M') \), where the novel items in each list are suitably bounded as either remote from both \( L \) and \( M \) or having iterated parents in one or the other. The new items in \( L' \) and \( M' \) are built using operations of complexity \( \leq \) the minimum element of \( A_{n-2} \), because they are argument list items in codes in \( Q_{n-1}(A) \), with underlying novel material limited to what we had in \( L \) and \( M \) (and we may unexceptionably assume that we know how to compute equivalence of range elements and items in \( L \) and \( M \)).

**Description of the complexity witnessing soundness of our recursion:**
We define complexity of objects of our classes relative to a small set \( S \) of objects belonging to \( C(A) \)’s, on which we assume we already know how to compute equivalences, in a form precisely designed to verify the recursion. For an element of a \( C(A) \) the value of the complexity will be zero if it is in \( S \) or if its parent is a small ordinal, and otherwise the same as that of its parent. For an element \( x \) of an \( N(A) \) the value of the complexity will be the supremum of the values of the complexity at the parent of \( x \) and at the elements of the third component of \( x \). For an element \((3,f,L) \) of \( Q(n,A)(n > 0) \), the value of the first component of the complexity will be the supremum of the complexity of the function code \( f \) (as defined above) and the complexities of range elements of \( L \).
This complexity is precisely designed so that the computation of equivalence of \((3, f, L)\) and \((3, g, M)\) in the same \(Q(n, A)\) will depend only on evaluations of equivalences of objects which have lower complexity relative to the union of the sets of items in \(L\) and \(M\), for which we assume we know how to compute equivalences, and further computations of equivalence that these depend on will be of no higher complexity with respect to this same set. It is also the case that the verification that any \((2, a, D)\) or \((3, f, L)\) meets the conditions involving \(\sim\) that it is required to meet will involve instances of \(\sim\) of the same or lower complexity, but of lower set theoretical rank, and that computation of instances of equivalence between elements of \(C\)- or \(N\)-basic classes depends only on instances of equivalence of the same or lower complexity but lower set theoretical rank.

Our inductive hypothesis in computing equivalence of \((3, f, L)\) and \((3, g, M)\) is that we know how to carry out all computations of equivalences between items in \(L\) or \(M\), and that we know how to carry out all computations of equivalences of lower complexity relative to any small set \(S\) on which we know how to evaluate equivalences. When we compute equivalence between formal elements \((3, f', L')\) and \((3, g', M')\) respectively of \((3, f, L)\) and \((3, g, M)\), which are themselves of lower complexity, we will be making computations of still lower complexity, but relative to a larger (but still small) set of items computed ultimately from items in \(L\) and \(M\) using function codes no more complex than \(f'\) and \(g'\) and found in the transitive closures of \(f'\) and \(g'\).

It is easy to see that computations of complexity zero relative to a set \(S\) can be carried out: if \((3, f, L)\) is of complexity 0 then \(n\) must be 1, and codes in \(Q(1, A)\) are very simple.

We have completed the demonstration that computation of equivalence succeeds; it is not difficult to see that if computation of equivalence succeeds, it must be an equivalence relation.

It remains to demonstrate that the basic classes are sets.

**Recapitulation re complexity of \(Q\)-basic classes and function codes:**

We recall that we have assigned each set \(Q(n, A)\), \(n > 0\), a complexity, equal to the minimum element of \(A_{n-1}\) if \(A_{n-1}\) is nonempty, and otherwise to \(\lambda\). Observe that for each \((3, f, L)\) in \(Q(n, A)\), we have each
element of \( A \) belonging to a \( C(B) \) or \( N(B) \) with \( B << A_{n-1} \). This means that each function code in a \( Q(m, D) \), \( m > 0 \), involved in the argument list type of \( f \), contains the parent of an element of \( N(B) \), and so we have \( D << B << A_{n-1} \) and \( m = |D| - |B| + 1 \), which tells us that the complexity of \( Q(m, D) \) is the minimum element of \( D_{‖D−‖B‖+1‖}−1 = B \), and so is less than or equal to the complexity of \( A_{n-1} \).

**Demonstration that the basic classes are sets:** Clearly the function codes with output type \( Q(0, A) \) make up a set.

As above, we define the complexity of a function code with output type \( Q(n, A) \), \( n > 0 \), as the complexity of \( Q(n, A) \).

Fix a function code of a given complexity and assume that the function codes of lower complexity make up a set. A function code with output type a given class \( Q(n, A) \) with \( n \) positive is determined by a set \( U \) of function codes with output type in \( Q(n-1, A) \) and so of lower complexity, and an argument list type code \( T \) containing as components a small collection of function codes of the same or lower complexity. Since we suppose as inductive hypothesis that the function codes of lower complexity than the given function code make up a set, we see that the set theoretical rank (the rank in the usual cumulative hierarchy) of the given function code exceeds the supremum of the set theoretical ranks of a small collection of function codes of the same complexity by no more than a fixed ordinal (determined by the structure of argument list codes, the hypothesized bound on the rank of codes of lower complexity, and known bounds on the ranks of small ordinals and finite subsets of \( \lambda \)). This cannot be true unless the function codes of the complexity of the given function code make up a set. Since there is only a set of complexities, there is only a set of function codes.

Once we know that the function codes make up a set, it is clear that all the classes \( C(A) \), \( N(A) \), \( Q(n, A) \) make up sets. Each element of each of these classes has set theoretical rank exceeding the supremum of the ranks of a small collection of components by no more than a fixed ordinal. In the case of \( C(A) \), we are talking about only one component on which we cannot immediately place a bound, its parent. In the case of \( N(A) \), there is the parent and a small collection of elements of \( C(A) \). In the case of \( Q(n, A) \), there is the function code \( [ \text{we know now that} \]
these are bounded in rank] and the small range of the argument list, taken from sets $C(B)$ and $N(B)$. 