The Algebra of Infinite Sequences: Notations and Formalization

Nachum Dershowitz[1⋆], Jean-Pierre Jouannaud[2,3], and Qian Wang[2]

[1] School of Computer Science, Tel Aviv University, Ramat Aviv, Israel
[2] School of Software, Tsinghua University, Beijing, China
[3] LIX, École Polytechnique, Palaiseau, France

Abstract. We propose some convenient notations for expressing complicated properties of finite and infinite, ordinal-indexed sequences. The algebra of ordinal-indexed sequences is being implemented in the proof-assistant Coq, together with the algebra of ordinals represented in Cantor normal form.

1 Purpose

In infinite combinatorics, program verification, and other subjects of mathematical interest, one often encounters a need to describe properties of finite and infinite sequences. For example, a "good" sequence in well-quasi-ordering (wqo) theory is an infinite sequence containing at least one element that is related (in a quasi-ordering $\precsim$) to a subsequent element.

We have been seeking convenient notations and operations that would enable us to easily express properties of sequences that are of interest. Not finding anything suiting our needs, we propose some in this note.

In programming and proof development, the idea of dependent types has established itself as most useful for various applications. In practice, such types depend either on natural numbers, such as lists depending on a natural number expressing their length, or on a proposition whose proof is required for type-checking purposes. Infinite lists, as studied in this note, will depend instead on ordinals. This paper therefore contains a formalization in Coq of ordinals in Cantor normal form, and their use for formalizing infinite sequences. Proofs of most statements are provided in the online supplement.

2 Finite Paths

We begin with the simpler finite case.

Let V be a (finite or infinite) alphabet, which we call *points* (akin to vertices), and let V^n be all n-tuples of points, thought of as n-element sequences, which we refer to as *paths*. The *length* of a path $s_0, s_1, \ldots, s_{n-1}$ is n, the number of its

⋆ The first author was on leave at INRIA-LIAMA (Sino French Lab in Computer Science, Automation and Applied Mathematics), Tsinghua University, Beijing, China.

points. This definition of length, although somewhat unusual, generalizes best to transfinite paths.

Note that the path s_0 has length one, the empty (pointless) path being obtained for $n = 0$. We will have little recourse to empty paths (but ε would be a natural symbolization): from here on, a path will always have $n > 0$ points.

A binary relation $E \subseteq V \times V$ consists of pairs of points, and may be thought of as a set of "colored" *steps* (edges) between points, the color being E. More generally, an n-ary relation (hyperedge) $R \subseteq V^n$ would represent a set of paths $s_0, s_1, \ldots, s_{n-1}$, of length n, consisting of n points and $n-1$ consecutive steps. The empty relation is $\varnothing$.

It will prove handy to view any single path as a singleton relation and to view an individual point $p \in V$ as a path of length 1 – the starting point and ending point of the path being one and the same. A set $P \subseteq V$ of such zero-step paths may be thought of as a monadic property of points.

The *concatenation* of two paths $r = r_0, r_1, \ldots, r_m$ and $s = s_0, s_1, \ldots, s_n$ over V is simply $r \frown s = r_0, r_1, \ldots, r_m, s_0, s_1, \ldots, s_n$, as in formal language theory. The concatenation of two paths need not be a path. More relevant is the multiplication-like *join* operation: those two paths can be joined only if the second starts where the first leaves off, that is, if $r_m = s_0$, in which case their join is the path

$$rs = r_0, r_1, \ldots, r_{m-1}, s_0, s_1, \ldots, s_n$$

For binary relations ($m = 1$, $n = 1$), there is also *composition*, where $r \circ s = r_0, s_n$.[1]

As we are more interested in relations between points than in the points themselves, the basic operation on paths that is of interest to us is join, not concatenation or composition, which is why we simply use juxtaposition for it. It is then natural to let $\mathbf{1}$ denote the set of all singleton paths, V, since it is the unit element vis-à-vis join (whereas ε is the identity element for concatenation). We also let $\mathbf{2} = V \times V$ be the full Cartesian product, that is all possible single steps (that is, paths of length 2) between pairs of points.

If R and S are sets of paths,[2] then we denote by

$$RS = \{rs \ : \ r \in R \text{ and } s \in S\}$$

the set of all (possible) joins between their elements, starting with a path in R. As is the norm for formal languages, we use additive notation for union of sets, so $R + S = \{q \ : \ q \in R \text{ or } q \in S\}$. We also use the customary exponents. So the join unit is

$$S^0 = \mathbf{1}$$

the n-fold join ($n > 0$) is

$$S^{n+1} = S^n S$$

[1] A suitable notion of *composition* for relations that are wider than binary is non-trivial. See, for example, [4].

[2] It is tempting to propose the term *gene* for a path (sequence of steps) and *genome* for a set of genes.

and finite iteration of join is

$$S^* = \sum_{n<\omega} S^n$$

Using this Kleene-star notation, $\mathbf{2}^*$ is the set of all finite paths.

3 Infinite Paths

Paths, in general, need not be finite; they may be infinite or transfinite. The length $|s|$ of a finite or transfinite path $s = \{s_\alpha\}_{\alpha<\beta}$ (over V), for countable ordinal β, is $\beta - 1$, the number of edges in the path ($\beta - 1 = \beta$ for limit ordinal β). Note that this definition coincides with the previous one in case β is a finite ordinal.

The join of two paths needs to be defined differently when the length of the first is a limit ordinal (like ω), in which case the two paths are concatenated, there being no last element for the first path:

$$rs = \begin{cases} r_0, \ldots, r_{|\alpha|}, s_0, \ldots & |r| = \alpha + 1, r_{|\alpha|} = s_0 \\ \text{nonexistent} & |r| = \alpha + 1, r_{|\alpha|} \neq s_0 \\ r_0, \ldots, s_0, \ldots & \text{otherwise} \end{cases}$$

We have

$$|rs| = (|r| - 1) + |s|$$

where addition is the standard non-commutative addition of ordinals, and subtraction is defined so that $\alpha - \beta$ is the unique γ such that $\beta + \gamma = \alpha$, for $\alpha \geq \beta$. In particular, the predecessor of a limit ordinal is itself.

Joins are indexed as follows:

$$(rs)_\alpha = \begin{cases} r_\alpha & \alpha < |r| \\ s_{\alpha-|r|} & |r| \leq \alpha \end{cases}$$

Exponentiation of sets of paths acts as expected for nonlimit ordinals.[3] Let S be a set of paths.

$$S^0 = \mathbf{1} \qquad S^{\beta+1} = S^\beta S$$

For exponentiation to a limit ordinal λ, matters are significantly more complicated. To begin with, let $\mathbf{s}$ be some ordinal-indexed *sequence* $\{\mathbf{s}^{(\beta)}\}_{\beta<\gamma}$ of paths $\mathbf{s}^{(\beta)} \in S$ for which we want to define the infinite join $\prod_{\beta<\gamma} \mathbf{s}^{(\beta)} = \mathbf{s}^{(0)}\mathbf{s}^{(1)} \cdots \mathbf{s}^{(\beta)} \cdots$ of its elements in the given order. Let's abbreviate $\mathbf{s}^{(<\gamma)} = \prod_{\beta<\gamma} \mathbf{s}^{(\beta)}$.

Before we can figure out how to index the points in joined paths, we need to know how to measure the length of joined paths. The easy cases are:

$$\left|\mathbf{s}^{(<0)}\right| = 0 \qquad \left|\mathbf{s}^{(<\gamma+1)}\right| = \left|\mathbf{s}^{(<\gamma)}\right| + \left|\mathbf{s}^{(\gamma)}\right|$$

[3] We do not consider 0 to be a limit ordinal.

But what is the size of a limit join? Naturally, it is the limit of longer and longer joins:

$$\left|\mathbf{s}^{(<\lambda)}\right| = \sup_{\gamma<\lambda} \left|\mathbf{s}^{(<\gamma)}\right|$$

which is well-defined by ordinal induction.

Now, we are ready define infinite joins by specifying its elements one by one. Given an ordinal $\alpha < |\mathbf{s}^{(<\lambda)}|$, we need to find the path $\mathbf{s}^{(\gamma)}$ in the sequence $\{\mathbf{s}^{(\beta)}\}_{\beta<\lambda}$, and the position δ in that path such that

$$\mathbf{s}_\alpha^{(<\lambda)} = \mathbf{s}_\delta^{(\gamma)}$$

Clearly, γ is the largest ordinal such that $|\mathbf{s}^{(<\gamma)}| \leq \alpha$, and $\delta = \alpha - |\mathbf{s}^{(<\gamma)}|$.

We can now complete the definition of exponentiation by adding the limit case:

$$S^\lambda = \left\{\mathbf{s}^{(<\lambda)} \,:\, \mathbf{s} \in [\lambda \to S]\right\}$$

where $[\lambda \to S]$ are all λ-long sequences of paths in S.

For example, a binary relation E is *well-founded*,[4] or *strongly normalizing*, if it admits no (ordinary) infinite (ω) paths: $E^\omega = \varnothing$.

It also pays to have

$$S^{<\alpha} = \sum_{\beta<\alpha} S^\beta$$

Then we may view the star notation as shorthand for the ω case and use it for infinite paths as well as finite ones:

$$S^* = S^{<\omega}$$

More generally we might want any range of ordinals in the exponent, as in $S^{[\omega..\omega^2)}$ for $\sum_{\omega\leq\beta<\omega^2} S^\beta$.

With these definitions in hand, the following equalities hold:

$$\varnothing S = S\varnothing = \varnothing$$
$$\mathbf{1}S = S$$
$$(QR)S = Q(RS)$$

On the other hand, $S\mathbf{1} \neq S$, when S has limit paths. In other words, $\mathbf{1}$ is a *left* unit only.

4 Path Operations

Let $\Omega = \mathbf{2}^{<\omega_1}$ be all (finite or countably transfinite) paths over V. We can define (modal) unary operators on a set of paths S:

$$\begin{aligned} \textit{eventually}: &\quad \Diamond S = \Omega S \\ \textit{complement}: &\quad !\,S = \Omega \setminus S \\ \textit{always}: &\quad \Box S = !\Diamond !\, S \end{aligned}$$

[4] We allow ourselves the luxury of using this term even for non-transitive relations.

Thus, $R \subseteq \Box S$ means that every tail of a path in R satisfies S (i.e. belongs to the set S).[5] The binary *until* modality, $P \,\mathcal{U}\, S = (P\mathbf{2})^* S$, states that monadic P holds at every point in a path until S holds of the continuation of the path.

Let

$$\langle S \rangle = \{s_0 : \ s \in S\}$$

give just the first (source) elements of paths in S. This may be used to *filter* paths by elements that initiate other paths. For example, we say that R *escapes* from S if $S^\omega \subseteq \Diamond \langle R(R+S)^\omega \rangle \, \Omega$, meaning that in any ω-long path of R- and S-steps there is a point from which an R-step (perhaps leading out of the path) initiates a path in $R + S$ of length ω (see [1]).

We found it useful to define

$$\lfloor S \rfloor = \big\{(s_0, s_{|s|}) : \ s \in S, \ |s| \text{ not a limit}\big\} \cup \big\{(s_0, a) : \ s \in S, \ a \in V, \ |s| \text{ limit}\big\}$$

that is, the binary relation consisting of all single steps composed of all first (s_0) and last ($s_{|s|}$) elements in paths $s \in S$. If there is no last element, then the first relates to everything (a). Let

$$\lceil S \rceil = \{r : \ \lfloor r \rfloor = \lfloor s \rfloor, \ s \in S\}$$

be a *span* containing all possible paths r with the same beginnings and ends as paths $s \in S$. Finally, another convenient notion is that of a residual (initial segment):

$$S/R = \{Q : \ QR = S\}$$

for which

$$RS \cap T \subseteq (R \cap T/\Omega)(S \cap \Omega T)$$

These definitions allow us to easily express the conditions for badness and goodness, as used in wqo theory (see, for example, [2, Chap. 12]). A quasi-ordered set S is *good* if $S \subseteq \Diamond \lceil Q \rceil \, \Omega$, meaning that every path in S has a pair of (not necessarily consecutive) points in Q, and is *bad* otherwise. It is *perfect* if $S \subseteq \Diamond \lceil Q \rceil^\omega$, meaning that S has a Q-chain as a (noncontiguous) subsequence (like stepping stones).[6]

One may also define the reverse of a finite path (starting at the end and ending at the start): $s^\intercal = s_{|s|}, \ldots, s_0$. Similarly, $S^\intercal = \{s^\intercal : \ s \in S\}$ is the set of reverses.[7]

---

[5] Typically, S is defined in terms of a property of its initial point, so this means that every point in every path has that property.

[6] By a simple case of Ramsey's Theorem, a good ω-sequence is perfect.

[7] The reverse of a transfinite path is not ordinal-indexed. Rather, the order type of the reverse of a path of order type α is the reverse order type α^*. We do not deal here with such paths.

5 Formalization of Ordinals and Infinite Sequences

All the above definitions can be easily modeled in a proof assistant like Coq, and their algebraic properties proved formally; this is what we have begun to do for some basic properties. To this end, we first need ordinal numbers. We could of course work with a theory of ordinal numbers without providing an explicit representation of them, but extraction is then impossible in Coq. Since ordinals are not present in the standard library of Coq, we decided to start our development with ordinals represented in Cantor normal form, in which an ordinal is written

$$\omega^{\alpha_1} n_1 + \ldots + \omega^{\alpha_k} n_k + n_{k+1}$$

where α_i is an ordinal, n_i is a natural number (hence, $n_i \neq 0$), $\alpha_1 > \alpha_2 \ldots \alpha_k > 0$, k is a non-negative integer, as well as n_{k+1}. There is also a variant in which $n_i = 1$ and $\alpha_i \geq \alpha_{i+1}$. Also, in classical Cantor normal form, n_{k+1} appears instead as $\omega^0 n_{k+1}$. We found it simpler, however, to omit ω^0.

Note that it is easy to check if an arbitrary succession of monomials $\omega^{\alpha_i} n_i$ ending in a natural number, let us call it an *ordinal notation*, satisfies the constraint of being in Cantor normal form ($\alpha_1, \ldots, \alpha_k$ is a decreasing sequence of ordinals in Cantor normal form). It is very easy as well to characterize if an ordinal notation is the smallest ordinal, zero ($k = n_1 = 0$), a limit ordinal ($k \neq 0 = n_{k+1}$), as well as a successor ordinal ($n_{k+1} \neq 0$). Ordinal notations are the elements of our basic inductive type Ord in Coq, and Cantor normal forms are the subset of Ord defined by a predicate checking its well-formedness.

The algebra of infinite sequences is still under development. Infinite sequences are indexed by ordinals in Cantor normal form, hence belong to a type dependent on ordinals. Usually, types depend on natural numbers (to measure the size of a given data structure), or on a proposition (to carry proofs within terms that are often used for type-checking purposes). To the best of our knowledge, the use of ordinals in dependent types is new.

5.1 Ordinals in Cantor Normal Form in Coq

We provide some explanations about our development below. The current version of the development can be found at `https://github.com/superwalter/Sequences`.

We hope these definitions, notations, and development will prove useful, not only to those interested in the theory of well-quasi-orders, or logics of processes, but also to those who use these concepts. Well-quasi orders, in particular, have been extensively used in the study and verification of transition systems. See, for example, [3], where the author insists on constructivity of proofs, which could therefore, in principle, be carried out in Coq.

```
(*************************************************************)
(*We use CNF as an abbreviation for Cantor Normal Form Ordinals*)
(*************************************************************)
```

```
(*Ordinal notations*)
Inductive Ord : Set :=
| fin : nat -> Ord
| inf : nat -> Ord -> Ord -> Ord.

(*All subsequent operations are defined on ordinal notations*)
(*but work provided they are in CNF*)

(*Degree of an ordinal*)
Fixpoint degree (o : Ord) : Ord :=
  match o with
    | fin _ => fin 0
    | inf n p Q => p
  end.

(*Equality of ordinals*)
Fixpoint beq_ord (o o' : Ord) : bool :=
  match o, o' with
    | fin m, fin n => beq_nat m n
    | inf n p Q, inf n' p' Q' =>
      (beq_nat n n') && (beq_ord p p') && (beq_ord Q Q')
    | _, _ => false
  end.

(*Order between ordinals*)
Fixpoint btb_ord (o o' : Ord) : bool :=
  match o, o' with
    | fin m, fin n => (ltb n m)
    | fin _, inf _ _ _  => false
    | inf _ _ _, fin _ => true
    | inf n p Q, inf n' p' Q' =>
      (btb_ord p p') || (beq_ord p p') && (ltb n' n) ||
        (beq_ord p p') && (beq_nat n n') && (btb_ord Q Q')
  end.

(*Max of two ordinals*)
Definition max_ord o o' := if (btb_ord o o') then o else o'.

(*Cantor Normal Form (CNF)*)
(*Note that CNF is a predicate, not a type*)
(*Having CNF as a (sub-) type will occur later*)
(*once most properties of CNFs are proved; *)
(*predicates make it simpler in Coq*)
Fixpoint CNF (o : Ord) : bool :=
```

```
  match o with
    | fin _ => true
    | inf n p Q => ((ltb 0 n)) && (CNF p) &&
      (CNF Q) && (btb_ord p (degree Q))
  end.

(*Plus*)
Fixpoint ord_plus (o o' : Ord) :=
  match o with
    | fin m  =>
      match o' with
        | fin n => fin (m+n)
        | inf _ _ _ => o'
      end
    | inf n p Q =>
      match o' with
        | fin _ => inf n p (ord_plus Q o')
        | inf n' p' Q' =>
          if (btb_ord p p') then (inf n p (ord_plus Q o')) else
            if (beq_ord p p') then (inf (n+n') p' Q') else o'
      end
  end.

(*Pred*)
Fixpoint ord_pred (o : Ord) : Ord :=
  match o with
    | fin 0 => o
    | fin (S n) => fin n
    | inf n p o' => inf n p (ord_pred o')
  end.

(*Minus*)
Fixpoint ord_minus (o o' : Ord) :=
  match o, o' with
    | fin n, fin m => fin (n - m)
    | fin n, inf _ _ _ => o
    | inf n p q, fin _ => inf n p (ord_minus q o')
    | inf n p q, inf n' p' q' =>
      if (btb_ord p p') then inf n p (ord_minus q o') else
        if (beq_ord p p') then
          (if (ltb n' n) then inf (n-n') p (ord_minus q q') else
            ord_minus q q')
          else o
  end.
```

```
(*********************************************************************)
(*This ends up the development of ordinal notations*)
(*We now have everything we need to introduce CNFs as types*)
(*under the name of Cantor Normal Form Ordinals*)
(*********************************************************************)

Definition CNFO := {o : Ord | CNF o = true}.

Definition CNFO_plus (o : CNFO) (o' : CNFO) : CNFO.
destruct o as (o, CNFo); destruct o' as (o', CNFo').
exists (ord_plus o o').
apply CNF_ord_plus; trivial.
Defined.

Definition CNFO_pred (o : CNFO) : CNFO.
destruct o as (o, CNFO).
exists (ord_pred o).
apply CNF_ord_pred; trivial.
Defined.

Definition CNFO_minus (o : CNFO) (o' : CNFO) : CNFO.
destruct o as (o, CNFo); destruct o' as (o', CNFo').
exists (ord_minus o o').
apply CNF_ord_minus; trivial.
Defined.

Definition CNFO_nat (n : nat) : CNFO.
exists (fin n); simpl; trivial.
Defined.

Definition CNFO_btb (o o' : CNFO) :=
  btb_ord (proj1_sig o) (proj1_sig o').

Definition CNFO_beq (o o' : CNFO) :=
  beq_ord (proj1_sig o) (proj1_sig o').
```

5.2 Infinite Paths

We shall now proceed in a similar way to define the join of two paths, by first concatenating them.

```
(*************************************************************************)
(*paths over an alphabet A, that is sequences on A of an ordinal length.*)
(*A is assumed to be equipped with an equality predicate*)
(*************************************************************************)
```

```
 Variable A : Type.
 Variable eq_A : A -> A -> bool.

 Definition domain (o : CNFO) := {i: CNFO|CNFO_btb o i = true}.

 Definition path (o : CNFO) := (domain o) -> A.

(*The definition of a join is complex, and requires much effort*)
(*we skip it and refer to the development*)
Definition join (o o' : CNFO) (p : path o) (p' : path o') :
   option (path (CNFO_plus (CNFO_pred o) o')).
 ...
 Defined.

(************************************)
(*Set of paths*)
(This part is still under development*)
(************************************)
```

Acknowledgement

The comments of Bernhard Gramlich on an early draft are greatly appreciated.

References

1. N. Dershowitz, "Jumping and Escaping: Modular Termination and the Abstract Path Ordering", *Theoretical Computer Science* 464 (2012), pp. 35–47.
2. R. Diestel, *Graph Theory*, Springer (2005).
3. Jean Goubault-Larrecq, "A Constructive Proof of the Topological Kruskal Theorem", to be published (2013).
4. M. Marx, *Algebraic Relativization and Arrow Logic*, PhD Dissertation, University of Amsterdam, 1995.