# PFPL Supplement: Natural and Co-Natural Numbers*

Robert Harper

September, 2020

## 1 Introduction

The concept of *polarity* characterizes type constructors according to the role of their introductory and eliminatory forms. A *positive* type constructor is one whose elements are characterized by its *introduction* rules, which determine all of its elements. Correspondingly, it enjoys a *mapping-out* property given by an elimination form that reduces the general case to each of the introductory cases. For example, nullary and binary sum types are positive: the type `0` has no introductory forms, and the type $\tau_1 + \tau_2$ has two, one for each injection. The mapping-out property of sums is given by the nullary and binary case analysis forms, which define mappings from the sum types into an arbitrary type in terms of the cases for their introductory forms: none, in the nullary case, and two in the binary case. Inductive types, as defined in Chapters 14 and 15 of **PFPL**, are positive types because their elements are given by a single introductory form determined by the type operator. Their mapping-out property is expressed by the *recursor*, whose behavior is defined using generic programming.

A *negative* type is, dually, characterized by its *elimination* rules, which specify how elements of the negative type behave when examined. Correspondingly, the introductory forms of a negative type are defined to engender the expected behavior. For example, function types are negative; they are characterized by their applicability to elements of the domain type to obtain elements of the codomain type. For functions the introductory form, $\lambda$-abstraction, which defines application by substitution for its bound variable.[1] Nullary and binary product types are also negative in that they are characterized by projection of their components—none in the nullary case, and two in the binary case. Consequently, the introduction rules package up zero or two components, and react to projection (if any) appropriately. Although it is trivial to say so, it may be helpful to note that it is possible for an ordered pair to be represented with the second component first and the first second, just so long as projection retrieves the appropriate component. Coinductive types, as defined in **PFPL**, are negative. Their elements must unfold in accordance with the defining type operator. A *generator* is a state machine that implements this behavior.

To develop these idea the inductive and coinductive types determined by the polynomial type



[1] More generally, the function type admits "foreign functions," those written in another language, by defining how to apply them. It does not matter what they "really are."

operator $t \,.\, \mathtt{1} + t$, are, respectively

$$\mathtt{nat} \triangleq \mu(\, t \,.\, \mathtt{1} + t\,)$$
$$\mathtt{conat} \triangleq \nu(\, t \,.\, \mathtt{1} + t\,).$$

The inductive interpretation is a repackaging of the formulation of the natural numbers given in Chapter 9. The co-inductive interpretation is its *dual*, which reverses the roles of the introduction and elimination rules for the inductive interpretation.

## 2 Inductive Types

For a type $\rho$ to be *closed* with respect to the type operator $t \,.\, \mathtt{1} + t$ means that it is equipped with $e_0 : \rho$, a *basis* element of $\rho$, and $x : \rho \vdash e_1 : \rho$, an *inductive step* for $\rho$.[2] The type $\mathtt{nat}$ is itself closed, with basis $\mathtt{z} : \mathtt{nat}$ being $\mathtt{fold}(\,\mathtt{l} \cdot \langle\rangle\,)$, and inductive step $x : \mathtt{nat} \vdash \mathtt{s}(\,x\,) : \mathtt{nat}$ being $\mathtt{fold}(\,\mathtt{r} \cdot x\,)$. The minimality of $\mathtt{nat}$ such closed types $\rho$ is expressed by the *iterator*, or *recursor*,

$$x : \mathtt{nat} \vdash \mathtt{iter}\{\rho\}(\, x \,;\, e_0 \,;\, x \,.\, e_1\,) : \rho$$

which is defined to be the recursor for the inductive type defining $\mathtt{nat}$,

$$\mathtt{rec}\{t \,.\, \mathtt{1} + t\}(\, e \,;\, x \,.\, \mathtt{case}(\, x \,;\, \_ \,.\, e_0 \,;\, x \,.\, e_1\,)\,).$$

For example, the "doubling" function on natural numbers is defined by

$$x : \mathtt{nat} \vdash \mathtt{twice}(\, x\,) \triangleq \mathtt{iter}\, e\, \{\mathtt{z} \hookrightarrow \mathtt{z} \mid \mathtt{s}(\, x\,) \hookrightarrow \mathtt{s}(\,\mathtt{s}(\, x\,))\} : \mathtt{nat}. \tag{1}$$

In essence this function replaces every successor in a number by two successors, thereby obtaining twice that number. Given this definition, one would expect to be able to prove the following properties of it:

1. $\mathtt{twice}(\,\mathtt{z}\,) =_{\mathtt{nat}} \mathtt{z}$, and
2. For any $e : \mathtt{nat}$, $\mathtt{twice}(\,\mathtt{s}(\, e\,)\,) =_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\, e\,)\,)\,)$.

This specification makes use of equality on closed expressions of type $\mathtt{nat}$, written $e =_{\mathtt{nat}} e'$. It is defined to be the *strongest*, or *least*, relation such that

1. If $e \longmapsto^* \mathtt{z}$ and $e' \longmapsto^* \mathtt{z}$, then $e =_{\mathtt{nat}} e'$, and
2. If $e \longmapsto^* \mathtt{s}(\, e_1\,)$, $e' \longmapsto^* \mathtt{s}(\, e_1'\,)$, and $e_1 =_{\mathtt{nat}} e_1'$, then $e =_{\mathtt{nat}} e'$.

The first property of doubling follows directly by evaluating $\mathtt{twice}(\,\mathtt{z}\,)$ to $\mathtt{z}$.

The entailment $x : \mathtt{nat} \vdash e_2 =_{\mathtt{nat}} e_2'$ means

$$\textit{if } e_1 =_{\mathtt{nat}} e_1', \textit{ then } [e_1/x]e_2 =_{\mathtt{nat}} [e_1'/x]e_2'.$$

The second property of doubling may be reformulated as the entailment

$$x : \mathtt{nat} \vdash \mathtt{twice}(\,\mathtt{s}(\, x\,)\,) =_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\, x\,)\,)\,),$$

from which the second property follows by reflexivity of equality at type $\mathtt{nat}$. The proof of this entailment is as follows:

[2] A given type $\rho$ may be thus closed in many ways, according to the choice of basis and inductive step.

1. If $e_1$ and $e_1'$ step to $\mathtt{z}$, then the result follows by a simple calculation.
2. If $e_1 \longmapsto^* \mathtt{s}(\,e_2\,)$, $e_1' \longmapsto^* \mathtt{s}(\,e_2'\,)$, and $e_2 =_{\mathtt{nat}} e_2'$, then

$$\begin{aligned}
\mathtt{twice}(\,\mathtt{s}(\,e_1\,)\,) &=_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\,e_1\,)\,)\,) \\
&=_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\,\mathtt{s}(\,e_2\,)\,)\,)\,) \\
&=_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\,e_2'\,)\,)\,)\,)\,) \quad \textit{(IH)} \\
&=_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\,\mathtt{s}(\,e_2'\,)\,)\,)\,) \\
&=_{\mathtt{nat}} \mathtt{s}(\,\mathtt{s}(\,\mathtt{twice}(\,e_1'\,)\,)\,).
\end{aligned}$$

As a corollary, $\mathtt{twice}(\,\overline{n}\,) =_{\mathtt{nat}} \overline{2 \times n}$ for all $n \in \mathbb{N}$, with the proof being a straightforward mathematical induction on $n$.

# 3 Co-Inductive Types

For a type $\sigma$ to be *consistent*, with respect to the type operator $t\,.\,\mathtt{1} + t$, means that it admits a *transition operation* $x\!:\!\sigma \vdash e_1 : \mathtt{1} + \sigma$ that determines whether the current state, an element of $e : \sigma$, is *terminal*, and, if not, what is the *next* state of the computation.[3] Coinductive types are defined to be maximal among consistent types. This is expressed by the generator,

$$x : \sigma \vdash \mathtt{gen}\{\sigma\}(\,x\,;x\,.\,e_1\,),$$

where $x : \sigma \vdash e_1 : \mathtt{1} + \sigma$ is a transition operation for $\sigma$. The type $\mathtt{conat}$ is itself consistent, with transition $x : \mathtt{conat} \vdash \mathtt{unfold}(\,x\,) : \mathtt{1} + \sigma$. The computation $\mathtt{unfold}(\,e\,)$ may be thought of as computing the (partial) predecessor of $e$, yielding $\mathtt{l} \cdot \langle\rangle$ when $e$ represents zero, and $\mathtt{l} \cdot e'$ when $e$ represents the successor of $e'$. Then the conditional on $\mathtt{conat}$ is defined in terms of this structure:

$$y : \mathtt{conat} \vdash \mathtt{coifz}\{\rho\}(\,y\,;e_0\,;x\,.\,e_1\,) : \rho \triangleq \mathtt{case}(\,\mathtt{unfold}(\,y\,)\,;\_\,.\,e_0\,;x\,.\,e_1\,). \tag{2}$$

Because the elements of a coinductive type are determined by the elimination form, it takes some practice getting used to how one creates co-natural numbers. To get started, define $\mathtt{coz}$, representing zero, by the equation

$$\mathtt{coz} \triangleq \mathtt{gen}\{\mathtt{1}\}(\,\langle\rangle\,;\_\,.\,\mathtt{l} \cdot \langle\rangle\,). \tag{3}$$

Note that $\mathtt{unfold}(\,\mathtt{coz}\,) \longmapsto^* \mathtt{l} \cdot \langle\rangle$, which is to say that it has no predecessor; in this regard it behaves like zero. As a second example, define the infinite co-natural number by

$$\infty \triangleq \mathtt{gen}\{\mathtt{1}\}(\,\langle\rangle\,;x\,.\,\mathtt{r} \cdot x\,). \tag{4}$$

Check that $\mathtt{unfold}(\,\infty\,) \longmapsto^* \mathtt{r} \cdot \infty$. Thus, $\infty$ is its own predecessor, which justifies it as being a co-natural number, the only requirement being that it either lacks, or has a unique, predecessor.

It is a little more tricky to define $\mathtt{cos}(\,n\,)$, the co-successor of a co-natural number. Intuitively, $\mathtt{cos}(\,n\,)$ always has a predecessor, namely $n$. When $n$ behaves like zero, co-successor emits one successor, and then behaves like zero. When $n$ has predecessor $n'$, then the predecessor of $\mathtt{cos}(\,n\,)$ should behave like $\mathtt{cos}(\,n'\,)$. With these thoughts in mind, define the state type $\sigma \triangleq \mathtt{conat} + \mathtt{1}$ whose elements represent the two possible states just described:

[3] A given type $\sigma$ may be consistent in many ways, according to the choice of transition.

1. State $\mathtt{l} \cdot n$: behave like the successor of $n$;

2. State $\mathtt{r} \cdot \langle\rangle$: behave like the successor of zero.

Then $\mathtt{cos}(\, n \,)$ may be defined by the following generator:

$$\mathtt{gen}\{\sigma\}(\, \mathtt{l} \cdot e \,;\, s \,.\, \mathtt{case}(\, s \,;\, n \,.\, \mathtt{coifz}\{1 + \sigma\}(\, n \,;\, \mathtt{r} \cdot \mathtt{r} \cdot \langle\rangle \,;\, n' \,.\, \mathtt{r} \cdot \mathtt{l} \cdot n' \,) \,;\, \_ \,.\, \mathtt{l} \cdot \langle\rangle \,)\,). \tag{5}$$

As an exercise, check that the behavior of $\mathtt{cos}(\, n \,)$, when unfolded, is as described above.

The doubling function for natural numbers may be extended to the co-natural numbers in such a way that it agrees with the former on finite co-natural numbers, and on $\infty$ behaves like $\infty$. Thus,

1. if $e$ behaves like $\mathtt{coz}$, then $\mathtt{unfold}(\, \mathtt{cotwice}(\, e \,)\,)$ also behaves like $\mathtt{coz}$;

2. if $e$ behaves like $\mathtt{cos}(\, e' \,)$, then $\mathtt{unfold}(\, \mathtt{cotwice}(\, e \,)\,)$ has a predecessor $e''$, itself with a predecessor that behaves like $\mathtt{cotwice}(\, e' \,)$.

The specification is given in terms of predecessors, because $\mathtt{conat}$ is a co-inductive type.

The code for doubling a co-natural number is a generator with state type $\sigma \triangleq \mathtt{conat} + \mathtt{conat}$. The two states correspond to the doubling a given co-natural number, and to the one more than the doubling of a given co-natural number. More precisely, define

$$\mathtt{cotwice}(\, e \,) \triangleq \mathtt{gen}\{\sigma\}(\, \mathtt{l} \cdot e \,;\, s \,.\, \mathtt{case}(\, s \,;\, n \,.\, \mathtt{coifz}\{\mathtt{conat}\}(\, n \,;\, \mathtt{l} \cdot \langle\rangle \,;\, n' \,.\, \mathtt{r} \cdot \mathtt{r} \cdot n' \,) \,;\, n \,.\, \mathtt{r} \cdot \mathtt{l} \cdot n \,)\,). \tag{6}$$

When in the doubling-plus-one state, the generator, when forced, indicates that it has a predecessor, which is the corresponding doubling state. When in the doubling state, it is necessary to test whether the number behaves like zero, in which case the doubling does as well; otherwise, it indicates that it has a predecessor, which is in the corresponding doubling-plus-one state.

All this talk of "same behavior" may be made precise by defining equality of co-natural numbers, $e =_{\mathtt{conat}} e'$, to be the *largest*, or *maximal*, binary relation consistent with unfolding:

$$\textit{if } e =_{\mathtt{conat}} e' \textit{ then } \mathtt{unfold}(\, e \,) =_{1+\mathtt{conat}} \mathtt{unfold}(\, e' \,).$$

Consistency of equality means that $e =_{\mathtt{conat}} e'$ implies that neither has a predecessor (that is, both behave like $\mathtt{coz}$), or both have predecessors that have again the same behavior under unfolding. For equality of co-natural numbers to be *maximal* means that

> Two co-natural numbers are equal, unless, a finite number of predecessors of both, results in one having a predecessor and the other not.

This is called *proof by coinduction*.

Coinduction corresponds to the legal principle "innocent (equal), unless proven guilty (unequal)." That is, two co-natural numbers are considered to be equal, unless that claim can be refuted by unfolding. The more familiar principle of proof by induction, on the other hand, corresponds to the legal principle "guilty (unequal) unless proven innocent (equal)." That is, two natural numbers are considerd to be unequal, unless they can by proved equal by folding.

For example, $\mathtt{cos}(\, \mathtt{coz} \,) \neq_{\mathtt{conat}} \mathtt{coz}$, because the former has a predecessor, $\mathtt{coz}$, and the latter does not, and this remains true after applying any number of successors to both sides. And, in a similar spirit, $\infty \neq_{\mathtt{conat}} n$ for any finite co-natural number $n$.

Let us consider how to prove that $\texttt{cos}(\,\infty\,) =_{\texttt{conat}} \infty$, which is to say that $\infty$ is its own successor. By coinduction this equal is deemed true, so long as it is consistent; that is, it suffices to check that

$$\texttt{unfold}(\,\texttt{cos}(\,\infty\,)\,) =_{\texttt{conat}} \texttt{unfold}(\,\infty\,).$$

Consulting the definitions of $\texttt{cos}(\,\infty\,)$ and $\infty$, the left-hand side steps to $\texttt{r}\cdot\infty$, as does the right-hand side. Both have the same predecessor, so no further examination can distinguish them. Thus, the equation being not inconsistent, holds true.

As another example, consider the intuitively obvious claim that "twice $\infty$ is $\infty$." Although such intuitions can be helpful, it is important to understand clearly what is being said in such a statement. The justification is *not* a hand-wavey argument based on the "size" of $\infty$. Rather, it is based on the definition of the extended doubling function, and, most importantly, on the maximality of equality of co-natural numbers. The idea of the proof is to assert two equations each of whose consistency rests on the other. Being jointly consistent, they are jointly true, and are sufficient for the desired result. Put in legal terms, the two equations "cross-reference each other's lies." In a regime in which the prosecution must refute such claims, if the "lies" are consistent, then they must be accepted as truths. Perhaps the two criminals really did rob that bank, but because their lies are self-consistent, they must be declared innocent in court!

The trick to determining the two equations to consider is based on the two states of the generator that arise during evaluation of an application of doubling to a co-natural number. Consulting definition 6, $\texttt{cotwice}(\,n\,)$ evaluates to the generator used in the definition in state $\texttt{l}\cdot n$, indicating that it represents the doubling of $n$. Were $n =_{\texttt{conat}} \texttt{coz}$, then the doubling generator in that state, written $g(\texttt{l}\cdot n)$, would be equal to $\texttt{coz}$ as well. But, because $\infty =_{\texttt{conat}} \texttt{cos}(\,\infty\,)$, this case does not arise, infinity being its own successor and therefore non-zero. Instead, the generator $g(\texttt{l}\cdot n)$ indicates that it is a "double successor," with predecessor $g(\texttt{r}\cdot n)$, representing twice $n$ plus one. It, in turn, is non-zero, and has $g(\texttt{l}\cdot n)$ are predecessor. Thus, the following two equations suffice to show that $\texttt{cotwice}(\,\infty\,) =_{\texttt{conat}} \infty$:

1. $g(\texttt{l}\cdot\infty) =_{\texttt{conat}} \infty$, and
2. $g(\texttt{r}\cdot\infty) =_{\texttt{conat}} \infty$.

These are mutually consistent in that

1. $g(\texttt{l}\cdot\infty)$ has predecessor $g(\texttt{r}\cdot\infty)$, and $\infty$ has itself as predecessor, and these are equal by the second equation.
2. $g(\texttt{r}\cdot\infty)$ has predecessor $g(\texttt{l}\cdot\infty)$, and $\infty$ has itself as predecessor, and these are equal by the first equation.

Thus, the two equations cross-reference each other, and are not refutable by unfolding, and so are true equations between co-natural numbers. It is then easy to check that $\texttt{cotwice}(\,\infty\,) =_{\texttt{conat}} \infty$ by a very similar argument, using the definition of doubling given by equation 6.

At first this sort of reasoning may seem "magical," or even invalid, but it is perfectly justified by the characterization of equality of co-natural numbers as the *maximal consistent binary relation* on co-natural numbers. Being maximal, "everything consistent is true," and this is the key to the preceding proof.

## References

Robert Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, Cambridge, England, Second edition, 2016.