INSTITUTE OF COMPUTER SCIENCE
Faculty of Mathematics and Natural Sciences

Heinrich Heine University Düsseldorf

Heinrich Heine Universität Düsseldorf

The Topos of Trees

Luca Leon Happel

from Mönchengladbach

Bachelorarbeit

Date of issue:	September 17, 2026
Date of submission:	December 17, 2026
Reviewers:	Dr. Peter Arndt
	Konrad Völkel

Acknowledgements

I wish to express my extraordinary gratitude to my supervisor, Dr. Peter Arndt, whose explanations helped me tremendously throughout the writing of this thesis, clarifying even highly abstract concepts in no time. I am especially thankful for all the times he explained concepts to me after hours, which went far beyond what one could normally expect. Not only were his explanations excellent, but working with him was also a breeze.

I would definitely recommend Dr. Arndt to any student who wants to write a thesis in topos theory or category theory, as he has been the best supervisor I have had so far. Although I have had some excellent ones before, Dr. Arndt raised the bar dramatically with his effort, patience, and expertise.

Contents

1 Introduction **1**

2 Preliminaries **3**

2.1 Step-indexing . . . 3

2.2 Category Theory . . . 3

2.3 Subobjects . . . 5

3 The topos of trees **6**

3.1 The later endofunctor . . . 7

3.2 Common objects and morphisms in the topos of trees . . . 8

3.3 Fixed points . . . 9

3.4 Global elements of the topos of trees . . . 10

3.5 The topos of graphs . . . 11

3.6 The topos of trees as a category of forests . . . 12

3.7 Graded graphs . . . 13

3.8 Limits, colimits and exponentials in the topos of trees . . . 14

3.9 Further common morphisms in the topos of trees . . . 17

4 Categorical logic **18**

4.1 Formal languages . . . 18

4.2 Categorical semantics . . . 20

4.3 Heyting algebras and adjunctions . . . 22

4.4 The truth values of $\mathcal{S}$ . . . 23

4.5 Connectives and Quantifiers in $\mathcal{S}$ . . . 27

4.6 The later modality . . . 30

4.7 A higher-order typed λ-calculus in a topos . . . 31

5 Applications **32**

5.1 Characterizing binary trees . . . 32

5.2 The Collatz tree . . . 36

6 Conclusion and future work **38**

A Further cases of Example 5.8 **39**

References **40**

1 Introduction

In this thesis we provide an introduction to the topos of trees, a model for the higher-order guarded λ-calculus, which allows for the definition of recursive types and functions in a way that prevents non-termination, and explore its intuitionistic internal logic through a category theoretic, logical and graph-theoretic lens. We introduce the necessary sheaf theory and define needed concepts from categorical logic from the ground up, with emphasis on intuition and practical applications, by use of the topos of trees' connection to graph theory, which we also make explicit.

Initial motivation stems from formal methods and type theory: Structural recursion on coinductively defined data types is a classical source of non-termination, which is notoriously hard to detect and debug, as can be seen in the following Python code which fails to terminate:

```python
from itertools import count
squares = map(lambda n: n * n, count())  # lazy, so this is fine
print(list(squares)[:5])                 # never terminates
```

In high-security and safety-critical systems, such errors must be avoided at all cost. But proving that a program is productive and will terminate in finite time still remains a challenge. languages like Lean4 sidestep this problem by by disallowing most coinductive types (*Coinductive and Inductive Predicates* 2026, Sec. 7.6.5), while others like Haskell can evaluate examples like the one above by lazy evaluation, but still fail at more subtle examples. Consider the misconception "if it compiles, it works", a mantra commonly found regarding Haskell's type system, which nonetheless fails to catch the following bug at compile time:

```haskell
toggle = 1 : 0 : toggle
interleave (x : xs) ys = x : interleave ys xs
-- does work
paperfolds = interleave toggle paperfolds
-- does not work
paperfolds' = interleave paperfolds' toggle
```

Here `paperfolds` is a productive stream, while `paperfolds'` is not, i.e. calculating the first element will never terminate and this is not caught by the type checker. *Guarded recursion* promises a solution: We introduce a new type constructor $\blacktriangleright$, called "later", which is modeled in the *topos of trees* through a synthetic form of step-indexing (Birkedal et al., 2011), stating that a value of type $\blacktriangleright A$ will only be available in the next computation/time step. Otherwise ill-formed self-referential type equations such as

$$\mathtt{Str}(T) \cong T \times \mathtt{Str}(T) \tag{1}$$

may then be reformulated as

$$\mathtt{Str}(T) \cong T \times \blacktriangleright\mathtt{Str}(T) \tag{2}$$

guaranteeing a unique solution, whose elements are also guaranteed to be productive. Following Clouston et al. (2015, p. 408), typing `interleave :: Str a -> ▶ Str a -> Str a` ensures a type error only in the case of `paperfolds'`, while `paperfolds` is still well-typed.

By encoding the passing of time into the type system, we naturally also gain denotational semantics for functional reactive programming (FRP), which allows for the definition of time-varying values, and reactions thereupon. For example, using $L(A) \cong A + \blacktriangleright L(A)$ we can model a value that is either available now, or will be available later, like async/await in imperative programming languages.

The passage of time encoded in the type system also allows for the expression of temporal properties, such as that some "good" event eventually happens, or that some "bad" event never occurs using only the type system. This recent development as proposed by Jaber and Riba (2021) therefore lifts model checking into the type level through the use of refinement types in the topos of trees, building further connections to formal methods. As an example, the formulae in Section 5.1 can be used to check whether a term $x : X$ modeling a nondeterministic process has, at each time step, exactly two distinct possible events which may occur to it.

Section 2 recalls step-indexing and the necessary category theory. Section 3 introduces the topos of trees and the later functor, and describes its objects as forests and graded graphs. Section 4 develops categorical logic and uses this graph-theoretic picture to compute the truth values, connectives, quantifiers and later modality of its internal logic. Section 5 applies this logic to characterize binary trees.

verify Jaber–Riba and Bizjak et al. claims against the papers

Related work. Step-indexed models were introduced by Appel and McAllester (2001) to give semantics to recursive types, and the modality $\blacktriangleright$ by Nakano (2000). Birkedal et al. (2011) and Birkedal et al. (2012) unify both in the topos of trees and study its internal logic. Clouston et al. (2015) build a guarded λ-calculus on top of it, and Kocsis (2025) details the constructions that Sections 2 and 4 follow. Guarded recursion is available in Guarded Cubical Agda (Veltri and Vezzosi, 2020), based on guarded cubical type theory (Birkedal et al., 2019), in the DSL Rattus for Haskell (Bahr, 2022), and recently also as an embedded language in Lean 4 (Stepanenko et al., 2026).

Contributions. We introduce the topos of trees from the ground up, show its connection to the topos of graphs and inspect it through this graph-theoretic lens. In the internal language of the topos of trees, we then

- give sentences $\mathrm{BinTree}_X^{\mathrm{Rel}}$ and $\mathrm{BinTree}_X^{\mathrm{iso}}$ (Definitions 5.1 and 5.2), quantifying over the exponentials X^R, respectively X^B and B^X, where the truth value of $\mathrm{BinTree}_X^{\mathrm{iso}}$ is the number of levels up to which the forest of X is a binary tree (Proposition 5.3);
- replace this quantification over morphisms by quantification over truth values, using the later modality, and obtain a sentence $\mathrm{BinTree}_X$ that holds if and only if $\mathrm{BinTree}_X^{\mathrm{iso}}$ does (Proposition 5.10).
- define the Collatz tree (Proposition 5.15)

2 Preliminaries

2.1 Step-indexing

Streams are coinductive types, meaning they are the maximal fixed point of the type equation $\texttt{Str}(A) \cong A \times \texttt{Str}(A)$, in contrast to inductive types, which are the minimal fixed point of a type equation, such as $\texttt{List}(A) \cong 1 + A \times \texttt{List}(A)$. Its terms $x : \texttt{Str}(A)$ are infinite sequences of terms $(x_i : A)_{i \in \mathbb{N}}$. Assume a function $f : A \to Y$ takes at least Δt time to compute its output. Using the functoriality of the stream type to lift f to $\bar{f} : \texttt{Str}(A) \to \texttt{Str}(Y)$ by applying f sequentially, its execution time $\sum_{i \in \mathbb{N}} \Delta t$ diverges to infinity. $\bar{f}$ is thus not computable in finite time.

In Haskell, by lazy evaluation, we can nonetheless process streams partially, by processing finite prefixes of the stream in finite time. More precisely, at computation step n, we can look at the prefix `take n xs` of the stream `xs`.

A similar approach is taken in the semantic tool of *step-indexing*, which allows us to stratify self-referential type definitions (such as streams) into a sequence of approximations (such as finite prefixes of streams): Given a type equation $T \cong F(T)$ we can define a sequence of approximations $(T_n)_{n \in \mathbb{N}}$ by $T_0 \cong F(1)$ and $T_{n+1} \cong F(T_n)$ where 1 is the unit type (the type with exactly one inhabitant).

For streams $\texttt{Str}(A) \cong A \times \texttt{Str}(A)$, this yields $\texttt{Str}_0(A) \cong A \times 1$ and $\texttt{Str}_{n+1}(A) \cong A \times \texttt{Str}_n(A)$ and expanding these definitions, we get

$$
\begin{aligned}
\texttt{Str}_0(A) &\cong A \\
\texttt{Str}_1(A) &\cong A \times A \\
\texttt{Str}_2(A) &\cong A \times A \times A \\
\texttt{Str}_3(A) &\cong A \times A \times A \times A \\
&\vdots
\end{aligned}
$$

An especially elegant approach to this technique comes from pure mathematics and more precisely *sheaf theory*. This will lead us to the *topos of trees*, but before we can get to it, we first need to lay down some groundwork of sheaf and category theory.

2.2 Category Theory

We assume the basic concepts of category theory, such as categories, functors and natural transformations are familiar and we follow the notation of Mac Lane and Moerdijk (1994). In the remainder, let $\mathcal{C}$ be an arbitrary small category.

Definition 2.1 (Presheaf)**.** *A* presheaf *on $\mathcal{C}$ is a functor $X : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$. Presheaves on $\mathcal{C}$ and natural transformations between them form the* presheaf category $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$.

Presheaves are initially motivated from topology: Given a topological space T, let $\mathrm{Open}(T)$ be the category whose objects are the open sets of T and whose arrows are inclusions. A presheaf $X\colon \mathrm{Open}(T)^{\mathrm{op}} \to \mathbf{Set}$ assigns some data $X(U)$ to each open set U, and for inclusions $V \to U$, it assigns restriction maps $X(V) \leftarrow X(U)$.

A prototypical example of a presheaf is $C^0(-,\mathbb{R})$, which assigns to each open set U the set of continuous functions $C^0(U,\mathbb{R})$. Say we have an inclusion $\iota\colon V \rightarrowtail U$, then we can restrict a function $f \in C^0(U,\mathbb{R})$ to V by precomposing with ι, i.e. $f \mapsto f\circ\iota$. We then have $f|_V := f\circ\iota \in C^0(V,\mathbb{R})$ (Figure 1).

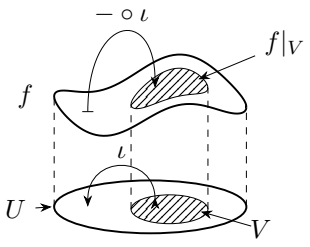


Figure 1: A function $f \in C^0(U,\mathbb{R})$ and its restriction $f|_V = f\circ\iota$ along $\iota\colon V \rightarrowtail U$.

This idea is then generalized: instead of considering only inclusions of open sets, we consider arbitrary arrows in $\mathcal{C}$. For $\mathcal{C} = 1$, the category consisting of only one object, a presheaf over it is simply a set. As such, the category of presheaves over 1 is equivalent to **Set**.

Presheaf categories come with a rich structure. In particular, by Mac Lane and Moerdijk, 1994, Ch. I, they are examples of *topoi*. These have successfully been used in pure mathematics to show the independence of the famous *continuum hypothesis*, and we now wish to make use of them in computer science.

Definition 2.2 (Elementary topos)**.** *A category $\mathcal{E}$ is an* (elementary) topos *if*

1. *it has all* finite limits*; in particular a* terminal object 1*, i.e. an object such that every X admits a unique morphism* $!\colon X \to 1$*, and* pullbacks*;*
2. *it is* cartesian closed*, i.e. for all objects X, Y there is an* exponential object Y^X *together with a natural bijection*

$$\mathit{Hom}_{\mathcal{E}}(Z \times X, Y) \cong \mathit{Hom}_{\mathcal{E}}(Z, Y^X);$$

3. *it has a* subobject classifier*, i.e. an object Ω together with a morphism $\top\colon 1 \to \Omega$ such that for every monomorphism $m\colon A \rightarrowtail X$ there is a unique* characteristic morphism *$\chi_A\colon X \to \Omega$ making*

$$\begin{array}{ccc} A & \rightarrowtail & X \\ {\scriptstyle !}\downarrow & & \downarrow{\scriptstyle \chi_A} \\ 1 & \xrightarrow[\top]{} & \Omega \end{array}$$

a pullback square.

For **Set**, the terminal object is the singleton set $1 = \{*\}$, the exponential object Y^X is the set of functions from X to Y, and the subobject classifier Ω is the set $\{\mathrm{true}, \mathrm{false}\}$, where $\top\colon 1 \to \Omega$ maps to true. The characteristic morphism $\chi_A\colon X \to \Omega$ of a subset $A \subseteq X$ is the function that maps elements of A to true and elements of $X \setminus A$ to false.

2.3 Subobjects

The subobject classifier naturally leads to the notion of subobjects, which are the categorical analogue of subsets.

Definition 2.3. *Let $\mathcal{C}$ be a category and X an object of $\mathcal{C}$. A* subobject *of X is an isomorphism class of monomorphisms $m\colon A \rightarrowtail X$, where m and $m'\colon A' \rightarrowtail X$ are isomorphic if $m = m' \circ \iota$ for an isomorphism $\iota\colon A \to A'$. The subobjects of X form a poset $\mathrm{Sub}_{\mathcal{C}}(X)$ with $m \leq m'$ if m factors through m'.*

Indeed, in a category $\mathcal{C}$ with pullbacks, such as a topos, $\mathrm{Sub}_{\mathcal{C}}$ forms a functor from $\mathcal{C}^{\mathrm{op}}$ to the category **Pos** of posets and monotone maps, which is given by

Definition 2.4. *Let $\mathcal{C}$ be a category with pullbacks. The* subobject functor $\mathrm{Sub}_{\mathcal{C}}\colon \mathcal{C}^{\mathrm{op}} \to \mathbf{Pos}$ *sends an object X to $\mathrm{Sub}_{\mathcal{C}}(X)$ and a morphism $f\colon X \to Y$ to the monotone map $f^*\colon \mathrm{Sub}_{\mathcal{C}}(Y) \to \mathrm{Sub}_{\mathcal{C}}(X)$ that sends $m\colon U \rightarrowtail Y$ to its pullback along f:*

$$\begin{array}{ccc} f^*U & \longrightarrow & U \\ {\scriptstyle f^*m}\downarrow & \lrcorner & \downarrow{\scriptstyle m} \\ X & \xrightarrow[f]{} & Y. \end{array}$$

A nice visual interpretation of this functor is provided in **Top**, the category of topological spaces and continuous maps. For a morphism $f \in \mathrm{Hom}_{\mathbf{Top}}(X, Y)$, Figure 2 draws f^*U on the mapping cylinder of f: it consists of the points of X that f sends into U, so only the part of U inside $f(X)$ matters. The subobject $U \rightarrowtail Y$ is drawn as a cross-hatched region, and the pullback $f^*U \rightarrowtail X$ is drawn similarly.

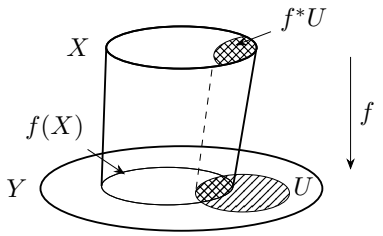


Figure 2: The pullback $f^*U = f^{-1}(U)$ along f.

In particular for a topos $\mathcal{E}$, we have:

Lemma 2.5. *Let $\mathcal{E}$ be a topos with subobject classifier Ω. Then for every object X in $\mathcal{E}$, there is a natural isomorphism*

$$\mathrm{Sub}_{\mathcal{E}}(X) \cong \mathit{Hom}_{\mathcal{E}}(X, \Omega).$$

Proof. By Definition 2.2, for every monomorphism $m\colon A \rightarrowtail X$ there is a unique characteristic morphism $\chi_A\colon X \to \Omega$ making the square

$$\begin{array}{ccc} A & \xrightarrow{m} & X \\ {\scriptstyle !}\downarrow & & \downarrow{\scriptstyle \chi_A} \\ 1 & \xrightarrow[\top]{} & \Omega \end{array}$$

a pullback. This gives a map $\mathrm{Sub}_{\mathcal{E}}(X) \to \mathrm{Hom}_{\mathcal{E}}(X, \Omega)$ sending m to χ_A. Conversely, for every morphism $f\colon X \to \Omega$, we can form the pullback

$$\begin{array}{ccc} P & \longrightarrow & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \top} \\ X & \xrightarrow[f]{} & \Omega \end{array}$$

which gives a map $\mathrm{Hom}_{\mathcal{E}}(X, \Omega) \to \mathrm{Sub}_{\mathcal{E}}(X)$ sending f to m. □

3 The topos of trees

Going back to step-indexing, we have assigned to each natural number some approximation of a given type. In our running example, we used the finite approximations of a stream, which in the limit would yield the actual stream.

As we have seen, we can assign to objects of a category some sets using the presheaf construction. Let us consider the finite von Neumann ordinal numbers ω, whose elements are $0 \leq 1 \leq 2 \leq \dots$ (commonly defined by $0 = \emptyset$ and $n{+}1 = n \cup \{n\}$, where $\omega = \{0, 1, \dots\}$ is the first infinite ordinal) and which form an *order-category*, meaning there is exactly one arrow $x \to y$ if and only if $x \leq y$, which can be represented as

$$0 \xrightarrow{\leq} 1 \xrightarrow{\leq} 2 \xrightarrow{\leq} \cdots$$

We use the presheaf category over ω to model step-indexing of types:

Definition 3.1. *The* topos of trees (*denoted* $\mathcal{S}$) *is the presheaf topos over the ordinal number* ω*, i.e. the functor category* $[\omega^{\mathrm{op}}, \mathbf{Set}]$.

Its objects are functors $X\colon \omega^{\mathrm{op}} \to \mathbf{Set}$ and its morphisms are natural transformations between such functors. Explicitly, an object $X\colon \omega^{\mathrm{op}} \to \mathbf{Set}$ is a sequence of sets $X(0), X(1), X(2), \dots$ together with functions $r_n^X\colon X(n{+}1) \to X(n)$ for each $n \in \mathbb{N}$, which can be depicted as

$$X(0) \xleftarrow{r_0^X} X(1) \xleftarrow{r_1^X} X(2) \xleftarrow{r_2^X} X(3) \xleftarrow{r_3^X} \cdots \tag{3}$$

Its morphisms are sequences of functions $f_n\colon X(n) \to Y(n)$ for each $n \in \mathbb{N}$ such that the following diagram commutes for all $n \in \mathbb{N}$:

$$\begin{array}{ccccccccc} X(0) & \longleftarrow & X(1) & \longleftarrow & X(2) & \longleftarrow & X(3) & \longleftarrow & \cdots \\ \downarrow{\scriptstyle f_0} & & \downarrow{\scriptstyle f_1} & & \downarrow{\scriptstyle f_2} & & \downarrow{\scriptstyle f_3} & & \\ Y(0) & \longleftarrow & Y(1) & \longleftarrow & Y(2) & \longleftarrow & Y(3) & \longleftarrow & \cdots \end{array} \tag{4}$$

Intuitively, ω counts time (or computation steps) and $X(n)$ is the data available at step n. Time only moves forward along ω, and by contravariance of X we can only see current and earlier data.

This is the structure of the finite stream approximations seen earlier. In the diagram below, the bottom row is ω, the row above it is $\mathtt{Str}(A)$, and the top two rows show its elements under $\mathtt{Str}(A)(n) \cong A^{n+1}$.

$$\begin{array}{ccccccccc}
[x_0] & \longleftarrow & [x_0,x_1] & \longleftarrow & [x_0,x_1,x_2] & \longleftarrow & [x_0,x_1,x_2,x_3] & \longleftarrow & \cdots \\
\cap & & \cap & & \cap & & \cap & & \\
A & \longleftarrow & A^2 & \longleftarrow & A^3 & \longleftarrow & A^4 & \longleftarrow & \cdots \\
\wr\| & & \wr\| & & \wr\| & & \wr\| & & \\
\mathtt{Str}(A)(0) & \longleftarrow & \mathtt{Str}(A)(1) & \longleftarrow & \mathtt{Str}(A)(2) & \longleftarrow & \mathtt{Str}(A)(3) & \longleftarrow & \cdots \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \\
0 & \longrightarrow & 1 & \longrightarrow & 2 & \longrightarrow & 3 & \longrightarrow & \cdots
\end{array} \tag{5}$$

Each successive approximation extends the previous one by one element $\mathtt{Str}(A)(n+1) \cong A \times \mathtt{Str}(A)(n)$. Unfolding this is guaranteed to reach the base case $\mathtt{Str}(A)(0) \cong A$ in finitely many steps. This ensures productivity.

3.1 The later endofunctor

A naive approach to defining infinite streams would be to write a type equation $\mathtt{Str}(A) \cong A \times \mathtt{Str}(A)$. Solving in **Set** yields as one possible (not unique) solution $\mathtt{Str}(A) \cong \prod_{n\in\mathbb{N}} A$, which lacks the step-indexing structure we desire. By extension this problem also arises in $\mathcal{S}$ for this type equation, which would yield $\mathtt{Str}(A)(n) \cong \prod_{k\in\mathbb{N}} A$ for all $n \in \mathbb{N}$.

One way to mitigate this problem is to introduce the *later* endofunctor $\blacktriangleright\colon \mathcal{S} \to \mathcal{S}$, such that we can write $\mathtt{Str}(A) \cong A \times \blacktriangleright\mathtt{Str}(A)$ instead, yielding a *unique* and *productive* solution.

Remark 3.2. *This approach is similar to the famous construction of streams in Abelson et al., 1996, Sec. 3.5.1, where a stream is a pair whose tail is* delayed*: `(cons-stream a b)` is `(cons a (delay b))`, and the tail `b` is only evaluated once it is* forced*. The later endofunctor plays the role of `delay` on the level of types.*

Definition 3.3. *The* later *endofunctor $\blacktriangleright\colon \mathcal{S} \to \mathcal{S}$ takes an X such as in* (3) *and maps it to $\blacktriangleright X$, represented as*

$$1 \xleftarrow{!} X(0) \xleftarrow{r_0^X} X(1) \xleftarrow{r_1^X} X(2) \xleftarrow{r_2^X} \cdots \tag{6}$$

Using $\blacktriangleright$, we get for each $X \in \mathcal{S}$ a new object $\blacktriangleright X \in \mathcal{S}$, and for each morphism $f\colon X \to Y$ a new morphism $\blacktriangleright f\colon \blacktriangleright X \to \blacktriangleright Y$. In fact, for each X we get a canonical $\mathcal{S}$-morphism $\mathrm{next}_X\colon X \to \blacktriangleright X$, the collection of which yields the natural transformation $\mathrm{next}\colon \mathrm{id} \Rightarrow \blacktriangleright$.

Definition 3.4 (next)**.** *The natural transformation* next*: $id \Rightarrow \blacktriangleright$ has components* $\mathrm{next}_X\colon X \to$

$\blacktriangleright X$ *given by* $(\text{next}_X)_0 = !$ *and* $(\text{next}_X)_{n+1} = r_n^X$*:*

$$\begin{array}{ccccccc} X(0) & \xleftarrow{r_0^X} & X(1) & \xleftarrow{r_1^X} & X(2) & \longleftarrow & \cdots \\ \downarrow{\scriptstyle !} & & \downarrow{\scriptstyle r_0^X} & & \downarrow{\scriptstyle r_1^X} & & \\ 1 & \xleftarrow{!} & X(0) & \xleftarrow{r_0^X} & X(1) & \longleftarrow & \cdots \end{array}$$

That is, next_X *sends the approximation at stage* $n+1$ *to the approximation at stage* n*.*

Remark 3.5. *In general, an endofunctor* F *need not admit a natural transformation* $\text{id} \Rightarrow F$*, as for the constant functor* $X \mapsto \emptyset$ *on* **Set** *there is not even a function* $1 \to \emptyset$*.*

3.2 Common objects and morphisms in the topos of trees

As we have seen, streams are an obvious application of the later endofunctor by $\texttt{Str}(A) \cong A \times \blacktriangleright\texttt{Str}(A)$. Unfolding this definition explicitly confirms the unique solution $\texttt{Str}(A)(n) \cong A^{n+1}$ for all $n \in \mathbb{N}$, and shows that recursing through $\blacktriangleright$ reaches the base case $\texttt{Str}(A)(0)$ in finitely many steps, which is the *guarded* recursion aspect of $\blacktriangleright$:

$$\begin{aligned} \texttt{Str}(A)(n) &\cong A \times \blacktriangleright\texttt{Str}(A)(n) \\ &\cong A^2 \times \blacktriangleright\texttt{Str}(A)(n-1) && (1^{\text{st}} \text{ recursion}) \\ &\cong A^3 \times \blacktriangleright\texttt{Str}(A)(n-2) && (2^{\text{nd}} \text{ recursion}) \\ &\vdots \\ &\cong A^n \times \blacktriangleright\texttt{Str}(A)(1) && ((n-1)^{\text{th}} \text{ recursion}) \\ &\cong A^{n+1} \times \blacktriangleright\texttt{Str}(A)(0) && (n^{\text{th}} \text{ recursion}) \\ &\cong A^{n+1} \times 1 \end{aligned}$$

This by extension also implies that each element of $\texttt{Str}(A)(n)$ is computable in finite time, i.e. the stream is productive.

Remark 3.6. *Implicitly, we have lifted the set* A *into the constant object* $\Delta A \in \mathcal{S}$ *by using the diagonal functor* $\Delta\colon \textbf{Set} \to \mathcal{S}$ *with* $\Delta A(n) = A$ *and* $r_n^{\Delta A} = \text{id}_A$ *for all* $n \in \mathbb{N}$*, which we may omit, when it is clear from the context. For* ΔA *we then get, for all* $\kappa, m \in \mathbb{N}$*,*

$$(\blacktriangleright^\kappa \Delta A)(m) \cong \begin{cases} 1 & \text{if } m < \kappa, \\ A & \text{if } m \geq \kappa. \end{cases}$$

where $\blacktriangleright^0 A := A$ *and* $\blacktriangleright^{\kappa+1} A := \blacktriangleright^\kappa \blacktriangleright A$*, and the index* κ *(called* clock*) counts how many computation/time steps are required until a value of* $\blacktriangleright^\kappa A$ *is available.*

Two common operations on streams are given by

$$\begin{array}{ll} \text{hd}\colon \texttt{Str}(A) \to A, & \text{tl}\colon \texttt{Str}(A) \to \blacktriangleright\texttt{Str}(A), \\ \text{hd}_n([x_0, \ldots, x_n]) = x_0, & \text{tl}_0 = !, \quad \text{tl}_{n+1}([x_0, \ldots, x_{n+1}]) = [x_1, \ldots, x_{n+1}]. \end{array}$$

Proposition 3.7. hd *and* tl *are morphisms in* $\mathcal{S}$.

Proof. We must show that hd and tl are natural transformations, which follows by the commutativity of the following squares for all $n \in \mathbb{N}$ (the square for tl between the stages 1 and 0 commutes trivially, since $(\blacktriangleright \mathtt{Str}(A))(0) = 1$):

$$\begin{array}{ccc} \mathtt{Str}(A)(n+1) & \xrightarrow{\mathrm{hd}_{n+1}} & A \\ {\scriptstyle r_n^{\mathtt{Str}(A)}}\downarrow & & \downarrow{\scriptstyle r_n^{\Delta A} = \mathrm{id}_A} \\ \mathtt{Str}(A)(n) & \xrightarrow[\mathrm{hd}_n]{} & A \end{array} \qquad \begin{array}{ccc} \mathtt{Str}(A)(n+2) & \xrightarrow{\mathrm{tl}_{n+2}} & (\blacktriangleright \mathtt{Str}(A))(n+2) \\ {\scriptstyle r_{n+1}^{\mathtt{Str}(A)}}\downarrow & & \downarrow{\scriptstyle r_{n+1}^{\blacktriangleright \mathtt{Str}(A)}} \\ \mathtt{Str}(A)(n+1) & \xrightarrow[\mathrm{tl}_{n+1}]{} & (\blacktriangleright \mathtt{Str}(A))(n+1) \end{array} \qquad \square$$

$$\begin{array}{ccc} [x_0, \ldots, x_{n+1}] & \longmapsto & x_0 \\ \downarrow & & \downarrow \\ [x_0, \ldots, x_n] & \longmapsto & x_0 \end{array} \qquad \begin{array}{ccc} [x_0, \ldots, x_{n+2}] & \longmapsto & [x_1, \ldots, x_{n+2}] \\ \downarrow & & \downarrow \\ [x_0, \ldots, x_{n+1}] & \longmapsto & [x_1, \ldots, x_{n+1}] \end{array}$$

Hence the head of the tail $\blacktriangleright\mathrm{hd} \circ \mathrm{tl}$ has type $\mathtt{Str}(A) \to \blacktriangleright A$, and the head of the tail of the tail $\blacktriangleright^2\mathrm{hd} \circ \blacktriangleright\mathrm{tl} \circ \mathrm{tl}$ has type $\mathtt{Str}(A) \to \blacktriangleright^2 A$, and so on, encoding the recursion depth requirements for computing these morphisms in their typing.

Corollary 3.8. *With* interleave$\colon \mathtt{Str}(A) \times \blacktriangleright\mathtt{Str}(A) \to \mathtt{Str}(A)$*, the definition of* `paperfolds` *from Section 1 is well-typed, but that of* `paperfolds'` *is not.*

Proof. By destructuring its first argument as `(x : xs)`, interleave implicitly applies hd and tl from Proposition 3.7 to it. For `paperfolds'`, this makes $\mathrm{hd}(\texttt{paperfolds'})\colon A$ depend on `paperfolds'`$\colon \mathtt{Str}(A)$ itself, whereas `paperfolds` only occurs as the second argument of type $\blacktriangleright\mathtt{Str}(A)$, so that $\mathrm{hd}(\texttt{paperfolds}) = \mathrm{hd}(\texttt{toggle})$. $\square$

Besides streams, the *lifting monad*[1] (Paviotti et al., 2015) is another useful example of the application of the later endofunctor, which is the unique solution of

$$L(A) \cong A + \blacktriangleright L(A), \qquad L(A)(n) \cong 1 + A(0) + \cdots + A(n),$$

which for $A = \Delta X$ yields $L(A)(n) \cong \{0, \ldots, n\} \times X + 1$, generalizing `async/await` from JavaScript and Rust, by stating that X is either available now, later, or never. Here $(k, a) \in \{0, \ldots, n\} \times X$ means that a became available after k steps, and $* \in 1$ that no value is available yet.

3.3 Fixed points

Notice that in the definition of $\mathtt{Str}(A)$ we do not require A to be of the form $\Delta A'$ for some set A'. In fact, we may use any object $A \in \mathcal{S}$. Analogously to $\mathtt{Str}(A)$, for adequate F, we may unfold $X \cong F(\blacktriangleright X)$ stagewise: since $(\blacktriangleright X)(0) = 1$ and $(\blacktriangleright X)(n+1) = X(n)$, the right-hand side at stage $n+1$ only refers to $X(0), \ldots, X(n)$, and we can construct X step by step:

$$X(0) := F(\blacktriangleright X)(0), \qquad X(n+1) := F(\blacktriangleright X)(n+1).$$

[1] Also known as the *delay monad*.

"Adequate" here means that F must be a *locally contractive* functor, which is defined concretely in Birkedal et al. (2011, Def. II.10), which for our applications boils down to every type expression built from products, coproducts, exponentials and constants in which X occurs only under $\blacktriangleright$ being adequate, and hence having a unique fixed point up to isomorphism.

3.4 Global elements of the topos of trees

We saw an "element" of $\mathtt{Str}(A)$ in (5), as a sequence of approximations $x_n \in \mathtt{Str}(A)(n)$ compatible with the restriction maps, $r_n^{\mathtt{Str}(A)}(x_{n+1}) = x_n$ approaching an infinite stream in the limit. Objects $X \in \mathcal{S}$ are however not sets but functors, which have no notion of "element". Instead, we generalize from **Set**, where an element $x \in X$ is the same as a morphism $x\colon 1 \to X$, to the following definition.

Definition 3.9. *A* global element *of X is a morphism $x\colon 1 \to X$.*

This definition uses the terminal object 1. It and the initial object 0 are represented as

$$\begin{aligned} 1 &= \big(1 \longleftarrow 1 \longleftarrow 1 \longleftarrow \cdots\big) \\ 0 &= \big(0 \longleftarrow 0 \longleftarrow 0 \longleftarrow \cdots\big) \end{aligned} \tag{7}$$

where the components 1 and 0 are the terminal and initial objects of **Set**. In the topos of trees, a global element $x\colon 1 \to X$ is thus the same as a sequence of elements $x_n \in X(n)$ such that $r_n^X(x_{n+1}) = x_n$ for all $n \in \mathbb{N}$.

Proposition 3.10. *For a set X, the global elements of $\mathtt{Str}(\Delta X)$ form the set $X^{\mathbb{N}}$.*

Proof. A global element is a sequence $x_n \in \mathtt{Str}(\Delta X)(n) \cong X^{n+1}$ with $r_n^{\mathtt{Str}(\Delta X)}(x_{n+1}) = x_n$. Since the restriction maps drop the last entry, $[a_0, \ldots, a_{n+1}] \mapsto [a_0, \ldots, a_n]$, we have $x_n = [a_0, \ldots, a_n]$ for a unique sequence $(a_k)_{k\in\mathbb{N}} \in X^{\mathbb{N}}$, and conversely every such sequence defines a global element. □

Proposition 3.11. *For a set X, the global elements of $L(\Delta X)$ form the set $\mathbb{N} \times X + 1$. That is, a global element x is either*

- *available now: $x_n = (0, a)$ for all $n \in \mathbb{N}$,*
- *available later: $x_n = *$ for $n < k$ and $x_n = (k, a)$ for $n \geq k$, for some $k > 0$, or*
- *never available: $x_n = *$ for all $n \in \mathbb{N}$.*

Proof. Unfolding $L(\Delta X) \cong \Delta X + \blacktriangleright L(\Delta X)$ gives $L(\Delta X)(0) \cong X + 1$ and $L(\Delta X)(n+1) \cong X + L(\Delta X)(n)$, where the restriction map is the identity on X and $r_{n-1}^{L(\Delta X)}$ (respectively ! for $n = 0$) on the second summand. Writing $L(\Delta X)(n) \cong \{0, \ldots, n\} \times X + 1$ as above, this means

$$r_n^{L(\Delta X)}(k, a) = \begin{cases} (k, a) & \text{if } k \leq n, \\ * & \text{if } k = n+1, \end{cases} \qquad r_n^{L(\Delta X)}(*) = *.$$

Hence, if $x_m = (k, a)$ for some m, then $x_n = (k, a)$ for all $n \geq k$ and $x_n = *$ for all $n < k$, giving the element $(k, a) \in \mathbb{N} \times X$. Otherwise $x_n = *$ for all n, giving the element of 1. □

How many global elements does $\mathtt{Str}(A)$ (or more generally an object $X \in \mathcal{S}$) have? Notice how at index $n+1$, by $\mathtt{Str}(A)(n+1) \cong A \times \mathtt{Str}(A)(n)$ and the surjection $r_n^{\mathtt{Str}(A)} \colon \mathtt{Str}(A)(n+1) \to \mathtt{Str}(A)(n)$, each fiber $(r_n^{\mathtt{Str}(A)})^{-1}(x)$ over an element $x \in \mathtt{Str}(A)(n)$ has exactly $\mathrm{card}(A)$ elements. Indeed, given a global element evaluated up to index n, there are exactly $\mathrm{card}(A)$ many choices for continuing it to index $n+1$. This produces the structure of a tree (Figure 3), which we will make precise in the next section.

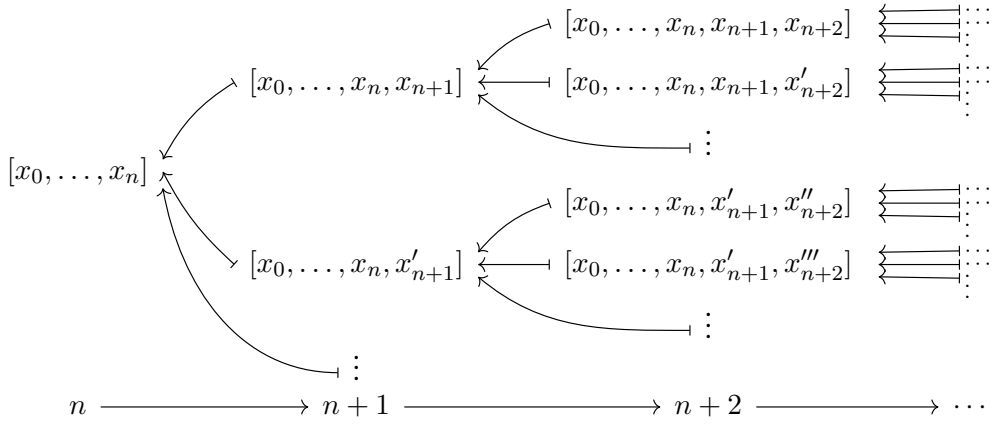
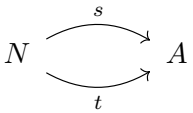


Figure 3: Elements of $\mathtt{Str}(A)$ over stages $n, n+1, n+2$, with $r_n^{\mathtt{Str}(A)}([x_0, \ldots, x_n, x_{n+1}]) = [x_0, \ldots, x_n]$.

3.5 The topos of graphs

As the name "topos of trees" implies, there is a strong connection to trees and their generalizations such as forests from graph theory. Explicitly, we may embed the topos of trees into the topos of graphs. This in turn will allow us to reason about the topos of trees using forests from graph theory instead of presheaves.

Definition 3.12. (*Vigna, 2003*) *The* topos of graphs *is the presheaf category* $\mathcal{G} = [\Gamma^{\mathrm{op}}, \mathbf{Set}]$, *where* Γ *is represented as*



An object $G \in \mathcal{G}$ is uniquely determined by $G(N)$, the set of nodes, $G(A)$, the set of arcs, and two functions $G(s), G(t) \colon G(A) \to G(N)$, which map an arc to its source and target node, respectively. In other words, G is a directed graph with nodes $G(N)$ and arcs $G(A)$.

A morphism $f\colon G \to H$ in $\mathcal{G}$ is a natural transformation between the presheaves G and H, or explicitly a pair of functions

$$f_N\colon G(N) \to H(N), \quad f_A\colon G(A) \to H(A)$$

such that the following diagrams commute:

$$\begin{array}{ccc} G(A) & \xrightarrow{G(s)} & G(N) \\ {\scriptstyle f_A}\downarrow & & \downarrow{\scriptstyle f_N} \\ H(A) & \xrightarrow[H(s)]{} & H(N) \end{array} \quad \text{and} \quad \begin{array}{ccc} G(A) & \xrightarrow{G(t)} & G(N) \\ {\scriptstyle f_A}\downarrow & & \downarrow{\scriptstyle f_N} \\ H(A) & \xrightarrow[H(t)]{} & H(N) \end{array}$$

Intuitively, this means that f maps nodes to nodes and preserves arcs, which is the standard definition of a graph homomorphism in graph theory.

Example 3.13. *Let G be the graph with $G(N) = \{a, b\}$, $G(A) = \{x\}$, and $G(s)(x) = a$, $G(t)(x) = b$, i.e. a single arc $a \to b$. Let H be the graph with $H(N) = \{c\}$, $H(A) = \{\ell\}$, and $H(s)(\ell) = H(t)(\ell) = c$, i.e. a single loop at c, as shown in Figures 4 and 5.*

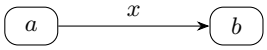


Figure 4: The graph G

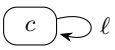


Figure 5: The graph H

Then $f\colon G \to H$ given by

$$f_N(a) = f_N(b) = c, \qquad f_A(x) = \ell$$

is a graph homomorphism, as both squares commute:

$$\begin{aligned} f_N(G(s)(x)) &= f_N(a) = c = H(s)(\ell) = H(s)(f_A(x)), \\ f_N(G(t)(x)) &= f_N(b) = c = H(t)(\ell) = H(t)(f_A(x)). \end{aligned}$$

3.6 The topos of trees as a category of forests

The topos of trees embeds into the topos of graphs via the following functor.

Definition 3.14. *The functor $F\colon \mathcal{S} \to \mathcal{G}$ is defined by*

$$F(X)(N) = \coprod_{i\in\omega} X(i), \quad F(X)(A) = \coprod_{i\in\omega} X(i+1) \tag{8}$$

$$F(X)(s) = \coprod_{i\in\omega} id_{X(i+1)}, \quad F(X)(t) = \coprod_{i\in\omega} r_i^X. \tag{9}$$

Intuitively $F(X)$ constructs a forest from X by taking the elements of $X(0)$ as roots, and for each $x_{n+1} \in X(n+1)$, adding an arc from x_{n+1} to its parent $x_n = r_n^X(x_{n+1}) \in X(n)$. A global element $x\colon 1 \to X$ is represented as an infinite path in the forest $F(X)$ ending at a root $x_0 \in X(0)$.

This is indeed an embedding, as F is injective on objects, since X is recovered from the levels of the coproduct $F(X)(N)$, and injective on morphisms, since a morphism $\phi\colon X \to Y$ in $\mathcal{S}$ is exactly a family $(\phi_i)_{i\in\omega}$ compatible with the restriction maps, which is recovered from $F(\phi) = \coprod_{i\in\omega} \phi_i$ as its restriction to the i-th summand.

Example 3.15. *Consider* $\mathtt{Str}(Bool) \cong Bool \times \blacktriangleright\mathtt{Str}(Bool)$*, the type of infinite streams of booleans. The roots of the forest* $F(\mathtt{Str}(Bool))$ *are the two booleans true and false, and there are two children for each node, corresponding to the two possible values of the next boolean in the stream. Figure 6 shows the first three levels of this forest, with the path corresponding to the stream* $x = (true, false, true, \dots)$*, given by the approximations* $x_0 = true$*,* $x_1 = (true, false)$*, and so on, drawn in bold.*

However, F is not *full*, as morphisms induced by it must preserve depth, while $\mathcal{G}$ contains morphisms that do not. We fix this in Section 3.7.

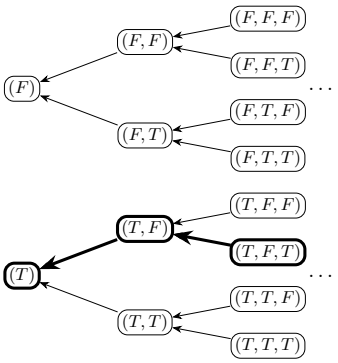
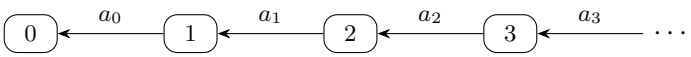


Figure 6: The forest $F(\mathtt{Str}(\mathrm{Bool}))$, with x in bold.

3.7 Graded graphs

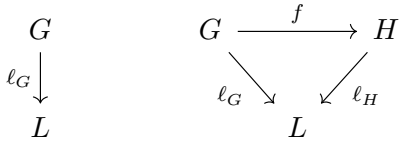
To make F full, we factor it through the slice $\mathcal{G}/L$ of *graded graphs* over the *level graph* L to keep track of depth information.

Definition 3.16 (The level graph)**.** *$L \in \mathcal{G}$ is the graph with nodes $L(N) = \mathbb{N}$ and arcs $L(A) = \{a_n \mid n \in \mathbb{N}\}$, where a_n has source $n+1$ and target n:*



It is the image $F(1)$ of the terminal object (7), and every $F(X)$ comes with a homomorphism to it (collapsing all nodes at level n to the node $n \in L(N)$).

Definition 3.17 (Graded graphs)**.** *A* graded graph *is an object of the slice category $\mathcal{G}/L$, i.e. a graph G together with a homomorphism ℓ_G to L, the* level *(left). A morphism of graded graphs is a* level-preserving *homomorphism f, i.e. one making the triangle commute (right):*



Since $F(1) = L$, sending X to $F(!)\colon F(X) \to L$ gives a full and faithful functor $\tilde{F}$ into the topos $\mathcal{G}/L$. Its essential image **RForest** consists of the graded graphs in which every node of positive level has exactly one outgoing arc. In particular, $\tilde{F}$ is full and faithful but not essentially surjective, as there exist $Y \in \mathcal{G}/L$ which are not isomorphic to any $F(X)$, for

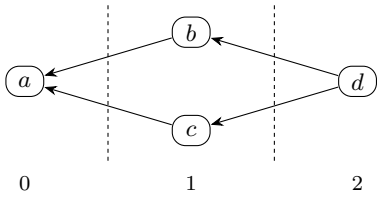
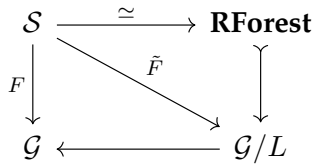


Figure 7: A graded graph that is not in **RForest**: the node d at level 2 has two outgoing arcs, so it is not isomorphic to any $\tilde{F}(X)$. Its level map sends $b \to a$ and $c \to a$ to a_0, and $d \to b$ and $d \to c$ to a_1.

example the graded graph in Figure 7, whose node d has two outgoing arcs, is an object of $\mathcal{G}/L$ but not of **RForest**.

By the resulting equivalence $\mathcal{S} \simeq$ **RForest**, we can regard the topos of trees as a full subcategory of a slice of the topos of graphs:



This allows us to regard morphisms in $\mathcal{S}$ as level-preserving forest morphisms, as shown in Figure 8.

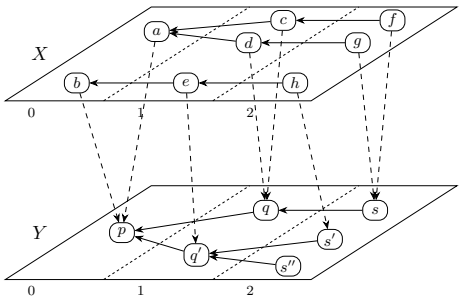


Figure 8: A morphism $X \to Y$ of **RForest** (dashed arrows): it sends nodes of level n to nodes of level n and parents to parents.

3.8 Limits, colimits and exponentials in the topos of trees

In Section 4.2, we will regard the objects of $\mathcal{S}$ as types in a type theory. An especially elegant way to construct complex data types is by combining simpler ones. *Product* and *sum* types built this way are also called *algebraic data types*. In $\mathcal{S}$, these as well as *sub-* and *quotient* types exist, and we will construct them using products, sums, equalizers and coequalizers.

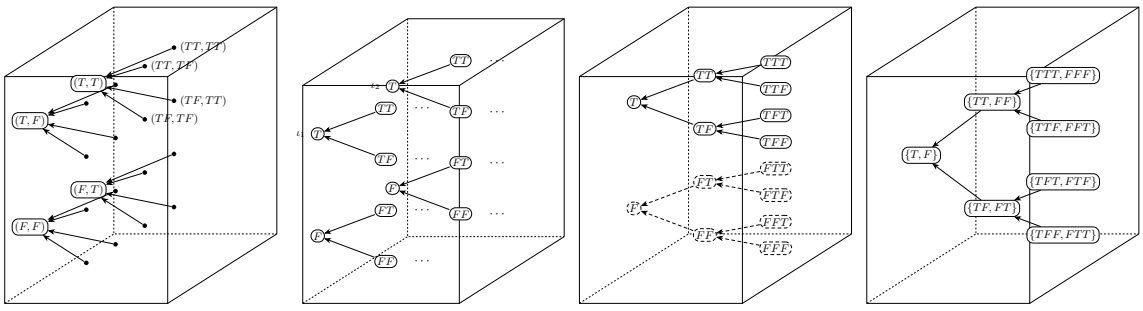
In presheaf categories, (co-)limits are computed pointwise (Mac Lane and Moerdijk, 1994, p. 22f.), which yields the following statement:

Lemma 3.18. *For $X, Y \in \mathcal{S}$ and every $n \in \mathbb{N}$:*

1. *products are given by* $(X \times Y)(n) = X(n) \times Y(n)$
2. *coproducts are given by* $(X \amalg Y)(n) = X(n) \amalg Y(n)$
3. *equalizers are given by* $\mathrm{Eq}(f, g)(n) = \mathrm{Eq}(f_n, g_n)$
4. *coequalizers are given by* $\mathrm{Coeq}(f, g)(n) = \mathrm{Coeq}(f_n, g_n)$.

On the level of forests (Figure 9), this means the following. Forests grow from left to right, a node at level n is labeled by its approximation x_n, abbreviating $(\text{true}, \text{false}, \text{true})$ as TFT, and arrows point from a node to its parent.

1. A node of $X \times Y$ at level n is a pair of nodes, one from each forest, and its children are the pairs of children. For $\mathtt{Str}(\mathrm{Bool}) \times \mathtt{Str}(\mathrm{Bool})$ this gives the four roots $(T, T), (T, F), (F, T), (F, F)$, each with four children, and 4^{n+1} nodes at level n (Figure 9a).
2. The forest of $X \amalg Y$ is the disjoint union of the two forests, placed side by side (Figure 9b). Using coproducts, we can for example construct $\mathrm{Bool} \cong 1 \amalg 1$.
3. The equalizer of $f, g \colon X \to Y$ consists of the nodes x_n on which f_n and g_n agree. By naturality the parent of such a node again lies in the equalizer, so $\mathrm{Eq}(f, g)$ is a sub-forest closed under taking parents. For $\mathrm{hd}, \mathrm{const}_{\mathrm{true}} \colon \mathtt{Str}(\mathrm{Bool}) \to \mathrm{Bool}$, with $\mathrm{const}_{\mathrm{true}}(-) = \mathrm{true}$, it is precisely the tree rooted at T (Figure 9c).
4. The coequalizer of $f, g \colon X \to Y$ identifies $f_n(x)$ with $g_n(x)$ level by level, so its nodes are equivalence classes of nodes. For $\mathrm{id}, \mathrm{not} \colon \mathtt{Str}(\mathrm{Bool}) \to \mathtt{Str}(\mathrm{Bool})$, where not negates every entry, each node is glued to its mirror image: the two trees of $F(\mathtt{Str}(\mathrm{Bool}))$ are folded onto a single one (Figure 9d).



(a) $F(X \times X)$ (b) $F(X \amalg X)$ (c) $F(\mathrm{Eq}(\mathrm{hd}, \mathrm{const}_{\mathrm{true}}))$ (d) $F(\mathrm{Coeq}(\mathrm{id}, \mathrm{not}))$

Figure 9: (Co-)limits of $\mathtt{Str}(\mathrm{Bool})$ as forests, with $X = \mathtt{Str}(\mathrm{Bool})$; in (c) the equalizer is drawn in bold, the rest of the forest dashed.

We can already see visually that $\blacktriangleright$ commutes with products and equalizers. More formally, due to the left adjoint $\blacktriangleleft \colon \mathcal{S} \to \mathcal{S}$, given by $(\blacktriangleleft X)(n) = X(n+1)$ and $r_n^{\blacktriangleleft X} = r_{n+1}^X$ (Birkedal et al., 2011, Sec. II), $\blacktriangleright$ must be a right adjoint, which gives us that it preserves all limits, in particular the terminal object, products and equalizers:

Lemma 3.19. $\blacktriangleright 1 \cong 1$, $\blacktriangleright(X \times Y) \cong \blacktriangleright X \times \blacktriangleright Y$ *and* $\blacktriangleright \mathrm{Eq}(f,g) \cong \mathrm{Eq}(\blacktriangleright f, \blacktriangleright g)$.

By virtue of being a topos, function types Y^X also exist in $\mathcal{S}$. By Yoneda (Mac Lane and Moerdijk, 1994, §I.6), we have $Y^X(n) \cong \mathrm{Nat}(\mathrm{Hom}_\omega(-,n) \times X, Y)$ (we elaborate on $\mathrm{Hom}_\omega(-,n)$ in Section 4.4), which gives us:

Lemma 3.20. *For $X, Y \in \mathcal{S}$ and every $n \in \mathbb{N}$,*

$$\begin{aligned} Y^X(n) &= \big\{(f_m \colon X(m) \to Y(m))_{m \le n} \;\big|\; r^Y_m \circ f_{m+1} = f_m \circ r^X_m \text{ for } m < n\big\} \\ &= \mathrm{Eq}\Big(\prod_{m \le n} Y(m)^{X(m)} \rightrightarrows \prod_{m<n} Y(m)^{X(m+1)}\Big), \end{aligned}$$

with $r^{Y^X}_n$ forgetting f_{n+1}.

In particular $Y^X(0) = Y(0)^{X(0)}$ and $\mathrm{Hom}_{\mathcal{S}}(1, Y^X) \cong \mathrm{Hom}_{\mathcal{S}}(X, Y)$. An element of $Y^X(n+1)$ is a ladder of commuting squares, and $r^{Y^X}_n$ removes its last (dashed) column:

$$\begin{array}{ccccccccc} X(0) & \xleftarrow{r^X_0} & X(1) & \xleftarrow{r^X_1} & \cdots & \xleftarrow{r^X_{n-1}} & X(n) & \xleftarrow{r^X_n} & X(n+1) \\ \downarrow{\scriptstyle f_0} & & \downarrow{\scriptstyle f_1} & & & & \downarrow{\scriptstyle f_n} & & \downarrow{\scriptstyle f_{n+1}} \\ Y(0) & \xleftarrow[r^Y_0]{} & Y(1) & \xleftarrow[r^Y_1]{} & \cdots & \xleftarrow[r^Y_{n-1}]{} & Y(n) & \xleftarrow[r^Y_n]{} & Y(n+1) \end{array}$$

Using (4), we see that this is an $\mathcal{S}$-morphism from $\mathrm{Hom}_\omega(-,n+1) \times X$ to $\mathrm{Hom}_\omega(-,n+1) \times Y$, i.e. from X to Y cut off above stage $n+1$. Using the equivalence $\mathcal{S} \simeq$ **RForest**, an element of $Y^X(n+1)$ thus corresponds to a graded forest morphism from $F(X)$ cut to the levels 0 to $n+1$ to $F(Y)$ cut to the levels 0 to $n+1$. The restriction $r^{Y^X}_n$ is then obtained by cutting off level $n+1$ of both forests (Figure 10).

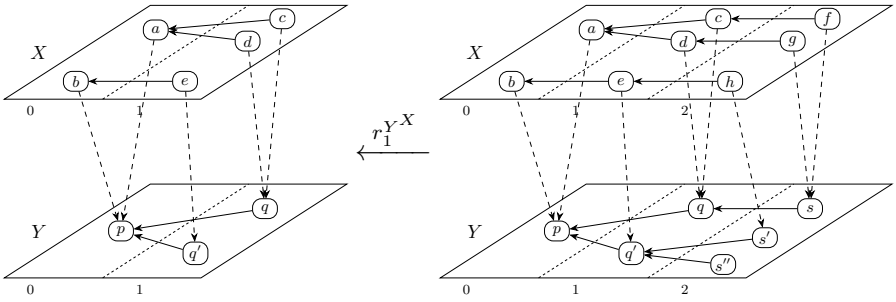


Figure 10: Restricting the morphism of Figure 8 (right), an element of $Y^X(2)$, to $Y^X(1)$ cuts off level 2 of both forests (left).

We can again visually see that $\blacktriangleright$ should commute, in this case with exponentials. This is in fact the case:

Lemma 3.21. $\blacktriangleright(Y^X) \cong (\blacktriangleright Y)^{\blacktriangleright X}$.

Proof. We compare stages using Lemma 3.20. For $n = 0$: $\blacktriangleright(Y^X)(0) = 1 = 1^1 = (\blacktriangleright Y)(0)^{(\blacktriangleright X)(0)} = (\blacktriangleright Y)^{\blacktriangleright X}(0)$. For $n+1$: the component $f_0 \colon 1 \to 1$ is unique, and $f_{m+1} \colon (\blacktriangleright X)(m+1) \to (\blacktriangleright Y)(m+1)$ is a map $X(m) \to Y(m)$, so $(\blacktriangleright Y)^{\blacktriangleright X}(n+1) \cong Y^X(n) = \blacktriangleright(Y^X)(n+1)$. Both restrictions forget the last component, so the isomorphisms are natural. □

3.9 Further common morphisms in the topos of trees

A common operation on streams is given by the unfold anamorphism.[2]

Definition 3.22. *For constant A (so $A = \Delta X$ for some set X), there exists the morphism*

$$\begin{gathered}\mathrm{unfold}_A \colon A \times A^A \to \mathtt{Str}(A),\\ (\mathrm{unfold}_A)_0(a,f) = [a], \qquad (\mathrm{unfold}_A)_k(a,f) = [a, f(a), f(f(a)), \dots, f^k(a)].\end{gathered}$$

Proof. Since $A = \Delta X$ is constant, an element of $A^A(k)$, i.e. a natural transformation $\mathrm{Hom}_\omega(-,k) \times A \to A$, has equal components at all stages, and hence amounts to a single function $f \colon X \to X$. Thus $(A \times A^A)(k) \cong X \times \mathrm{Hom}_{\mathbf{Set}}(X,X)$ with identities as restriction maps, and we need to show that unfold_A is a natural transformation, i.e. that the following diagram commutes for all $k \in \mathbb{N}$:

$$\begin{array}{ccc}(A \times A^A)(k+1) & \xrightarrow{(\mathrm{unfold}_A)_{k+1}} & \mathtt{Str}(A)(k+1)\\ {\scriptstyle r_k^{A\times A^A}}\downarrow & & \downarrow{\scriptstyle r_k^{\mathtt{Str}(A)}}\\ (A \times A^A)(k) & \xrightarrow[(\mathrm{unfold}_A)_k]{} & \mathtt{Str}(A)(k)\end{array} \tag{10}$$

This follows by

$$\begin{array}{ccc}(a,f) & \longmapsto & [a, f(a), \dots, f^{k+1}(a)]\\ \downarrow & & \downarrow\\ (a,f) & \longmapsto & [a, f(a), \dots, f^k(a)]\end{array}$$

□

Notice however, that for non-constant A this morphism need not be defined, as otherwise the commutativity of (10) would require

$$\begin{array}{ccc}(a_{k+1}, f_{k+1}) & \longmapsto & [a_0, f_1(a_1), f_2(f_1(a_2)), \dots, f_{k+1}(f_k(a_{k+1}))]\\ \downarrow & & \downarrow\\ (a_k, f_k) & \longmapsto & [a_0, f_1(a_1), f_2(f_1(a_2)), \dots, f_k(f_{k-1}(a_k))]\end{array}$$

to commute, but there is no way to retrieve a_m from a_{k+1} in general for $m \leq k$.

We can now use this unfold to get constant streams:

Definition 3.23. *We can define the constant stream for constant objects A by*

$$\mathrm{const}_{\mathtt{Str}(A)}(a) := \mathrm{unfold}_A(a, id) = [a, a, a, \dots].$$

[2] For anamorphisms see Meijer et al. (1991).

4 Categorical logic

So far, we have seen how we can construct products, coproducts, equalizers and coequalizers in $\mathcal{S}$, akin to algebraic data types from programming. We have also previously seen that we can consider a generalized notion of "elements" of an object $X \in \mathcal{S}$. In set theory, we can glob all elements of $Y \in \mathbf{Set}$ which satisfy a formula φ with one free variable y into a subset $\{y \in Y \mid \varphi(y)\} \subseteq Y$. Can we do the same for objects in $\mathcal{S}$, akin to refinement types in programming, for example "$\{x \in \mathtt{Str}(\mathbb{N}) \mid x \text{ starts with } 0\}$" for all streams of natural numbers starting with 0? The problem we face here is that objects of $\mathcal{S}$ are functors and as such do not have a notion "subset". They do however have *subobjects*! The syntax and semantics in which subobjects such as

$$[\![x : X \mid \varphi(x)]\!] \rightarrowtail X$$

are defined are specified in the field of *categorical logic*, and each topos comes with a first-order logic, given by its *internal logic*. In fact, for a topos, one may also consider its *Mitchell–Bénabou language*, which even yields an entire (intuitionistic, higher-order) typed λ-calculus (Mac Lane and Moerdijk, 1994, Sec. VI.5), effectively letting us *program* in a typed λ-calculus in $\mathcal{S}$ (see Section 4.7), but we will limit ourselves to first-order logic for now.

We follow the conventions as laid out in (Mac Lane and Moerdijk, 1994, Sec. VI.5, VI.6). The reader already familiar with categorical logic may skip Sections 4.1 to 4.3 and 4.7, as only Sections 4.4 to 4.6 contain information specific to the topos of trees.

4.1 Formal languages

So far, we have used types and sets interchangeably, as seen in $\mathtt{Str}(A)$, which is meant to be a type, but is a presheaf, i.e. an ω-indexed family of sets. To define subobjects $[\![x : X \mid \varphi(x)]\!] \rightarrowtail X$, and with them refinement types, we first need to make precise what types, terms and formulae are, and what their relationship to objects, (global) elements and subobjects is. In categorical logic as in model theory, we fix a *formal language* Σ and define the terms and formulae over Σ recursively.

Definition 4.1 (Johnstone, 2002b, p. D1.1.1)**.** *A* first-order language *or* (first-order) signature Σ *consists of the following data.*

(*a*) *A set* Σ-Type *of* types*, also called* sorts*.*

(*b*) *A set* Σ-Fun *of* function symbols*, together with a map assigning to each* $f \in \Sigma$-Fun *its* type*, which consists of a finite non-empty list of types* (*with the last type in the list enjoying a distinguished status*)*: we write*

$$f \colon A_1 \times \cdots \times A_n \to B$$

to indicate that f *has type* $A_1, \ldots, A_n, B$*. The number* n *is called the* arity *of* f*; in the case* $n = 0$*,* f *is called a* constant *of type* B*.*

(*c*) *A set* Σ-Rel *of* relation symbols*, together with a map assigning to each* $R \in \Sigma$-Rel *its* type*, which consists of a finite list of types: we write*

$$R \rightarrowtail A_1 \times \cdots \times A_n$$

to indicate that R *has type* $A_1, \ldots, A_n$*. The number* n *is called the* arity *of* R*; in the case* $n = 0$*,* R *is called an* atomic proposition*.*

We assume that each type A has a countably infinite collection V_A of *variables* of type A.

Definition 4.2 (Johnstone, 2002b, p. D1.1.2)**.** *The collection* Σ-Term *of* terms *over* Σ *is defined recursively by the clauses below; simultaneously, we define the* type *of each term and write* $t : A$ *to denote that* t *is a term of type* A*.*

(*a*) $x : A$*, if* $x \in V_A$ *is a variable of type* A*;*

(*b*) $f(t_1, \ldots, t_n) : B$*, if* $f \colon A_1 \times \cdots \times A_n \to B$ *is a function symbol and* $t_1 : A_1, \ldots, t_n : A_n$*.*

Given f *a constant, we will write* f *instead of* $f()$ *as is customary. Along the same recursion, the set* $\mathrm{FV}(t)$ *of* free variables *of* t *is given by* $\mathrm{FV}(x) := \{x\}$ *and* $\mathrm{FV}(f(t_1, \ldots, t_n)) := \bigcup_{i=1}^{n} \mathrm{FV}(t_i)$*.*

A *formula* is a logical expression denoting a predicate on the inhabitants of the types, which will be assigned a "truth value".

Definition 4.3 (Johnstone, 2002b, p. D1.1.3)**.** *The collection* Σ-Form *of* first-order formulae *over* Σ *is defined recursively by the following clauses, together with, for each formula* φ*, the finite set* $\mathrm{FV}(\varphi)$ *of its* free variables*.*

(*i*) Relations*:* $R(t_1, \ldots, t_n)$*, if* $R \rightarrowtail A_1 \times \cdots \times A_n$ *is a relation symbol and* $t_1 : A_1, \ldots, t_n : A_n$ *are terms;* $\mathrm{FV}(R(t_1, \ldots, t_n)) := \bigcup_{i=1}^{n} \mathrm{FV}(t_i)$*. If* R *has arity* 0*, we write* R *instead of* $R()$*.*

(*ii*) Equality*:* $(s = t)$*, if* s *and* t *are terms of the same type;* $\mathrm{FV}(s = t) := \mathrm{FV}(s) \cup \mathrm{FV}(t)$*.*

(*iii*) Truth *and* falsity*:* $\top$ *and* $\bot$*;* $\mathrm{FV}(\top) := \mathrm{FV}(\bot) := \emptyset$*.*

(*iv*) Binary connectives*:* $(\varphi \star \psi)$ *for* $\star \in \{\wedge, \vee, \Rightarrow\}$*, if* φ *and* ψ *are formulae;* $\mathrm{FV}(\varphi \star \psi) := \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$*.*

(*v*) Negation*:* $\neg\varphi$*, if* φ *is a formula;* $\mathrm{FV}(\neg\varphi) := \mathrm{FV}(\varphi)$*.*

(*vi*) Quantification*:* $(\exists x : A)\varphi$ *and* $(\forall x : A)\varphi$*, if* φ *is a formula and* x *a variable of type* A*;* $\mathrm{FV}((\exists x : A)\varphi) := \mathrm{FV}((\forall x : A)\varphi) := \mathrm{FV}(\varphi) \setminus \{x\}$*.*

Given two formulae φ *and* ψ *we will regard them as* α*-equivalent if they differ only in the names of their bound variables, and we omit brackets where no ambiguity arises. A formula* φ *with* $\mathrm{FV}(\varphi) = \emptyset$ *is called* closed*, or a* sentence*.*

Example 4.4 (Mac Lane and Moerdijk, 1994, Ch. VI)**.** *Ordinary mathematical statements can be formulated in first-order logic. The language of real numbers has a single type,* $\Sigma\text{-Type} = \{R\}$*, constants* $0, 1 \colon \to R$ *and function symbols* $+, \cdot \colon R \times R \to R$*, and relation symbols* $<, \geq \rightarrowtail R \times R$*. Using infix notation,* $x \cdot x$ *and* $x + y$ *are terms of type* R*,* $x < y$ *and* $x + y = z$ *are formulae, and*

$$(\forall x : R)\big((\exists y : R)(x < y) \wedge (x < 0 \vee x \geq 0)\big)$$

is a sentence.

Definition 4.5 (Johnstone, 2002b, p. D1.1.4)**.** *A* context *is a finite list $\vec{x} = x_1 : X_1, \ldots, x_n : X_n$ of distinct variables. The empty list is the* empty context $[]$*. The* type *of $\vec{x}$ is $X_1 \times \cdots \times X_n$.*

A context $\vec{x}$ is *suitable* for a formula φ if $\mathrm{FV}(\varphi) \subseteq \{x_1, \ldots, x_n\}$, and a *formula-in-context* $\vec{x}.\varphi$ is a formula φ together with a suitable context $\vec{x}$ (also written $\vec{x} \mid \varphi$, or $\Gamma \mid \varphi$ with $\Gamma = \vec{x}$). The *canonical context* of φ consists of the distinct free variables of φ, listed in the order of their first appearance. Similarly, a *term-in-context* $\vec{x}.t$ is a term t together with a context $\vec{x}$ containing all variables of t.

4.2 Categorical semantics

Given a formula φ in context $\vec{x} = x : X$ over a signature Σ, how can we *interpret* $[\![x : X \mid \varphi(x)]\!]$ as a subobject of X? Even more so, how can we interpret general formulae over the signature $\Sigma_{\mathcal{S}}$? *Categorical semantics* provides a way to give meaning to formulae and terms by interpreting the types, function symbols and relation symbols of Σ in a category $\mathcal{C}$ with finite products.

We follow Johnstone (2002b, Sec. D1.2), and first define how a signature Σ is interpreted in a category $\mathcal{C}$ with sufficient structure. In the following, let $\mathcal{C}$ be a category with finite products and Σ a signature.

Definition 4.6 (Johnstone, 2002b, p. D1.2.1)**.** *A Σ-*structure *M in $\mathcal{C}$ is specified by the following data.*

(*i*) *A function assigning to each type $A \in \Sigma$-*Type *an object MA of $\mathcal{C}$. It is extended to finite strings of types by $M(A_1, \ldots, A_n) := MA_1 \times \cdots \times MA_n$, where the empty string $[]$ is sent to the terminal object 1 of $\mathcal{C}$.*

(*ii*) *A function assigning to each function symbol $f : A_1 \times \cdots \times A_n \to B$ in Σ-*Fun *a morphism $Mf : M(A_1, \ldots, A_n) \to MB$ in $\mathcal{C}$.*

(*iii*) *A function assigning to each relation symbol $R \rightarrowtail A_1 \times \cdots \times A_n$ in Σ-*Rel *a subobject $MR \rightarrowtail M(A_1, \ldots, A_n)$ in $\mathcal{C}$.*

In particular, the *internal language* of $\mathcal{C}$ is the signature $\Sigma_{\mathcal{C}}$ with types (interpreted as) the objects of $\mathcal{C}$, function symbols (interpreted as) the morphisms of $\mathcal{C}$, and relation symbols (interpreted as) the subobjects of $\mathcal{C}$, which admits a *tautological* $\Sigma_{\mathcal{C}}$-structure interpreting every type and every function symbol by itself.

Example 4.7. *In the internal language of* **Set***, types are sets A, and a constant $c : A$, i.e. a function symbol of arity 0, is interpreted as a function $1 \to A$, that is, an element $c \in A$. Relation symbols $R \rightarrowtail A_1 \times \cdots \times A_n$ are subsets $R \subseteq A_1 \times \cdots \times A_n$.*

Example 4.8. *In the internal language of* **Top***, types are topological spaces, function symbols are continuous maps, and a constant $c : A$ is a continuous map $1 \to A$ from the one-point space, i.e. a point of A. Relation symbols are monomorphisms in* **Top***, i.e. injective continuous maps $R \rightarrowtail A_1 \times \cdots \times A_n$, which need not carry the subspace topology.*

More generally, by recursion, we can also interpret terms in the following way:

Definition 4.9 (Johnstone, 2002b, p. D1.2.3)**.** *Let $\vec{x}.t$ be a term in the context $\vec{x} = x_1 : A_1, \ldots, x_n : A_n$ and $t : B$. The morphism*

$$[\![\vec{x}.t]\!]_M \colon M(A_1, \ldots, A_n) \to MB$$

is defined recursively by the following clauses.

(*a*) *If t is a variable, it is x_i for a unique $i \leq n$, and $[\![\vec{x}.t]\!]_M := \pi_i$, the i-th product projection.*

(*b*) *If t is $f(t_1, \ldots, t_m)$ with $t_j : C_j$, then $[\![\vec{x}.t]\!]_M$ is the composite*

$$M(A_1, \ldots, A_n) \xrightarrow{([\![\vec{x}.t_1]\!]_M, \ldots, [\![\vec{x}.t_m]\!]_M)} M(C_1, \ldots, C_m) \xrightarrow{Mf} MB.$$

We omit the subscript M when the structure is clear from the context.

Formulae need more structure on $\mathcal{C}$ than finite products. Relations only need finite limits, disjunction needs $\mathcal{C}$ to be *coherent*, and implication and quantification need it to be *Heyting*. For simplicity, we assume that $\mathcal{C}$ is a topos, which covers all of these. The following definition uses Heyting algebras, which we introduce in Section 4.3 right after it.

Definition 4.10 (Johnstone, 2002b, p. D1.2.6)**.** *Let M be a Σ-structure in a topos $\mathcal{C}$ and let $\vec{x}.\varphi$ be a formula-in-context with $x_i : A_i$. The subobject*

$$[\![\vec{x}.\varphi]\!]_M \rightarrowtail M(A_1, \ldots, A_n)$$

is defined recursively by the following clauses.

(*i*) *If φ is $R(t_1, \ldots, t_m)$ with $R \rightarrowtail B_1 \times \cdots \times B_m$, then $[\![\vec{x}.\varphi]\!]$ is the pullback*

$$\begin{array}{ccc} [\![\vec{x}.\varphi]\!] & \longrightarrow & MR \\ \downarrow & \lrcorner & \downarrow \\ M(A_1, \ldots, A_n) & \xrightarrow[([\![\vec{x}.t_1]\!], \ldots, [\![\vec{x}.t_m]\!])]{} & M(B_1, \ldots, B_m). \end{array}$$

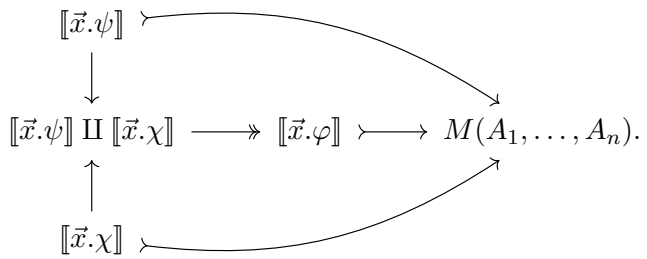
(*ii*) *If φ is $(s = t)$, then $[\![\vec{x}.\varphi]\!]$ is the equalizer of $[\![\vec{x}.s]\!]$ and $[\![\vec{x}.t]\!]$.*

(*iii*) *If φ is $\top$ or $\bot$, then $[\![\vec{x}.\varphi]\!]$ is the top or bottom element of $\mathrm{Sub}_{\mathcal{C}}(M(A_1, \ldots, A_n))$.*

(*iv*) *If φ is $(\psi \wedge \chi)$, then $[\![\vec{x}.\varphi]\!]$ is the intersection (pullback)*

$$\begin{array}{ccc} [\![\vec{x}.\varphi]\!] & \rightarrowtail & [\![\vec{x}.\psi]\!] \\ \downarrow & \lrcorner & \downarrow \\ [\![\vec{x}.\chi]\!] & \rightarrowtail & M(A_1, \ldots, A_n). \end{array}$$

(*v*) *If φ is $(\psi \vee \chi)$, then $[\![\vec{x}.\varphi]\!]$ is the union of $[\![\vec{x}.\psi]\!]$ and $[\![\vec{x}.\chi]\!]$, i.e. the image of the induced map out of their coproduct:*



(*vi*) *If φ is $(\psi \Rightarrow \chi)$ or $\neg\psi$, then $[\![\vec{x}.\varphi]\!]$ is the implication or negation of $[\![\vec{x}.\psi]\!]$ and $[\![\vec{x}.\chi]\!]$ in the Heyting algebra $\mathrm{Sub}_{\mathcal{C}}(M(A_1, \ldots, A_n))$. (Notably $\neg\varphi$ is equivalent to $\varphi \Rightarrow \bot$.)*

(*vii*) *If φ is $(\exists y : B)\psi$, then $[\![\vec{x}.\varphi]\!]$ is the image of the composite*

$$[\![\vec{x}, y.\psi]\!] \rightarrowtail M(A_1, \ldots, A_n, B) \xrightarrow{\pi} M(A_1, \ldots, A_n),$$

where π is the projection onto the first n factors.

(*viii*) *If φ is $(\forall y : B)\psi$, then $[\![\vec{x}.\varphi]\!] := \forall_\pi([\![\vec{x}, y.\psi]\!])$, where π is the projection as in (vii) and $\forall_\pi$ is the right adjoint of pulling back along π.*

4.3 Heyting algebras and adjunctions

Definition 4.10 uses Heyting algebras for implication and negation ((vi)) and adjoints of pullback for the quantifiers ((vii), (viii)).

Definition 4.11 (Heyting algebra)**.** *A* Heyting algebra *is a poset $(H, \leq)$ with finite meets and joins (i.e. a lattice) such that for all $a, b \in H$ there is a greatest element $c \in H$ such that $a \wedge c \leq b$. This element is denoted by $a \Rightarrow b$.*

Example 4.12. *The following are Heyting algebras:*

- $\mathrm{Sub}_{\mathbf{Set}}(1) = \{\bot, \top\}$,
- *every finite distributive lattice,*
- *every Boolean algebra,*
- *the open sets $\mathcal{O}(T)$ of a topological space T, ordered by inclusion, with $U \Rightarrow V$ the interior of $(T \setminus U) \cup V$,*
- *the open elements of an interior algebra.*

If H is totally ordered, then $a \wedge c = \min(a, c)$, which yields the following statement:

Lemma 4.13. *For all a, b in a totally ordered Heyting algebra H,*

$$a \Rightarrow b = \max\{c \in H \mid \min(a, c) \leq b\} = \begin{cases} \top & \text{if } a \leq b, \\ b & \text{if } a > b. \end{cases}$$

Proof. If $a \leq b$, then $\min(a, c) \leq a \leq b$ for all c. If $a > b$, then $\min(a, c) \leq b \iff c \leq b$. □

In a topos $\mathcal{E}$ (and more generally in a *Heyting category* (Johnstone, 2002a, A1.4, p. 39)), every subobject poset $\mathrm{Sub}_{\mathcal{E}}(X)$ is a Heyting algebra. Taking $\mathrm{Sub}_{\mathcal{E}}(1) \cong \mathrm{Hom}_{\mathcal{E}}(1, \Omega)$ (Lemma 2.5) as the truth values and its Heyting operations as the logical connectives, we interpret intuitionistic logic in $\mathcal{E}$.

This settles the implication and negation clauses. For the projection $\pi\colon [\![B]\!] \times [\![A]\!] \to [\![B]\!]$, the quantifiers are the two adjoints of pulling back along π:

$$\begin{array}{ccc} & \mathrm{Sub}_{\mathcal{E}}([\![B]\!] \times [\![A]\!]) & \\ \exists_\pi \circlearrowleft & \dashv\ \pi^{-1}\ \dashv & \circlearrowright \forall_\pi \\ & \mathrm{Sub}_{\mathcal{E}}([\![B]\!]) & \end{array} \tag{11}$$

This means that, for all $U \in \mathrm{Sub}_{\mathcal{E}}([\![B]\!] \times [\![A]\!])$ and $V \in \mathrm{Sub}_{\mathcal{E}}([\![B]\!])$,

$$\begin{aligned} \mathrm{Hom}_{\mathrm{Sub}_{\mathcal{E}}([\![B]\!])}(\exists_\pi U, V) &\cong \mathrm{Hom}_{\mathrm{Sub}_{\mathcal{E}}([\![B]\!]\times[\![A]\!])}(U, \pi^{-1}V) \quad \text{and} \\ \mathrm{Hom}_{\mathrm{Sub}_{\mathcal{E}}([\![B]\!]\times[\![A]\!])}(\pi^{-1}V, U) &\cong \mathrm{Hom}_{\mathrm{Sub}_{\mathcal{E}}([\![B]\!])}(V, \forall_\pi U). \end{aligned}$$

Both are posets, and in a poset P the set $\mathrm{Hom}_P(x, y)$ is inhabited iff $x \leq y$. Hence $\exists_\pi U \leq V \iff U \leq \pi^{-1}V$ and $\pi^{-1}V \leq U \iff V \leq \forall_\pi U$, which yields the following statement:

Lemma 4.14. *For all $U \in \mathrm{Sub}_{\mathcal{E}}([\![B]\!] \times [\![A]\!])$,*

$$\exists_\pi U = \min\{V \in \mathrm{Sub}_{\mathcal{E}}([\![B]\!]) \mid U \leq \pi^{-1}V\} \quad \textit{and} \quad \forall_\pi U = \max\{V \in \mathrm{Sub}_{\mathcal{E}}([\![B]\!]) \mid \pi^{-1}V \leq U\}.$$

In **Set** this gives

$$\exists_\pi U = \{b \in [\![B]\!] \mid \exists a \in [\![A]\!].\ (b, a) \in U\} \quad \text{and} \quad \forall_\pi U = \{b \in [\![B]\!] \mid \forall a \in [\![A]\!].\ (b, a) \in U\}.$$

For example, for $[\![A]\!] = [\![B]\!] = \mathbb{N}$ and $U = \{(b, a) \mid b \leq a\}$ we get $\exists_\pi U = \mathbb{N}$ and $\forall_\pi U = \{0\}$.

4.4 The truth values of $\mathcal{S}$

We are particularly interested in defining formulae over a signature for the topos of trees, and we will therefore consider the internal language $\Sigma_{\mathcal{S}}$ of $\mathcal{S}$. As seen in Section 4.2, a formula φ can be interpreted in two ways: in a context $x : X$ as the subobject $[\![x.\varphi]\!] \rightarrowtail X$ of all "elements" of X satisfying φ, or, if φ is a sentence, as a truth value $[\![\varphi]\!] \rightarrowtail 1$. In this section we are interested in the latter.

For instance, assume there is a sentence φ expressing that the forest $F(X)$ of an object X is a binary tree. What would it mean for φ to have a truth value besides $\top$ or $\bot$? What truth values does $\mathcal{S}$ have at all? By Lemma 2.5, truth values $[\![\varphi]\!] \rightarrowtail 1$ correspond to global elements $1 \to \Omega$, so we investigate Ω. Since Ω is determined by the sets $\Omega(n)$

and the restriction maps r_n^Ω, we start with $\Omega(n)$. By the contravariant Yoneda lemma $\mathrm{Nat}(\mathrm{Hom}_\mathcal{C}(-,a),F) \cong F(a)$, we have

$$\begin{aligned}
\Omega(n) &\cong \mathrm{Nat}(\mathrm{Hom}_\omega(-,n),\Omega) && \mid \text{Yoneda} && (12)\\
&= \mathrm{Hom}_\mathcal{S}(\mathrm{Hom}_\omega(-,n),\Omega) && \mid \mathit{Definition}\ 3.1 && (13)\\
&\cong \mathrm{Sub}_\mathcal{S}(\mathrm{Hom}_\omega(-,n)) && \mid \mathit{Lemma}\ 2.5 && (14)\\
&\cong \{u\colon U \rightarrowtail \mathrm{Hom}_\omega(-,n)\}/_{\cong}. && \mid \mathit{Definition}\ 2.3 && (15)
\end{aligned}$$

$\mathrm{Hom}_\omega(i,n)$ has at most one element by

$$\mathrm{Hom}_\omega(i,n) = \begin{cases} 1 & 0 \le i \le n \\ 0 & i > n, \end{cases}$$

and u is defined componentwise by maps $u_i\colon U(i) \to \mathrm{Hom}_\omega(i,n)$. These can, however, only be monomorphisms if $U(i)$ has at most one element. So u is determined up to isomorphism by the choices of $U(i)$ between 0 (the empty set, i.e. the initial object of **Set**) and 1 (the set with one element, i.e. the terminal object of **Set**) for $0 \le i \le n$, with $U(i) = 0$ for $i > n$. By the restriction maps $r_i^U : U(i+1) \to U(i)$ however, $U(i+1) = 1$ forces $U(i) = 1$, as there is no map $1 \to 0$, and thus every element of $\Omega(n)$ is represented by some

$$U_m(i) = \begin{cases} 1 & 0 \le i < m \\ 0 & i \ge m, \end{cases}$$

for $m \in [0..n+1]$. These U_m correspond to the downward-closed subsets of $[0..n]$, and under F are represented by the following forests:

$F(U_0)$	
$F(U_1)$	$\boxed{0}$
$F(U_2)$	$\boxed{0} \leftarrow \boxed{1}$
$\vdots$	
$F(U_n)$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1}$
$F(U_{n+1})$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1} \leftarrow \boxed{n}$

Hence we get:

Lemma 4.15. *For all $n \in \mathbb{N}$, we have bijections (i.e. isomorphisms in* **Set***)*

$$\begin{aligned}
\Omega(n) &\cong \{U_0, \ldots, U_{n+1}\} \\
&\cong \{[0..(m-1)] \mid 0 \le m \le n+1\} \\
&\cong [0..(n+1)],
\end{aligned}$$

where $[0..(-1)] := \emptyset$.

Lastly, we need to determine the restriction maps $r_n^\Omega\colon \Omega(n+1) \to \Omega(n)$, which by Definition 3.1 are given by $\Omega(s)$ for the morphism $s\colon n \to n+1$ in ω. By Definition 2.4, r_n^Ω

is given by pulling back along s. Specifically, as in Equation (12) and Equation (14), we have

$$r_n^\Omega \colon \{U \rightarrowtail \mathrm{Hom}_\omega(-,n+1)\}/\cong \to \{V \rightarrowtail \mathrm{Hom}_\omega(-,n)\}/\cong,$$
$$U_m \mapsto s^* U_m$$

with

$$\begin{aligned}(s^*U_m)(i) &= U_m(i) \times_{\mathrm{Hom}_\omega(i,n+1)} \mathrm{Hom}_\omega(i,n) \\ &= \begin{cases} 1 & 0 \leq i < \min(m, n+1) \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

So all U_m with $m \leq n+1$ are left invariant, and only U_{n+2} is mapped to U_{n+1}.

From now on, we identify $U_m \in \Omega(n)$ with $m \in [0..(n+1)]$, i.e. $\Omega(n) = [0..(n+1)]$. Unfortunately, 1 then denotes both U_1 and the terminal object of $\mathcal{S}$, whose truth value $\top \colon 1 \to \Omega$ corresponds to ω (see Proposition 4.18). The meaning is usually clear from context, and in ambiguous contexts we write U_m instead. This gives us:

Lemma 4.16. *For all $n \in \mathbb{N}$ and $m \in [0..(n+2)]$, the restriction map $r_n^\Omega \colon \Omega(n+1) \to \Omega(n)$ is given by*

$$r_n^\Omega(m) = \min(m, n+1).$$

Example 4.17. *Under F, r_n^Ω cuts off the nodes at level $n+1$, so only the longest chain $F(U_{n+2})$ changes:*

$\Omega(n)$			$\Omega(n+1)$	
$F(r_n^\Omega U_0)$			$F(U_0)$	
$F(r_n^\Omega U_1)$	$\boxed{0}$		$F(U_1)$	$\boxed{0}$
$F(r_n^\Omega U_2)$	$\boxed{0} \leftarrow \boxed{1}$	$\xleftarrow{r_n^\Omega}$	$F(U_2)$	$\boxed{0} \leftarrow \boxed{1}$
$\vdots$			$\vdots$	
$F(r_n^\Omega U_n)$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1}$		$F(U_n)$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1}$
$F(r_n^\Omega U_{n+1})$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1} \leftarrow \boxed{n}$		$F(U_{n+1})$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1} \leftarrow \boxed{n}$
$F(r_n^\Omega U_{n+2})$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1} \leftarrow \boxed{n}$		$F(U_{n+2})$	$\boxed{0} \leftarrow \boxed{1} \leftarrow \cdots \leftarrow \boxed{n-1} \leftarrow \boxed{n} \leftarrow \boxed{n+1}$

A truth value $1 \to \Omega$ is a family $(x_i \in \Omega(i))_{i \in \mathbb{N}}$ with $x_i = r_i^\Omega(x_{i+1}) = \min(x_{i+1}, i+1)$ by Lemma 4.16. There are two possible sections $\sigma_{i,i+1}$ and $\sigma_{i,i+2}$ of r_i^Ω, uniquely defined by $\sigma_{i,k}(i+1) = k$, yielding two possible choices $x_{i+1} \in \{i+1, i+2\}$ for $x_i = i+1$, but no choice for $x_i \leq i$, where $x_{i+1} = x_i$.

As such, a truth value is uniquely determined by the least $n \in \mathbb{N}$ with $x_n = n$ (if it exists), i.e. the depth of the bifurcation point of $F(\Omega)$ (see Figure 11) after which x_n stops growing. If there is no such n, i.e. $x_i = i+1$ for all $i \in \mathbb{N}$, we write $n = \omega$. This yields by Lemma 2.5 the following bijection:

Proposition 4.18. *The truth values of $\mathcal{S}$ satisfy the bijection* $\mathrm{Sub}_\mathcal{S}(1) \cong \overline{\mathbb{N}}$, *where* $\overline{\mathbb{N}} := \mathbb{N} \cup \{\omega\}$.

Another way to see this is that $\mathrm{Sub}_{\mathcal{S}}(1)$ is isomorphic to the profinite set $\varprojlim_n [0..(n+1)] \cong \overline{\mathbb{N}}$, the limit of the $\Omega(n)$ along the restriction maps of Lemma 4.16.

Corollary 4.19. *In fact, since $U_j \leq U_i \iff j \leq i$ for all $i, j \in \overline{\mathbb{N}}$, the bijection of Proposition 4.18 is a poset isomorphism $\mathrm{Sub}_{\mathcal{S}}(1) \cong \omega + 1$.*

Under this bijection, for a sentence φ with empty context, we have $[\![\varphi]\!] \in \overline{\mathbb{N}}$.

Moreover, $\Omega \cong L(1)$ via $\Omega(n) = \{0, \ldots, n+1\} \cong \{0, \ldots, n\} + 1 = L(1)(n)$, sending the top element $n+1$ to 1. This in turn induces a bijection between the global elements of $L(1)$ and of Ω. Intuitively, this isomorphism allows us to think of truth values as `async` computations whose `await` resolves once φ stops holding, i.e. at time step $[\![\varphi]\!]$, and never if $[\![\varphi]\!] = \omega$.

Remark 4.20. *A common notation that φ is true at stage m, i.e. that $\mathrm{Hom}_\omega(-, m) \to 1$ factors through $[\![\varphi]\!] \rightarrowtail 1$, is given by the* Kripke–Joyal semantics*, written $m \Vdash \varphi$ ("m* forces *φ"). We therefore have $m \Vdash \varphi$ if and only if $m < [\![\varphi]\!]$. The term stems from Cohen's forcing for the continuum hypothesis (Mac Lane and Moerdijk, 1994, §VI.2).*

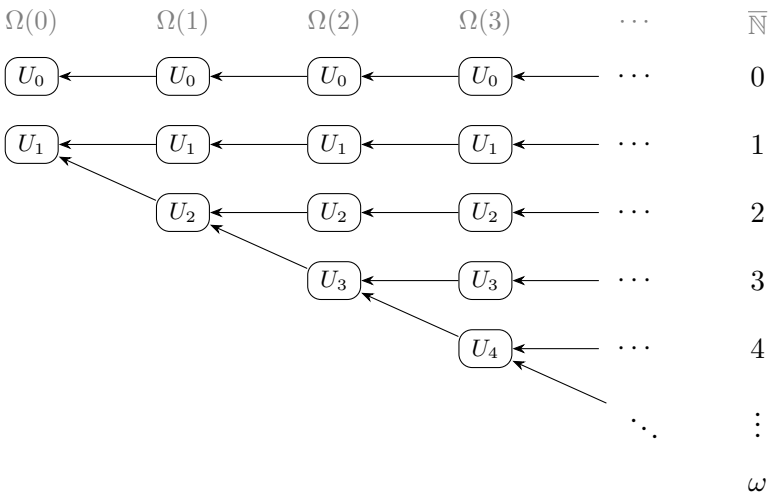


Figure 11: The forest $F(\Omega)$ with columns $\Omega(i)$. Global elements $1 \to \Omega$ are paths $(x_i)_{i \in \mathbb{N}}$ increasing in level for each $n \in \mathbb{N}$.

Example 4.21. *Let $x, y \colon 1 \to \mathtt{Str}(Bool)$ be global elements (Definition 3.9), i.e. streams as in Example 3.15. They are constants of $\Sigma_{\mathcal{S}}$, so they have no free variables and empty context. The formula $x = y$ is therefore a sentence. By Definition 4.10, $[\![x = y]\!] \rightarrowtail 1$ is a truth value in $\mathcal{S}$, given by the equalizer*

$$[\![x = y]\!] \rightarrowtail 1 \underset{y}{\overset{x}{\rightrightarrows}} \mathtt{Str}(Bool).$$

encoding up to which stage $n \in \overline{\mathbb{N}}$ the two streams x and y are equal.

Example 4.21 is depicted under F in Figure 12 for two concrete streams $x = (T, F, T, \ldots)$ and $y = (T, F, F, \ldots)$.

We can similarly interpret the logical connectives and quantifiers.

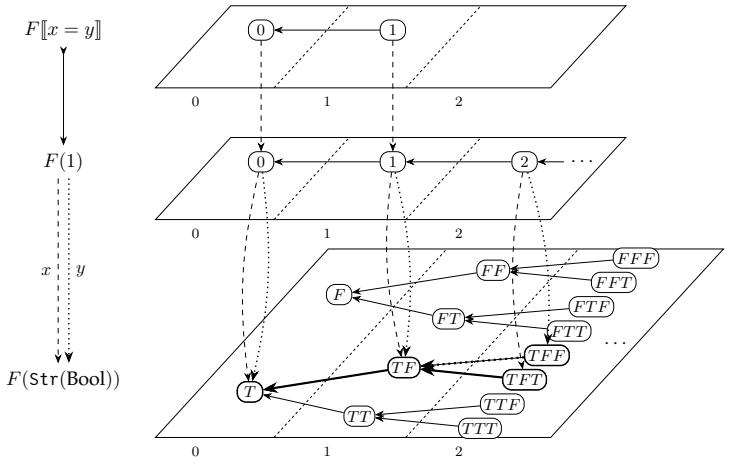


Figure 12: The equalizer $[\![x = y]\!] \rightarrowtail 1 \rightrightarrows \mathtt{Str}(\mathrm{Bool})$ under F, with $[\![x = y]\!] = U_2$.

4.5 Connectives and Quantifiers in $\mathcal{S}$

Having classified the truth values of $\mathcal{S}$ in Proposition 4.18, computing the logical connectives between them and quantifiers over them is the obvious next step. Let φ, ψ be formulae in the empty context, with $[\![\varphi]\!], [\![\psi]\!] \in \overline{\mathbb{N}}$ by Proposition 4.18.

Proposition 4.22. *We have*

$$[\![\varphi \wedge \psi]\!] = \min([\![\varphi]\!], [\![\psi]\!]).$$

Proof. By Definition 4.10(iv), $[\![\varphi \wedge \psi]\!]$ is the pullback of $[\![\varphi]\!]$ and $[\![\psi]\!]$ over 1, computed stagewise (Section 3.8). So stage i forces $\varphi \wedge \psi$ iff it forces both, i.e. by Remark 4.20 iff $i < [\![\varphi]\!]$ and $i < [\![\psi]\!]$. □

Figure 13 shows the corresponding pullback in **RForest** under F, and we can check that it is indeed computed stagewise: at each level, a node lies in $F[\![\varphi \wedge \psi]\!]$ precisely if it lies in both $F[\![\varphi]\!]$ and $F[\![\psi]\!]$.

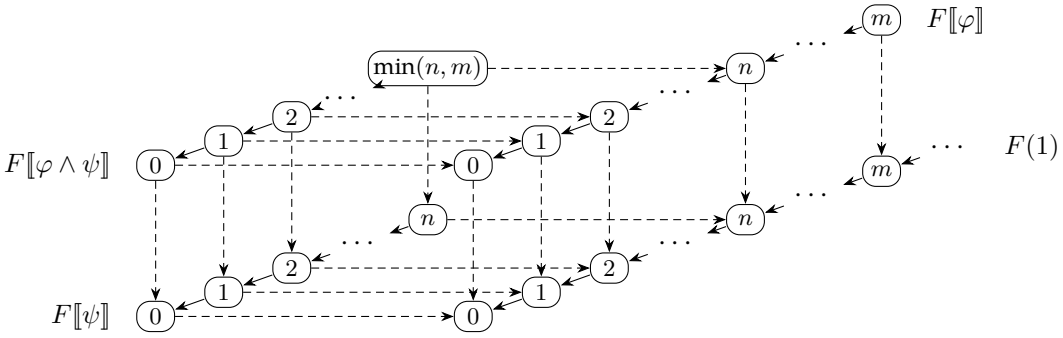


Figure 13: Definition 4.10(iv) under F for $[\![\varphi]\!] = m > n = [\![\psi]\!]$.

Proposition 4.23. *We have*

$$[\![\varphi \vee \psi]\!] = \max([\![\varphi]\!], [\![\psi]\!]).$$

Proof. By Definition 4.10(v), $[\![\varphi \vee \psi]\!]$ is the image of $[\![\varphi]\!] \amalg [\![\psi]\!] \to 1$, computed stagewise (Section 3.8). So stage i forces $\varphi \vee \psi$ iff it forces one of them, i.e. by Remark 4.20 iff $i < [\![\varphi]\!]$ or $i < [\![\psi]\!]$. □

We forgo depicting the union of Definition 4.10(v) in **RForest** for disjunction, as it is analogous to conjunction.

After conjunction and disjunction, *implication* is the next connective of great interest. By the Heyting algebra structure of $\mathrm{Sub}_{\mathcal{S}}(1)$, its definition easily follows:

Proposition 4.24. *We have*

$$[\![\varphi \Rightarrow \psi]\!] = \begin{cases} [\![\psi]\!] & \text{if } [\![\varphi]\!] > [\![\psi]\!], \\ \omega & \text{if } [\![\varphi]\!] \leq [\![\psi]\!]. \end{cases}$$

Proof. Under Proposition 4.18, $\mathrm{Sub}_{\mathcal{S}}(1) \cong \overline{\mathbb{N}}$ is totally ordered by $U_a \leq U_b \iff a \leq b$ with top ω, so Definition 4.10(vi) and Lemma 4.13 give the claim. □

This gives the following truth table for $[\![\varphi \Rightarrow \psi]\!]$:

$[\![\varphi]\!] \setminus [\![\psi]\!]$	0	1	2	3	$\cdots$	ω
0	ω	ω	ω	ω	$\cdots$	ω
1	0	ω	ω	ω	$\cdots$	ω
2	0	1	ω	ω	$\cdots$	ω
3	0	1	2	ω	$\cdots$	ω
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$
ω	0	1	2	3	$\cdots$	ω

Restricting to $[\![\varphi]\!], [\![\psi]\!] \in \{0, \omega\}$, we recover the classical implication. Intuitively, we just keep track of "how long" the implication relation holds.

Finally, by Definition 4.10(vi) we can also *negate* formulae now:

Proposition 4.25. *We have*

$$[\![\neg\varphi]\!] = \begin{cases} \omega & \text{if } [\![\varphi]\!] = 0, \\ 0 & \text{if } [\![\varphi]\!] > 0. \end{cases}$$

Proof. By Definition 4.10(vi), $[\![\neg\varphi]\!] = [\![\varphi \Rightarrow \bot]\!]$, and $[\![\bot]\!] = 0$ by Definition 4.10(iii). Now apply Proposition 4.24 with $[\![\psi]\!] = 0$. □

It follows immediately that the internal logic of $\mathcal{S}$ is intuitionistic, as the law of excluded middle fails: if $0 < [\![\varphi]\!] < \omega$, then $[\![\varphi \vee \neg\varphi]\!] = \max([\![\varphi]\!], 0) = [\![\varphi]\!] \neq \omega$ by Proposition 4.23 and Proposition 4.25.

We end this section by computing the *quantifiers* using Equation (11) in $\mathcal{S}$. Let φ now denote a formula in context $y : A$, let $S := [\![y.\varphi]\!] \rightarrowtail 1 \times [\![A]\!]$ and let $\pi \colon 1 \times [\![A]\!] \to 1$ be the projection. Recall that $m \in \overline{\mathbb{N}}$ corresponds to $U_m \rightarrowtail 1$ with $U_m(i) = 1$ if $i < m$ and $U_m(i) = 0$ otherwise.

Proposition 4.26. *We have*

$$[\![(\exists y : A)\varphi]\!] = \sup\{i+1 \mid i \in \mathbb{N},\ S(i) \neq \emptyset\}$$

in $\overline{\mathbb{N}}$, *where* $\sup \emptyset := 0$.

Proof. By Definition 4.10(vii), Lemma 4.14 and Corollary 4.19, $[\![(\exists y : A)\varphi]\!]$ is the least m with $S \leq \pi^{-1}U_m$. As $(\pi^{-1}U_m)(i) = U_m(i) \times [\![A]\!](i)$ (Section 3.8),

$$S \leq \pi^{-1}U_m \iff \forall i \geq m.\ S(i) = \emptyset \iff \sup\{i+1 \mid S(i) \neq \emptyset\} \leq m. \qquad \square$$

Under F, the global elements of $S \rightarrowtail 1 \times [\![A]\!] \cong [\![A]\!]$ are pairs of paths in $F(1)$ and $F([\![A]\!])$. By Proposition 4.18, the possible paths in $F(1)$ are the $F(U_m)$, which cut off at level m, therefore encoding the length m of a path. $[\![(\exists y : A)\varphi]\!]$ returns the longest such path occurring as a first component, i.e. the height of the forest $F(S)$. Taking $[\![y.\varphi]\!](n)$ as encoding the set of possible values at time step n, $\exists$ therefore encodes up to which time step there exists an element of $[\![A]\!]$ satisfying φ.

Proposition 4.27. *We have*

$$[\![(\forall y : A)\varphi]\!] = \min\{i \in \mathbb{N} \mid S(i) \neq [\![A]\!](i)\}$$

in $\overline{\mathbb{N}}$, *where* $\min \emptyset := \omega$.

Proof. By Definition 4.10(viii), Lemma 4.14 and Corollary 4.19, $[\![(\forall y : A)\varphi]\!]$ is the largest m with $\pi^{-1}U_m \leq S$. As $(\pi^{-1}U_m)(i) = U_m(i) \times [\![A]\!](i)$ (Section 3.8),

$$\pi^{-1}U_m \leq S \iff \forall i < m.\ S(i) = [\![A]\!](i) \iff m \leq \min\{i \in \mathbb{N} \mid S(i) \neq [\![A]\!](i)\}. \qquad \square$$

Analogously to existential quantification, universal quantification measures up to which time step n the formula φ holds for all $y \in [\![A]\!](n)$.

Example 4.28. *Let $X \in \mathcal{S}$ consist of the finite sequences*

$$\begin{aligned}
X(0) &= \{(2),(4)\},\\
X(1) &= \{(2,4),(2,6),(4,8),(4,10)\},\\
X(2) &= \{(2,4,6),(2,6,7),(4,8,12),(4,10,11)\},\\
X(3) &= \{(2,4,6,7),(2,6,7,9),(4,8,12,13),(4,10,11,13)\},
\end{aligned}$$

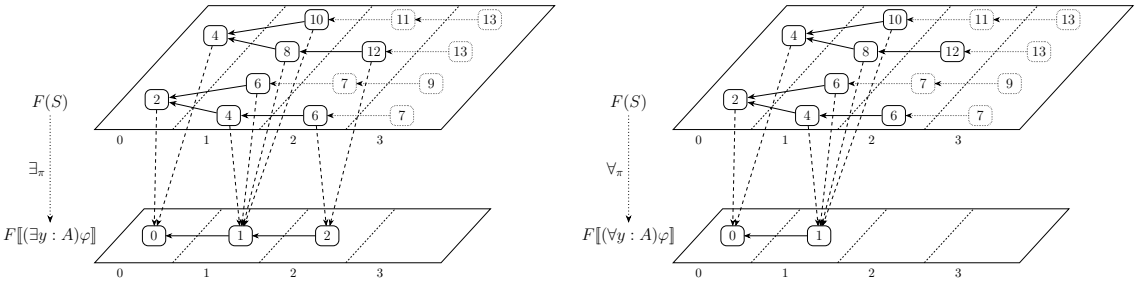
with $X(n) = \emptyset$ for $n > 3$ and restriction maps dropping the last entry. Let $E \rightarrowtail X$ consist of the sequences with even last entry. It is a subobject, since the parent of each such sequence again ends in an even number. Its characteristic map $\chi_E \colon X \to \Omega$ is a function symbol of $\Sigma_{\mathcal{S}}$, so for $y : A$ with $[\![A]\!] = X$ we define the formula "y is even" as

$$\varphi := (\chi_E(y) = \top),$$

with $S = [\![y.\varphi]\!] \cong E$. Half of the sequences in $X(2)$ and none in $X(3)$ are even, so by Propositions 4.26 and 4.27

$$[\![(\exists y : A)\varphi]\!] = \sup\{1,2,3\} = 3, \qquad [\![(\forall y : A)\varphi]\!] = \min\{2,3\} = 2,$$

i.e. some y is even up to time step 2 and all y are even up to time step 1 (Figure 14).



(a) $[\![(\exists y : A)\varphi]\!] = U_3$ (b) $[\![(\forall y : A)\varphi]\!] = U_2$

Figure 14: Example 4.28 under F, with nodes labeled by their last entry and S drawn bold inside $F(1 \times X) \cong F(X)$, the rest dotted. The dashed arrows are $\pi|_S$ in (a) and π restricted to $\pi^{-1}U_2 \leq S$ in (b).

4.6 The later modality

Given a formula φ in the empty context, we have established that its truth value corresponds to the number of time steps up to which it holds (Proposition 4.18). Naturally, the question arises how we can manipulate and work with time regarding such formulae. Say, when evaluating streams $x, y : \mathtt{Str}(A)$ at stage $n+1$, how can we write a formula that expresses that the streams are equal up to time step n?

Assume $[\![\varphi]\!] \rightarrowtail 1$. By Proposition 4.18 we know that $[\![\varphi]\!] \cong U_m$ for some $m \in \overline{\mathbb{N}}$, and we can simply consider $\blacktriangleright[\![\varphi]\!] \cong U_{m+1} \rightarrowtail \blacktriangleright 1 \cong 1$ to obtain a truth value that encodes "φ holds in the previous time step". However, $\blacktriangleright$ only applies to the interpretation $\blacktriangleright[\![\varphi]\!]$, whereas we would like a modal operator $\triangleright$ on the level of syntax, $[\![\triangleright\varphi]\!]$.

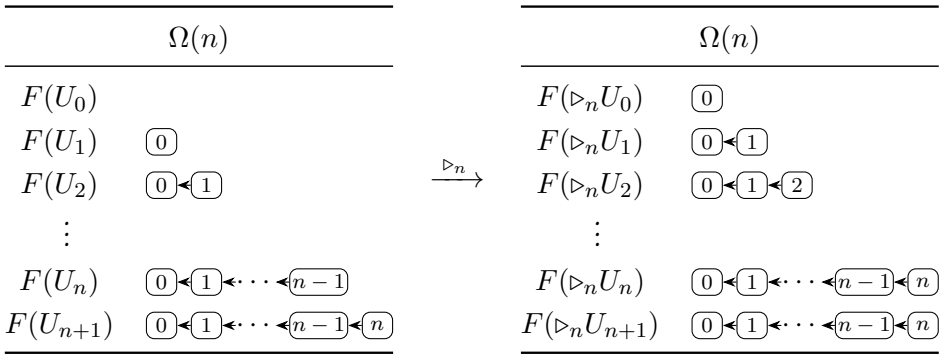
Definition 4.29. *Let $\vec{x} = x_1 : A_1, \ldots, x_n : A_n$ be a context and $X := [\![A_1]\!] \times \cdots \times [\![A_n]\!]$. We extend Definition 4.3 by the clause $\triangleright\varphi$ with $\mathrm{FV}(\triangleright\varphi) := \mathrm{FV}(\varphi)$, interpreted by the pullback*

$$\begin{array}{ccc} [\![\vec{x}.\triangleright\varphi]\!] & \longrightarrow & \blacktriangleright[\![\vec{x}.\varphi]\!] \\ \downarrow & \lrcorner & \downarrow \\ X & \xrightarrow[\mathrm{next}_X]{} & \blacktriangleright X. \end{array}$$

Stagewise, $[\![\vec{x}.\triangleright\varphi]\!](0) = X(0)$ and $a \in [\![\vec{x}.\triangleright\varphi]\!](n+1) \iff r_n^X(a) \in [\![\vec{x}.\varphi]\!](n)$ for all $n \in \mathbb{N}$ and $a \in X(n+1)$.

Intuitively, we consider $x'_{n+1} \in \blacktriangleright[\![\vec{x}.\varphi]\!](n+1)$ and "cut out" those $x_{n+1} \in X(n+1)$ which are sent to such an x'_{n+1} under next_X. Specifically, for $\top : 1 \rightarrowtail \Omega$, pulling back $\blacktriangleright\top$ along next_Ω we get a subobject of Ω whose characteristic morphism we also denote by $\triangleright$, given by $\triangleright : \Omega \to \Omega$ with $\triangleright_n(m) = \min(m+1, n+1)$.

Example 4.30. *Under F, $\triangleright_n$ appends a node at level m to every chain $F(U_m)$ with $m \leq n$, while the longest chain $F(U_{n+1})$ stays unchanged:*



Immediate consequences of Definition 4.29 are, for $X = 1$, that $n + 1 \Vdash \triangleright\varphi$ iff $n \Vdash \varphi$, and that $0 \Vdash \triangleright\varphi$ always (Remark 4.20), or phrased differently:

Corollary 4.31. *For a formula φ in the empty context, we have*

$$[\![\triangleright\varphi]\!] = \begin{cases} [\![\varphi]\!] + 1 & \text{if } [\![\varphi]\!] < \omega, \\ \omega & \text{if } [\![\varphi]\!] = \omega. \end{cases}$$

Example 4.32. *Let $x\colon 1 \to \mathtt{Str}(\mathit{Bool})$ be a global element with approximations x_n as in Example 3.15. By Definition 4.29, the subobject $[\![y : \mathtt{Str}(\mathit{Bool}) \mid \triangleright(x = y)]\!] \rightarrowtail \mathtt{Str}(\mathit{Bool})$ contains all of $\mathtt{Str}(\mathit{Bool})(0)$ at stage 0, and at stage $n + 1$ exactly those $y_{n+1} = (b_0, \dots, b_n, b_{n+1})$ with $r_n(y_{n+1}) = (b_0, \dots, b_n) = x_n$, i.e. all streams that agree with x up to time step n, while b_{n+1} is arbitrary. For $x = (\mathit{true}, \mathit{true}, \dots)$, Figure 15 depicts $F([\![y : \mathtt{Str}(\mathit{Bool}) \mid \triangleright(x = y)]\!])$.*

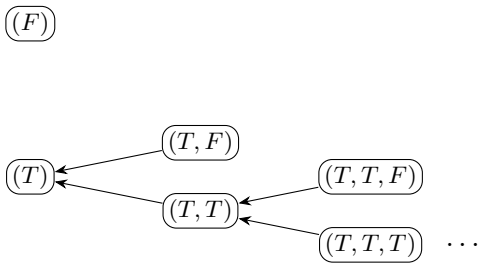


Figure 15: The forest $F([\![y : \mathtt{Str}(\mathrm{Bool}) \mid \triangleright(x = y)]\!])$ for $x = (\mathrm{true}, \mathrm{true}, \dots)$.

4.7 A higher-order typed λ-calculus in a topos

For topoi $\mathcal{E}$ like $\mathcal{S}$, we can extend the internal language $\Sigma_{\mathcal{E}}$ by the *Mitchell–Bénabou language* (Mac Lane and Moerdijk, 1994, Sec. VI.5), yielding a higher-order typed λ-calculus, effectively allowing us to *program inside $\mathcal{E}$*. The following terms and formulae are added:

Definition 4.33 (Mac Lane and Moerdijk, 1994, Sec. VI.5)**.** *The terms and formulae of the* Mitchell–Bénabou language *of $\mathcal{E}$ are given by the clauses of Definition 4.2 and Definition 4.3 over $\Sigma_{\mathcal{E}}$, together with the following.*

(*c*) Application*: $\sigma(t) : B$, if $\sigma : B^A$ and $t : A$ are terms;* $\mathrm{FV}(\sigma(t)) := \mathrm{FV}(\sigma) \cup \mathrm{FV}(t)$.

(*d*) Abstraction*: $\lambda x{:}A.\, t : B^A$, if $t : B$ is a term and x a variable of type A;* $\mathrm{FV}(\lambda x{:}A.\, t) := \mathrm{FV}(t) \setminus \{x\}$.

(*vii*) Membership*: $(t \in E)$, if $t : A$ and $E : \Omega^A$ are terms;* $\mathrm{FV}(t \in E) := \mathrm{FV}(t) \cup \mathrm{FV}(E)$.

The term $\sigma(t)$ for $\sigma : B^A$ is not to be confused with $f(t)$ for a function symbol $f\colon A \to B$; the types of σ and f tell them apart.

Definition 4.34 (Mac Lane and Moerdijk, 1994, Sec. VI.5)**.** *Let $\vec{x}$ be a context of type X. We extend Definition 4.9 and Definition 4.10 by the following clauses.*

(*c*) *$[\![\vec{x}.\sigma(t)]\!]$ is the composite $X \xrightarrow{([\![\vec{x}.\sigma]\!],[\![\vec{x}.t]\!])} B^A \times A \xrightarrow{\mathrm{ev}} B$, where* ev *is the counit of the adjunction $(-) \times A \dashv (-)^A$.*

(*d*) *$[\![\vec{x}.\lambda x{:}A.\, t]\!]\colon X \to B^A$ is the exponential transpose of $[\![\vec{x}, x.t]\!]\colon X \times A \to B$.*

(*ix*) *$[\![\vec{x}.(t \in E)]\!] \rightarrowtail X$ is the pullback of $\top\colon 1 \to \Omega$ along $X \xrightarrow{([\![\vec{x}.E]\!],[\![\vec{x}.t]\!])} \Omega^A \times A \xrightarrow{\mathrm{ev}} \Omega$, i.e. the subobject classified by $[\![\vec{x}.E(t)]\!]$.*

In particular, $t \in E$ and $E(t) = \top$ are interpreted as the same subobject.

5 Applications

5.1 Characterizing binary trees

As we have seen, there are multiple different types which are binary trees under F, such as $F(\mathrm{Eq}(\mathsf{hd}, \mathsf{const}_{\mathsf{true}}))$ and $F(\mathrm{Coeq}(\mathrm{id}, \mathrm{not}))$ from Figures 9c and 9d. We wish to construct a sentence $\mathrm{BinTree}_X$ capturing that $F(X)$ is a binary tree.

One immediate idea is to ask "does there exist a unary relation symbol R such that R is a binary tree and all elements of X satisfy R?". Such a relation symbol will be interpreted as a monomorphism $m\colon R \rightarrowtail X$ for R a binary tree and then using $[\![(\forall x : X)(\exists y : R)m(y) = x]\!]$ to express that X is a binary tree. This however requires the existence of such a monomorphism m, which is not guaranteed. We can express that $m\colon X^R$ is a monomorphism by $[\![(\forall y_1, y_2 : R)m(y_1) = m(y_2) \Rightarrow y_1 = y_2]\!]$, and by setting $R = \mathrm{Eq}(\mathsf{hd}, \mathsf{const}_{\mathsf{true}})$, one possible construction of $\mathrm{BinTree}_X^{\mathrm{Rel}}$ is achieved by existentially quantifying over $m : X^R$, given we use the higher-order typed λ-calculus of Section 4.7 which permits such:

Definition 5.1. *For $X \in \mathcal{S}$, we define the sentence*

$$\mathrm{BinTree}_X^{Rel} := (\exists m : X^R)\big((\forall y_1, y_2 : R)m(y_1) = m(y_2) \Rightarrow y_1 = y_2 \\ \wedge\, (\forall x : X)(\exists y : R)\, m(y) = x\big).$$

In a similar fashion, we could have required the existence of an isomorphism between X and a known binary tree such as $\mathrm{Eq}(\mathsf{hd}, \mathsf{const}_{\mathsf{true}})$. Again using the higher-order typed λ-calculus of Section 4.7 to quantify over morphisms and require the existence of an isomorphism to a fixed binary tree such as $B := \mathrm{Eq}(\mathsf{hd}, \mathsf{const}_{\mathsf{true}})$, we express this as follows:

Definition 5.2. *For $X \in \mathcal{S}$, we define the sentence*

$$\mathrm{BinTree}_X^{iso} := (\exists f : X^B)(\exists g : B^X)\big((\forall s : B)\, g(f(s)) = s \\ \wedge (\forall x : X)\, f(g(x)) = x\big).$$

Using the description of exponentials in Lemma 3.20, we can see that the truth value of $\mathrm{BinTree}_X^{\mathrm{iso}}$ measures up to which depth $F(X)$ is a binary tree.

Proposition 5.3. *Let $M \in \overline{\mathbb{N}}$ be the least stage such that there is no family of isomorphisms $(X(m) \cong B(m))_{m \leq M}$ compatible with the restriction maps, and $M = \omega$ if no such stage exists. Then $[\![\mathrm{BinTree}_X^{iso}]\!] = M$.*

Proof. By Proposition 4.26, Definition 4.10(ii), (iv), (viii), Definition 4.34(c) and Lemma 3.20,

$$\begin{aligned}[\![\mathrm{BinTree}_X^{\mathrm{iso}}]\!] &= \sup\{k+1 \mid \exists f \in (X^B)(k),\ g \in (B^X)(k)\ \forall m \leq k. \\ &\qquad g_m \circ f_m = \mathrm{id}_{B(m)} \wedge f_m \circ g_m = \mathrm{id}_{X(m)}\} \\ &= \sup\{k+1 \mid k < M\} = M.\end{aligned}$$

□

However, both $\mathrm{BinTree}_X^{\mathrm{Rel}}$ and $\mathrm{BinTree}_X^{\mathrm{iso}}$ quantify over exponentials (X^R, respectively X^B and B^X), i.e. over morphisms rather than nodes under the equivalence $\mathcal{S} \simeq$ **RForest**. We might instead strive for a formula quantifying only over X, $\blacktriangleright X$ and Ω rather than over exponentials, to reduce complexity. This requires constructing a formula that checks that each node has exactly two children. We construct formulae that check

1. whether two nodes $x_1, x_2 : X$ have the same parent (Sib_X),
2. whether each parent has at most two children ($\mathrm{AtMostTwo}_X$),
3. whether each parent has at least two children ($\mathrm{AtLeastTwo}_X$),
4. whether X has only one root ($\mathrm{OneRoot}_X$), and
5. whether every node has a child (Total_X).

Assembling them, we define $\mathrm{BinTree}_X$ in Definition 5.9.

Two nodes are siblings if they have the same parent, i.e. if they agree up to the previous stage. As in Example 4.32, this is exactly what the later modality applied to equality expresses:

Definition 5.4. *For $X \in \mathcal{S}$ and the context $\vec{x} = x_1 : X, x_2 : X$, we define*

$$\vec{x}.\mathrm{Sib}_X(x_1, x_2) := \vec{x}.\, \triangleright\, (x_1 = x_2).$$

By Definition 4.29 and Definition 4.10(ii), $[\![\vec{x}.\mathrm{Sib}_X(x_1, x_2)]\!](0) = X(0) \times X(0)$, meaning all roots are siblings, and $[\![\vec{x}.\mathrm{Sib}_X(x_1, x_2)]\!](n+1) = \{(a_1, a_2) \mid r_n^X(a_1) = r_n^X(a_2)\}$ consists of all pairs of siblings at stage $n+1$.

Using siblings, we can bound the number of children of every node.

Definition 5.5. *For $X \in \mathcal{S}$, we define the sentence*

$$\mathrm{AtMostTwo}_X := (\forall x_1, x_2, x_3 : X)\big(\mathrm{Sib}_X(x_1, x_2) \wedge \mathrm{Sib}_X(x_2, x_3) \Rightarrow x_1 = x_2 \vee x_1 = x_3 \vee x_2 = x_3\big).$$

Proposition 5.6. *Let $m \in \overline{\mathbb{N}}$ be the least stage such that $X(m)$ contains three pairwise distinct siblings x_1, x_2, x_3, and $m = \omega$ if there is none. Then $[\![\mathrm{AtMostTwo}_X]\!] = m$.*

Proof. We prove this by induction. First, let $m = 0$. We need to show that if $X(m)$ contains three pairwise distinct siblings, then $[\![\mathrm{AtMostTwo}_X]\!] = 0$. In the case $m = 0$, all nodes are siblings, and given $x_1, x_2, x_3 \in X(0)$ pairwise distinct, we have that all of the equalities inside the disjunction $x_1 = x_2 \vee x_1 = x_3 \vee x_2 = x_3$ fail and $[\![\mathrm{AtMostTwo}_X]\!] = 0$. If there are no three distinct x_1, x_2, x_3, then at least one of the equalities holds and as such $[\![\mathrm{AtMostTwo}_X]\!] > m$.

Let now $m = m' + 1$ for some $m' \in \mathbb{N}$ and assume $[\![\mathrm{AtMostTwo}_X]\!] > m'$. If there exist three distinct $x_1, x_2, x_3 \in X(m)$ which are siblings, then $\mathrm{Sib}_X(x_1, x_2) \wedge \mathrm{Sib}_X(x_2, x_3)$ holds, but all of the equalities in the disjunction $x_1 = x_2 \vee x_1 = x_3 \vee x_2 = x_3$ fail again, and we have $[\![\mathrm{AtMostTwo}_X]\!] = m$. If there are no three distinct x_1, x_2, x_3, then at least one of the equalities must hold and the disjunction also holds, meaning $[\![\mathrm{AtMostTwo}_X]\!] > m$.

Finally, assume that for no m there exist three distinct $x_1, x_2, x_3 \in X(m)$ which are siblings. Then the implication is obviously true, and as such $[\![\mathrm{AtMostTwo}_X]\!]$ must be greater than any natural number, meaning $[\![\mathrm{AtMostTwo}_X]\!] = \omega$. □

Hence $\mathrm{AtMostTwo}_X$ says that $F(X)$ has at most two roots and every node has at most two children. We now need to make this "at most two" into an "exactly two" regarding the number of children. It would be tempting to think that something using negation like $[\![(\forall x : X)(\exists y : X)\mathrm{Sib}_X(x, y) \wedge \neg(x = y)]\!]$ would be helpful, but inequality is already determined at stage 0, so $\mathrm{Sib}_X(x, y)$ already implies that $\neg(x = y)$ fails by Proposition 4.25. Instead, by quantifying over truth values $p : \Omega$, we get:

Definition 5.7. *For $X \in \mathcal{S}$, we define the sentence*

$$\mathrm{AtLeastTwo}_X := (\forall x : X)(\forall p : \Omega)(\exists y : X)\big((\triangleright p \Rightarrow x = y) \wedge (x = y \Rightarrow \triangleright p)\big).$$

At stage n, $\triangleright p$ ranges over $1, \ldots, n+1$, so for every $a \in X(n)$ and $k < n$ there must be $b \in X(n)$ agreeing with a up to stage k and differing at stage $k+1$. Thus, if X has no leaves, $\mathrm{AtLeastTwo}_X$ holds iff every node of $F(X)$ has at least two children, as for $X = \mathtt{Str}(\mathrm{Bool})$.

Example 5.8. *Let $X = \mathtt{Str}(Bool)$ and $x = (true, true, true, \ldots)$, and fix a global truth value p with $[\![p]\!] = 1$, so that $[\![\triangleright p]\!] = 2$ by Corollary 4.31. For candidates y with different separation stages $[\![x = y]\!]$, Proposition 4.24 and Proposition 4.22 give the following values of the left implication $L := \triangleright p \Rightarrow x = y$, the right implication $R := x = y \Rightarrow \triangleright p$, and their conjunction:*

(*case:* $[\![\triangleright p]\!] = 2$)

y	$[\![x = y]\!]$	$[\![L]\!]$	$[\![R]\!]$	$[\![L \wedge R]\!]$
$(false, \ldots)$	0	0	ω	0
$(true, false, \ldots)$	1	1	ω	1
$(true, true, false, \ldots)$	2	ω	ω	ω
$(true, true, true, false, \ldots)$	3	ω	2	2
$(true, true, true, \ldots)$	ω	ω	2	2

The left implication L holds whenever x and y agree at least as long as $\triangleright p$ does, i.e. $[\![x = y]\!] \geq 2$, ruling out y which separate from x too early. The right implication R holds whenever they agree at most as long as $\triangleright p$ does, i.e. $[\![x = y]\!] \leq 2$, ruling out y which separate too late or not at all. Together, $L \wedge R$ holds exactly for $[\![x = y]\!] = 2$, i.e. for y that agrees with x up to stage 1 and differs from it at stage 2. Hence $[\![(\exists y : X)(L \wedge R)]\!] = \omega$ for this p.

For every other p, we likewise have $[\![(\exists y : X)(L \wedge R)]\!] = \omega$ if and only if there exists a stream y that agrees with x below stage $[\![\triangleright p]\!]$ and differs from it at stage $[\![\triangleright p]\!]$, i.e. $[\![x = y]\!] = [\![\triangleright p]\!]$. The tables for $[\![p]\!] = 0$, $[\![p]\!] = 2$ and $[\![p]\!] = 3$ are provided in Section A. In the case $[\![p]\!] = \omega$, we just ask for a y with $[\![x = y]\!] = \omega$, which is given by $y = x$. No p asks for $[\![x = y]\!] = 0$, since $[\![\triangleright p]\!] \geq 1$, so distinct roots are never required. In contrast, for $X = 1$ the only choice is $y = x$ with $[\![x = y]\!] = \omega$. For $[\![p]\!] = 0$, the right implication then has $[\![R]\!] = 1$: $x = y$ still holds at stage 1, where $\triangleright p$ has stopped. Hence $[\![\mathrm{AtLeastTwo}_1]\!] = 1$, reflecting that the root of $F(1)$ has only one child.

It remains to demand a single root and to exclude leaves. With $\triangleright\bot$, which holds only at stage 0, we define

$$\begin{aligned} \mathrm{OneRoot}_X &:= (\forall x, y : X)(\triangleright\bot \Rightarrow x = y), \\ \mathrm{Total}_X &:= (\forall z : \blacktriangleright X)(\exists x : X)\, \mathrm{next}_X(x) = z, \end{aligned}$$

where $\mathrm{OneRoot}_X$ says that all nodes share their root, and Total_X that $X(0)$ is inhabited and every node has a child, since $(\blacktriangleright X)(0) = 1$ and next_X is the restriction map otherwise. Combining all of them, we get:

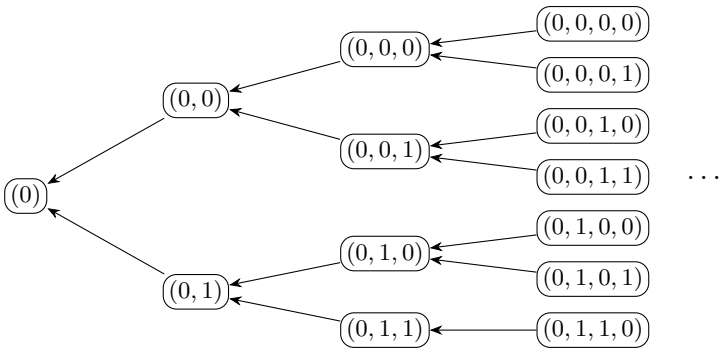
Definition 5.9. *For $X \in \mathcal{S}$, we define the sentence*

$$\mathrm{BinTree}_X := \mathrm{OneRoot}_X \wedge \mathrm{Total}_X \wedge \mathrm{AtMostTwo}_X \wedge \mathrm{AtLeastTwo}_X.$$

Proposition 5.10. *For $X \in \mathcal{S}$, we have $[\![\mathrm{BinTree}_X]\!] = [\![\mathrm{BinTree}_X^{iso}]\!]$.*

Proof. By Remark 4.20 and Proposition 4.22, it suffices to show $n \Vdash \mathrm{BinTree}_X$ iff $n < [\![\mathrm{BinTree}_X^{\mathrm{iso}}]\!]$. By the above, $n \Vdash \mathrm{OneRoot}_X \wedge \mathrm{Total}_X$ iff $X(0)$ is a singleton and no node in $X(i)$ for $i < n$ is a leaf, and then $n \Vdash \mathrm{AtMostTwo}_X \wedge \mathrm{AtLeastTwo}_X$ iff no such node has more than two or exactly one child. This says $(X(m) \cong B(m))_{m \leq n}$ compatibly, i.e. $n < [\![\mathrm{BinTree}_X^{\mathrm{iso}}]\!]$ by Proposition 5.3. □

Example 5.11. *Consider $X \in \mathcal{S}$ given by the tree*



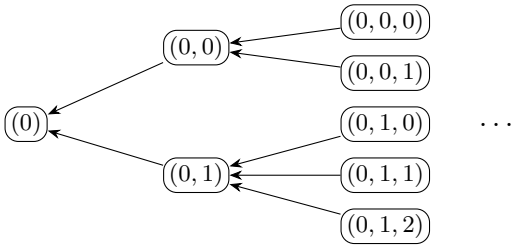
Let x be the global element with $x_3 = (0,1,1,0)$ and $[\![p]\!] = 2$, so $[\![\triangleright p]\!] = 3$ (Corollary 4.31). Then $[\![x = y]\!] \in \{1,2\} \cup \{4,5,\ldots,\omega\}$ for all y, and

$[\![x = y]\!]$	$[\![\triangleright p \Rightarrow x = y]\!]$	$[\![x = y \Rightarrow \triangleright p]\!]$	$[\![L \wedge R]\!]$
1	1	ω	1
2	2	ω	2
$k \geq 4,\ \omega$	ω	3	3

by Propositions 4.22 and 4.24, where $L \wedge R$ abbreviates the body of $\mathrm{AtLeastTwo}_X$. Hence, by Propositions 4.22, 4.26 and 4.27,

$$[\![\mathrm{BinTree}_X]\!] = [\![\mathrm{AtLeastTwo}_X]\!] = [\![(\exists y : X)(L \wedge R)]\!] = \max\{1,2,3\} = 3.$$

Example 5.12. *Consider $X \in \mathcal{S}$ given by the tree*



Let x_1, x_2, x_3 be global elements with $(x_i)_2 = (0, 1, i-1)$. Then $[\![x_i = x_j]\!] = 2$ for $i \neq j$, so $[\![\mathrm{Sib}_X(x_i, x_j)]\!] = [\![\triangleright(x_i = x_j)]\!] = 3$, and by Propositions 4.22 to 4.24,

$$[\![\mathrm{Sib}_X(x_1,x_2) \wedge \mathrm{Sib}_X(x_2,x_3) \Rightarrow x_1 = x_2 \vee x_1 = x_3 \vee x_2 = x_3]\!] = 2,$$

since $\min\{3,3\} = 3 > 2 = \max\{2,2,2\}$. Hence, by Propositions 4.22 and 4.27,

$$[\![\mathrm{BinTree}_X]\!] = [\![\mathrm{AtMostTwo}_X]\!] = 2.$$

5.2 The Collatz tree

Consider the Collatz function $c\colon \mathbb{N} \to \mathbb{N}$ defined by

$$c(n) := \begin{cases} n/2, & \text{if } n \text{ is even,} \\ 3n+1, & \text{if } n \text{ is odd.} \end{cases}$$

The Collatz conjecture states that for every $n \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that $c^k(n) = 1$. Equivalently, given the seed $C_0 := \{1\}$, we may constuct $C_{k+1} := c^{-1}(C_k)$, and wonder whether $\bigcup_{k\in\mathbb{N}} C_k = \mathbb{N}$. We only need to define restriction maps $r_k^C\colon C_{k+1} \to C_k$ by $r_k^C(n) := c(n)$, and we have an object $C \in \mathcal{S}$.

However, is it possible to rephrase this construction in terms of the internal language $\Sigma_{\mathcal{S}}$?

Consider $\mathtt{Str}(\mathbb{N})$. We wish to refine it up to isomorphism to a subobject $C \rightarrowtail \mathtt{Str}(\mathbb{N})$ such that $F(C)$ is the Collatz tree, meaning up to isomorphism it has one root $[1]$, then the nodes $\{[1,x] \mid x \in c^{-1}(1)\}$, and so on, i.e.

$$C(n) = \{[x_0,\ldots,x_n] \in \mathtt{Str}(\mathbb{N})(n) \mid x_0 = 1 \text{ and } c(x_{i+1}) = x_i \text{ for all } i < n\}.$$

To fix the root, we can use a formula like $\mathrm{OneRoot}_X$, but with the constant $\mathrm{const}_{\mathtt{Str}(\mathbb{N})}(1)\colon 1 \to \mathtt{Str}(\mathbb{N})$ instead of y, which gives

$$\mathrm{Root}_1(x) := \triangleright\bot \Rightarrow x = \mathrm{const}_{\mathtt{Str}(\mathbb{N})}(1).$$

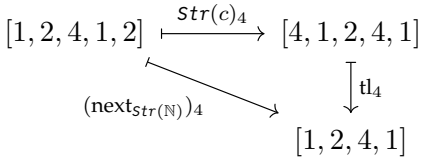
Now we are left finding a formula $\mathrm{Collatz}(x)$ such that $C = [\![x.\, \mathrm{Root}_1(x) \wedge \mathrm{Collatz}(x)]\!]$. By functoriality and the diagonal map, we can lift $c : \mathbb{N} \to \mathbb{N}$ to $\mathtt{Str}(c) : \mathtt{Str}(\mathbb{N}) \to \mathtt{Str}(\mathbb{N})$, by applying c entrywise $\mathtt{Str}(c)_n([x_0, \ldots, x_n]) := [c(x_0), \ldots, c(x_n)]$.

Example 5.13. *For* $[1,2,4,1,2] \in \mathtt{Str}(\mathbb{N})(4)$*, we get* $\mathtt{Str}(c)_4([1,2,4,1,2]) = [5,1,2,4,1]$.

Notice, that exactly for global elements $x : C$, we have that the tail of $\mathtt{Str}(c)(x)$ is guaranteed to be a global element of $\blacktriangleright C$. This allows us to define

$$\mathrm{Collatz}(x) := (\blacktriangleright\mathtt{Str}(c))(\mathrm{tl}(x)) = \mathrm{next}_{\mathtt{Str}(\mathbb{N})}(x).$$

Example 5.14. *Consider* $x = [1,2,4,1,2] \in \mathtt{Str}(\mathbb{N})(4)$ *as before. Since* tl *is natural, applying* $\mathtt{Str}(c)$ *before or after* tl *makes no difference, and* $\mathrm{Collatz}(x)$ *holds at stage* 4*, as the following diagram commutes:*



Proposition 5.15. $C = [\![x.\, \mathrm{Root}_1(x) \wedge \mathrm{Collatz}(x)]\!]$.

Proof. We compare stagewise. Obviously we have $C(0) = [1]$ for the left hand side, and for the right hand side by $\mathrm{Root}_1(x)$ only $[1]$ is permissible. $\mathrm{Collatz}(x)$ holds for any x at stage 0, as both tl and $\mathrm{next}_{\mathtt{Str}(\mathbb{N})}$ neccessarily are morphisms into the terminal object 1. At stage $n+1$, an element $x = [x_0, \ldots, x_{n+1}]$ is sent to

$$(\blacktriangleright\mathtt{Str}(c))_{n+1}(\mathrm{tl}_{n+1}(x)) = [c(x_1), \ldots, c(x_{n+1})], \qquad (\mathrm{next}_{\mathtt{Str}(\mathbb{N})})_{n+1}(x) = [x_0, \ldots, x_n],$$

so $[\![x.\, \mathrm{Collatz}(x)]\!](n)$ consists of those $[x_0, \ldots, x_n]$ with $c(x_{i+1}) = x_i$ for all $i < n$. By the implication from $\triangleright\bot$, we only consider those $[x_0, \ldots, x_n]$ with $x_0 = 1$. Intersecting both subobjects yields exactly $C(n)$. □

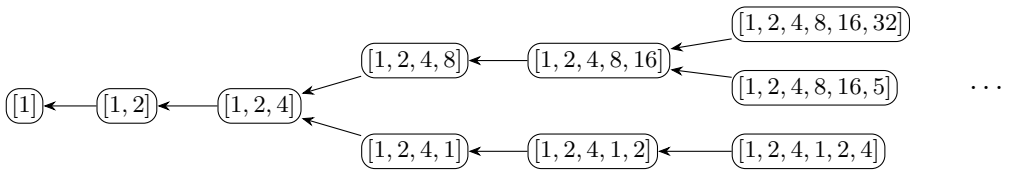


Figure 16: The first six levels of the Collatz tree $F(C)$.

6 Conclusion and future work

We have succeeded in our goal of defining the topos of trees from the ground up, including its categorical logic and sheaf-theoretic foundations. We also succeeded in defining a sentence to check whether a type represents a binary tree, which we did in varying degrees of abstraction. Moreover, we successfully constructed a formula to build the Collatz tree.

In this thesis we focused mainly on the application of the topos of trees to model coinductive types which possess finite approximations. As such, a binary tree type can be regarded as the type of all streams of zeros and ones whose head is 0 (or vice versa 1).

However, as hinted at in the introduction and in Jaber and Riba (2021), the temporal nature of $\mathcal{S}$ and refinement types allow for *model checking* to be expressed using only the type system. A type X with $[\![\text{BinTree}_X]\!] = k$ then encodes that a system is ensured to have exactly two possible state transitions at every time step i with $i + 1 < k$. A state without any outgoing transitions, i.e. an element at stage n with no children at stage $n + 1$, is then a *deadlock*.

Detecting deadlocks, or providing a sentence that checks up to which depth no deadlocks occur, like Total_X from Section 5.1, would be of great utility to prove safety-critical properties of systems.

As an outlook, we propose research into geometric morphisms between $[\omega^{\text{op}}, \mathbf{Set}]$ and $[\zeta^{\text{op}}, \mathbf{Set}]$, where ζ is a linear order extending ω, to express concepts such as the behavioral semantics of Event-B refinement (Schneider et al., 2014), as well as research into how a minimal sublanguage of $\Sigma_{\mathcal{S}}$ can be expressed such that all such state spaces are definable.

A Further cases of Example 5.8

We give the tables of Example 5.8 for $[\![p]\!] = 0$, $[\![p]\!] = 2$ and $[\![p]\!] = 3$, with $X = \mathtt{Str}(\mathsf{Bool})$, $x = (\mathsf{true}, \mathsf{true}, \mathsf{true}, \dots)$, $L := \triangleright p \Rightarrow x = y$ and $R := x = y \Rightarrow \triangleright p$. In each case, $L \wedge R$ holds exactly for the y with $[\![x = y]\!] = [\![\triangleright p]\!] = [\![p]\!] + 1$.

(case: $[\![\triangleright p]\!] = 1$)

y	$[\![x = y]\!]$	$[\![L]\!]$	$[\![R]\!]$	$[\![L \wedge R]\!]$
$(\mathsf{false}, \dots)$	0	0	ω	0
$(\mathsf{true}, \mathsf{false}, \dots)$	1	ω	ω	ω
$(\mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	2	ω	1	1
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \dots)$	ω	ω	1	1

(case: $[\![\triangleright p]\!] = 3$)

y	$[\![x = y]\!]$	$[\![L]\!]$	$[\![R]\!]$	$[\![L \wedge R]\!]$
$(\mathsf{false}, \dots)$	0	0	ω	0
$(\mathsf{true}, \mathsf{false}, \dots)$	1	1	ω	1
$(\mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	2	2	ω	2
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	3	ω	ω	ω
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	4	ω	3	3
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \dots)$	ω	ω	3	3

(case: $[\![\triangleright p]\!] = 4$)

y	$[\![x = y]\!]$	$[\![L]\!]$	$[\![R]\!]$	$[\![L \wedge R]\!]$
$(\mathsf{false}, \dots)$	0	0	ω	0
$(\mathsf{true}, \mathsf{false}, \dots)$	1	1	ω	1
$(\mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	2	2	ω	2
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	3	3	ω	3
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	4	ω	ω	ω
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{false}, \dots)$	5	ω	4	4
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \dots)$	ω	ω	4	4

References

Harold Abelson, Gerald Jay Sussman, and Julie Sussman (1996). *Structure and Interpretation of Computer Programs*. 2nd ed. MIT Press.

Andrew W. Appel and David McAllester (2001). "An Indexed Model of Recursive Types for Foundational Proof-Carrying Code". In: *ACM Transactions on Programming Languages and Systems* 23.5, pp. 657–683.

Patrick Bahr (2022). "Modal FRP for All: Functional Reactive Programming without Space Leaks in Haskell". In: *Journal of Functional Programming* 32, e15.

Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi (Aug. 1, 2019). "Guarded Cubical Type Theory". In: *Journal of Automated Reasoning* 63.2, pp. 211–253.

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring (June 2011). "First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees". In: *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. 2011 IEEE 26th Annual Symposium on Logic in Computer Science, pp. 55–64. (Visited on 08/19/2026).

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring (Oct. 2012). "First Steps in Synthetic Guarded Domain Theory: Step-indexing in the Topos of Trees". In: *Logical Methods in Computer Science* 8.4:1. Extended journal version of Birkedal, Møgelberg, Schwinghammer, and Støvring (2011) ., pp. 1–45. (Visited on 08/29/2026).

Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal (2015). "Programming and Reasoning with Guarded Recursion for Coinductive Types". In: *Foundations of Software Science and Computation Structures (FoSSaCS 2015)*. Ed. by Andrew Pitts. Vol. 9034. Lecture Notes in Computer Science. Springer, pp. 407–421.

Coinductive and Inductive Predicates (2026). The Lean Language Reference. Lean FRO. URL: `https://lean-lang.org/doc/reference/latest/Definitions/Recursive-Definitions/#coinductive-predicates` (visited on 11/08/2026).

Guilhem Jaber and Colin Riba (2021). "Temporal Refinements for Guarded Recursive Types". In: *Programming Languages and Systems: 30th European Symposium on Programming (ESOP 2021)*. Ed. by Nobuko Yoshida. Vol. 12648. Lecture Notes in Computer Science. Springer, pp. 548–578.

Peter T. Johnstone (2002a). *Sketches of an Elephant: A Topos Theory Compendium*. Vol. 1. Oxford Logic Guides 43. Oxford University Press.

Peter T. Johnstone (2002b). *Sketches of an Elephant: A Topos Theory Compendium*. Vol. 2. Oxford Logic Guides 44. Oxford University Press.

Bálint Kocsis (May 23, 2025). *Guarded Recursion in the Topos of Trees*. Internship report (updated version).

Saunders Mac Lane and Ieke Moerdijk (1994). *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer.

Erik Meijer, Maarten Fokkinga, and Ross Paterson (1991). "Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire". In: *Functional Programming Languages and Computer Architecture*. Ed. by John Hughes. Vol. 523. Lecture Notes in Computer Science. Springer, pp. 124–144.

Hiroshi Nakano (2000). "A Modality for Recursion". In: *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS 2000)*. IEEE Computer Society, pp. 255–266.

Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal (2015). "A Model of PCF in Guarded Type Theory". In: *Electronic Notes in Theoretical Computer Science* 319, pp. 333–349.

Steve Schneider, Helen Treharne, and Heike Wehrheim (2014). "The Behavioural Semantics of Event-B Refinement". In: *Formal Aspects of Computing* 26.2, pp. 251–280.

Sergei Stepanenko, Patrick Bahr, and Rasmus Ejlers Møgelberg (2026). *Garlene: Guarded Recursion in Lean*. arXiv: 2609.24345 [cs.LO].

Niccolò Veltri and Andrea Vezzosi (2020). "Formalizing π-Calculus in Guarded Cubical Agda". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*. ACM, pp. 270–283.

Sebastiano Vigna (2003). *A Guided Tour in the Topos of Graphs*. arXiv: math/0306394.

Declaration

I hereby declare that I have written this bachelor's thesis independently and have used no sources or aids other than those indicated.

Use of AI tools. Claude Code (model: Opus 5.5) was used for initial outlining ideas, language editing, creating figures, and transcribing handwritten notes into LaTeX. All technical statements and sources were checked independently, and I take full responsibility for them.

Düsseldorf, December 17, 2026

(Luca Leon Happel)