Connecting Constructive Notions of Ordinals in Homotopy Type Theory

Nicolai Kraus  
University of Nottingham, UK

Fredrik Nordvall Forsberg  
University of Strathclyde, UK

Chuangjie Xu  
fortiss GmbH, München, Germany

Abstract

In classical set theory, there are many equivalent ways to introduce ordinals. In a constructive setting, however, the different notions split apart, with different advantages and disadvantages for each. We consider three different notions of ordinals in homotopy type theory, and show how they relate to each other: A notation system based on Cantor normal forms, a refined notion of Brouwer trees (inductively generated by zero, successor and countable limits), and wellfounded extensional orders. For Cantor normal forms, most properties are decidable, whereas for wellfounded extensional transitive orders, most are undecidable. Formulations for Brouwer trees are usually partially decidable. We demonstrate that all three notions have properties expected of ordinals: their order relations, although defined differently in each case, are all extensional and wellfounded, and the usual arithmetic operations can be defined in each case. We connect these notions by constructing structure preserving embeddings of Cantor normal forms into Brouwer trees, and of these in turn into wellfounded extensional orders. We have formalised most of our results in cubical Agda.

2012 ACM Subject Classification  Theory of computation → Type theory

Keywords and phrases  Constructive ordinals, Cantor normal forms, Brouwer trees

Digital Object Identifier  10.4230/LIPIcs.MFCS.2021.70

Related Version  Full proofs are available at: https://arxiv.org/abs/2104.02549

Supplementary Material  A formalisation is available: Software (Agda source code): https://bitbucket.org/nicolaikraus/constructive-ordinals-in-hott/src; archived at svh:1:dir:8f62d289d51a35a3f22f3b3bd9def18dfcaf909

Funding  Nicolai Kraus: The Royal Society, grant reference URF\R1\191055.
Fredrik Nordvall Forsberg: UK National Physical Laboratory Measurement Fellowship project “Dependent types for trustworthy tools”.
Chuangjie Xu: The Humboldt Foundation and the LMUexcellent program.

Acknowledgements  We thank the participants of the conferences Developments in Computer Science and TYPES, as well as Helmut Schwichtenberg and Thorsten Altenkirch for fruitful discussions on this work. We are also grateful to the anonymous reviewers, whose remarks helped us improve the paper.

1 Introduction

The use of ordinals is a powerful tool when proving that processes terminate, when justifying induction and recursion [20, 24], or in (meta)mathematics generally. Unfortunately, the standard definition of ordinals is not very well-behaved constructively, meaning that additional work is required before this tool can be deployed in constructive mathematics or program verification tools based on constructive type theory such as Agda [33], Coq [15] or Lean [18]. Constructively, the classical notion of ordinal fragments into a number of inequivalent
Connecting Constructive Notions of Ordinals

definitions, each with pros and cons. For example, “syntactic” ordinal notation systems [10, 36, 38] are popular with proof theorists, as their concrete character typically mean that equality and the order relation on ordinals are decidable. However, truly infinitary operations such as taking the limit of a countable sequence of ordinals are usually not constructible. We will consider a simple ordinal notation system based on Cantor normal forms [32], designed in such a way that there are no “junk” terms not denoting real ordinals.

Another alternative (based on notation systems by Church [14] and Kleene [28]), popular in the functional programming community, is to consider “Brouwer ordinal trees” \( O \) inductively generated by zero, successor and a “supremum” constructor

\[
\text{sup} : (\mathbb{N} \to O) \to O
\]

which forms a new tree for every countable sequence of trees [8, 16, 26]. By the inductive nature of the definition, constructions on trees can be carried out by giving one case for zero, one for successors, and one for suprema, just as in the classical theorem of transfinite induction. However calling the constructor \( \text{sup} \) is wishful thinking; \( \text{sup}(s) \) does not faithfully represent the suprema of the sequence \( s \), since we do not have that e.g. \( \text{sup}(s_0, s_1, s_2, \ldots) = \text{sup}(s_1, s_0, s_2, \ldots) \) – each sequence gives rise to a new tree, rather than identifying trees representing the same suprema. We use the notion of higher inductive types [17, 30] from homotopy type theory [40] to remedy the situation and make a type of Brouwer trees which faithfully represents ordinals. Since our ordinals now can be infinitary, we lose decidability of equality and order relations, but we retain the possibility of classifying an ordinal as a zero, a successor or a limit.

One can also consider extensional wellfounded orders, a variation on the classical set-theoretical axioms more suitable for a constructive treatment [39], which was transferred to the setting of homotopy type theory in the HoTT book [40, Chapter 10], and significantly extended by Escardó [23]. One is then forced to give up most notions of decidability – it is not even possible to decide if a given ordinal is zero, a successor or a limit. However many operations can still be defined on such ordinals, and properties such as wellfoundedness can still be proven. This is also the notion of ordinal most closely related to the traditional notion, and thus the most obviously “correct” notion in a classical setting.

All in all, each of these approaches gives quite a different feel to the ordinals they represent: Cantor normal forms emphasise syntactic manipulations, Brouwer trees how every ordinal can be classified as a zero, successor or limit, and extensional wellfounded orders the set theoretic properties of ordinals. As a consequence, each notion of ordinals is typically used in isolation, with no interaction or opportunities to transfer constructions and ideas from one setting to another – e.g., do the arithmetic operations defined on Cantor normal forms obey the same rules as the arithmetic operations defined on Brouwer trees? The goal of this paper is to answer such questions by connecting together the different notions. We do this firstly by introducing an abstract axiomatic framework of what we expect of any notion of ordinal, and explore to what extent the notions above satisfy these axioms, and secondly by constructing faithful embeddings between the notions, which shows that they all represent a correct notion of ordinal from the point of view of classical set theory.

Contributions

- We identify an axiomatic framework for ordinals and ordinal arithmetic that we use to compare the situations above in the setting of homotopy type theory.
- We define arithmetic operations on Cantor normal forms [32] and prove them uniquely correct with respect to our abstract axiomatisation. This notion of correctness has not been verified for Cantor normal forms previously, as far as we know.
- We construct a higher inductive-inductive type of Brouwer trees, and prove that their order is both wellfounded and extensional – properties which do not hold simultaneously for previous definitions of ordinals based on Brouwer trees. Further, we define arithmetic operations, and show that they are uniquely correct.
- We prove that the “set-theoretic” notion of ordinals [40, Section 10.3] satisfies our axiomatisation of addition and multiplication, and give constructive “taboos”, showing that many operations on these ordinals are not possible constructively.
- We relate and connect these different notions of ordinals by constructing order preserving embeddings from more decidable notions into less decidable ones.

Formalisation and Full Proofs

We have formalised the material on Cantor normal forms and Brouwer trees in cubical Agda [43] at https://cj-xu.github.io/agda/constructive-ordinals-in-hott/; see also Escardó’s formalisation [23] of many results on “set-theoretic” ordinals in HoTT. We have marked theorems with formalised and partly formalised proofs using the QED symbols \(\begin{smallmatrix} \text{\texttt{\# TERMINATING \#}} \end{smallmatrix}\) and \(\begin{smallmatrix} \text{\texttt{\#}} \end{smallmatrix}\) respectively; they are also clickable links to the corresponding machine-checked statement. Moreover, pen-and-paper proofs for all our results can be found in the the arXiv version of the paper.

Our formalisation uses the \(\texttt{(-# TERMINATING #-)}\) pragma to work around one known bug (issue #4725) and one limitation of the termination checker of Agda: recursive calls hidden under a propositional truncation are not seen to be structurally smaller. Such recursive calls when proving a proposition are justified by the eliminator presentation of [21] (although it would be non-trivial to reduce our mutual definitions to eliminators).

2 Underlying Theory and Notation

We work in and assume basic familiarity with homotopy type theory (HoTT), i.e. Martin-Löf type theory extended with higher inductive types and the univalence axiom [40]. The central concept of HoTT is the Martin-Löf identity type, which we write as \(a \equiv b\) – we write \(a = b\) for definitional equality. We use Agda notation \(x : A \rightarrow B(x)\) for the type of dependent functions, and write simply \(A \rightarrow B\) if \(B\) does not depend on \(x : A\). If the type in the domain can be inferred from context, we may simply write \(\forall x.B(x)\) for \((x : A) \rightarrow B(x)\). Freely occurring variables are assumed to \(\forall\)-quantified.

We denote the type of dependent pairs by \(\Sigma((x : A).B(x))\), and its projections by \(\text{fst}\) and \(\text{snd}\). We write \(A \times B\) if \(B\) does not depend on \(x : A\). We write \(\mathcal{U}\) for a universe of types; we assume that we have a cumulative hierarchy \(\mathcal{U}_i : \mathcal{U}_{i+1}\) of such universes closed under all type formers, but we will leave universe levels typically ambiguous.

We call a type \(A\) a proposition if all elements of \(A\) are equal, i.e. if \((x : A) \rightarrow (y : A) \rightarrow x = y\) is provable. We write \(\text{hProp} = \Sigma(A : \mathcal{U}).\text{isProp}(A)\) for the type of propositions, and we implicitly insert a first projection if necessary, e.g. for \(A : \text{hProp}\), we may write \(x : A\) rather than \(x : \text{fst}(A)\). A type \(A\) is a set, \(A : \text{hSet}\), if \((x = y) : \text{hProp}\) for every \(x, y : A\).

By \(\exists(x : A).B(x)\), we mean the propositional truncation of \(\Sigma(x : A).B(x)\), and if \((a, b) : \Sigma(x : A).B(x)\) then \(|(a, b)| : \exists(x : A).B(x)\). The elimination rule of \(\exists(x : A).B(x)\) only allows to define functions into propositions. By convention, we write \(\exists k.P(k)\) for \(\exists(k : \mathbb{N}).P(k)\). Finally, we write \(A + B\) for the sum type, \(\mathbf{0}\) for the empty type, \(\mathbf{1}\) for the type with exactly one element \(\ast\), \(\mathbf{2}\) for the type with two elements \(\mathbf{ff}\) and \(\mathbf{tt}\), and \(\sim A\) for \(A \rightarrow A\).
The law of excluded middle (LEM) says that, for every proposition \( P \), we have \( P \lor \neg P \). Since we explicitly work with constructive notions of ordinals, we do not assume LEM, but rather use it as a taboo: a statement is not provable constructively if it implies LEM. Another, weaker, constructive taboo is the weak limited principle of omniscience WLPO: It says that any sequence \( s : \mathbb{N} \to 2 \) is either constantly \( \text{ff} \), or it is not constantly \( \text{ff} \).

### 3 Three Constructions of Types of Ordinals

We consider three concrete notions of ordinals in this paper, together with their order relations \(<\) and \(\leq\). The first notion is the one of Cantor normal forms, written \( \text{Cnf} \), whose order is decidable. The second, written \( \text{Brw} \), are Brouwer Trees, implemented as a higher inductive-inductive type. Finally, we consider the type \( \text{Ord} \) of ordinals that were studied in the HoTT book [40], whose order is undecidable, in general. In the current section, we briefly give the three definitions and leave the discussion of results for afterwards.

#### 3.1 Cantor Normal Forms as a Subset of Binary Trees

In classical set theory, every ordinal \( \alpha \) can be written uniquely in Cantor normal form

\[
\alpha = \omega^{\beta_1} + \omega^{\beta_2} + \cdots + \omega^{\beta_n} \quad \text{with} \quad \beta_1 \geq \beta_2 \geq \cdots \geq \beta_n
\]  

for some natural number \( n \) and ordinals \( \beta_i \). If \( \alpha < \varepsilon_0 \), then \( \beta_i < \alpha \), and we can represent \( \alpha \) as a finite binary tree (with a condition) as follows [10, 12, 25, 32]. Let \( T \) be the type of unlabeled binary trees, i.e. the inductive type with suggestively named constructors \( 0 : T \) and \( \omega^{\leftarrow +} : T \times T \to T \). Let the relation \( < \) be the lexicographical order, i.e. generated by the following clauses:

\[
\begin{align*}
0 &< \omega^a + b \\
\omega^a + b &< \omega^c + d \\
b &< d \quad \text{where} \quad \omega^a + b < \omega^c + d.
\end{align*}
\]

We have the map \( \text{left} : T \to T \) defined by \( \text{left}(0) :\equiv 0 \) and \( \text{left}(\omega^a + b) :\equiv a \) which gives us the left subtree (if it exists) of a tree. A tree is a Cantor normal form (CNF) if, for every \( \omega^s + t \) that the tree contains, we have \( \text{left}(t) \leq s \), where \( s \leq t :\equiv (s < t) \lor (s = t) \); this enforces the condition in (1). For instance, both trees \( 1 :\equiv \omega^0 + 0 \) and \( \omega :\equiv \omega^1 + 0 \) are CNFs. Formally, the predicate \( \text{isCNF} \) is defined inductively by the two clauses

\[
\begin{align*}
\text{isCNF}(0) \\
\text{isCNF}(s) \to \text{isCNF}(t) \to \text{left}(t) \leq s \to \text{isCNF}(\omega^s + t).
\end{align*}
\]

We write \( \text{Cnf} :\equiv \Sigma(t : T).\text{isCNF}(t) \) for the type of Cantor normal forms. We often omit the proof of \( \text{isCNF}(t) \) and call the tree \( t \) a CNF if no confusion is caused.

#### 3.2 Brouwer Trees as a Quotient Inductive-Inductive Type

As discussed in the introduction, Brouwer ordinal trees (or simply Brouwer trees) are in functional programming often inductively generated by the usual constructors of natural numbers (zero and successor) and a constructor that gives a Brouwer tree for every sequence of Brouwer trees. To state a refined (correct in a sense that we will make precise and prove) version, we need the following notions:
Let $A$ be a type and $\preceq : A \to A \to \text{hProp}$ be a binary relation. If $f$ and $g$ are two sequences $\mathbb{N} \to A$, we say that $f$ is simulated by $g$, written $f \preceq g$, if $f \preceq g : \equiv \forall k. \exists n. f(k) \preceq g(n)$. We say that $f$ and $g$ are bisimilar with respect to $\prec$, written $f \approx \prec g$, if we have both $f \preceq g$ and $g \preceq f$. A sequence $f : \mathbb{N} \to A$ is increasing with respect to $\prec$ if we have $\forall k. f(k) \prec f(k + 1)$.

Our type of Brouwer trees is a quotient inductive-inductive type \cite{Brw}, where we simultaneously construct the type $\text{Brw} : \text{hSet}$ together with a relation $\leq : \text{Brw} \to \text{Brw} \to \text{hProp}$. The constructors for $\text{Brw}$ are

- \text{zero} : \text{Brw}
- \text{succ} : \text{Brw} \to \text{Brw}
- \text{limit} : (\mathbb{N} \to \text{Brw}) \to \text{Brw}
- \text{bisim} : (f : \mathbb{N} \to \text{Brw}) \to f \approx \leq g \to \text{limit} f = \text{limit} g,

where we write $x < y$ for $\text{succ} x \leq y$ in the type of limit. Simulations thus use $\leq$ and the increasing predicate uses $<$, as one would expect. The truncation constructor, ensuring that $\text{Brw}$ is a set, is kept implicit in the paper (but is explicit in the Agda formalisation).

The constructors for $\leq$ are the following, where each constructor is implicitly quantified over the variables $x, y, z$:

- $\leq$-zero : $\text{zero} \leq x$
- $\leq$-trans : $x \leq y \to y \leq z \to x \leq z$
- $\leq$-succ-mono : $x \leq y \to \text{succ} x \leq \text{succ} y$
- $\leq$-cocone : $(k : \mathbb{N}) \to x \leq f(k) \to x \leq \text{limit} f$
- $\leq$-limiting : $(\forall k. f(k) \leq x) \to \text{limit} f \leq x$

The truncation constructor, which ensures that $x \leq y$ is a proposition, is again kept implicit.

We hope that the constructors of $\text{Brw}$ and $\leq$ are self-explanatory. $\leq$-cocone ensures that $\text{limit} f$ is indeed an upper bound of $f$, and $\leq$-limiting witnesses that it is the least upper bound or, from a categorical point of view, the (co)limit of $f$.

By restricting to limits of increasing sequences, we can avoid multiple representations of the same ordinal (as otherwise e.g. $a = \text{limit} (\lambda _\_ a)$). It is possible to drop this restriction, if one also strengthens the bisim constructor to witness antisymmetry – however we found this version of $\text{Brw}$ significantly harder to work with.

### 3.3 Extensional Wellfounded Orders

The third notion of ordinals that we consider is the one studied in the HoTT book \cite{HoTT}. This is the notion which is closest to the classical definition of an ordinal as a set with a trichotomous, wellfounded, and transitive order, without a concrete representation. Requiring trichotomy leads to a notion that makes many constructions impossible in a setting where the law of excluded middle is not assumed. Therefore, when working constructively, it is better to replace the axiom of trichotomy by extensionality.
Concretely, an ordinal in the sense of [40, Def 10.3.17] is a type\(^1\) \(X\) together with a relation \(\prec: X \to X \to \text{hProp}\) which is transitive, extensional (any two elements of \(X\) with the same predecessors are equal), and wellfounded (every element is accessible, where accessibility is the least relation such that \(x\) is accessible if every predecessor of \(x\) is accessible.) – we will recall the precise definitions in Section 4. We write \(\text{Ord}\) for the type of ordinals in this sense. Note the shift of universes that happens here: the type \(\text{Ord}\) of ordinals with \(X : \mathcal{U}_i\) is itself in \(\mathcal{U}_{i+1}\). We are mostly interested in \(\text{Ord}_0\), but note that \(\text{Ord}_0\) lives in \(\mathcal{U}_1\), while \(\text{Cnf}\) and \(\text{Brw}\) both live in \(\mathcal{U}_0\).

We also have a relation on \(\text{Ord}\) itself. Following [40, Def 10.3.11 and Cor 10.3.13], a simulation between ordinals \((X, \prec_X)\) and \((Y, \prec_Y)\) is a function \(f : X \to Y\) such that:

\begin{enumerate}[(a)]
  \item \(f\) is monotone: \((x_1 \prec_X x_2) \Rightarrow (f x_1 \prec_Y f x_2)\); and
  \item for all \(x : X\) and \(y : Y\), if \(y \prec_Y f x\), then we have an \(x_0 \prec_X x\) such that \(f x_0 = y\).
\end{enumerate}

We write \(X \leq Y\) for the type of simulations between \((X, \prec_X)\) and \((Y, \prec_Y)\). Given an ordinal \((X, \prec)\) and \(x : X\), the initial segment of elements below \(x\) is given as \(X_{/x} : = \Sigma(y : X).y \prec x\).

Following [40, Def 10.3.19], a simulation \(f : X \leq Y\) is bounded if we have \(y : Y\) such that \(f\) induces an equivalence \(X \simeq Y_{/y}\). We write \(X < Y\) for the type of bounded simulations. This completes the definition of \(\text{Ord}\) together with type families \(\leq\) and \(<\).

## 4 An Abstract Axiomatic Framework for Ordinals

Which properties do we expect a type of ordinals to have? In this section, we go up one level of abstraction. We consider a type \(A\) with type families \(<\) and \(\leq : A \to A \to \mathcal{U}\), and discuss the properties that \(A\) with \(<\) and \(\leq\) can have. In Section 3, we introduced each of the types \(\text{Cnf}, \text{Brw},\) and \(\text{Ord}\) together with its relations \(<\) and \(\leq\). Note that \(\leq\) is the reflexive closure of \(<\) in the case of \(\text{Cnf}\), but for \(\text{Brw}\) and \(\text{Ord}\), this is not constructively provable. In this section, we consider which properties they satisfy.

### 4.1 General Notions

\(A\) is a set if it satisfies the principle of unique identity proofs, i.e. if every identity type \(a = b\) with \(a, b : A\) is a proposition. Similarly, \(<\) and \(\leq\) are valued in propositions if every \(a < b\) and \(a \leq b\) is a proposition. A relation \(<\) is reflexive if we have \(a < a\), irreflexive if it is pointwise not reflexive \(\neg(a < a)\), transitive if \(a < b \Rightarrow b < c \Rightarrow a < c\), and antisymmetric if \(a < b \Rightarrow b < a \Rightarrow a = b\). Further, the relation \(<\) is connex if \((a < b) \lor (b < a)\) and trichotomous if \((a < b) \lor (a = b) \lor (b < a)\).

\textbf{Theorem 1.} Each of \(\text{Cnf}, \text{Brw},\) and \(\text{Ord}\) is a set, and their relations \(<\) and \(\leq\) are all valued in propositions. In each case, both \(\leq\) and \(\leq\) are transitive, \(<\) is irreflexive, and \(\leq\) is reflexive and antisymmetric. For \(\text{Cnf}\), the relation \(<\) is trichotomous and \(\leq\) connex; for \(\text{Ord}\), these statements are equivalent to the law of excluded middle.

Proving that \(\leq\) for \(\text{Brw}\) is antisymmetric is challenging because of the path constructors in the inductive-inductive definition of Brouwer trees. Antisymmetry and other technical properties discussed below require us to characterise the relation \(\leq\) more explicitly, using an encode-decode argument [29]. By induction on \(x\) and \(y\), we define the family \(\text{Code}\) such that \((\text{Code}\, x\, y) \leftrightarrow (x \leq y)\). The cases for point constructors are unsurprising; for example, we define

\footnote{Note that [40, Def 10.3.17] asks for \(X\) to be a set, but this follows from the rest of the definition and we therefore drop this requirement.}
The difficult part is defining $\text{Code}$ for the path constructor $\text{bisim}$. If for example we have $g \approx h$, we need to show that $\text{Code}(\text{limit } f)(\text{limit } g) = \text{Code}(\text{limit } f)(\text{limit } h)$. The core argument is easy; using the bisimulation $g \approx h$, one can translate between indices for $g$ and $h$ with the appropriate properties. However, this example already shows why this becomes tricky: The bisimulation gives us inequalities $(\leq)$, but the translation requires instances of $\text{Code}$, which means that $\text{toCode} : (x \leq y) \to (\text{Code } x y)$ has to be defined mutually with $\text{Code}$. This is still not sufficient: In total, the mutual higher inductive-inductive construction needs to simultaneously prove and construct $\text{Code}$, $\text{toCode}$, versions of transitivity and reflexivity of $\text{Code}$ as well several auxiliary lemmas. The complete definition is presented in the Agda formalisation (file $\text{BrouwerTree.Code}$). Once the definition of $\text{Code}$ is shown correct, many technical properties are simple consequences.

From now on, we will assume that $A$ is a set and that $<$ and $\leq$ are valued in propositions.

## 4.2 Extensionality and Wellfoundedness

Following [40, Def 10.3.9], we call a relation $<$ extensional if, for all $a, b : A$, we have $(\forall c. c < a \leftrightarrow c < b) \to b = a$, where $\leftrightarrow$ denotes “if and only if” (functions in both directions). Extensionality of $<$ for $\text{Brw}$ is true, but non-trivial – note that it fails for the “naive” version of $\text{Brw}$, where the path constructor $\text{bisim}$ is missing.

$\blacktriangleright$ **Theorem 2.** For each of $\text{Cnf}$, $\text{Brw}$, $\text{Ord}$, both $<$ and $\leq$ are extensional.

We use the inductive definition of accessibility and wellfoundedness (with respect to $<$) by Aczel [1]. Concretely, the type family $\text{acc} : A \to \mathbb{U}$ is inductively defined by the constructor

$$\text{access} : (a : A) \to ((b : A) \to b < a \to \text{acc}(b)) \to \text{acc}(a).$$

An element $a : A$ is called accessible if $\text{acc}(a)$, and $<$ is wellfounded if all elements of $A$ are accessible. It is well known that the following induction principle can be derived from the inductive presentation [40]:

$\blacktriangleright$ **Lemma 3 (Transfinite Induction).** Let $<$ be wellfounded and $P : A \to \mathbb{U}$ be a type family such that $\forall a. (\forall b < a. P(b)) \to P(a)$. Then, it follows that $\forall a. P(a)$.

In turn, transfinite induction can be used to prove that there is no infinite decreasing sequence if $<$ is wellfounded: $\neg (\Sigma(f : \mathbb{N} \to A). (i : \mathbb{N}) \to f(i + 1) < f(i))$. A direct corollary is that if $<$ is wellfounded and valued in propositions, then its reflexive closure $(x < y) \equiv(x = y)$ is also valued in propositions, as $b < a$ and $b = a$ are mutually exclusive propositions.

$\blacktriangleright$ **Theorem 4.** For each of $\text{Cnf}$, $\text{Brw}$, $\text{Ord}$, the relation $<$ is wellfounded.

The proof for $\text{Brw}$ again makes crucial use of our encode-decode characterisation of $\leq$. Whenever $x < \text{limit } f$, we can use the characterisation to find an $n : \mathbb{N}$ such that $x < f(n)$, which allows us to proceed with an inductive proof of wellfoundedness. Note that the results stated so far in particular mean that $\text{Cnf}$ and $\text{Brw}$ can be seen as elements of $\text{Ord}$ themselves.
4.3 Classification as Zero, a Successor, or a Limit

All standard formulations of ordinals allow us to determine a minimal ordinal \(\text{zero}\) and (constructively) calculate the \(\text{successor}\) of an ordinal, but only some allows us to also calculate the \(\text{supremum}\) or \(\text{limit}\) of a collection of ordinals.

4.3.1 Assumptions

We have so far not required a relationship between \(<\) and \(\leq\), but we now need to do so in order for the concepts we define to be meaningful. We assume:

(A1) \(<\) is transitive and irreflexive;
(A2) \(\leq\) is reflexive, transitive, and antisymmetric;
(A3) we have \((<) \subseteq (\leq)\) and \((< \circ \leq) \subseteq (<)\).

The third condition 3 means that \((b < a) \rightarrow (b \leq a)\) and \((c < b) \rightarrow (b \leq a) \rightarrow (c < a)\). The “symmetric” variation

\[
(\leq \circ <) \subseteq (<)
\]

is true for \(\text{Cnf}\) and \(\text{Brw}\), but for \(\text{Ord}\), it is equivalent to the law of excluded middle – hence, we do not assume it. This constructive failure is known, and can be seen as motivation for \(\text{plump}\) ordinals [39, 37]. Of course, the above assumptions are satisfied if \(\leq\) is the reflexive closure of \(<\), but we again emphasise that this is not necessarily the case.

▶ Theorem 5. For each of \(\text{Cnf, Brw, Ord}\), assumptions 1 to 3 are satisfied. ◆

For the remaining concepts, we assume that \(<\) and \(\leq\) satisfy the discussed assumptions.

4.3.2 Zero and (Strong) Successors

Let \(a\) be an element of \(A\). It is \(\text{zero}\), or \(\text{bottom}\), if it is at least as small as any other element

\[
is\text{-zero}(a) := \forall b. a \leq b,
\]

and we say that the triple \((A, <, \leq)\) has a zero if we have an inhabitant of the type \(\Sigma(z : A).\text{is\text{-zero}(z)}\). Both the types “being a zero” and “having a zero” are propositions.

▶ Theorem 6. \(\text{Cnf, Brw, Ord}\) each have a zero. ◆

We say that \(a\) is a \(\text{successor}\) of \(b\) if it is the least element strictly greater\(^2\) than \(b\):

\[
(a \text{ is-suc\text{-of } } b) := (b < a) \times \forall x > b. x \geq a.
\]

We say that \((A, <, \leq)\) has successors if there is a function \(s : A \rightarrow A\) which calculates successors, i.e. such that \(\forall b. s(b)\) is-suc\text{-of } \(b\). “Calculating successors” and “having successors” are propositional properties, i.e. if a function that calculates successors exists, then it is unique. The following statement is simple but useful. Its proof uses assumption 3.

▶ Lemma 7. Let \(s : A \rightarrow A\) be given. The function \(s\) calculates successors if and only if \(\forall bx.(b < x) \leftrightarrow (s b \leq x)\). ◆

\(^2\) Note that \(>\) and \(\geq\) are the obvious symmetric notations for \(<, \leq\); they are \textit{not} newly assumed relations.
Dual to “$a$ is the least element strictly greater than $b$” is the statement that “$b$ is the greatest element strictly below $a$”, in which case it is natural to call $b$ the predecessor of $a$. If $a$ is the successor of $b$ and $b$ the predecessor of $a$, then we call $a$ the strong successor of $b$:

\[ a \text{ is-str-suc-of } b \equiv a \text{ is-suc-of } b \times \forall x < a. x \leq b. \]

We say that $A$ has strong successors if there is $s : A \to A$ which calculates strong successors, i.e. such that $\forall b. s(b)$ is-str-suc-of $b$. The additional information contained in a strong successor play an important role in our technical development. A function $f : A \to A$ is $<$-monotone or $\leq$-monotone if it preserves the respective relation.

**Theorem 8.** Each of the three types $\text{Cnf}$, $\text{Brw}$, $\text{Ord}$ has strong successors. The successor functions of $\text{Cnf}$ and $\text{Brw}$ are both $<$- and $\leq$-monotone. For the successor function of $\text{Ord}$, either monotonicity property is equivalent to the law of excluded middle.

For $\text{Cnf}$, the successor function is given by adding a leaf, for $\text{Brw}$ by the constructor with the same name, and for $\text{Ord}$, one forms the coproduct with the unit type.

### 4.3.3 Suprema and Limits

Finally, we consider suprema/least upper bounds of $\mathbb{N}$-indexed sequences. We say that $a$ is a supremum or the least upper bound of $f : \mathbb{N} \to A$, if $a$ is at least as large as every $f_i$, and if any other $x$ with this property is at least as large as $a$:

\[ (a \text{ is-sup-of } f) : \equiv (\forall i. f_i \leq a) \times (\forall x. (\forall i. f_i \leq x) \to a \leq x). \]

We say that $(A, \leq, <)$ has suprema if there is a function $\sqcup : (\mathbb{N} \to A) \to A$ which calculates suprema, i.e. such that $(f : \mathbb{N} \to A) \to (\sqcup f)$ is-sup-of $f$. The supremum of a sequence is unique if it exists, i.e. the type of suprema is propositional for a given sequence $f$. Both the properties “calculating suprema” and “having suprema” are propositions.

Every $a : A$ is trivially the supremum of the sequence constantly $a$, and therefore, “being a supremum” does not describe the usual notion of *limit ordinals*. One might consider $a$ a proper supremum of $f$ if $a$ is pointwise strictly above $f$, i.e. $\forall i. f_i < a$. This is automatically guaranteed if $f$ is increasing with respect to $<$, and in this case, we call $a$ the limit of $f$:

\[ _{-}\text{is-limit-of }_{-} : A \to (\mathbb{N} \xrightarrow{\leq} A) \to U \]

\[ a \text{ is-limit-of } (f, q) : \equiv a \text{ is-sup-of } f. \]

We say that $A$ has limits if there is a function limit : $(\mathbb{N} \xrightarrow{\leq} A) \to A$ that calculates limits.

Note that $\text{Cnf}$ cannot have limits since one can construct a sequence (see Theorem 22) which comes arbitrarily close to $\varepsilon_0$. This motivates the restriction to bounded sequences, i.e. a sequence $f$ with a $b : \text{Cnf}$ such that $f_i < b$ for all $i$.

**Theorem 9.** $\text{Cnf}$ does not have suprema or limits. $\text{Brw}$ has limits of increasing sequences by construction. $\text{Ord}$ also has limits of increasing sequences, and moreover limits of weakly increasing sequences (i.e. sequences increasing with respect to $\leq$).

Assuming the law of excluded middle, $\text{Cnf}$ has suprema (and thus limits) of arbitrary bounded sequences. If $\text{Cnf}$ has limits of bounded increasing sequences, then the weak limited principle of omniscience (WLPO) is derivable.

We expect that it is not constructively possible to calculate suprema (or even binary joins) in $\text{Brw}$, as it seems this would make it possible to decide if a limit reaches past $\omega + 1$ or not, which is a constructive taboo.
4.3.4 Classifiability

For classical set-theoretic ordinals, every ordinal is either zero, a successor, or a limit. We say that a notion of ordinals which allows this is has classification. This is very useful, as many theorems that start with “for every ordinal” have proofs that consider the three cases separately. In the same way as not all definitions of ordinals make it possible to calculate limits, only some formulations make it possible to constructively classify any given ordinal. We already defined what it means to be a zero in (2). We now also define what it means for a \( a : A \) to be a strong successor or a limit:

\[
is\text{-}\text{str}\text{-}\text{suc}(a) \equiv \Sigma(b : A). (a \text{ is}\text{-}\text{str}\text{-}\text{suc}\text{-}\text{of} b)
is\text{-}\text{lim}(a) \equiv \exists f : \mathbb{N} \to A. a \text{ is}\text{-}\text{lim}\text{-}\text{of} f.\]

All of \( \text{is}\text{-}\text{zero}(a) \), \( \text{is}\text{-}\text{str}\text{-}\text{suc}(a) \) and \( \text{is}\text{-}\text{lim}(a) \) are propositions; note that this is true even though \( \text{is}\text{-}\text{str}\text{-}\text{suc}(a) \) is defined without a propositional truncation.

\[\text{Lemma 10.} \text{ Any } a : A \text{ can be at most one out of } \{\text{zero}, \text{strong successor}, \text{limit}\}, \text{ and in a unique way. In other words, the type } \text{is}\text{-}\text{zero}(a) \cup \text{is}\text{-}\text{str}\text{-}\text{suc}(a) \cup \text{is}\text{-}\text{lim}(a) \text{ is a proposition.} \]

We say that an element of \( A \) is classifiable if it is zero or a strong successor or a limit. We say \( (A, <, \leq) \) has classification if every element of \( A \) is classifiable. By Lemma 10, \( (A, <, \leq) \) has classification exactly if the type \( \text{is}\text{-}\text{zero}(a) \cup \text{is}\text{-}\text{str}\text{-}\text{suc}(a) \cup \text{is}\text{-}\text{lim}(a) \) is contractible.

\[\text{Theorem 11.} \text{ Cnf and Brw have classification. Ord having classification would imply the law of excluded middle.} \]

Classifiability corresponds to a case distinction, but the useful principle from classical ordinal theory is the related induction principle:

\[\text{Definition 12 (classifiability induction).} \text{ We say that } (A, <, \leq) \text{ satisfies the principle of classifiability induction if the following holds: For every family } P : A \to h\text{Prop} \text{ such that} \]

\[
is\text{-}\text{zero}(a) \to P(a)
(a \text{ is}\text{-}\text{str}\text{-}\text{suc}\text{-}\text{of} b) \to P(b) \to P(a)
(a \text{ is}\text{-}\text{lim}\text{-}\text{of} f) \to (\forall i. P(f_i)) \to P(a),\]

we have \( \forall a. P(a) \).

Note that classifiability induction does not ask for successors or limits to be computable. Using Lemma 10, we get that classifiability induction implies classification. For the reverse, we need a further assumption:

\[\text{Theorem 13. Assume } (A, <, \leq) \text{ has classification and satisfies the principle of transfinite induction. Then } (A, <, \leq) \text{ satisfies the principle of classifiability induction.} \]

It is also standard in classical set theory that classifiability induction implies transfinite induction: showing \( P \) by transfinite induction corresponds to showing \( \forall x < a. P(x) \) by classifiability induction. In our setting, this would require strong additional assumptions, including the assumption that \( (x \leq a) \) is equivalent to \( (x < a) \lor (x = a) \), i.e. that \( \leq \) is the reflexive closure of \( < \). The standard proof works with several strong assumptions of this form, but we do not consider this interesting or useful, and concentrate on the results which work for the weaker assumptions that are satisfied for Brw and Ord (see Section 4.3.1).

\[\text{Theorem 14. Cnf and Brw satisfy classifiability induction, while Ord satisfying it again implies excluded middle.} \]
4.4 Arithmetic

Using the predicates is-zero(a), a is-suc-of b, and a is-sup-of f, we can define what it means for \((A, <, \leq)\) to have the standard arithmetic operations. We still work under the assumptions declared in Section 4.3.1 – in particular, we do not assume that e.g. limits can be calculated, which is important to make the theory applicable to Cnf.

▶ Definition 15 (having addition). We say that \((A, <, \leq)\) has addition if there is a function \(+ : A \to A \to A\) which satisfies the following properties:

\[
\begin{align*}
\text{is-zero}(a) & \to c + a = c \\
\text{a is-suc-of } b \to d \text{ is-suc-of } (c + b) & \to c + a = d \\
\text{a is-sup-of } f \to b \text{ is-sup-of } (\lambda i. c + f_i) & \to c + a = b
\end{align*}
\]

We say that \(A\) has unique addition if there is exactly one function \(+\) with these properties.

Note that (3) makes an assumption only for (strictly) increasing sequences \(f\); this suffices to define a well-behaved notion of addition, and it is not necessary to include a similar requirement for arbitrary sequences. Since \((\lambda i. c + f_i)\) is a priori not necessarily increasing, the middle term of (3) has to talk about the supremum, not the limit.

Completely analogously to addition, we can formulate multiplication and exponentation, again without assuming that successors or limits can be calculated:

▶ Definition 16 (having multiplication). Assuming that \(A\) has addition, we say that it has multiplication if we have a function \(\cdot : A \to A \to A\) that satisfies the following properties:

\[
\begin{align*}
\text{is-zero}(a) & \to c \cdot a = a \\
\text{a is-suc-of } b \to c \cdot a = c \cdot b + c \\
\text{a is-sup-of } f \to b \text{ is-sup-of } (\lambda i. c \cdot f_i) & \to c \cdot a = b
\end{align*}
\]

\(A\) has unique multiplication if it has unique addition and there is exactly one function \(\cdot\) with the above properties.

▶ Definition 17 (having exponentation). Assume \(A\) has addition and multiplication. We say that \(A\) has exponentation with base \(c\) if we have a function \(\exp(c, -) : A \to A\) that satisfies the following properties:

\[
\begin{align*}
\text{is-zero}(b) & \to a \text{ is-suc-of } b \to \exp(c, b) = a \\
\text{a is-suc-of } b \to \exp(c, a) = \exp(c, b) \cdot c \\
\text{a is-sup-of } f \to \neg \text{is-zero}(c) \to b \text{ is-sup-of } (\exp(c, f_i)) & \to \exp(c, a) = b \\
\text{a is-sup-of } f \to \text{is-zero}(c) \to \exp(c, a) = c
\end{align*}
\]

\(A\) has unique exponentation with base \(c\) if it has unique addition and multiplication, and if \(\exp(c, -)\) is unique.

▶ Theorem 18. Cnf has addition, multiplication, and exponentiation with base \(\omega\) (all unique), Brw has addition, multiplication and exponentiation with every base (all unique), and Ord has addition and multiplication.
For Cnf, arithmetic is defined by pattern matching on the trees. Addition\(^3\) is given as

\[
0 + b :\equiv b \\
\omega a + 0 :\equiv a \\
(\omega a + c) + (\omega b + d) :\equiv \begin{cases} 
\omega b + d & \text{if } a < b \\
\omega a + (c + (\omega b + d)) & \text{otherwise},
\end{cases}
\]

multiplication as

\[
0 \cdot b :\equiv 0 \\
\omega a \cdot 0 :\equiv 0 \\
a \cdot (\omega b + d) :\equiv a + a \cdot d \\
(\omega a + c) \cdot (\omega b + d) :\equiv (\omega a + b + 0) + (\omega c) \cdot d \quad \text{if } b \neq 0,
\]

and exponentiation with base \(\omega\) by \(\omega a :\equiv \omega a \cdot 0\). These definitions are standard. Novel is our proof of correctness in the sense of Definitions 15–17, which we achieve by defining the inverse operations of subtraction and division.

Arithmetic on Brw is defined by recursion on the second argument, following the clauses of the specifications in Definitions 15–17. Since the constructor limit only accepts an increasing sequence, it is necessary to prove mutually with the definition that the operations are monotone and preserve increasing sequences. However, the case \(c = 0\) needs to be treated separately since neither pointwise multiplication nor exponentiation with 0 preserves increasingness. This makes it crucial to use classification (Theorem 11) and, in particular, that it is decidable whether \(c :\Brw\) is zero.

Addition on Ord is given by disjoint union \(A \uplus B\) (with \(\text{inl}(a) \prec_{A \uplus B} \text{inr}(b)\)), and multiplication by Cartesian product \(A \times B\) with the reverse lexicographical order. We expect that exponentiation cannot be defined constructively: the “obvious” definition via function spaces gives a wellfounded order assuming the law of excluded middle [27], but it seems unlikely that it can be avoided.

5 Interpretations Between the Notions

In this section, we show how our three notions of ordinals can be connected via structure preserving embeddings.

5.1 From Cantor Normal Forms to Brouwer Trees

The arithmetic operations of Brw allow the construction of a function \(\CtoB : \Cnf \to \Brw\) in a canonical way. We define \(\CtoB : \Cnf \to \Brw\) by:

\[
\CtoB(0) :\equiv \text{zero} \\
\CtoB(\omega a + b) :\equiv \omega^{\CtoB(a)} + \CtoB(b)
\]

▶ **Theorem 19.** The function \(\CtoB\) preserves and reflects \(<\) and \(\leq\), i.e., \(a < b \leftrightarrow \CtoB(a) < \CtoB(b)\), and \(a \leq b \leftrightarrow \CtoB(a) \leq \CtoB(b)\).

\(^3\) Caveat: \(\uplus\) is a notation for the tree constructor, while \(+\) is an operation that we define. We use parenthesis so that all operations can be read with the usual operator precedence.
To show that \( \text{CtoB} \) preserves \(<\), we first prove that Brouwer trees of the form \( \omega^x \) are additive principal: if \( a < \omega^x \) then \( a + \omega^x = \omega^x \) – a property not true for the “naive” version of Brouwer trees without path constructors. By reflecting \( \leq \) and antisymmetry, we have:

\[ \text{Corollary 20. The function CtoB is injective.} \]

We note that \( \text{CtoB} \) also preserves all arithmetic operations on \( \text{Cnf} \). For multiplication, this relies on \( \iota(n) \cdot \omega^x = \omega^x \) for \( \text{Brw} \), where \( \iota : \mathbb{N} \to \text{Brw} \) embeds the natural numbers as Brouwer trees, and \( \omega \equiv \text{limit} \iota \) – see our formalisation for details.

\[ \text{Theorem 21. CtoB commutes with addition, multiplication, and exponentiation with base} \omega. \]

Lastly, as expected, Brouwer trees define bigger ordinals than Cantor normal forms: when embedded into \( \text{Brw} \), all Cantor normal forms are below \( \varepsilon_0 \), the limit of the increasing sequence \( \omega, \omega^\omega, \omega^{\omega^\omega}, \ldots \)

\[ \text{Theorem 22. For all} a : \text{Cnf}, \text{ we have CtoB}(a) < \text{limit}(\lambda k. \omega \uparrow \uparrow k), \text{where} \omega \uparrow \uparrow 0 \equiv \omega \text{ and} \omega \uparrow \uparrow (k + 1) \equiv \omega^{\omega \uparrow \uparrow k}. \]

5.2 From Brouwer Trees to Extensional Wellfounded Orders

As \( \text{Brw} \) comes with an order that is extensional, wellfounded, and transitive, it can itself be seen as an element of \( \text{Ord} \). Every “subtype” of \( \text{Brw} \) (constructed by restricting to trees smaller than a given tree) inherits this property, giving a canonical function from Brouwer trees to extensional, wellfounded orders. We define

\[ \text{BtoO}(a) = \Sigma(y : \text{Brw}).(y < a). \]

with order relation \((y, p) < (y', p')\) if \( y < y' \). This extends to a function \( \text{BtoO} : \text{Brw} \to \text{Ord} \). The first projection gives a simulation \( \text{BtoO}(a) \leq \text{Brw} \). Using extensionality of \( \text{Brw} \), this implies that \( \text{BtoO} \) is an embedding from \( \text{Brw} \) into \( \text{Ord} \). Using that \(<\) on \( \text{Brw} \) is propositional, and that carriers of orders are sets, it is also not hard to see that \( \text{BtoO} \) is order-preserving:

\[ \text{Lemma 23. The function BtoO : Brw \to Ord is injective, and preserves < and \leq.} \]

A natural question is whether the above result can be strengthened further, i.e. whether \( \text{BtoO} \) is a simulation. Using LEM to find a minimal simulation witness, this is possible:

\[ \text{Theorem 24. Under the assumption of the law of excluded middle, the function BtoO : Brw \to Ord is a simulation.} \]

We do not know whether the reverse of Theorem 24 is provable, but from the assumption that \( \text{BtoO} \) is a simulation, we can derive another constructive taboo:

\[ \text{Theorem 25. If the map BtoO : Brw \to Ord is a simulation, then WLPO holds.} \]

We trivially have \( \text{BtoO}(\text{zero}) = 0 \). One can further prove that \( \text{BtoO} \) commutes with limits, i.e. \( \text{BtoO}(\text{limit}(f)) = \text{lim}(\text{BtoO} \circ f) \). However, \( \text{BtoO} \) does not commute with successors; it is easy to see that \( \text{BtoO}(x \uplus 1) = \text{BtoO}(\text{succ} x) \), but the other direction implies WLPO. This also means that \( \text{BtoO} \) does not preserve the arithmetic operations but “over-approximates” them, i.e. we have \( \text{BtoO}(x + y) \geq \text{BtoO}(x \uplus \text{BtoO}(y)) \) and \( \text{BtoO}(x \cdot y) \geq \text{BtoO}(x \times \text{BtoO}(y)) \).
6 Conclusions and Future Directions

We have demonstrated that three very different implementations of ordinal numbers, namely Cantor normal forms ($Cnf$), Brouwer ordinal trees ($Brw$), and extensional wellfounded orders ($Ord$), can be studied in a single abstract setting in the context of homotopy type theory. We hope that our development may shed light on other constructive or formalised approaches to ordinals also in other settings [31, 7, 6, 34].

Cantor normal forms are a formulation where most properties are decidable, while the opposite is the case for extensional wellfounded orders. Brouwer ordinal trees sit in the middle, with some of its properties being decidable. This aspect is not discussed in full in this paper; we only have included Theorem 14. It is easy to see that, for $x : Brw$, it is decidable whether $x$ is finite; in other words, the predicate $(\omega \leq \_ : Brw \to hProp$ is decidable, while $(\omega < \_)$ is decidable if and only if WLPO holds.

If $x$ is finite, then the predicates $(x = \_)$, $(x \leq \_)$, and $(x < \_)$ are also decidable. We have a further proof that, if $c : Cnf$ is smaller than $\omega^2$, then the families $(CtoB c \leq \_)$ and $(CtoB c < \_)$ are semidecidable, where semidecidability can be defined using the Sierpinski space [3, 13, 42].

Thus, each of the canonical maps $CtoB : Cnf \to Brw$ and $BtoO : Brw \to Ord$ embeds the “more decidable” formulation of ordinals into the “less decidable” one. Naturally, they both also include a “smaller” type of ordinals into a “larger” one: While every element of $Cnf$ represents an ordinal below $\epsilon_0$, $Brw$ can go much further. It would be interesting to consider more powerful ordinal notation systems such as those based on the Veblen functions [41, 35] or collapsing functions [4, 9], and see how these compare to Brouwer trees. Another avenue for potentially extending Cantor normal forms would be using superleaves [19]; we do not know how such a “bigger” version of $Cnf$ would compare to $Brw$.

Since $Brw$ can be viewed as an element of $Ord$, the latter can clearly reach larger ordinals than the former. This is of course not surprising; the Burali-Forti argument [5, 11] shows that lower universes cannot reach the same ordinals as higher universes. Another obstruction for $Brw$ to reach the full power of $Ord$ is the fact that $Brw$ only includes limits of $N$-indexed sequences. To overcome this problem, one can similarly construct higher number classes as quotient inductive-inductive types, e.g. a type $Brw_3$ closed under limits of $Brw$-indexed sequences, and then more generally types $Brw_{n+1}$ closed under limits of $Brw_n$-indexed sequences, and so on.

Finally, there are interesting connections between the ordinals we can represent and the proof-theoretic strength of the ambient type theory: each proof of wellfoundedness for a system of ordinals is also a lower bound for the strength of the type theory it is constructed in. It is well known that definitional principles such as simultaneous inductive-recursive definitions [22] and higher inductive types [30] can increase the proof-theoretical strength, and so, we hope that they can also be used to faithfully represent even larger ordinals.

References


