A datatype for the first inaccessible ordinal.

Peter Hancock

Version of Mon Dec 4 14:16:52 GMT 2000

The form of specifications One model for programs takes their specifications to boil down to a formula with the following form.

$$\forall x. Pre(x) \rightarrow \exists y. \{e\}(x) \simeq y \wedge Post(x, y)$$

Here e is the thing wanted, which is a program for some computing device such as a universal Turing[1] machine, and the formula (in which e occurs free) is its specification. The proof shows that whenever e is loaded and run on the computer with input data x that satisfies the precondition Pre, execution will eventually terminate with output data y that stands in a certain relation $Post$ to the input. I call this a machine dependent specification.

What does abstract and concrete mean?

A less machine-dependent model is to take a program *simply to be* a proof of a statement

$$\forall x. Pre(x) \rightarrow \exists y. Post(x, y)$$

After all, if such a statement is proved constructively by methods for which there is a well-behaved realisability interpretation, the proof will 'contain' a concrete program in the machine dependent sense, together with a proof that the concrete program satisfies a specification in the first sense. Such a model may be preferable for some purposes, since it abstracts from the computing device. According to this picture, a specification (which may not be realisable) is given by a (unary) predicate expressing that the input is valid, and a (binary) relation expressing that the output on termination is properly related to the input.

There is an apparent imbalance in the representation of a specification in terms of a predicate required to hold in an input state and a relation guaranteed to hold between an input state and an output state. The imbalance is only apparent, as we are dealing with both 'rigid' and 'flexible' variables. It is better to think of the general form of a specification of the kind considered here as a pair of *relations*, $Pre(x, v)$ from input variables x to constants[2], and $Post(v, y)$ from constants v to output variables y.

[1] The computational mechanism can be Turing machines, register machines, von Neumann machines, lambda-calculus, .. or any abstract computational mechanism or architecture.

[2] Or 'logical' variables, as opposed to program variables.

Separate point: such a pair is a represention or presentation of a certain predicate transformer

$$P \mapsto \{s \mid \exists c.Pre(s,c) \wedge \forall s'.Post(s',c) \rightarrow P(s')\}$$

and so compare such pairs modulo the partial ordering induced by the natural partial order of predicate transformers. (Needs thought.)

This picture, in either the machine-dependent or the machine-independent form is appropriate chiefly for *batch* programs. These are programs that behave as if all input is made available in advance, and all output occurs when the computation (proceeding in isolation) has eventually terminated. The type of data such a program can consume and produce consists of fully explorable[3], finitistic values, encodable as natural numbers.

There is another class of program model more appropriate for designing systems from components that interact over and over again in time. We think of a running program as interacting with an environment, and focus on property we want of a sequence of messages that might be exchanged or communicated by a properly functioning component. We can think of these interactions as communicating progressively more and more information about an infinitary value, which can never be explored exhaustively (or only 'at the end of time'. According to this view, a specification is a formula.

$$\forall s = (s_0, s_1, \ldots) : S^\omega.Rely(s) \rightarrow Guarantee(s)$$

Here S is the set of transmissible data values, or visible states of the interface between the running program and its environment. The first predicate says what it means for the environment to have behaved properly throughout an infinite sequence or stream of states. The second predicate says what it means for the program to have delivered the required service.[45]

What are the computing mechanisms in this case? What corresponds to the $\simeq$ statement, or a Turing machine?

A important pattern of interaction is handshaking. The characteristic thing here is that every input to a component is followed by at most one output, fulfilling as it were a obligation or promise. The output can be a mere acknowledgement, when no information need be passed, merely keeping the two parties synchronised. For this and other specialised protocols a restricted class of models may be appropriate. There are two kinds of programs: masters and slaves, or clients and servers.

[3] This is a little like surveyability in Hilbert, Übersichtlichkeit, anschaulichkeit, in Wittgenstein. Visualisable. Grokkable in one gulp. Hilbert's word apples to free-variable equations.

[4] We can represent predicates of finite or infinite sequences – *ie* predicates of the final coalgebra for $(S\times)$ (if this carries less connotation of decidability) – as predicates of infinite sequences that are compatible with stuttering-equivalence.

[5] This raises the whole question of what it means to have a predicate on a final coalgebra. What does it mean to have a predicate of the final coalgebra $S \mapsto (\, c : C\,) \times S^{R(c)}$ where $\langle R, C\rangle$ is a family of sets? A set valued function, for sure, but one which is compatible with some equivalence relation. If it is stuttering equivalence for $(S\times)$, what is it for $S \mapsto (\, c : C\,) \times S^{R(c)}$?

Of course the two kinds of model are linked. A single step in the course of a sequence of interactions can be implemented by a batch program. In turn, a batch program can be implemented by a terminating sequence of interactions between internal components. (The output of a Turing machine computation is the result of a terminating sequence of interactions between the tape and automaton components.)

Termination proofs A proof of termination need not use a wellordering; it can be 'by construction', in the sense that a type system is used which constrains programs so that they terminate.

Nevertheless, termination proofs are sometimes presented with an explicit well-ordering defined between the states of a machine. (This often seems to be bad style; proofs by construction are more elegant, understandable and efficient.) The wellordering quite often has transfinite length. For example [Conor] sometimes an algorithm for finding a most general unifier for a set of equations between first order terms is proved to terminate by induction on the number of free variables in the terms, and within that by induction on the size of the equation-set. Each state of the algorithm is assigned an ordinal $< \omega^2$, and each step of the algorithm reduces the ordinal assigned to the current state.

To take another example: to prove strong normalisation for a rewriting system, one sometimes interprets the expressions as ordinal notations in such a way that any allowed rewriting lowers the ordinal that is the denotation of an expression.

And now I want to say something like; *implicitly* there is always a transfinite order, even if termination is 'by construction'. The order is defined between certain 'clock-values', that are structures which record the amount of 'time' a program has left.

Applications Applications of ordinals as data structures: as *magnitudes* through various order relations; as *exponents* in iteration. Is this a real distinction in applications?

Ordinals as magnitudes abstract from the particular fundamental sequence structure used by a notation system; if one can simulate the other, the magnitude (or applicability) of the first is at least as great as that of the second.

There is a deep connection between wellfounded structures, the representation of their magnitude with ordinals, the arithmetic of ordinals and certain key ideas in programming; but it is submerged.

The theory of ordinal arithmetic is appropriate to questions of the *scale* of programs. This does not mean resource usage, but extent of applicability.

If we want to compare different programming systems, one interesting question is whether in one we can write programs capable of provoking deadlocks in programs written in the other system.

The need for a computationally oriented theory of ordinals By computationally oriented I mean that a limit ordinal is equipped with a fundamental sequence or family, with a specific domain or index set. This lets us interpret an ordinal as a program which *reads* an element of its input set/alphabet, and evolves to a new program that depends on the value read. On this basis, there are a variety of equivalence and order relations that can be erected; but the transition structure is primary. The order relations express the existence of certain concrete translations (chiefly, of inputs to inputs) which allow one ordinal to simulate another, viewed from some perspective. (Transfinitely vague.)

Ordinals are abstractions of programs, where we have forgotten everything about what service a program does, or its output, but just remember that it occurs (all outputs become 'beep') and focus on the pattern of inputs and outputs. [6] Execution consists of tracing out a descending chain. (One cannot skip any steps in this chain.) Ordinals are of course *terminating* programs.

They are 'time' values for a clock, which allow us to enforce termination of a free-running state-machine. Every input is fed into the clock, which counts down the "time" remaining. One unit of time is subtracted at a successor ordinal. (This is counting outputs.)

There do not seem to be much to cite for a development of ordinal arithmetic in a constructive setting. Troelstra. Brouwer.

What has been done. Arithmetic for recursive order types. Crossley. There is a theory of recursive number classes Church, Kleene, Markwald, Spector, Kreider and Rogers.

recursion theoretic stuff. Not only Church-Kleene, but also the theory of admissible sets. The theory of admissible sets (and its relations with the hierarchy of constructible sets). What is the theory of admissible sets? Barwise. Constructible sets. Closed under ordinary inductive definitions.

This isn't really programming oriented. *Do we lack a computational theory of ordinals?*

There has been investigation of finite tree classes or constructive number classes from a computational perspective [3]. There is Setzer's thesis.

Notation systems. We lack a computational development of the theory of those ordinal notation systems based on the idea of collapsing higher number classes into the countable ordinals. We need algorithms and programs for reduction to normal form, comparision of normal forms.

The theory of ordinal notation systems Church. Its practice (the 'panzer divisions' of ordinals). The practice of defining notation systems leans heavily on

[6] Cantor arrived at ordinals by abstracting 'away' the specific nature of the elements of a linear order, remembering only their relative position, as preserved by the appropriate kind of morphism. An unpublished draft review [2, Draft towards a review of Cantor's Gesammlte Abhandlungen zur Lehre vom Transfiniten] by Frege of Cantor's book has a deliciously venemous caricature of Cantor's abstraction. We take a pair of mice, and abstract from the length of their tails, the beadiness of their eyes, the sharpness of their teeth, and so on, till they become indistinguishable from the number 2. The published version of this review was more civilised, but the original is to be recommended as entertaining, for those whose sense of humour is tickled by displays of sheer unrestrained bile.

Abstraction really means having a notion of structure preserving mapping, and dealing with everything modulo a structure preserving map.

normal forms such as Cantor Normal form that are constructively problematic. It all comes right in the end, but the motivation is hocus-pocus.

The framework is classical set theory, used as inspiration for a notation system. It is like taking a drug to get into contact with the spirit world.

Haven't quite large notation systems actually been implemented as as Prolog programs for deciding a magnitude relation? Levitz, Bucholtz.

Problem with notation systems construed merely as recursive relations between notations. If implementing a notation system is implementing an order-relation, then we do not need to assign fundamental sequences to notations for limit ordinals. We could exhaust the expressions of the notation system and filter out all notations at least as large as the limit, getting a fundamental sequence for the limit ordinal – this is obviously an artificial assignment. (We could even produce a fundamental sequence which is strictly increasing.) It is not *computational*. We cannot program the simulation of one thing by another.

How many we want The datatype of ordinals should be reasonably unrestricted as to cardinality; it should not be confined to the countable ordinals, or even the ordinals with cardinal $< \aleph_\omega$; for each ordinal, there should be a next greater regular ordinal, and for any ordinal valued function indexed by the predecessors of an ordinal, there should be an ordinal which is its supremum.

Why do we want so many? One sometimes sees the use of higher number classes in functional programs – though of course, not 'as such'. Sometimes the datatypes are parsers.

And to treat such constructions generically, we need to be able to collect them into families.

The ordinals In this note I define a datatype of ordinals below the first inaccessible. The definition may serve as a basis for a theory of ordinals that is computationally oriented, and capable of describing the strength of quite strong principles.

We take a dependent type theory as framework, extended by constants introduced as fixed-points for certain operators; for example constants for inductively defined objects. These may be sets, predicates, families of sets, tuples of such things, and functions defined on such structures by recursion on their build-up. We will consider 'coalgebras for *Fam*(.)' which are inductively defined (as least fixpoints). That is to say, we define a coalgebra for *Fam*(.) which is the least fixpoint of a certain operation whose type is the following

$$((\, S : \mathrm{Set}\,) \times S \to \mathit{Fam}(S)) \to (\, S : \mathrm{Set}\,) \times S \to \mathit{Fam}(S)$$

The principle of recursion used here may be problematic; it involves both induction and recursion, yet it is not an inductive-recursive definition in the sense of Dybjer and Setzer [1], which allows us to take the least fixpoint of certain operations of type

$$\mathit{Fam}(D) \to \mathit{Fam}(D)$$

for a fixed (large) type D.

We define the type in question as the least fixed point of a certain operation on objects of type

$$\begin{array}{l} (\ S : \mathrm{Set} \\ ,\ f\ : S \to \mathit{Fam}(S)) \end{array}$$

I call such objects *transition systems*. Here $\mathit{Fam}(X)$ is the type $(\ I : \mathrm{Set}\ ) \times X^I$ of indexed families of elements of a type X.

A transition system can alternatively be given as a triple

$$\begin{array}{l} (\ S : \mathrm{Set} \\ ,\ T : \mathbb{P}(S) \\ ,\ \delta\ : (\ s : S\ ) \to S^{T(s)}) \end{array}$$

Here $\mathbb{P}(X)$ is the type $X \to \mathrm{Set}$ of set valued functions defined on a type X. If δ can be inferred, I shall often use the notation $s[t]$ for $\delta(s, t)$ where $s : S$ and $t : T(s)$.

The elements of S are states; where s is a state, the elements of $T(s)$ are transitions from s; where s is a state, and t a transition from s, $\delta(s, t)$ is the destination state of t. When defining an operation on transition systems, the output S (states) component may refer not just to the input T (transition) component but also to the input δ (destination) component.

In categorical terminology, a transition system is a coalgebra for the functor $\mathit{Fam}(\_)$.

Operators on transition systems have the following general form

$$\begin{pmatrix} U \\ T \\ \delta \end{pmatrix}' = \begin{pmatrix} U'(U,T,\delta) \\ T'(U,T,\delta) \\ \delta'(U,T,\delta) \end{pmatrix}$$

An operation on transition systems may make use of all three components of the input system for the definition of any component of the output.

Universes The type of universes is simpler than that of transition systems.

$$\begin{array}{l} (\ U : \mathrm{Set} \\ ,\ T\ : \mathbb{P}(U)) \end{array}$$

We can get a universe from a transition system by forgetting the δ component.

When we define a universe as a least fixed point of an operation, the operations of which we take least fixed points can have the following general form.

$$\begin{pmatrix} U \\ T \end{pmatrix}' = \begin{pmatrix} U'(U,T) \\ T'(U,T) \end{pmatrix}$$

In general, U' (as well as T') may depend on T. We have an inductive recursive definition of an object of type $\mathit{Fam}(D)$, where $D = \mathrm{Set}$.

Our definition is of an object of the type $(\ S : \mathrm{Set}\ ) \times (S \to \mathit{Fam}(S))$.

1 Successor operations

1.1 Sets

The successor operation on sets has the following definition.

$$S(X) = \textbf{data}\ \ 0 \\ \qquad\qquad\qquad \mid\ s(x) \quad (x : X)$$

The initial algebra $\mu X.\, S(X)$ is the same as the set $\mathbb{N}$ of natural numbers.

1.2 Families of sets

Here are two ways to extend this operation on sets to work on families of sets.

- The output family contains the empty set, and the successor of any set in the input family. I call this the *lifting* successor.

  $$\begin{pmatrix} U' \\ T' \end{pmatrix} = \begin{pmatrix} S(U) & & & \\ \textbf{case} & 0 & \mapsto & \{\} \\ & \mid\ s(\beta) & \mapsto & S(T(\beta)) \end{pmatrix}$$

  $S(\_)$ was defined in 1.1

  This does not automatically extend the input family – if it was for example the family assigning the empty set to 0, the successor of the empty set to $s(0)$ and so one, in this case the output family is the same.

- The other extends the input family with its own index set. I call this the *von Neumann* successor.

  $$\begin{pmatrix} U' \\ T' \end{pmatrix} = \begin{pmatrix} S(U) & & & \\ \textbf{case} & 0 & \mapsto & U \\ & \mid\ s(\beta) & \mapsto & T(\beta) \end{pmatrix}$$

  The output family automatically extends the input family, and moreover contains it as an internal subfamily (given by $(0, s)$).

1.3 Transition systems

Here are two ways to extend the successor operation yet further so that it takes transition systems to transition systems.

- the *lifting* successor.

  $$\begin{pmatrix} U' \\ T' \\ \delta' \end{pmatrix} = \begin{pmatrix} S(U) & & & & & & \\ \textbf{case} & 0 & \mapsto & \{\} & & & \\ & \mid\ s(\beta) & \mapsto & S(T(\beta)) & & & \\ \textbf{case} & 0 & \mapsto & \textbf{empty} & & & \\ & \mid\ s(\beta) & \mapsto & \textbf{case} & 0 & \mapsto & 0 \\ & & & & \mid\ s(\alpha) & \mapsto & s(\beta[\alpha]) \end{pmatrix}$$

The idea here is that 0 goes 'on the bottom'; everything is lifted up (β is lifted to $s(\beta)$), and acquires an extra transition to the new bottom.

This successor can be thought of as $1 + \alpha$.

- *von Neumann* successor.

$$\begin{pmatrix} U' \\ T' \\ \delta' \end{pmatrix} = \begin{pmatrix} S(U) \\ \mathbf{case}\ 0 \quad\ \ \mapsto U \\ \quad\ \mid s(\beta) \mapsto T(\beta) \\ \mathbf{case}\ 0 \quad\ \ \mapsto \mathbf{fun}\ \alpha : U \mapsto s(\alpha) \\ \quad\ \mid s(\beta) \mapsto \mathbf{fun}\ \alpha : T(\beta) \mapsto s(\beta[\alpha]) \end{pmatrix}$$

The idea here is that 0 goes 'on the top'. The transitions from input states are as they were; there is a transition to every input state from 0.

This successor can be thought of as $\alpha + 1$.

The definition may seem a bit strange, in that using the notation 0 carries some connotation of 'leastness'. All the same, it is natural in that the 'least' transition 0 goes the least distance.

But 0 and $s(\_)$ are just labels that distinguish a given family of things from a new thing.

2 Regular universes

A regular ordinal is an infinite ordinal that is closed under suprema of sequences of smaller length. So I define a 'regular' universe to be a family of sets that contains the empty set, is closed under the successor operation on sets (and is therefore infinite), and is closed under disjoint unions.

The first regular universe is the following least fixed point.

$$\mu \begin{pmatrix} U \\ T \end{pmatrix} . \begin{pmatrix} \mathbf{data}\ 0 \\ \quad \mid s(\beta) \quad (\beta : U) \\ \quad \mid \sqcup_\beta f \quad (\beta : U, f : U^{T(\beta)}) \\ \mathbf{case}\ 0 \quad \mapsto \{\} \\ \quad \mid s(\beta) \mapsto S(T(\beta)) \\ \quad \mid \sqcup_\beta f \mapsto (t : T(\beta)) \times T(f(t)) \end{pmatrix}$$

Intuitively, this consists of 'pure' finite sets. It is itself infinite.

Note that the U component refers to the T component, in the clause for suprema. However the suprema are in fact superfluous, and we get the same family of sets as the standard finite sets over $\mathbb{N}$.

We extend this with the definition of a closure operator producing from a family of sets (A, B) the least family of sets extending it which is a regular universe. I give this the name $Reg(A, B)$, and call it the *next regular universe* operator.

$$
\begin{array}{lcl}
\mathit{Reg}(\_,\_) & : & \mathit{Fam}(\mathrm{Set}) \to \mathit{Fam}(\mathrm{Set}) \\
\mathit{Reg}(A,B) & = & \mu \begin{pmatrix} U \\ T \end{pmatrix} . \left(\begin{array}{llll}
\textbf{data} & g(a) & & (a : A) \\
& \mid 0 & & \\
& \mid s(\beta) & & (\beta : U) \\
& \mid \sqcup_\beta f & & (\beta : U, f : U^{T(\beta)}) \\
\textbf{case} & g(a) & \mapsto & B(a) \\
& \mid 0 & \mapsto & \{\} \\
& \mid s(\beta) & \mapsto & S(T(\beta)) \\
& \mid \sqcup_\beta f & \mapsto & (\, t : T(\beta)\,) \times T(f(t))
\end{array} \right)
\end{array}
$$

Notice that it just *extends* the given family of sets, not necessarily properly. It may already be closed under 0, $S(.)$ and disjoint unions of families. To get the next (strictly) greater regular universe, take the next (non-strictly) greater regular universe of the (von-Neumann) successor.

It seems to be useful to define an operator $(.)^*$ of type $(\mathit{Fam}(\mathrm{Set}) \to \mathit{Fam}(Set)) \to \mathit{Fam}(\mathrm{Set}) \to \mathit{Fam}(Set)$, which given an operator $\textbf{fun}\ \langle U, Y\rangle \mapsto \langle \phi(U,T), \textbf{fun}\ x \mapsto \psi(U,T,x)\rangle$ and a family of sets $\langle A, B\rangle$ makes the least family extending $\langle A, B\rangle$ closed under $\langle \phi, \psi\rangle$.

$$
\begin{array}{l}
\_^* : (\mathit{Fam}(\mathrm{Set}) \to \mathit{Fam}(\mathrm{Set})) \to \mathit{Fam}(\mathrm{Set}) \to \mathit{Fam}(\mathrm{Set}) \\
(\textbf{fun} \begin{pmatrix} U \\ T \end{pmatrix} \mapsto \begin{pmatrix} \phi(U,T) \\ \textbf{fun}\ x \mapsto \psi(U,T,x) \end{pmatrix})^* \begin{pmatrix} A \\ B \end{pmatrix} \\
\qquad = \mu \begin{pmatrix} U \\ T \end{pmatrix} . \left(\begin{array}{llll}
\textbf{data} & g(a) & & (a : A) \\
& \mid h(x) & & (x : \phi(U,T)) \\
\textbf{case} & g(a) & \mapsto & B(a) \\
& \mid h(x) & \mapsto & \psi(U,T,x)
\end{array} \right)
\end{array}
$$

For example, we can take

$$
\begin{array}{ll}
\phi(U,T) & = S(U) \\
\psi(U,T,0) & = \{\} \\
\psi(U,T,s(u)) & = S(T(u))
\end{array}
$$

This closes up a family so it contains the empty set and is closed under set successor.

Another example is

$$
\begin{array}{ll}
\phi(U,T) & = S(U) \\
\psi(U,T,0) & = U \\
\psi(U,T,s(u)) & = T(u)
\end{array}
$$

The next step will be to define something like a ∇ operation (below somewhere). This will take a $\langle \phi, \psi\rangle$ (analogy: a normal function) and return a universe which is closed under $\{\}$, von-Neumann successor, and the closure operation $\langle \phi, \psi\rangle^*$ (analogy: the club of fixpoints of the normal function). The presence of the *successor* operation means we can have a hierarchy of cumulative universes.

2.1 Transition systems

We can extend the next regular universe operation to an operation on transition systems.

$$
\mu \begin{pmatrix} U \\ T \\ .[.] \end{pmatrix} . \left(\begin{array}{l}
\textbf{data}\ \ g(a) \quad (a : A) \\
\quad\quad \mid 0 \\
\quad\quad \mid s(\beta) \quad (\beta : U) \\
\quad\quad \mid \sqcup_\beta f \quad (\beta : U, f : U^{T(\beta)}) \\
\textbf{case}\ \ g(a) \mapsto B(a) \\
\quad\quad \mid 0 \quad\ \ \mapsto \{\} \\
\quad\quad \mid s(\beta) \mapsto S(T(\beta)) \\
\quad\quad \mid \sqcup_\beta f \mapsto (\,t : T(\beta)\,) \times T(f(t)) \\
\textbf{case}\ \ g(a) \mapsto a[b] \\
\quad\quad \mid 0 \quad\ \ \mapsto \textbf{empty} \\
\quad\quad \mid s(\beta) \mapsto \textbf{case}\ \ 0 \quad\ \ \mapsto \beta \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid s(t) \mapsto \beta[t] \\
\quad\quad \mid \sqcup_\beta f \mapsto \textbf{fun}\ \langle t, t' \rangle \mapsto f(t)[t']
\end{array} \right)
$$

However there is no genuine mutuality as the third component is not used in the definition of the first two.

3 Inaccessible universe

$$
\mu \begin{pmatrix} U \\ T \\ \_[\_] \end{pmatrix} . \left(\begin{array}{l}
\textbf{data}\ \ 0 \\
\quad\quad \mid s(\beta) \quad (\beta : U) \\
\quad\quad \mid \sqcup_\beta f \quad (\beta : U, f : U^{T(\beta)}) \\
\quad\quad \mid \rho(\beta) \quad (\beta : U) \\
\textbf{case}\ \ 0 \quad\quad\quad\quad \mapsto \{\} \\
\quad\quad \mid s(\beta) \quad\quad\ \ \mapsto S(T(\beta)) \\
\quad\quad \mid \sqcup_\beta f \quad\quad\ \ \mapsto (\,t : T(\beta)\,) \times T(f(t)) \\
\quad\quad \mid \alpha = \rho(\beta) \mapsto \mu X.\ \textbf{data}\ \ g(\gamma) \quad (\gamma : T(\beta)) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid 0 \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid s(\gamma) \quad (\gamma : X) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid \sqcup_\gamma f \quad (\gamma : X, f : X^{T(\alpha[\gamma])}) \\
\textbf{case}\ \ 0 \quad\quad\quad\quad \mapsto \textbf{empty} \\
\quad\quad \mid s(\beta) \quad\quad\ \ \mapsto \textbf{case}\ \ 0 \quad\ \ \mapsto \beta \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid s(t) \mapsto \beta[t] \\
\quad\quad \mid \sqcup_\beta f \quad\quad\ \ \mapsto \textbf{fun}\ \langle t, t' \rangle \mapsto f(t)[t'] \\
\quad\quad \mid \alpha = \rho(\beta) \mapsto \textbf{case}\ \ g(\gamma) \mapsto \beta[\gamma] \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid 0 \quad\ \ \mapsto 0 \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid s(\gamma) \mapsto s(\alpha[\gamma]) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mid \sqcup_\gamma f \mapsto \sqcup_{\alpha[\gamma]}(\textbf{fun}\ x \mapsto \alpha[f(x)])
\end{array} \right)
$$

We call the triple $(I, \mathit{Seg}\,(.), .[.])$. Remark that the third component (the indexing function $.[.]$) occurs in the definition of the second, so we have genuine mutuality. The I component of this is our inaccessible ordinal. The indexed set $\{\alpha[t] \mid t : \mathit{Seg}\,(\alpha)\}$ consists of all structural predecessors of α.

The ordinal $\rho(\alpha)$ is the least regular containing α (not necessarily properly). We want ρ to be a closure operator, in other words idempotent and inflationary. (These notions depend on the order relation which we have not defined yet.)

Say something about the map

$$\textbf{fun}\ \alpha : S \mapsto \sqcup_\alpha(\textbf{fun}\ t : \mathit{Seg}\,(\alpha) \mapsto \alpha[t])$$

It is predecessor at successors, otherwise the identity.

4 Iteration

One use of ordinals is as iterative exponents.

Iteration involves a set X of values, a unary operation $f : X \to X$ on them (associated with successor), and a way of taking limits of families of a certain form. For each ordinal α we can define $f^\alpha : X \to X$ (the α-th iterate of f).

The 'introverted' use to iterate operations on ordinals. It is clear what the limit operation should be in this case.

$$\begin{array}{ll}\phi^\alpha(x) = \textbf{case}\ \alpha\ \textbf{of} & \\ \qquad\qquad 0 & \mapsto x \\ \qquad\qquad s(\beta) & \mapsto \phi(\phi^\beta(x)) \\ \qquad\qquad \_ & \mapsto \sqcup_\alpha(\textbf{fun}\ t : \mathit{Seg}\,(\alpha) \mapsto \phi^{\alpha[t]}(x))\end{array}$$

We don't need to consider any other cases. Say why.

For an example of iteration we can define the function indexing the infinite initial ordinals

$$\omega_\alpha = (\textbf{fun}\ \beta \mapsto \rho(s(\beta)))^\alpha(\rho(0))$$

ω_0 is ω, which is regular, as are ω_1, ω_2, ..., and $\omega_{\alpha+1}$. ω_ω is not regular, nor is the least fixpoint of $\beta \mapsto \omega_\beta$. I is the first regular fixed point of the function $\omega_.$. The initial ordinals $\{\omega_\alpha \mid \alpha : I\}$ (which I also call cardinals and write $\aleph_\alpha$ are the closure of the regulars under limits.

The number classes can then be defined.

$$\mathcal{T}_\alpha = \mathit{Seg}\,(\omega_\alpha)$$

Other examples. The function here is

$$\nabla f \alpha = (f \circ (+1))^\alpha(f(0))$$

$\omega_\alpha = \nabla \rho\, \alpha$. $\phi_0 = \omega_\alpha$. $\phi_{\alpha+1} = \nabla(\phi_\alpha)^\omega$. $\phi_\lambda = \nabla(\sqcup_n \phi_{\lambda[n]})$.

5 Order

There are two interpretations we can give to $\sqcup$: the least strict upper bound (which subsumes successor), or the least upper bound. (For some reason obscure to me the second is better.)

$$
\begin{array}{lcl}
\alpha \leq \alpha' & = & \textbf{case } \alpha \textbf{ of} \\
& & \quad \begin{array}{ll}
0 & \mapsto N_1 \\
s(\beta) & \mapsto \textbf{case } \alpha' \textbf{ of} \\
& \quad \begin{array}{ll}
0 & \mapsto N_0 \\
s(\beta') & \mapsto (\alpha \leq \beta') + (\beta \leq \beta') \\
\_ & \mapsto (\,\gamma : \mathit{Seg}\,(\alpha')\,) \times \alpha \leq \alpha'[\gamma]
\end{array} \\
\_ & \mapsto (\,\gamma : \mathit{Seg}\,(\alpha)\,) \rightarrow \alpha[\gamma] \leq \alpha'
\end{array}
\end{array}
$$

As $\sqcup$ is interpreted as least upper bound it is not decidable whether an ordinal is a genuine limit. A genuine limit contains zero and is closed under successor; for each ordinal there is a (non-strictly) greater limit.

There is lots of 'trash' among these ordinals; definitions by recursion will not often assign equal values to (extensionally) equal ordinals.

Remark how we are using universes; we are defining a set-valued function (that can be thought of as a predicate) by recursion on a wellfounded set, by recursion into some universe's set of codes. The universe has to contain codes for N_0 and N_1, to be closed under $+$, and to be closed under cartesian product and disjoint union of families indexed over initial segments of some ordinal.

The set $\alpha \leq \beta$ is the set of simulations of α by β. we define it by recursion on α, and within that on β. But of course simulation is meaningful (especially) for non-wellfounded objects.

Prove reflexive and transitive. Define equality to be symmetric closure.

$$
\begin{array}{l}
\alpha < \beta = s(\alpha) \leq \beta \\
\alpha = \beta = \alpha \leq \beta \times \beta \leq \alpha
\end{array}
$$

We can define what it means for an ordinal to be proper:

$$
\begin{array}{ll}
P(0) & = N_1 \\
P(s(\alpha)) & = P(\alpha) \\
P(\alpha) & = \forall t : \mathit{Seg}\,(\alpha).P(\alpha[t]) \times \exists t' : \mathit{Seg}\,(\alpha).\alpha[t] < \alpha[t']
\end{array}
$$

This says that the supremums are always of 'infinite' families, or true limits. Some directedness property may also be interesting.

$$
\begin{array}{lll}
Q(0) & = N_1 & \\
Q(s(\alpha)) & = Q(\alpha) & \\
Q(\alpha) & = \forall t : \mathit{Seg}\,(\alpha). & Q(\alpha[t]) \\
& & \times\ \forall t' : \mathit{Seg}\,(\alpha).\exists t'' : \mathit{Seg}\,(\alpha). \\
& & \qquad \alpha[t] < \alpha[t''] \times \alpha[t'] < \alpha[t'']
\end{array}
$$

6 Collapsing functions

We let $\mathcal{T}_1$ denote the countable ordinals (the second number class, the first being the finite ordinals), and $\mathcal{T}_2$ the third number class.

Schwichtenberg

$$\begin{array}{lcl} D & : & \mathcal{T}_2 \to \mathcal{T}_1 \\ D(0) & = & \omega \\ D(\alpha+1)[n] & = & D(\alpha).(n+1) \\ D(\lambda)[n] & = & D(\lambda[n]) \\ D(\Lambda)[n] & = & (\mathbf{fun}\ \alpha \mapsto D(\Lambda[\alpha]))^{\omega}(\omega) \end{array}$$

Martin-Löf

$$\begin{array}{lcl} P & : & \mathcal{T}_2 \to \mathcal{T}_1 \\ P(0) & = & 1 \\ P(\alpha+1) & = & P(\alpha)+1 \\ P(\lambda)[n] & = & P(\lambda[n]) \\ P(\Lambda)[n] & = & (\mathbf{fun}\ \alpha \mapsto P(\Lambda[\alpha]))^{\omega}(0) \end{array}$$

Scwichtenberg's definition is constructed so that its range consists of infinite ordinals closed under addition, and notations are built up by finite sums from 1 and D-expressions.

$$\begin{array}{lcl} D(\alpha) & = & \omega^{1+\alpha} \\ D(\Omega) & = & \epsilon_0 \\ D(\Omega+\alpha) & = & \epsilon_0.\omega^{\alpha} \\ D(\Omega.2) & = & \epsilon_1 \\ D(\Omega.(1+\alpha)) & = & \epsilon_\alpha \\ D(\Omega^2) & = & \phi_2 0 \\ D(\Omega^\alpha) & = & \phi_\alpha 0 \\ D(\Omega^\alpha.(1+\beta)) & = & \phi_\alpha \beta \\ D(\Omega^\Omega) & = & \Gamma_0 \end{array}$$

$$\begin{array}{lcl} P(\alpha) & = & 1+\alpha \\ P(\Omega) & = & \omega \\ P(\Omega+\alpha) & = & \omega+\alpha \\ P(\Omega.2) & = & \omega^2 \\ P(\Omega.\alpha+\beta) & = & \omega^\alpha+\beta \\ P(\Omega^{1+\alpha}.(1+\beta)) & = & \phi_\alpha \beta \\ P(\Omega^\Omega) & = & \Gamma_0 \end{array}$$

In fact it is clear how to extend these functions to collapse or project $\mathcal{T}_{n+1}$ to $\mathcal{T}_n$ for arbitrary finite non-zero n. One can compose these functions to project $\mathcal{T}_{n+m}$ to $\mathcal{T}_n$ for $n, m > 0$, or define these functions directly in another way Schwichtenberg has used [3, p84].

The problem is to define $\theta_{\alpha,t} : \mathcal{T}_\alpha \to \mathcal{T}_{\alpha[t]}$ for each ordinal α and $t : \mathit{Seg}\,(\alpha)$.

What we want is a usable notation system for ordinals up to the strength of **KPi**. It should be clear how to extend it (and the universe) to hyper-inaccessibles, Mahlo numbers, etc.

At limits, it is very tempting to form a kind of inverse limit of collapsing functions.

$$\{f : \prod_n \mathcal{T}_n \,|\, \forall n. f(n) = \theta(f(n+1))\}$$

which we can write as the fixpoint of an operator on $\prod_n \mathcal{T}_n$

$$\{f : \prod_n \mathcal{T}_n \,|\, f = \theta \circ f \circ (+1)\}$$

It does not matter which projection we take - we can take the 0-th.

References

[1] P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In *Proceedings of TLCA 1999*, volume 1581 of *lncs*, pages 129–146, 1999.

[2] G. Frege. *Posthumous writings.* Blackwell, 1979. Originally published in German under the title Nachgelassene Schriften und wissenschaftlicher Briefwechsel.

[3] H. Schwichtenberg. Proofs as programs. In P. Aczel, H. Simmons, and S. Wainer, editors, *Proof Theory*, pages 79–115, Cambridge, 1992. Cambridge University Press.