JFP **27**, e13, 20 pages, 2017.
doi:10.1017/S0956796817000041

*Finiteness and rational sequences, constructively**

TARMO UUSTALU and NICCOLÒ VELTRI

Institute of Cybernetics, Tallinn University of Technology
Akadeemia tee 21, 12618 Tallinn, Estonia
(*e-mails:* tarmo@cs.ioc.ee, niccolo@cs.ioc.ee)

Abstract

Rational sequences are possibly infinite sequences with a finite number of distinct suffixes. In this paper, we present different implementations of rational sequences in Martin–Löf type theory. First, we literally translate the above definition of rational sequence into the language of type theory, i.e., we construct predicates on possibly infinite sequences expressing the finiteness of the set of suffixes. In type theory, there exist several inequivalent notions of finiteness. We consider two of them, listability and Noetherianness, and show that in the implementation of rational sequences the two notions are interchangeable. Then we introduce the type of lists with backpointers, which is an inductive implementation of rational sequences. Lists with backpointers can be unwound into rational sequences, and rational sequences can be truncated into lists with backpointers. As an example, we see how to convert the fractional representation of a rational number into its decimal representation and vice versa.

1 Introduction

Rational sequences, also called ultimately periodic sequences, are possibly infinite sequences with a finite number of distinct suffixes. The paradigmatic example of rational sequences that most people are familiar with is the decimal representation of rational numbers. For example, the decimal representation of $2/8$ is 0.25, and the one of $21/26$ is $0.807692307692307\ldots$, which is typically written $0.8\overline{076923}$ or $0.8(076923)$. The set of suffixes of 0.25 is $\{0.25, 25, 5, []\}$, where $[]$ is the empty sequence, while the one of $0.8(076923)$ is

$$\{0.8(076923), 8(076923), (076923), (769230), (692307), (923076), (230769), (307692)\}$$

Both sets of suffixes are finite. This is not the case for the set of suffixes of irrational numbers like π or $\sqrt{2}$.

* This work was supported by the ERDF funded project Coinduction, the Estonian Ministry of Education and Research institutional research grant no. IUT33-13 and the Estonian Science Foundation grant no. 9475.

More generally, rational sequences arise as solutions of systems of equations of the form

$$\begin{aligned} x_1 &\sim a_1 :: x_{i_1} \\ x_2 &\sim a_2 :: x_{i_2} \\ &\vdots \\ x_n &\sim a_n :: x_{i_n} \end{aligned} \tag{1}$$

where $a_1, \ldots, a_n$ are constants from a given alphabet, $::$ is a constructor, $x_1, \ldots, x_n$ are variables and $i : \{1, \ldots, n\} \to \{1, \ldots, n\}$. The system of equations in Equation (1) is an example of a system of iterative equations. Such systems have been studied in the 1970s in connection with potentially infinite computations (Elgot, 1975; Bloom and Elgot, 1976; Elgot *et al.*, 1978; Ginali, 1979), and have more recently received a categorical generalization (Adámek *et al.*, 2003).

In this paper, we present two different implementations of rational sequences in Martin–Löf type theory. The first is the literal translation of the definition of rational sequence into the language of type theory, i.e., the set of possibly infinite sequences with finitely many suffixes. What does it mean for a set to be finite in type theory? Constructively, several inequivalent notions of finiteness exist (Coquand and Spiwack, 2010). In this paper, we focus on two notions: listability and Noetherianness. Intuitively, a set A is listable, if one is able to construct a list xs containing all the elements in A; a set A is Noetherian, if one is eventually able to find a duplicate when given arbitrary elements from A one after the other. Listability and Noetherianness are classically equivalent predicates, but constructively listability is strictly stronger than Noetherianness. Interestingly, given a possibly infinite sequence xs, we will see that, if the set of suffixes of xs is Noetherian, then it is also listable. In fact, there exists a class of subsets of a given set, that we call orbits, for which Noetherianness implies listability, and the subset of all sequences over A given by suffixes of xs happens to be an orbit.

Afterwards, we also give an inductive implementation of rational sequences, reminiscent of the decimal representation of a rational number. We consider the type of lists with backpointers, i.e., lists with possibly an index at the very end, revealing the position in the list where the period starts. We show how a list with backpointers can be unwound into a rational sequence and how a rational sequence can be truncated into a list with backpointers. Rational sequences and lists with backpointers turn out to be isomorphic types in the appropriate proof-relevant sense.

We have fully formalized the results of this paper in the dependently typed programming language Agda (Norell, 2009). The formalization is available at `http://cs.ioc.ee/~niccolo/rational/`.

1.1 Structure of the paper

This paper is organized as follows. In Section 2, we give an overview of the type theory we are working in and we introduce some datatypes and terms that we use

throughout the paper. In Section 3, we discuss listable and Noetherian sets. We show how to extend these notions to subsets represented by predicates and prove that Noetherianness implies listability for orbits. We also introduce an alternative but equivalent implementation of finiteness specific for orbits. In Section 4, we introduce rational sequences as possibly infinite sequences with a rationality proof. We show that the choice of the notion of finiteness does not matter in this definition. In Section 5, we construct a corecursion principle for rational sequences. In Section 6, we introduce lists with backpointers and show their connection with rational sequences. In Section 7, we implement the decimal expansion of rational numbers as an example of programming with rational sequences and lists with backpointers. In Section 8, we describe related work, to finally draw some conclusions and discuss future work in Section 9.

2 Preliminaries

We consider Martin–Löf type theory with inductive and coinductive types with one universe $\mathscr{U}$. To define functions from inductive types or to coinductive types, we use guarded (co)recursion. When we write statements like "A is a type" or "A is a set", we mean $A : \mathscr{U}$. We allow dependent functions to have implicit arguments and indicated implicit argument positions with curly brackets (as in Agda). We write $\equiv$ for propositional equality (identity types) and $=$ for judgmental (definitional) equality. A type A is said to be a *proposition*, if it has at most one inhabitant, i.e., if the type

$$\mathsf{isProp}\, A = \Pi x, y : A.\, x \equiv y$$

is inhabited.

Let A be a type. The type $\mathsf{List}\, A$ of lists over the type A is inductively defined by the rules

$$\frac{}{[] : \mathsf{List}\, A} \qquad \frac{x : A \quad xs : \mathsf{List}\, A}{x :: xs : \mathsf{List}\, A}$$

The length of a list can be recursively defined as follows:

$$\begin{aligned} &\mathsf{length} : \mathsf{List}\, A \to \mathbb{N} \\ &\mathsf{length}\ [] \quad\ = \mathsf{zero} \\ &\mathsf{length}\ (x :: xs) = \mathsf{suc}\ (\mathsf{length}\ xs) \end{aligned}$$

Membership in a list is introduced as a relation $\in$ between elements of A and lists over A, and it is inductively defined by the rules

$$\frac{}{\mathsf{here} : x \in x :: xs} \qquad \frac{p : x \in xs}{\mathsf{there}\, p : x \in y :: xs}$$

We construct a function $\mathsf{index}\in$ that given a proof of $x \in xs$, returns the position of x in the list xs.

$$\begin{aligned} &\mathsf{index}\in : \Pi\{xs : \mathsf{List}\, A\}\Pi\{x : A\}.\, x \in xs \to \mathsf{Fin}\,(\mathsf{length}\, xs) \\ &\mathsf{index}\in\ \mathsf{here}\quad\ = \mathsf{zero} \\ &\mathsf{index}\in\ (\mathsf{there}\ r) = \mathsf{suc}\ (\mathsf{index}\in\ r) \end{aligned}$$

Given a proof $p : x \in xs$, we can construct the list $\mathsf{remove}\, xs\, p$ which is xs without the occurrence of x in the position specified by p:

$$\begin{aligned}
&\mathsf{remove} : \Pi xs : \mathsf{List} A\, \Pi\{x : A\}.\, x \in xs \to \mathsf{List}\, A \\
&\mathsf{remove}\ (x :: xs)\ \mathsf{here} \quad\ = xs \\
&\mathsf{remove}\ (x :: xs)\ (\mathsf{there}\, p) = \mathsf{remove}\ xs\ p
\end{aligned}$$

We say that a list has its head duplicated, if the head appears also in the tail.[1] The duplicate predicate Dup is defined by the rule

$$\frac{x \in xs}{\mathsf{Dup}\,(x :: xs)}$$

We indicate with 1 the unit type, whose only inhabitant is $* : 1$, with $\mathbb{N}$ the type of natural numbers and with $\mathsf{Fin}\, n$ the type of the first n natural numbers. The type $\mathsf{Vec}_n\, A$ of vectors over the type A with length n is inductively defined by the rules

$$\frac{}{[] : \mathsf{Vec}_{\mathsf{zero}}\, A} \qquad \frac{x : A \quad xs : \mathsf{Vec}_n\, A}{x :: xs : \mathsf{Vec}_{\mathsf{suc}\, n}\, A}$$

Membership in a vector (denoted $\in_V$) is introduced similarly to membership in a list $\in$. We also introduce a function $\mathsf{index}_{\in_V} : \Pi\{n : \mathbb{N}\}\Pi\{xs : \mathsf{Vec}_n\, A\}\Pi\{x : A\}.\, x \in_V xs \to \mathsf{Fin}\, n$, defined analogously to $\mathsf{index}_{\in}$.

Possibly infinite sequences over a given type A, that we simply call *sequences*,[2] are defined coinductively by the rules

$$\frac{}{[] : \mathsf{Seq}\, A} \qquad \frac{x : A \quad xs : \mathsf{Seq}\, A}{x :: xs : \mathsf{Seq}\, A}$$

The type $\mathsf{Seq}\, A$ is the final coalgebra of the functor $F\, X = 1 + A \times X$. Given another F-coalgebra $f : C \to 1 + A \times C$, the coalgebra morphism given by the finality of $\mathsf{Seq}\, A$ is typically called $\mathsf{unfold}\, f$. We give here an explicit definition of it.

$$\begin{aligned}
&\mathsf{unfold} : (C \to 1 + A \times C) \to C \to \mathsf{Seq}\, A \\
&\mathsf{unfold}\ f\ z \ \text{with}\ f\ z \\
&\mathsf{unfold}\ f\ z \quad | \quad \mathsf{inl}\ * \quad\ = [] \\
&\mathsf{unfold}\ f\ z \quad | \quad \mathsf{inr}\ (x, z') = x :: \mathsf{unfold}\ f\ z'
\end{aligned}$$

Extensional equality for sequences is called strong bisimilarity, and it is coinductively defined by the rules

$$\frac{}{[] \sim []} \qquad \frac{xs \sim ys}{x :: xs \sim x :: ys}$$

While it ought to be the case morally, one cannot prove that strongly bisimilar computations are equal in Martin–Löf type theory. Therefore, we postulate an inhabitant for

$$\mathsf{SExt} = \Pi\{A : \mathscr{U}\}\Pi\{xs, ys : \mathsf{Seq}\, A\}.\, xs \sim ys \to xs \equiv ys$$

[1] It is also meaningful to allow detection of duplicates deeper inside the list, but for the purposes of this paper, it is more appropriate to be concerned with duplication of the head.

[2] They are often also called colists.

3 Finiteness

In order to define rational sequences in type theory, we need to first define what it means for a set to be finite. In this section, we consider two notions of finiteness: listability and Noetherianness. More notions of finite set have been proposed and analyzed. We give a brief overview of finiteness in constructive mathematics in Section 8.

3.1 Listability

A set A is called *listable*, if the type

$$\mathsf{Listable}\, A = \Sigma xs : \mathsf{List}\, A.\, \Pi x : A.\, x \in xs$$

is inhabited. Listability says that we can fit all elements of A into a list. The length of the list is an upper bound on the size of the set A. Note that in general, a proof of $\mathsf{Listable}\, A$ does not give the exact size of A since the list may contain duplicates. Nonetheless, the exact size of the set can be inferred: Since propositional equality on listable sets is always decidable, as shown for example in Firsov and Uustalu (2015), we can remove all duplicates from the given list.

Proposition 1
Propositional equality on listable sets is decidable.

Proof
Let A be a listable set, i.e., there exists $xs : \mathsf{List}\, A$ such that $p : \Pi\, x : A.\, x \in xs$. Let $x, y : A$. In order to check if two elements x and y are equal terms of type A, it is sufficient to check if $\mathsf{index}{\in}\, (p\, x)$ and $\mathsf{index}{\in}\, (p\, y)$ are equal terms of type $\mathsf{Fin}\, (\mathsf{length}\, xs)$ (notice that propositional equality on $\mathsf{Fin}\, n$ is decidable, for all $n : \mathbb{N}$). If $\mathsf{index}{\in}\, (p\, x) \equiv \mathsf{index}{\in}\, (p\, y)$, then x and y occupy the same position in xs and therefore they are equal. If $\mathsf{index}{\in}\, (p\, x) \not\equiv \mathsf{index}{\in}\, (p\, y)$, then $x \not\equiv y$. Indeed, if x and y were equal, then also $\mathsf{index}{\in}\, (p\, x)$ would be equal to $\mathsf{index}{\in}\, (p\, y)$, which is not the case. □

3.2 Noetherianness

A set A is said to be *Noetherian*, if the type

$$\mathsf{Noeth}\, A = \mathsf{Noeth}'_{[]}\, A$$

is inhabited, where the auxiliary predicate Noeth' is inductively defined by the rules

$$\frac{d : \mathsf{Dup}\, acc}{\mathsf{dup}\, d : \mathsf{Noeth}'_{acc}\, A} \qquad \frac{n : \Pi x : A.\, \mathsf{Noeth}'_{x :: acc}\, A}{\mathsf{ask}\, n : \mathsf{Noeth}'_{acc}\, A}$$

Noetherianness says that, if we are shown elements from A one after another, sooner or later we will have seen some element twice.[3] From a proof of Noetherianness, we

[3] The standard definition of Noetherianness (Coquand and Spiwack, 2010) uses a version of Dup that accepts duplicates anywhere in the given list and not only duplication of the head. The corresponding version of Noeth is logically equivalent to the one considered here.

cannot generally infer a bound on the size of A. Nevertheless, propositional equality on Noetherian sets is always decidable (Firsov *et al.*, 2016).

Proposition 2
Propositional equality on Noetherian sets is decidable.

Proof
We describe an informal proof of the proposition, a detailed proof can be found in our Agda formalization.

Let A be a Noetherian set and let $x, y : A$. First, we repeatedly feed x into the Noetherianness proof until we reach a leaf. The Noetherianness proof tells us that we fed the same element at the last iteration and at some earlier, say pth, iteration.

Next, we repeat the procedure described above, i.e., we repeatedly feed x into the Noetherianness proof, but at the pth iteration we feed y instead. The Noetherianness proof now tells us that, in the new experiment, we fed it the same element at whichever was now the last iteration and at some earlier, say p'th, iteration.

If $p \equiv p'$, then clearly $x \equiv y$, since the element fed in the p'th (i.e., pth) iteration in the second experiment was y, but the element fed last in the same experiment was x, however these must have been the same element. But if $p \not\equiv p'$, then $x \not\equiv y$, since if $x \equiv y$ were the case, the two experiments would have been identical and ought to have given the same result, i.e., we should have $p \equiv p'$. □

Listability is stronger than Noetherianness.

Proposition 3
Every listable set is Noetherian, i.e., the type

$$\Pi A : \mathscr{U}.\, \mathsf{Listable}\, A \to \mathsf{Noeth}\, A$$

is inhabited.

Proof
Let A be a type. We construct an inhabitant f of the type

$$\Pi xs, acc : \mathsf{List}\, A.(\Pi x : A.\, x \in xs + x \in acc) \to \Pi x : A.\, \mathsf{Noeth}'_{x :: acc}\, A$$

The statement of the Proposition then follows easily. In fact, if we have $xs : \mathsf{List}\, A$ with $r : \Pi x : A.\, x \in xs$, we consider the term $\mathsf{ask}\,(f\, xs\, []\, (\lambda x.\, \mathsf{inl}\,(r\, x))$ of type $\mathsf{Noeth}\, A$.

So fix two lists xs and acc containing together all elements in A, and fix $x : A$. We have the following two cases:

- $x \in acc$, therefore the list $x :: acc$ has a duplicate and we are done.
- $q : x \in xs$, then by inductive hypothesis we have a term $g = f\,(\mathsf{remove}\, xs\, q)\,(x :: acc)\, p$ of type $\Pi y : A.\, \mathsf{Noeth}'_{y :: x :: acc}\, A$, where p is a proof that the lists $\mathsf{remove}\, xs\, q$ and $x :: acc$ together contain all elements of A. Therefore, we are done, since $\mathsf{ask}\, g : \mathsf{Noeth}'_{x :: acc}\, A$.

□

The converse of Proposition 3 is generally not true. In fact, there exists a Noetherian set X such that, from a given proof of listability of X, one is able to

derive the limited principle of omniscience (Coquand and Spiwack, 2010). Another weak counterexample is the following: Let A be a proposition, then such a general A is Noetherian but not listable. If every proposition were listable, then one could derive the law of excluded middle for propositions.

Next, we see how to extend listability and Noetherianness to subsets represented by predicates.

3.3 Finite subsets

Given a set A, a predicate $P : A \to \mathscr{U}$ should intuitively specify a subset of A, i.e., a set such that, for all $x : A$, x is a member if and only if the type $P\,x$ is inhabited. But we do not have a subset type former in type theory. The type $\Sigma x : A.\, P\,x$ comes close, but is not a faithful implementation of such a set. The reason is that the predicate P can be proof-relevant, i.e., for some $x : A$, the type $P\,x$ may fail to be a proposition. This has bad consequences. If for some $x : A$, we have two proofs $p, p' : P\,x$, then this implementation has two "copies" of the element x, namely (x, p) and (x, p'). In connection to finiteness, this has the further effect that, if A is finite (either listable or Noetherian), then $\Sigma x : A.\, P\,x$ is not generally so.[4]

As we do not have a type former for subsets, in the rest of this paper, we work with sets with a predicate as a poor man's substitute for subsets. But for lucidity, we can informally refer to these representations as subsets proper.

If we want to reason about finiteness of subsets on the level of sets with a predicate, we need to fine-tune the definitions introduced above.

Let A be a set and $P : A \to \mathscr{U}$ a predicate on it. We call the subset *listable*, if the type

$$\mathsf{ListableSub}\,P = \Sigma xs : \mathsf{List}\,A.\, \Pi x : A.\, P\,x \to x \in xs$$

is inhabited.

We call the subset *Noetherian*, if the type

$$\mathsf{NoethSub}\,P = \mathsf{NoethSub}'_{[]}\,P$$

is inhabited, where the auxiliary predicate $\mathsf{NoethSub}'$ is inductively defined by the rules

$$\frac{d : \mathsf{Dup}\,acc}{\mathsf{dup}\,d : \mathsf{NoethSub}'_{acc}\,P} \qquad \frac{n : \Pi x : A.\, P\,x \to \mathsf{NoethSub}'_{x::acc}\,P}{\mathsf{ask}\,n : \mathsf{NoethSub}'_{acc}\,P}$$

Notice that the above definitions define finiteness of a subset of A meaningfully, even if the predicate P representing the subset is proof-relevant. In fact, both listability and Noetherianness are closed under subsets in the following sense[5]:

$$\Pi A : \mathscr{U}\, \Pi P : A \to \mathscr{U}.\, \mathsf{Listable}\,A \to \mathsf{ListableSub}\,P$$

$$\Pi A : \mathscr{U}\, \Pi P : A \to \mathscr{U}.\, \mathsf{Noeth}\,A \to \mathsf{NoethSub}\,P$$

[4] With propositional truncation, we could define the subset for P by $\Sigma x : A.\, \|P\,x\|$.

[5] For subsets implemented with propositional truncation, this is not true: Noetherianness is closed under subsets while listability is not.

For subsets, listability is still stronger than Noetherianness. The proof is similar to the one of Proposition 3.

Proposition 4
Every listable subset of a set A is Noetherian, i.e., the following type is inhabited:

$$\Pi A : \mathscr{U}\, \Pi P : A \to \mathscr{U}.\, \mathsf{ListableSub}\, P \to \mathsf{NoethSub}\, P$$

The converse of Proposition 4 is generally not true. However, there exists a non-trivial class of subsets for which the two notions of finiteness are logically equivalent. We call such subsets orbits.

3.4 Finiteness of orbits

Let A be a set. Let $f : A \to 1 + A$ and $x : A$. The function f and the value x generate a subset of A, containing all the elements of A that are obtained from repeated applications of the function f on x. Formally, the *orbit* of x wrt. f is represented by the predicate $\mathsf{Orb}\, f\, x = \lambda y.\, x \Rightarrow_f^* y$, where the relation $\Rightarrow_f^*$ is inductively defined by the rules

$$\frac{}{\mathsf{first} : x \Rightarrow_f^* x} \qquad \frac{p : x \Rightarrow_f^* y \quad q : f\, y \equiv \mathsf{inr}\, z}{\mathsf{next}\, p\, q : x \Rightarrow_f^* z}$$

The relation $\Rightarrow_f^*$ is the reflexive–transitive closure of the relation $\lambda y, z.\, f\, y \equiv \mathsf{inr}\, z$, that we write $\Rightarrow_f$ for short. For the subset $\mathsf{Orb}\, f\, x$ of A, listability and Noetherianness are logically equivalent notions.

Proposition 5
For orbits, Noetherianness implies listability, i.e., the type

$$\Pi A : \mathscr{U}\, \Pi f : A \to 1 + A\, \Pi x : A.\, \mathsf{NoethSub}\, (\mathsf{Orb}\, f\, x) \to \mathsf{ListableSub}\, (\mathsf{Orb}\, f\, x)$$

is inhabited.

Proof
Let A be a set, $f : A \to 1 + A$ and $x : A$. We construct an inhabitant g of the type

$$\Pi\, y : A\, \Pi\, p : x \Rightarrow_f^* y.\, \mathsf{NoethSub}'_{\mathsf{genAcc}\, p}\, (\mathsf{Orb}\, f\, x) \to \mathsf{ListableSub}\, (\mathsf{Orb}\, f\, x)$$

where the list $\mathsf{genAcc}\, p$ is the list of elements of A between x and y in p, i.e., if p is of the form

$$x = x_0 \Rightarrow_f x_1 \Rightarrow_f x_2 \Rightarrow_f \ldots \Rightarrow_f x_k = y$$

with $k \geqslant 0$, then $\mathsf{genAcc}\, p = [x_k, \ldots, x_1, x_0]$.

$$\begin{aligned} &\mathsf{genAcc} : \Pi\{y : A\}.\, x \Rightarrow_f^* y \to \mathsf{List}\, A \\ &\mathsf{genAcc}\ \mathsf{first} \quad\ = x :: [] \\ &\mathsf{genAcc}\ (\mathsf{next}\ p\ q) = y :: \mathsf{genAcc}\ p \end{aligned}$$

After constructing such term g, we can prove the proposition. First, apply g to x and first, obtaining a term of type $\mathsf{NoethSub}'_{x::[]}\, (\mathsf{Orb}\, f\, x) \to \mathsf{ListableSub}\, (\mathsf{Orb}\, f\, x)$, and from this the statement of the proposition follows easily.

Fix $y : A$ such that $p : x \Rightarrow_f^* y$. Suppose $n : \mathsf{NoethSub}'_{\mathsf{genAcc}\, p}(\mathsf{Orb}\, f\, x)$. The proof proceeds by induction on the Noetherianness proof n.

If $n = \mathsf{dup}\, d$, then $\mathsf{genAcc}\, p$ contains a duplicate. In this case, the proof p is of the form

$$x \Rightarrow_f^* y \Rightarrow_f y_1 \Rightarrow_f \ldots \Rightarrow_f y_k = y$$

with $k \geqslant 1$, therefore $\mathsf{genAcc}\, p = [y, \ldots, y_1, y, \ldots, x]$. Notice that every element z such that $y \Rightarrow_f^* z$ is already a member of $\mathsf{genAcc}\, p$. Moreover, every element z such that $x \Rightarrow_f^* z$ is already a member of $\mathsf{genAcc}\, p$. Therefore, the subset $\mathsf{Orb}\, f\, x$ is listable.

Otherwise, we have $n = \mathsf{ask}\, n'$ with $n' : \Pi z : A.\, x \Rightarrow_f^* z \to \mathsf{NoethSub}'_{z :: \mathsf{genAcc}\, p}$ $(\mathsf{Orb}\, f\, x)$. The proof now proceeds by case analysis on $f\, y$. There are following two cases:

- If we have $f\, y \equiv \mathsf{inl}\, *$, then there is no element z such that $y \Rightarrow_f^* z$. Therefore, every element z such that $x \Rightarrow_f^* z$ is already a member of $\mathsf{genAcc}\, p$, so the subset $\mathsf{Orb}\, f\, x$ is listable.
- If we have $q : f\, y \equiv \mathsf{inr}\, z$, then we can conclude by taking $g\, z\, (\mathsf{next}\, p\, q)$ $(n'\, z\, (\mathsf{next}\, p\, q)) : \mathsf{ListableSub}\, (\mathsf{Orb}\, f\, x)$ given by the inductive hypothesis.

□

3.5 An alternative notion of finiteness for orbits

Orbits admit a special notion of finiteness. In fact, intuitively, in order to claim that the subset represented by $\mathsf{Orb}\, f\, x$ is finite, it ought to be sufficient to establish that the iterated application of f on x either terminates or outputs the same element twice. Formally, we introduce a predicate FinOrb as

$$\mathsf{FinOrb} : \Pi\{A : \mathscr{U}\}.\, (A \to 1 + A) \to A \to \mathscr{U}$$
$$\mathsf{FinOrb}\, f\, x = \mathsf{FinOrb}'_{[]}\, f\, x$$

where the auxiliary predicate FinOrb' is inductively defined by the rules

$$\frac{d : x \in acc}{\mathsf{dup}\, d : \mathsf{FinOrb}'_{acc}\, f\, x} \qquad \frac{p : f\, x \equiv \mathsf{inl}\, *}{\mathsf{nil}\, p : \mathsf{FinOrb}'_{acc}\, f\, x} \qquad \frac{p : f\, x \equiv \mathsf{inr}\, y \quad r : \mathsf{FinOrb}'_{x :: acc}\, f\, y}{\mathsf{cons}\, y\, p\, r : \mathsf{FinOrb}'_{acc}\, f\, x}$$

A function f and a value x satisfy the predicate FinOrb if, visiting all the elements generated by f starting from x, eventually we either find an element x' such that $f\, x' \equiv \mathsf{inl}\, *$ or we encounter the same element twice. The inhabitedness of the type $\mathsf{FinOrb}\, f\, x$ expresses the finiteness of the orbit, since it is equivalent to $\mathsf{NoethSub}\, (\mathsf{Orb}\, f\, x)$ (and then also $\mathsf{ListableSub}\, (\mathsf{Orb}\, f\, x)$ by Proposition 5).

Proposition 6

The types $\mathsf{FinOrb}\, f\, x$ and $\mathsf{NoethSub}\, (\mathsf{Orb}\, f\, x)$ are logically equivalent, for all $f : A \to 1 + A$ and $x : A$.

Proof

($\Leftarrow$) Let $f : A \to 1 + A$ and $x : A$. We construct an inhabitant g of the type

$$\Pi\, acc : \mathsf{List}\, A\, \Pi\{y : A\}\Pi\, y' : A.\, x \Rightarrow_f^* y \to y \Rightarrow_f y' \to \mathsf{NoethSub}'_{y::acc}\,(\mathsf{Orb}\, f\, x) \to \mathsf{FinOrb}'_{acc}\, f\, y$$

The statement of the Proposition then follows easily as a corollary.
So fix two elements y and y' such that $p : x \Rightarrow_f^* y$ and $p' : y \Rightarrow_f y'$, and let $acc : \mathsf{List}\, A$. Suppose we have a proof $n : \mathsf{NoethSub}'_{y::acc}\,(\mathsf{Orb}\, f\, x)$. The proof proceeds by induction on n.

- If $n = \mathsf{dup}\, d$, i.e., there is a duplicate in $y :: acc$, i.e., $y \in acc$. Therefore, we are done.
- If $n = \mathsf{ask}\, n'$, with $n' : \Pi z : A.\, x \Rightarrow_f^* z \to \mathsf{NoethSub}'_{z::y::acc}\,(\mathsf{Orb}\, f\, x)$, we proceed by case analysis on $f\, y'$.
  - — If $q : f\, y' \equiv \mathsf{inl}\, *$, then we conclude by taking $\mathsf{cons}\, y'\, p'\, (\mathsf{nil}\, q)$.
  - — If $q : f\, y' \equiv \mathsf{inr}\, z$, then by inductive hypothesis, we have a term $h = g\,(y :: acc)\, z\, (\mathsf{next}\, p\, p')\, q\, (n'\, y'\, (\mathsf{next}\, p\, p')) : \mathsf{FinOrb}'_{y::acc}\, f\, y'$. Then we conclude by taking $\mathsf{cons}\, y'\, p'\, h$.

($\Rightarrow$) We prove $\mathsf{FinOrb}\, f\, x \to \mathsf{ListableSub}\,(\mathsf{Orb}\, f\, x)$. We then conclude using Proposition 4. Fix $f : A \to A + 1$. We first construct a list that collects all the elements of A visited in a proof of $\mathsf{FinOrb}\, f\, x$.

$$\begin{aligned}&\mathsf{orbList} : \Pi\, x : A.\, \mathsf{FinOrb}\, f\, x \to \mathsf{List}\, A\\ &\mathsf{orbList}\, x\, r = \mathsf{orbList}'\, x\, []\, r\end{aligned}$$

where the auxiliary function $\mathsf{orbList}'$ is defined as follows:

$$\begin{aligned}&\mathsf{orbList}' : \Pi\, x : A \Pi\, acc : \mathsf{List}\, A.\, \mathsf{FinOrb}'_{acc}\, f\, x \to \mathsf{List}\, A\\ &\mathsf{orbList}'\, x\, acc\, (\mathsf{dup}\, d) \quad = x :: acc\\ &\mathsf{orbList}'\, x\, acc\, (\mathsf{nil}\, p) \quad = x :: []\\ &\mathsf{orbList}'\, x\, acc\, (\mathsf{cons}\, y\, p\, r) = x :: \mathsf{orbList}'\, y\, (x :: acc)\, r\end{aligned}$$

Let $x, y : X$ such that $x \Rightarrow_f^* y$. In order to conclude, it is sufficient to construct an inhabitant g of the type

$$\Pi\{w : A\}\Pi\, p : w \Rightarrow_f^* x\, \Pi\, r : \mathsf{FinOrb}'_{\mathsf{genAcc2}\, p}\, f\, x.\, y \in \mathsf{orbList}'\, x\, (\mathsf{genAcc2}\, p)\, r$$

where $\mathsf{genAcc2}$ is the following variation of genAcc introduced in the proof of Proposition 5:

$$\begin{aligned}&\mathsf{genAcc2} : \Pi\{y : A\}.\, x \Rightarrow_f^* y \to \mathsf{List}\, A\\ &\mathsf{genAcc2}\ \mathsf{first} \quad = []\\ &\mathsf{genAcc2}\ (\mathsf{next}\ p\ q) = \mathsf{genAcc}\ p\end{aligned}$$

The proof proceeds by induction on r. We describe the proof for the case $r = \mathsf{dup}\, d$, for $d : x \in acc$. In this case, we have that the list $\mathsf{orbList}'\, x\, acc\, (\mathsf{dup}\, d) = x :: acc = \mathsf{genAcc}\, p$ has a duplicate. Therefore, as already discussed in the proof of Proposition 5, we know that $y \in \mathsf{orbList}'\, x\, acc\, (\mathsf{dup}\, d)$.

The proof of Proposition 6 is completed: Given $r : \mathsf{FinOrb}\, f\, x$, we conclude by taking $g\, \mathsf{first}\, r : y \in \mathsf{orbList}\, x\, r$.

□

In the next section, we introduce rational sequences over a type A as sequences with finitely many suffixes where "finite" could mean either Noetherian or listable. The suffix predicate turns out to be an orbit. Therefore, Propositions 5 and 6 both apply to it, which means that the two notions of finiteness become equivalent.

4 Rational sequences

The definition of rational sequence is based on the notion of *suffix*. Formally, we introduce a binary relation $\trianglelefteq$ between sequences that is satisfied whenever the first sequence is a suffix of the second. The relation $\trianglelefteq$ is inductively defined by the rules

$$\frac{}{\mathsf{here} : xs \trianglelefteq xs} \qquad \frac{p : ys \trianglelefteq xs}{\mathsf{there}\, p : ys \trianglelefteq x :: xs}$$

The set of all suffixes of xs is represented by the predicate $\mathsf{Suffix}\, xs = \lambda ys.\, ys \trianglelefteq xs$.

A sequence is *rational*, if it has a finite number of distinct suffixes. In Section 3, we have introduced two generally inequivalent notions of finiteness, listability and Noetherianness. Therefore, we can write down two different proof-relevant definitions of rational sequences:

$$\mathsf{SeqRL}\, A = \Sigma xs : \mathsf{Seq}\, A.\, \mathsf{ListableSub}\, (\mathsf{Suffix}\, xs)$$

$$\mathsf{SeqRN}\, A = \Sigma xs : \mathsf{Seq}\, A.\, \mathsf{NoethSub}\, (\mathsf{Suffix}\, xs)$$

Interestingly, the types $\mathsf{SeqRL}\, A$ and $\mathsf{SeqRN}\, A$ are isomorphic up to the first projection (i.e., ignoring proofs of finiteness)—this is how we can express that the intended subsets of $\mathsf{Seq}\, A$ are isomorphic. Therefore, it does not matter which notion of finiteness we pick in the definition of rational sequence.

Proposition 7

Let xs be a sequence over the type A. The types $\mathsf{ListableSub}\, (\mathsf{Suffix}\, xs)$ and $\mathsf{NoethSub}\, (\mathsf{Suffix}\, xs)$ are logically equivalent.

Proof

By Proposition 4, we already have the left-to-right direction. For the other direction, we show that the subset given by $\mathsf{Suffix}\, xs$ is an orbit and conclude using Proposition 5. We define a function tail that returns the tail of a sequence whenever the sequence is non-empty.

$$\begin{aligned} &\mathsf{tail} : \mathsf{Seq}\, A \to 1 + \mathsf{Seq}\, A \\ &\mathsf{tail}\, [] \quad\;\; = \mathsf{inl}\, * \\ &\mathsf{tail}\, (x :: xs) = \mathsf{inr}\, xs \end{aligned}$$

It is easy to see that the type $ys \trianglelefteq xs$ is isomorphic to the type $xs \Rightarrow^{*}_{\mathsf{tail}} ys$, which proves that the subset $\mathsf{Suffix}\, xs$ is the orbit of xs wrt. tail. □

Since $\mathsf{Suffix}\, xs$ is an orbit, rational sequences admit an additional encoding:

$$\mathsf{SeqRO}\, A = \Sigma xs : \mathsf{Seq}\, A.\, \mathsf{FinOrb}\, \mathsf{tail}\, xs$$

By Proposition 6, the types $\mathsf{SeqRO}\, A$,$\mathsf{SeqRN}\, A$ and $\mathsf{SeqRL}\, A$ are all isomorphic up to the first projection.

We present a variation of the predicate FinOrb specific for rational sequences that will become handy in Section 6. First, we construct the function $\mathsf{prefixes}$ that, given a vector $ys = [y_1, y_2, \ldots, y_n]$ and a sequence xs (both over a certain type A), returns the following vector of supersequences of xs:

$$[y_1 :: xs, y_2 :: y_1 :: xs, \ldots, y_n :: \ldots :: y_2 :: y_1 :: xs]$$

$$\begin{aligned}
&\mathsf{prefixes} : \Pi\{n : \mathbb{N}\}.\, \mathsf{Vec}_n\, A \to \mathsf{Seq}\, A \to \mathsf{Vec}_n\, (\mathsf{Seq}\, A)\\
&\mathsf{prefixes}\ []\quad\ \ xs = []\\
&\mathsf{prefixes}\ (y :: ys)\ xs = (y :: xs) :: \mathsf{prefixes}\ ys\ (y :: xs)
\end{aligned}$$

The rationality predicate $\mathsf{FinOrbV}$ is given as

$$\mathsf{FinOrbV}\, xs = \mathsf{FinOrbV}'_{[]}\, xs$$

where the auxiliary predicate $\mathsf{FinOrbV}'$ is inductively defined by the rules

$$\frac{d : xs \in_V \mathsf{prefixes}\, acc\, xs}{\mathsf{dup}\, d : \mathsf{FinOrbV}'_{acc}\, xs} \qquad \frac{}{\mathsf{nil} : \mathsf{FinOrbV}'_{acc}\, []} \qquad \frac{r : \mathsf{FinOrbV}'_{x :: acc}\, xs}{\mathsf{cons}\, r : \mathsf{FinOrbV}'_{acc}\, (x :: xs)}$$

The predicate $\mathsf{FinOrbV}'$ takes as argument an accumulator $acc : \mathsf{Vec}_n\, A$, while FinOrb' would take an accumulator $acc : \mathsf{List}\, (\mathsf{Seq}\, A)$ when applied to tail. In fact, when checking whether a sequence is rational, we only need to store the heads of the suffixes we visit. In order to prove that we have seen the same suffix twice, we reconstruct the vector of sequences we visited using the function $\mathsf{prefixes}$ and show that it contains a duplicate. The type of the accumulator is $\mathsf{Vec}_n\, A$ instead of $\mathsf{List}\, A$ because in Section 6 we work with vectors.

Another possible encoding of rational sequences is the following:

$$\mathsf{SeqROV}\, A = \Sigma xs : \mathsf{Seq}\, A.\, \mathsf{FinOrbV}\, xs$$

Proposition 8
Let xs be a sequence over the type A. The types $\mathsf{FinOrb}\, \mathsf{tail}\, xs$ and $\mathsf{FinOrbV}\, xs$ are logically equivalent. As a consequence, the types $\mathsf{SeqRO}\, A$ and $\mathsf{SeqROV}\, A$ are isomorphic.

5 Corecursion

In this section, we describe a corecursion principle for rational sequences. Let A and X be types. The corecursion principle can be formulated as follows: if X is finite, then every sequence of the form $\mathsf{unfold}\, f\, x$ is rational, for any $f : X \to 1 + A \times X$ and $x : X$. In Section 4, we have seen that it does not matter which notion of rationality we choose, they are all equivalent. But what do we mean by the statement "X is finite"? It turns out that it is enough to require X to be a Noetherian set in order to prove the corecursion principle.

Proposition 9
There is an inhabitant (which we will denote $\mathsf{unfoldR}$) of the following type, describing the corecursion principle for rational sequences:

$$\Pi\{A, X : \mathscr{U}\}.\, \mathsf{Noeth}\, X \to (X \to 1 + A \times X) \to X \to \mathsf{SeqRN}\, A$$

Proof
Fix $f : X \to 1 + A \times X$. We define a function $\mathsf{elemUnfold}$ as follows:

$$\begin{array}{lll}
\mathsf{elemUnfold} : \Pi x : X\, \Pi\{xs : \mathsf{Seq}\, A\}.\, xs \trianglelefteq \mathsf{unfold}\, f\, x \to X & & \\
\mathsf{elemUnfold}\ x\ p & \mathsf{with}\ f\ x & \\
\mathsf{elemUnfold}\ x\ p & \mid\ \mathsf{inl}\ * & = x \\
\mathsf{elemUnfold}\ x\ \mathsf{here} & \mid\ \mathsf{inr}\ (a, x') & = x \\
\mathsf{elemUnfold}\ x\ (\mathsf{there}\ p) & \mid\ \mathsf{inr}\ (a, x') & = \mathsf{elemUnfold}\ x'\ p
\end{array}$$

Given a value $x : X$ and a suffix xs of $\mathsf{unfold}\, f\, x$, the function $\mathsf{elemUnfold}$ constructs an element $y : X$ such that $\mathsf{unfold}\, f\, y \equiv xs$, i.e., $\mathsf{elemUnfold}\, x\, p$ is the initial value that by corecursion generates the sequence xs.

Now fix $x : X$. We construct an inhabitant g of the following type:

$$\Pi\, acc : \mathsf{List}\, X.\, \mathsf{Noeth}'_{acc}\, X \to \mathsf{NoethSub}'_{\mathsf{map}\,(\mathsf{unfold}\, f)\, acc}\, (\mathsf{Suffix}\,(\mathsf{unfold}\, f\, x))$$

Having such g, we can define corecursion as $\mathsf{unfoldR}\, n\, f\, x = (\mathsf{unfold}\, f\, x, g\, []\, n)$. So let $acc : \mathsf{List}\, X$ be an accumulator and $n : \mathsf{Noeth}'_{acc}\, X$. The proof proceeds by induction on n.

- If $n = \mathsf{dup}\, d$, then acc contains a duplicate. Also, $\mathsf{map}\,(\mathsf{unfold}\, f)\, acc$ contains a duplicate, therefore we are done.
- Otherwise, we have $n = \mathsf{ask}\, n'$ with $n' : \Pi y : X.\mathsf{Noeth}'_{y :: acc}\, X$. It is sufficient to construct an inhabitant of $\mathsf{NoethSub}'_{xs :: \mathsf{map}\,(\mathsf{unfold}\, f)\, acc}\, (\mathsf{Suffix}\,(\mathsf{unfold}\, f\, x))$, for all $xs : \mathsf{Seq}\, A$ such that $p : xs \trianglelefteq \mathsf{unfold}\, f\, x$, and then conclude using the constructor ask on this construction. Let $y = \mathsf{elemUnfold}\, x\, p$, by inductive hypothesis, we have a term

  $$g\,(y :: acc)(n'\, y) : \mathsf{NoethSub}'_{\mathsf{unfold}\, f\, y :: \mathsf{map}\,(\mathsf{unfold}\, f)\, acc}\, (\mathsf{Suffix}\,(\mathsf{unfold}\, f\, x)).$$

  As we observed after the definition of the function $\mathsf{elemUnfold}$, we have $\mathsf{unfold}\, f\, y \equiv xs$ and therefore we are done.

□

We will see an application of the corecursion principle in Section 7, where we will build the decimal representation of a rational number starting from its fractional representation.

6 Lists with backpointers

We now move to an inductive implementation of rational sequences. One can think of this as a syntactic representation of rational sequences (the one a programmer would actually like to work with), as opposed to the more semantic representation

given in Section 4. In this section, we introduce the type of *lists with backpointers*, i.e., lists with possibly an index at the very end, pointing to the position in the list where the period starts. Formally, we introduce a type $\mathsf{ListP}\,A$ of lists with backpointers over A as follows:

$$\mathsf{ListP}\,A = \mathsf{ListP}'_{\mathsf{zero}}\,A$$

where the auxiliary predicate ListP' is inductively defined by the rules

$$\frac{i : \mathsf{Fin}\,n}{\mathsf{ptr}\,i : \mathsf{ListP}'_n\,A} \qquad \frac{}{[] : \mathsf{ListP}'_n\,A} \qquad \frac{x : A \quad xs : \mathsf{ListP}'_{\mathsf{suc}\,n}\,A}{x :: xs : \mathsf{ListP}'_n\,A}$$

As a convention, the argument $i : \mathsf{Fin}\,n$ of the constructor ptr points to the position in the list where the period starts, counting from the end of the list. For example, the term $x_0 :: x_1 :: x_2 :: x_3 :: \mathsf{ptr}\,2$ represent the sequence $[x_0, x_1, x_2, x_3, x_1, x_2, x_3, \ldots]$. So the index 2 refers to the element at distance 2 from the end of the list, in this case x_1 (the last element of the list having distance 0).

A term $xs : \mathsf{ListP}\,A$ can be unwound into a possibly infinite sequence over A.

$$\begin{aligned}&\mathsf{listp2seq} : \mathsf{ListP}\,A \to \mathsf{Seq}\,A\\ &\mathsf{listp2seq} = \mathsf{listp2seq}'\,[]\end{aligned}$$

where the auxiliary function $\mathsf{listp2seq}'$ is defined (by mutual corecursion with the function $\mathsf{listp2seqRewind}$) as follows:

$$\begin{aligned}&\mathsf{listp2seq}' : \Pi\{n : \mathbb{N}\}.\mathsf{Vec}_n\,A \to \mathsf{ListP}'_n\,A \to \mathsf{Seq}\,A\\ &\mathsf{listp2seq}'\,acc\,(\mathsf{ptr}\,i) \;\; = \mathsf{listp2seqRewind}\,acc\,(\mathsf{ptr}\,i)\,i\\ &\mathsf{listp2seq}'\,acc\,[] \qquad\;\; = []\\ &\mathsf{listp2seq}'\,acc\,(x :: xs) = x :: \mathsf{listp2seq}'\,(x :: acc)\,xs\end{aligned}$$

$$\begin{aligned}&\mathsf{listp2seqRewind} : \Pi\{n : \mathbb{N}\}.\mathsf{Vec}_n\,A \to \mathsf{ListP}'_n\,A \to \mathsf{Fin}\,n \to \mathsf{Seq}\,A\\ &\mathsf{listp2seqRewind}\,(x :: acc)\,xs\,\mathsf{zero} \;\; = x :: \mathsf{listp2seq}'\,(x :: acc)\,xs\\ &\mathsf{listp2seqRewind}\,(x :: acc)\,xs\,(\mathsf{suc}\,i) = \mathsf{listp2seqRewind}\,acc\,(x :: xs)\,i\end{aligned}$$

The function $\mathsf{listp2seq}'$ takes as arguments a vector acc of length n and an element $p : \mathsf{ListP}'_n\,A$. One has to think at the pair (acc, p) as a location inside an element of $\mathsf{ListP}\,A$, i.e., a zipper in the sense of Huet (1997). Therefore, the vector acc corresponds to a context containing the elements that we have already visited, while p is still to be explored. The function $\mathsf{listp2seq}'$ calls $\mathsf{listp2seqRewind}$ in the case $p = \mathsf{ptr}\,i$. The function $\mathsf{listp2seqRewind}$ rewinds the zipper back i steps, and then calls $\mathsf{listp2seq}'$. For example, the function $\mathsf{listp2seq}$ proceeds as follows when applied to the input $x_0 :: x_1 :: x_2 :: x_3 :: \mathsf{ptr}\,2$:

$$\begin{aligned}\mathsf{listp2seq}\,(x_0 :: x_1 :: x_2 :: x_3 :: \mathsf{ptr}\,2) &= \mathsf{listp2seq}'\,[]\,(x_0 :: x_1 :: x_2 :: x_3 :: \mathsf{ptr}\,2)\\ &= x_0 :: x_1 :: x_2 :: x_3 :: (\mathsf{listp2seqRewind}\,(x_3 :: x_2 :: x_1 :: x_0 :: [])\,(\mathsf{ptr}\,2)\,2)\\ &= x_0 :: x_1 :: x_2 :: x_3 :: x_1 :: (\mathsf{listp2seq}'\,(x_0 :: [])\,(x_2 :: x_3 :: \mathsf{ptr}\,2))\\ &= x_0 :: x_1 :: x_2 :: x_3 :: x_1 :: x_2 :: x_3 ::\\ &\quad (\mathsf{listp2seqRewind}\,(x_3 :: x_2 :: x_1 :: x_0 :: [])(\mathsf{ptr}\,2)\,2)\\ &= \ldots\end{aligned}$$

A sequence constructed using listp2seq can be proved rational, in the sense that it satisfies one of the rationality predicates introduced in Section 4. Here, we use the predicate FinOrbV since the accumulator in listp2seq′ is a vector and not a list. The proof of rationality is constructed as follows:

$$\begin{aligned}&\mathsf{listp2rat} : \Pi xs : \mathsf{ListP}\, A.\, \mathsf{FinOrbV}\,(\mathsf{listp2seq}\, xs)\\&\mathsf{listp2rat} = \mathsf{listp2rat}'\ []\end{aligned}$$

where the auxiliary function listp2rat′ does all the work:

$$\begin{aligned}&\mathsf{listp2rat}' : \Pi\{n : \mathbb{N}\}\Pi\, acc : \mathsf{Vec}_n\, A\, \Pi xs : \mathsf{ListP}'_n\, A.\, \mathsf{FinOrbV}'_{acc}(\mathsf{listp2seq}'\, acc\, xs)\\&\mathsf{listp2rat}'\ acc\ (\mathsf{ptr}\ i) \quad = \mathsf{dup}\ r\\&\mathsf{listp2rat}'\ acc\ [] \qquad = \mathsf{nil}\\&\mathsf{listp2rat}'\ acc\ (x :: xs) = \mathsf{cons}\ (\mathsf{listp2rat}'\ (x :: acc)\ xs)\end{aligned}$$

The term r above is a consequence of the fact that the sequence listp2seqRewind acc $xs\, i$ is a member of the vector prefixes acc (listp2seq′ $acc\, xs$), a statement easily provable by induction on the index i and the accumulator acc.

From a rational sequence, it is also possible to construct a list with backpointers. Again, we choose FinOrbV as the rationality predicate.

$$\begin{aligned}&\mathsf{seqrat2listp} : \Pi xs : \mathsf{Seq}\, A.\, \mathsf{FinOrbV}\, xs \to \mathsf{ListP}\, A\\&\mathsf{seqrat2listp} = \mathsf{seqrat2listp}'\ []\end{aligned}$$

where the auxiliary function seqrat2listp′ is defined as follows:

$$\begin{aligned}&\mathsf{seqrat2listp}' : \Pi\{n : \mathbb{N}\}\Pi acc : \mathsf{Vec}_n\, A\, \Pi\ xs : \mathsf{Seq}\, A.\, \mathsf{FinOrbV}'_{acc}\, xs \to \mathsf{ListP}'_n\, A\\&\mathsf{seqrat2listp}'\ acc\ xs \qquad (\mathsf{dup}\ r) \ = \mathsf{ptr}\ (\mathsf{index}{\in_V}\ r)\\&\mathsf{seqrat2listp}'\ acc\ [] \qquad \mathsf{nil} \qquad = []\\&\mathsf{seqrat2listp}'\ acc\ (x :: xs)\ (\mathsf{cons}\ r) = x :: \mathsf{seqrat2listp}'\ (x :: acc)\ xs\ r\end{aligned}$$

In the case where the rationality proof is dup r, the function seqrat2listp′ returns a list with backpointers ptr (index$\in_V$ r). (Remember that the list version of index$\in_V$ was introduced in Section 2.) The proof r tells us that the current sequence xs is a member of the vector prefixes $acc\, xs$, and index$\in_V$ r tells us the position of xs in the vector. This position corresponds exactly to the period of the sequence.

Not only can we move between the inductive and the coinductive implementation of rational sequences, but the two implementations (ListP A and SeqRO A) are isomorphic, in the sense that the following types are inhabited:

$$\begin{aligned}&\mathsf{ListPIso}_1 : \Pi\{A : \mathscr{U}\}\Pi\, xs : \mathsf{Seq}\, A\, \Pi\, r : \mathsf{FinOrbV}\, xs.\\&\qquad (xs, r) \equiv (\mathsf{listp2seq}\,(\mathsf{seqrat2listp}\, xs\, r), \mathsf{listp2rat}(\mathsf{seqrat2listp}\, xs\, r))\\&\mathsf{ListPIso}_2 : \Pi\{A : \mathscr{U}\}\Pi\, xs : \mathsf{ListP}\, A.\, xs \equiv \mathsf{seqrat2listp}\,(\mathsf{listp2seq}\, xs)\,(\mathsf{listp2rat}\, xs)\end{aligned}$$

We do not present here the proofs of $\mathsf{ListPIso}_1$ and $\mathsf{ListPIso}_2$. They can be found in our Agda formalization.

7 Rational numbers

We now proceed to an example. We show how to convert the fractional representation of a rational number into a decimal expansion and vice versa. We restrict our study to positive rational numbers between 0 and 1.

The conversion between fractions and decimal expansion has type:

$$\mathsf{decimal} : \Pi\, d : \mathbb{N}.\, \mathsf{Fin}\, d \to \mathsf{SeqRN}\,(\mathsf{Fin}\, 10)$$

Here, we represent fractions using the type $\Pi\, d : \mathbb{N}.\, \mathsf{Fin}\, d$. An element (d, n) corresponds to the fraction n/d, and by construction n is strictly less than d. Notice that this representation forbids division by zero, since the type $\mathsf{Fin}\,\mathsf{zero}$ is empty. The result of $\mathsf{decimal}\, d\, n$ is the decimal expansion of n/d together with a proof that the expansion is rational (as a sequence). Such expansion is necessarily of the form $0, x_1 x_2 x_3 \ldots$ and $\mathsf{decimal}\, d\, n$ is the sequence $x_1 :: x_2 :: x_3 :: \ldots$.

We consider two functions, div and mod, calculating the quotient and remainder after integer division of two natural numbers, respectively. Both functions are already available in Agda's standard library. We construct decimal using the corecursion principle of rational sequences (described in Section 5). A fundamental ingredient for doing so is to have a Noetherian state space. Given a denominator $d : \mathbb{N}$, we take $\mathsf{Fin}\, d$ as the state space. The set $\mathsf{Fin}\, d$ is listable, and therefore Noetherian by Proposition 3. We define a function divmod that, given a number d and n with $n < d$, returns the pair (q, r) with $10 * n = q * d + r$ whenever $n \neq 0$.

$$\begin{aligned}
&\mathsf{divmod} : \Pi\, d : \mathbb{N}.\, \mathsf{Fin}\, d \to 1 + \mathsf{Fin}\, 10 \times \mathsf{Fin}\, d\\
&\mathsf{divmod}\ \mathsf{zero}\quad\ \ () \\
&\mathsf{divmod}\ (\mathsf{suc}\ d)\ \mathsf{zero}\quad\ \ = \mathsf{inl}\ *\\
&\mathsf{divmod}\ (\mathsf{suc}\ d)\ (\mathsf{suc}\ n) = \mathsf{inr}\ (q', r)\\
&\quad \mathsf{where}\\
&\qquad q = (10 * \mathsf{toℕ}\ (\mathsf{suc}\ n))\ \mathsf{div}\ \mathsf{suc}\ d\\
&\qquad q' = \mathsf{fromℕ≤}\ (\mathsf{divfin}{<}\ d\ 9\ (\mathsf{suc}\ n))\\
&\qquad r = (10 * \mathsf{toℕ}\ (\mathsf{suc}\ n))\ \mathsf{mod}\ \mathsf{suc}\ d
\end{aligned}$$

The infix operations div and mod take natural numbers as arguments, therefore we convert the index $\mathsf{suc}\, n$ using the function $\mathsf{toℕ} : \Pi\, \{n : \mathbb{N}\}.\, \mathsf{Fin}\, n \to \mathbb{N}$. Moreover, notice that the quotient q is a natural number, while we have to return an inhabitant of $\mathsf{Fin}\, 10$. But we know that q is strictly smaller than 10. Formally, we construct a term $\mathsf{divfin}{<}$ of type:

$$\Pi\, d, m : \mathbb{N}\ \Pi\, n : \mathsf{Fin}\,(\mathsf{suc}\, d).\, (\mathsf{suc}\, m * \mathsf{toℕ}\, n)\ \mathsf{div}\, (\mathsf{suc}\, d) < \mathsf{suc}\, m$$

Therefore, using the map $\mathsf{fromℕ≤} : \Pi\, \{m, n : \mathbb{N}\} \to m < n \to \mathsf{Fin}\, n$ we are done. We can now apply the corecursion principle.

$$\begin{aligned}
&\mathsf{decimal} : \Pi\, d : \mathbb{N}.\, \mathsf{Fin}\, d \to \mathsf{SeqRN}\,(\mathsf{Fin}\, 10)\\
&\mathsf{decimal}\ d\ n = \mathsf{unfoldR}\ \mathsf{NoethFin}\ (\mathsf{divmod}\ d)\ n
\end{aligned}$$

The term NoethFin is the proof of Noetherianness of $\mathsf{Fin}\, d$. We give an explicit description of the behavior of the function decimal. We divide $10 * n$ by d. This produces a quotient q_0 (the first digit of the decimal expansion) and a remainder r_0.

If $r_0 = 0$, we halt, otherwise we divide $10 * r_0$ by d. This produces a quotient q_1 (the second digit of the decimal expansion) and a remainder r_1. If $r_1 = 0$, we halt, otherwise we continue dividing $10 * r_1$ by d. Proceeding in this way, we obtain the decimal expansion of the fraction n/d, together with a proof of rationality.

We now move to the conversion of rational decimal expansions into fractions. The construction is performed using the recursion principle of ListP. Formally, the conversion is defined as follows:

$$\begin{aligned}
&\mathsf{fraction} : \Pi\{k : \mathbb{N}\}.\, \mathsf{ListP}'_k\,(\mathsf{Fin}\,10) \to \mathbb{N} \times \mathbb{N}\\
&\mathsf{fraction}\ (\mathsf{ptr}\ i) && = 1\ ,\ 0\\
&\mathsf{fraction}\ [] && = 1\ ,\ 0\\
&\mathsf{fraction}\ (x :: xs)\ \mathsf{with}\ \mathsf{startPeriod}\ xs \stackrel{?}{=} 1 \mid \mathsf{fraction}\ xs\\
&\mathsf{fraction}\ (x :: xs) \quad | \quad \mathsf{yes}\ p \mid d\ ,\ n = 10 * d - 1\ ,\ \mathsf{toℕ}\ x * d + n\\
&\mathsf{fraction}\ (x :: xs) \quad | \quad \mathsf{no}\ p \mid d\ ,\ n = 10 * d\ ,\ \mathsf{toℕ}\ x * d + n
\end{aligned}$$

The term $\stackrel{?}{=}$ is a decider for propositional equality on natural numbers. The map startPeriod returns a natural number that tells us whether in a list with backpointers we are visiting an element of the period or not. In particular, when startPeriod xs changes from 0 to 1, it means that we are entering the period.

$$\begin{aligned}
&\mathsf{startPeriod} : \Pi\{k : \mathbb{N}\}.\, \mathsf{ListP}'_k\,(\mathsf{Fin}\,10) \to \mathbb{N}\\
&\mathsf{startPeriod}\ (\mathsf{ptr}\ i) && = \mathsf{suc}\ (\mathsf{toℕ}\ i)\\
&\mathsf{startPeriod}\ [] && = \mathsf{zero}\\
&\mathsf{startPeriod}\ (x :: xs)\ \mathsf{with}\ \mathsf{startPeriod}\ xs\\
&\mathsf{startPeriod}\ (x :: xs) \quad | \quad \mathsf{zero} && = \mathsf{zero}\\
&\mathsf{startPeriod}\ (x :: xs) \quad | \quad \mathsf{suc}\ n && = n
\end{aligned}$$

In the definition of fraction, when we have a list with backpointers $x :: xs$, we first check whether we are entering the period by checking if startPeriod xs is equal to 1. If this is the case, we multiply the denominator by 10 and we subtract 1, otherwise we just multiply by 10. We subtract 1 when we enter the period, because for a periodic number $z = 0.(x_1 \ldots x_k)$ the following equation holds:

$$(10^k - 1) * z = 10^{k-1} * x_1 + \cdots + 10 * x_{k-1} + x_k$$

Remark 1

What about 0.5 and 0.4(9)? They both refer to the same rational number $1/2$, but using our encoding (either rational sequences or lists with backpointers) they are distinct terms. Notice that the type $\mathbb{Q}$ of rational numbers can be defined as $\mathbb{N} \times \mathbb{N}$ quotiented by the equivalence relation: $(d_1, n_1) \approx (d_2, n_2)$ iff $n_1 * d_2 \equiv n_2 * d_1$. If we want terms like 0.5 and 0.4(9) to be equal, we have to quotient ListP (Fin 10) by the equivalence relation: $xs \approx_{\mathsf{ListP}} ys$ iff fraction $xs \approx$ fraction ys. Similarly, we would have to quotient the type SeqRO (Fin 10) (and analogously SeqRN (Fin 10) and SeqRL (Fin 10)) by the equivalence relation: $xs \approx_{\mathsf{SeqRO}} ys$ iff seqrat2frac $xs \approx$ seqrat2frac ys, where seqrat2frac is the following composite map:

$$\mathsf{seqrat2frac} :\ \mathsf{SeqRO}\,(\mathsf{Fin}\,10) \xrightarrow{\mathsf{seqrat2listp}} \mathsf{ListP}\,(\mathsf{Fin}\,10) \xrightarrow{\mathsf{fraction}} \mathbb{N} \times \mathbb{N}$$

Remark 2

Notice that the function decimal (more precisely its curried form) has input type $\Sigma\, d : \mathbb{N}.\, \mathsf{Fin}\, d$, while the return type of fraction is $\mathbb{N} \times \mathbb{N}$. The codomain of fraction cannot be tightened to $\Sigma\, d : \mathbb{N}.\, \mathsf{Fin}\, d$, since e.g. $\mathsf{fraction}\,(0.99(999)) = (99900, 99900)$. In general, all elements $xs : \mathsf{ListP}\,(\mathsf{Fin}\,10)$ that represent 1, i.e., periodic xs made up of 9s only, are problematic in this sense. We can tighten the codomain by restricting the application of fraction to lists with backpointers with period not consisting of 9s only. Formally, we can define a predicate Period9s on $\mathsf{ListP}'_k\,(\mathsf{Fin}\,10)$, for all $k : \mathbb{N}$, as follows:

$$\frac{}{\mathsf{Period9s}\,(\mathsf{ptr}\, i)} \qquad \frac{\mathsf{Period9s}\, xs \quad (\mathsf{startPeriod}\, xs > 0 \to x \equiv 9)}{\mathsf{Period9s}\,(x :: xs)}$$

Then we can define $\mathsf{ListP}_{\neg 9s}\,(\mathsf{Fin}\,10) = \Sigma\, xs : \mathsf{ListP}\,(\mathsf{Fin}\,10).\, (\mathsf{Period9s}\, xs \to \bot)$ and we can construct a map $\mathsf{fraction}_{\neg 9s} : \mathsf{ListP}_{\neg 9s}\,(\mathsf{Fin}\,10) \to \Sigma\, d : \mathbb{N}.\, \mathsf{Fin}\, d$.

Notice that if we generally work with the type $\mathsf{ListP}_{\neg 9s}\,(\mathsf{Fin}\,10)$ instead of $\mathsf{ListP}\,(\mathsf{Fin}\,10)$, we can avoid the problem discussed in Remark 1. For example, 0.4(9) is not a representative of $1/2$ anymore. Equality on $\mathsf{ListP}_{\neg 9s}\,(\mathsf{Fin}\,10)$ can be defined as $(xs, p) \approx_{\neg 9s} (ys, q)$ iff $\mathsf{listp2seq}\, xs \sim \mathsf{listp2seq}\, ys$. Moreover, every rational number x in $[0, 1)$ has a canonical representative $xs : \mathsf{ListP}_{\neg 9s}\,(\mathsf{Fin}\,10)$: among all the elements ys such that $\mathsf{fraction}_{\neg 9s}\, ys \equiv x$, xs has the shortest length.

8 Related work

Finiteness in constructive mathematics has been investigated by various authors recently. Coquand and Spiwack (2010) studied four constructively inequivalent notions of finite sets in set theory à la Bishop: enumerated sets (that we call listable sets), bounded size sets, Noetherian sets and streamless sets. They showed how these different notions are connected and proved several closure properties. The study of streamless sets was furthered by Parmann (2014) in the setting of Martin–Löf type theory. He showed that streamless sets are closed under Cartesian product if at least one of the sets has decidable equality. Firsov *et al.* (2016) further studied a number of variations of Noetherianness in detail. Firsov and Uustalu (2015) developed a toolbox for practical programming with listable subsets of base sets with decidable equality in Agda. Bezem *et al.* (2012) focused on the interrelationships between specializations of different notions of finiteness for decidable subsets of natural numbers.

Rational trees (also known as regular trees) were thoroughly studied in the 1970s in the context of solving finite systems of guarded equations describing potentially infinite computations (Elgot, 1975; Bloom and Elgot, 1976; Elgot *et al.*, 1978; Ginali, 1979; Courcelle, 1983). Adámek *et al.* (2003) reworked the algebraic treatment into a categorical analysis.

Turbak and Wells (2001) and Ghani *et al.* (2006) investigated representation and manipulation of rational data in functional programming. Ancona (2013) studied rational corecursion in the context of logic programming.

Jeannin *et al.* (2013) were specifically interested in recursion on rational or non-well-founded data.

Spadotti (2015; 2016) described an implementation of rational trees in Coq. Similarly to us, he characterized rational trees both coinductively, as a subset of the coinductive type of non-well-founded trees with finitely many distinct subtrees, and inductively, as well-founded trees with backpointers. Our paper focuses on the relation between listability, Noetherianness and rational sequences. Such analysis is missing in Spadotti (2015; 2016), where only listability is considered.

9 Conclusions

In the paper, we introduced rational sequences in type theory. The different rationality predicates presented in Section 4 are all logically equivalent. Particularly noteworthy is the isomorphism between SeqRN and SeqRL, a consequence of Proposition 7. We did not discuss basing rationality on streamlessness and boundedness. But bounded size is between listability and Noetherianness and is hence also equivalent to them for orbits. Streamlessness is generally weaker than Noetherianness, but for orbits it is again equivalent.

A user of this rational sequences library would prefer to program with lists with backpointers (a finitary representation, a kind of syntax) and then possibly evaluate the results into rational sequences (the intended denotation or semantics). It is also possible to view general sequences as the intended denotations, but then one should not forget that the types NoethSub (Suffix xs), ListableSub (Suffix xs) and FinOrb tail xs are not propositions. This means that in general, when constructing a map f of type SeqRN $A \to X$, for some type X, one should prove that $f\,(xs, r) \equiv f\,(xs, r')$, where r and r' are two proofs of rationality of the sequence xs. The same holds for maps out of SeqRL A and SeqRO A. For maps out of ListP A, one should prove that equivalent lists with backpointers are mapped to the same result.

This paper serves as a starting point for an implementation of general rational trees in Martin–Löf type theory. Rational trees are non-well-founded trees with a finite number of distinct subtrees. They arise as solutions of systems of guarded iterative equations. The representations of rational sequences described in this paper (the ones in Section 4 and lists with backpointers in Section 6) have been chosen because they scale well to general rational trees. Lists with backpointers are isomorphic to pairs of lists, and those could even work smoother for the example of decimal expansions of rational numbers. But this isomorphism does not scale to rational trees.

References

Adámek, J., Milius, S. & Velebil, J. (2003) Free iterative theories: A coalgebraic view. *Math. Struct. Comput. Sci.* **13**(2), 259–320.

Ancona, D. (2013) Regular corecursion in Prolog. *Comput. Lang. Syst. Struct.* **39**(4), 142–162.

Bezem, M., Nakata, K. & Uustalu, T. (2012) On streams that are finitely red. *Log. Meth. Comput. Sci.* **8**(4), article 4.

Bloom, S. L. & Elgot, C. C. (1976) The existence and construction of free iterative theories. *J. Comput. Syst. Sci.* **12**(3), 305–318.

Coquand, T. & Spiwack, A. (2010) Constructively finite? In *Scientific Contributions in Honor of Mirian Andrés Gómez*, Laureano Lambán, L., Romero, A. & Rubio, J. (eds), Universidad de La Rioja, pp. 217–230.

Courcelle, B. (1983) Fundamental properties of infinite trees. *Theor. Comput. Sci.* **25**, 95–169.

Elgot, C. C. (1975) Monadic computation and iterative algebraic theories. In *Logic Colloquium '73*, Studies in Logic and Foundations of Mathematics, vol. 80, Amsterdam: North-Holland, pp. 175–230.

Elgot, C. C., Bloom, S. L. & Tindell, R. (1978) On the algebraic structure of rooted trees. *J. Comput. Syst. Sci.* **16**(3), 361–399.

Firsov, D. & Uustalu, T. (2015) Dependently typed programming with finite sets. In Proceedings of 11th ACM SIGPLAN Wksh. on Generic Programming, WGP '15, New York: ACM Press, pp. 33–44.

Firsov, D., Uustalu, T. & Veltri, N. (2016) Variations on Noetherianness. In Proceedings of 6th Wksh. on Mathematically Structured Functional Programming, MSFP 2016, Atkey, R. & Krishnaswami, N. (eds), Electron. Proc. in Theor. Comput. Sci. Sydney: Open Publishing Assoc.

Ghani, N., Hamana, M., Uustalu, T. & Vene, V. (2006) Representing cyclic structures as nested datatypes (2006). In Proceedings of 7th Symp. on Trends in Functional Programming, TFP 2006, Nilsson, H. (ed), Univ. of Nottingham, pp. 173–188.

Ginali, S. (1979) Regular trees and the free iterative theory. *J. Comput. Syst. Sci.* **18**(3), 228–242.

Huet, G. (1997) The zipper. *J. Funct.Program.* **7**(5), 549–554.

Jeannin, J.-B., Kozen, D. & Silva, A. (2013) Language constructs for non-well-founded computation. In Proceedings of 22nd European Symposium on Programming, ESOP 2013, Felleisen, M. & Gardner, P. (eds), Lect. Notes in Comput. Sci., vol. 7792, Heidelberg: Springer, pp. 61–80.

Norell, U. (2009) Dependently typed programming in Agda. In *Revised Lectures from 6th International School. on Advanced Functional Programming, AFP 2008*, Koopman, P., Plasmeijer, R. & Swierstra, S. D. (eds), Lect. Notes in Comput. Sci., vol. 5832, Heidelberg: Springer, pp. 230–266.

Parmann, E. (2014) Investigating streamless sets. In Proceedings of 20th International Conference on Types for Proofs and Programs, TYPES 2014, Herbelin, H., Letouzey, P. & Sozeau, M. (eds), Leibniz Int. Proc. in Informatics, vol. 39, Saarbrücken/Wadern: Dagstuhl Publishing, pp. 187–201.

Spadotti, R. (2015) A mechanized theory of regular trees in dependent type theory. In Proceedings of 6th International Conference on Interactive Theorem Proving, ITP 2015, Urban, C. & Zhang, X. (eds), Lect. Notes in Comput. Sci., vol. 9236, Heidelberg: Springer, pp. 405–420.

Spadotti, R. (2016) Une théorie mécanisée des arbres réguliers en théorie des types dépendants. PhD Thesis, Université Toulouse 3 Paul Sabatier.

Turbak, F. & Wells, J. (2001) Cycle therapy: A prescription for fold and unfold on regular trees. In Proceedings of 3rd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, PPDP 2001, New York: ACM Press, pp. 137–149.