A Monad For Randomized Algorithms

Tyler Barker
Tulane University[1]

Domains XII
August 26, 2015

[1]Supported by AFOSR

Randomized Algorithms

A **randomized algorithm** can be represented by a probabilistic Turing machine.

A **probabilistic Turing machine** is a Turing machine that can use random coin flips to choose transitions.

The goal is to provide a domain-theoretic monad to use in denotational semantics for randomized algorithms.

Related Work: Probabilistic Powerdomain

The **probabilistic powerdomain** of Jones and Plotkin gives a monad over domains.

However, there is no Cartesian closed category of domains known to be invariant under the probabilistic powerdomain.

Also, the probabilistic powerdomain does not share a distributive law with any of the nondeterministic powerdomains.

Motivation for the Functor

One of most well known randomized algorithms is the **Miller-Rabin primality test**.

The algorithm is in the complexity class **randomized polynomial time (RP)**.

This means that it always runs in polynomial time and has a one-sided error probability.

Motivation for the Functor

To test whether a given number n is prime, a random number is chosen between 2 and $n - 2$.

Tests using modular arithmetic are performed with this number before returning **false** or **probably true**.

A test on a prime number will always return **probably true**.

A test on a composite number will sometimes return **probably true**.

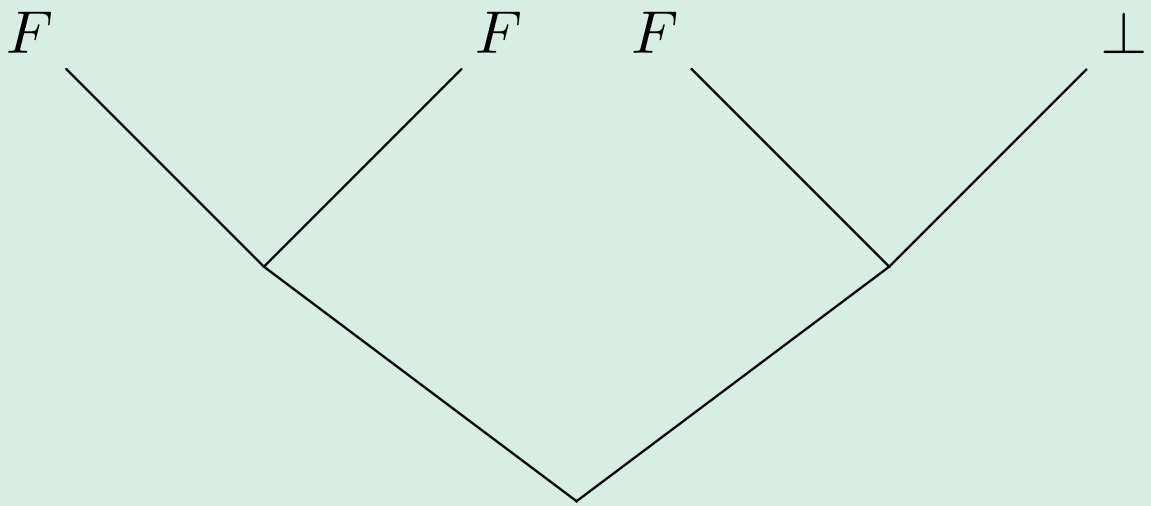
Motivation for the Functor

For a composite number, at most $\frac{1}{4}$ of the possible random choices between 2 and $n-2$ will result in the test returning **probably true**.

To minimize the error probability, we can repeat the test (choosing a new random number) only when the test returns **probably true**.

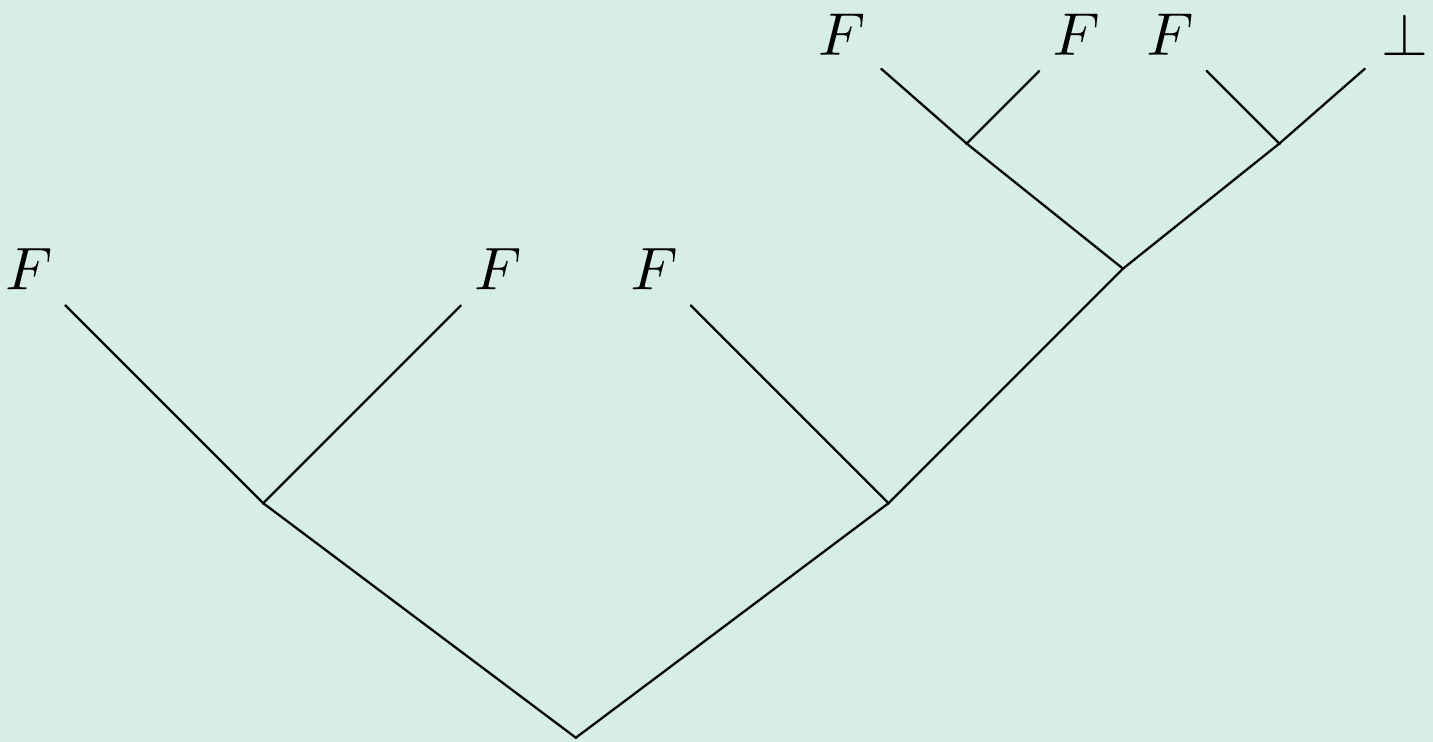
Running the test m times results in an error probability of at most $\frac{1}{4^m}$.

Simplified View of Miller-Rabin - One Iteration

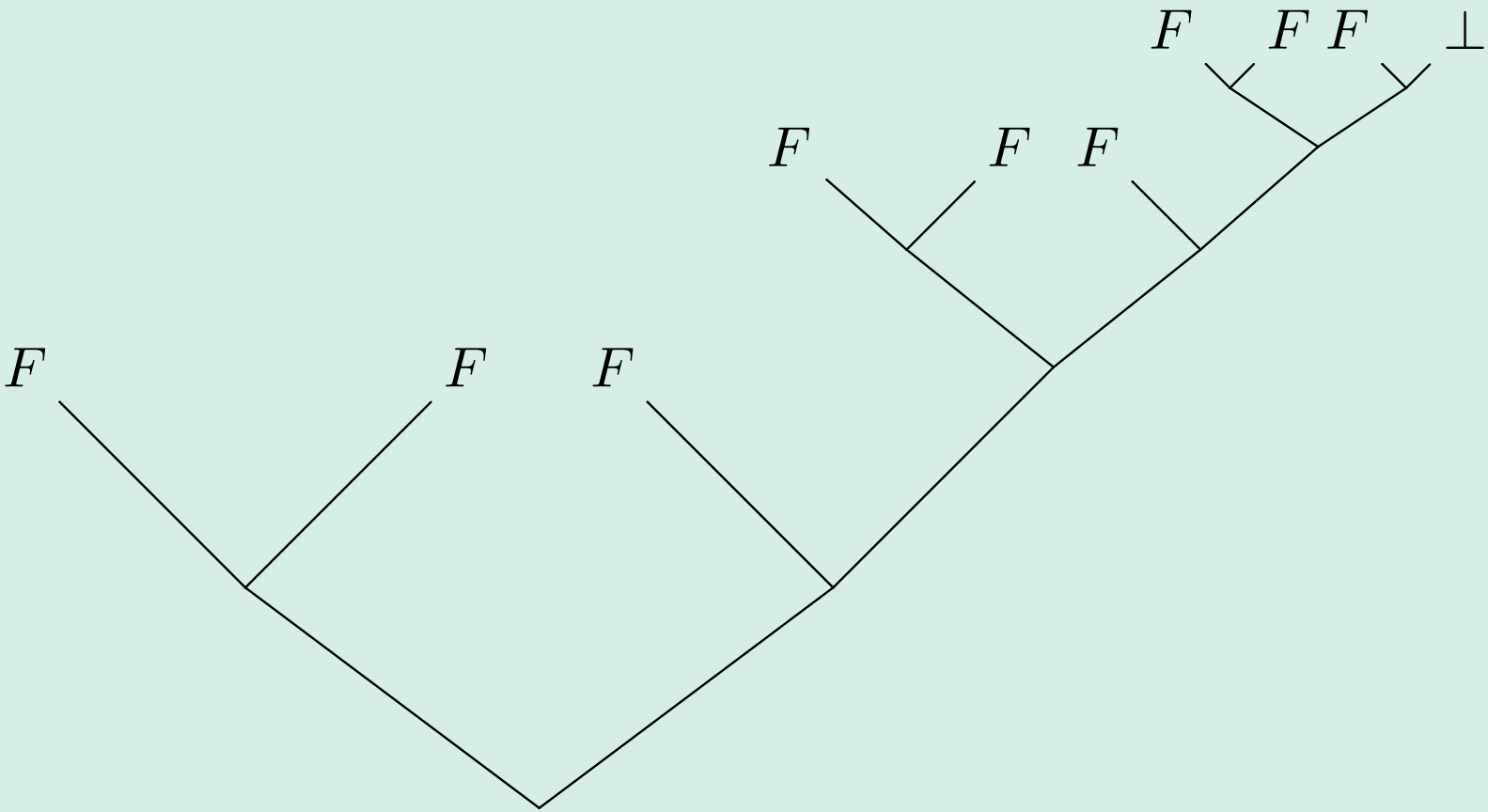


Here F represents "composite" while $\perp$ represents "probably prime", or "not sure".

Simplified View of Miller-Rabin - Two Iterations



Simplified View of Miller-Rabin - Three Iterations



The Functor Defined

Definition

An **antichain** *of* $\{0,1\}^\infty$ *is a subset of words such that no two distinct words are comparable (no word is a prefix of another word).*

Definition

An antichain M *is* **full** *if* $\{0,1\}^\omega \subseteq \uparrow M$*, or equivalently,* $M \sqsubseteq_{EM} \{0,1\}^\omega$*. Denote the full antichains by* $AC(\{0,1\}^\infty)$*.*

The Functor Defined

For a category of domains, D, the functor of random choice, RC, is defined on objects by

$$\boldsymbol{RC(D) = \left\{(M, f) \mid M \in AC(\{0,1\}^\infty), f : M \to D\right\}}$$

where f is Scott continuous (giving M the subspace topology from the Scott topology of $\{0,1\}^\infty$).

For $a : D \to D'$ and $(M, f) \in RC(D)$, we define

$$\boldsymbol{RC(a)(M, f) = (M, a \circ f)}$$

The Functor Defined

$(M, f) \sqsubseteq (N, g)$ iff

- $\boldsymbol{M \sqsubseteq_{EM} N}$
- $\boldsymbol{w \leq z \Rightarrow f(w) \sqsubseteq g(z), \forall w \in M, z \in N}$

Since the antichains are required to be full, $M \sqsubseteq_{EM} N$ is equivalent to $M \subseteq \downarrow N$, or dually, $N \subseteq \uparrow M$.

Another characterization for the order on functions is $\forall z \in N, f \circ \pi_M(z) \sqsubseteq g(z)$, where $\pi_M(z)$ sends z to the unique element of M below z.

The Functor Defined

Proposition

If D is a dcpo, then so is $RC(D)$.

Proposition

If D is a bounded complete domain, then so is $RC(D)$.

The Monad

A monad can be formed exhibiting a unit, $\eta : D \to RC(D)$, and for any morphism, $h : D \to RC(E)$, a Kleisli extension, $h^\dagger : RC(D) \to RC(E)$.

The unit is defined by

$$\eta(d) = (\epsilon, \chi_d)$$

where ϵ is the antichain only containing the empty word, with the constant function whose value is d.

Motivation for the Kleisli Extension

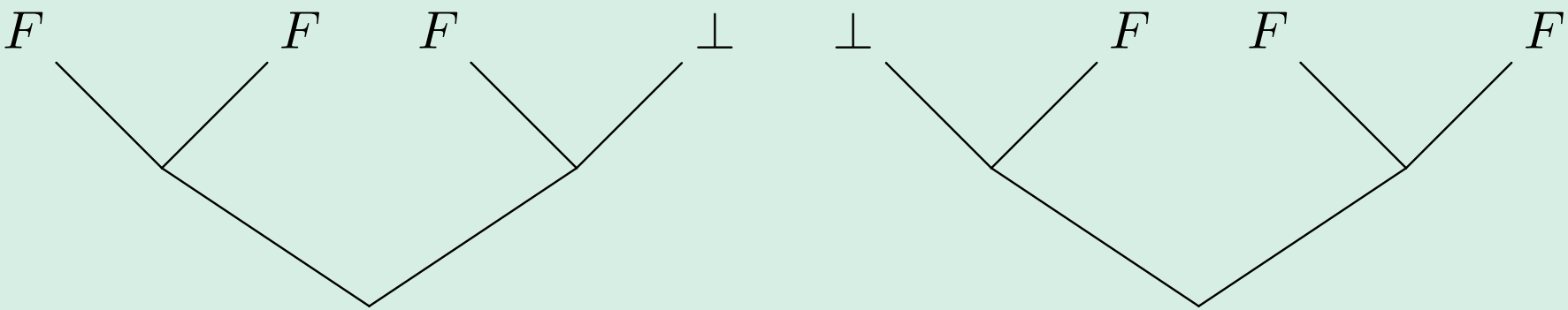
One function of the Kleisli extension is to lift binary operations.

If we have a binary operation $* : D \times E \to F$, the Kleisli extension lifts this operation to $*^\dagger : RC(D) \times RC(E) \to RC(F)$. This is achieved by setting

$$*^\dagger = (\lambda a.RC(\lambda b.a * b))^\dagger$$

Motivation for the Kleisli Extension

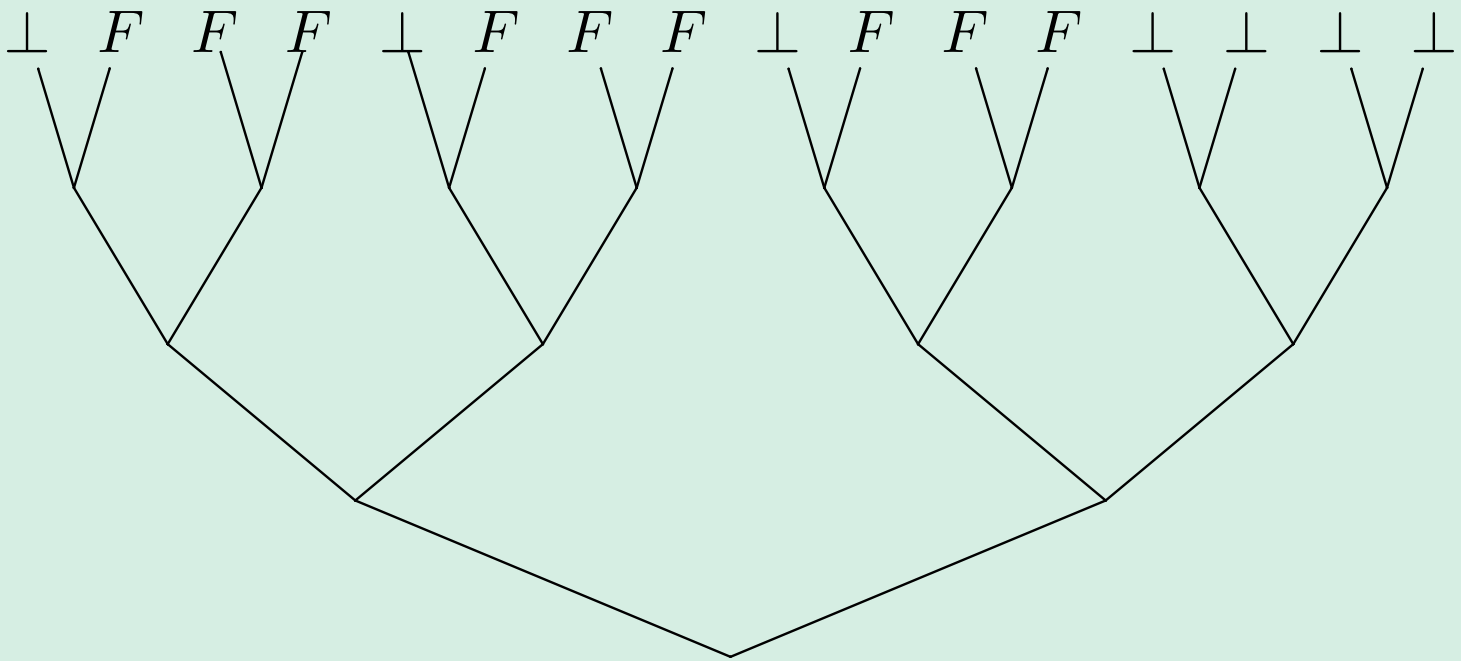
Suppose these are Miller-Rabin tests for two different composite numbers.



How should the **or** operation be lifted?

Motivation for the Kleisli Extension

It may seem natural to perform the two tests one after the other. This would result in:



The probability of error in this case is $\frac{7}{16}$.

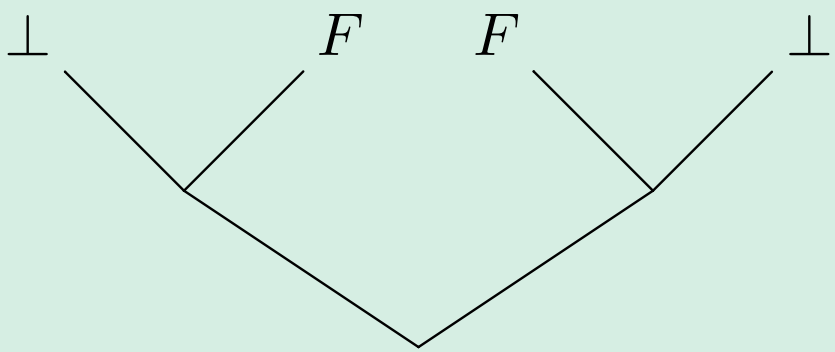
Motivation for the Kleisli Extension

However, this approach has two main problems.

- What do we do in the infinite case?
- The Kleisli extension that results in this behavior is not monotone in our order.

Motivation for the Kleisli Extension

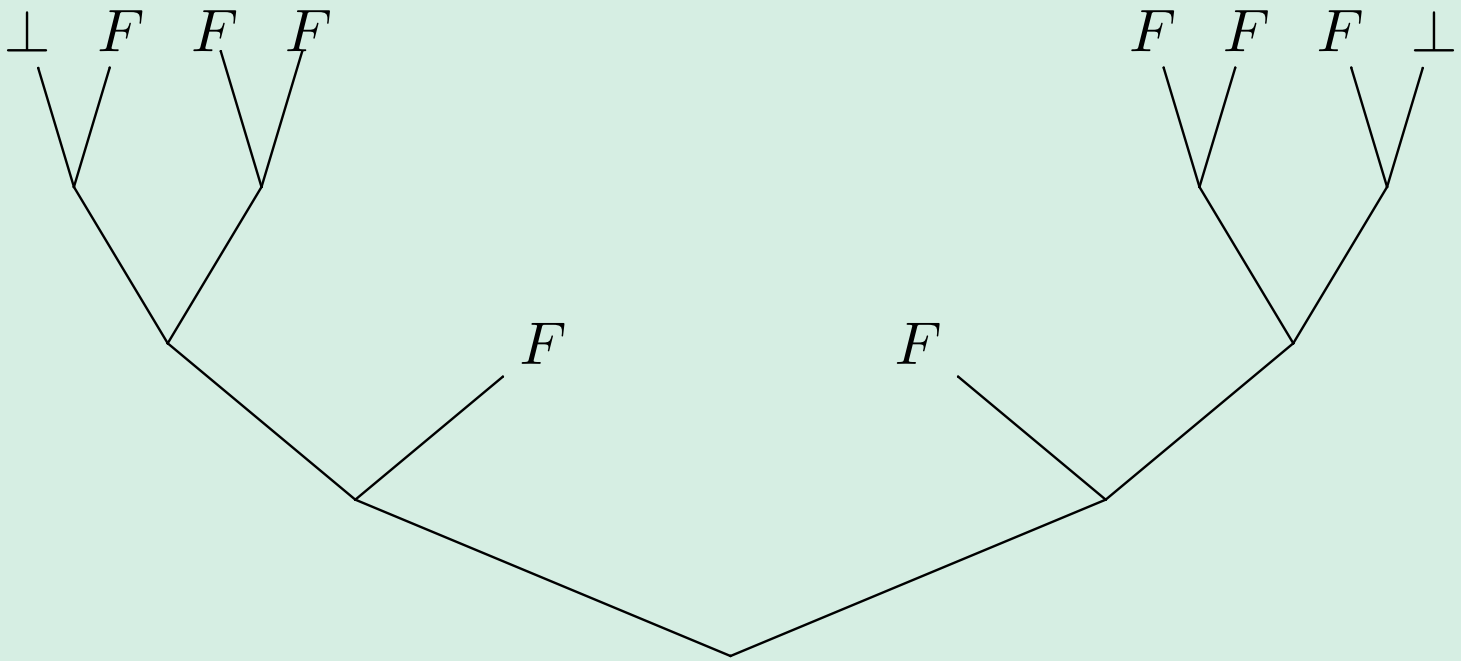
Instead of performing the tests one after the other, we will perform them concurrently, feeding the result of each coin flip to both tests. This results in:



This results in an error probability of $\frac{1}{2}$. However, this only uses two coin flips.

Motivation for the Kleisli Extension

Using four coin flips gives:



which only has an error probability of $\frac{1}{8}$.

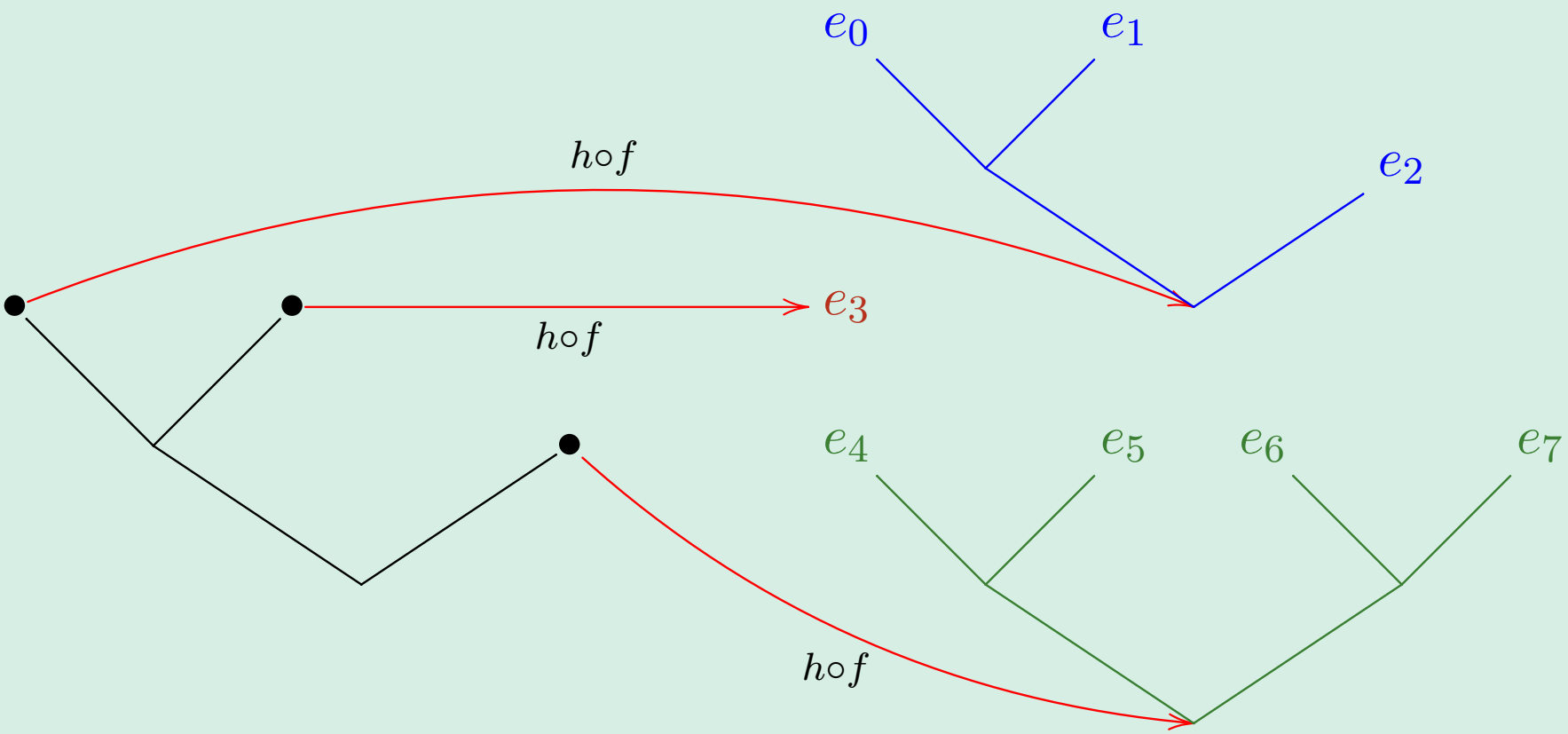
The Kleisli Extension Defined

Consider $h : D \to RC(E)$. To define the extension $h^\dagger : RC(D) \to RC(E)$ on (M, f), $h \circ f(w)$ must be considered for each $w \in M$.

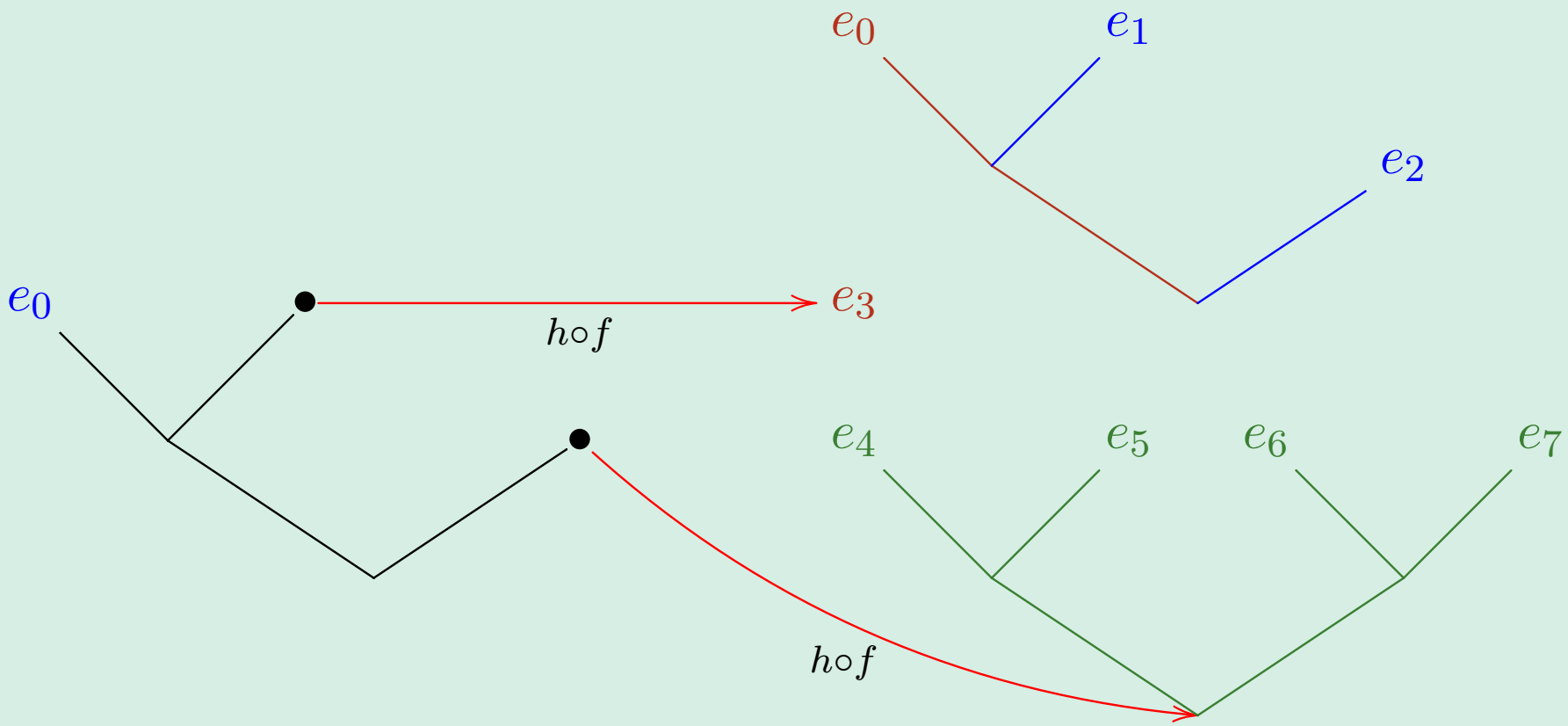
$\pi_1 \circ h \circ f(w)$ gives an antichain, but the entire antichain is not used.

Instead, the Kleisli extension only considers the part of $\pi_1 \circ h \circ f(w)$ that is on the same "branch" of the tree as w, namely $\uparrow w \cup \downarrow w$.

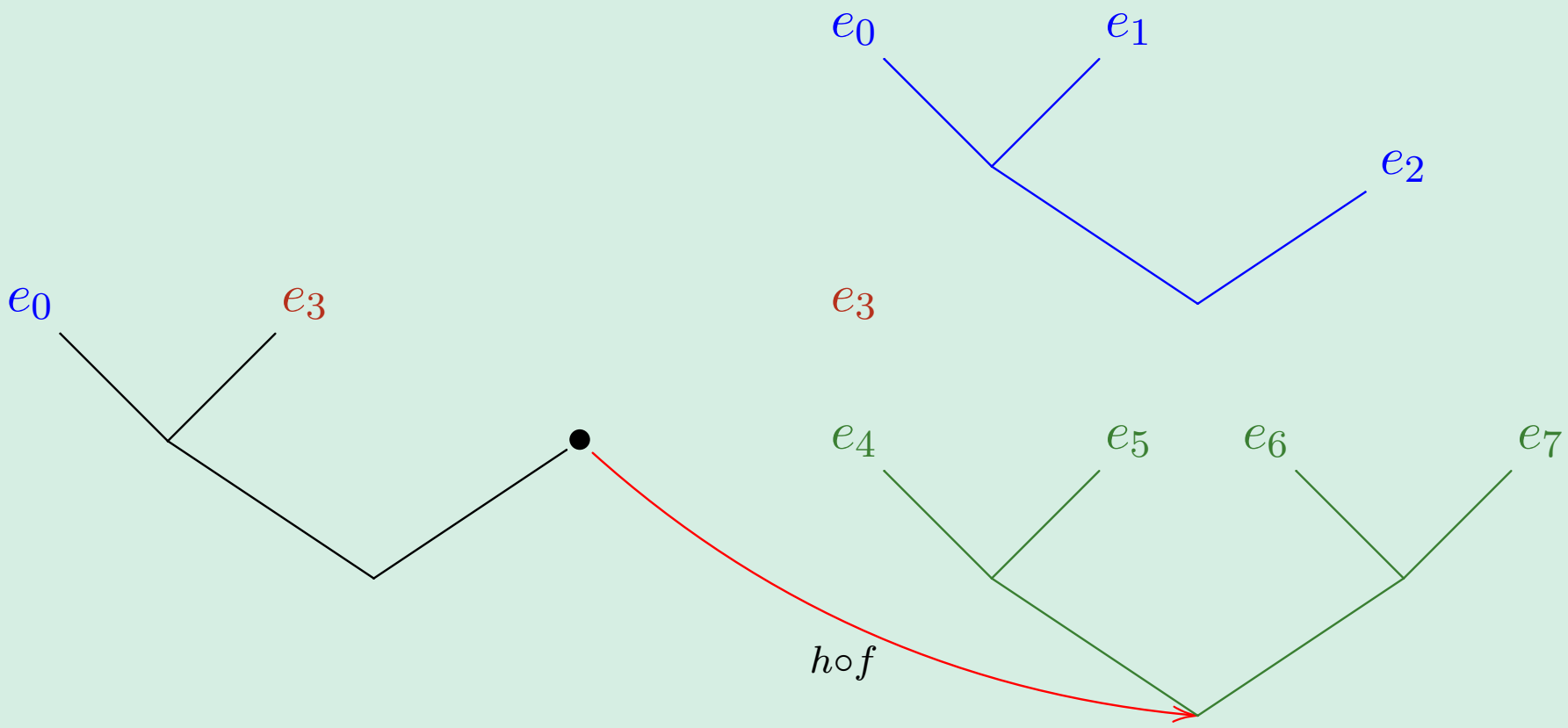
The Kleisli Extension Defined



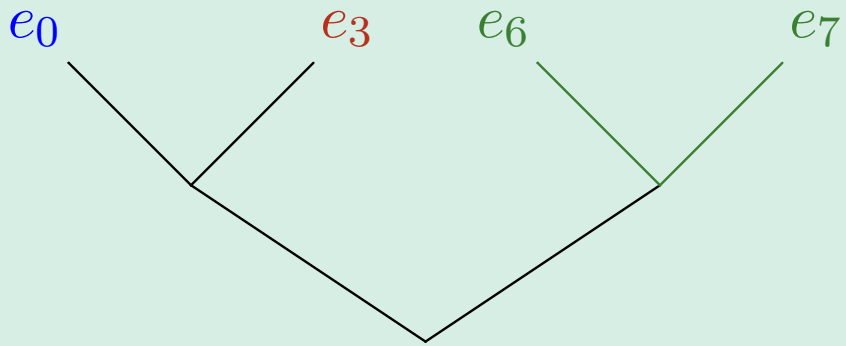
The Kleisli Extension Defined

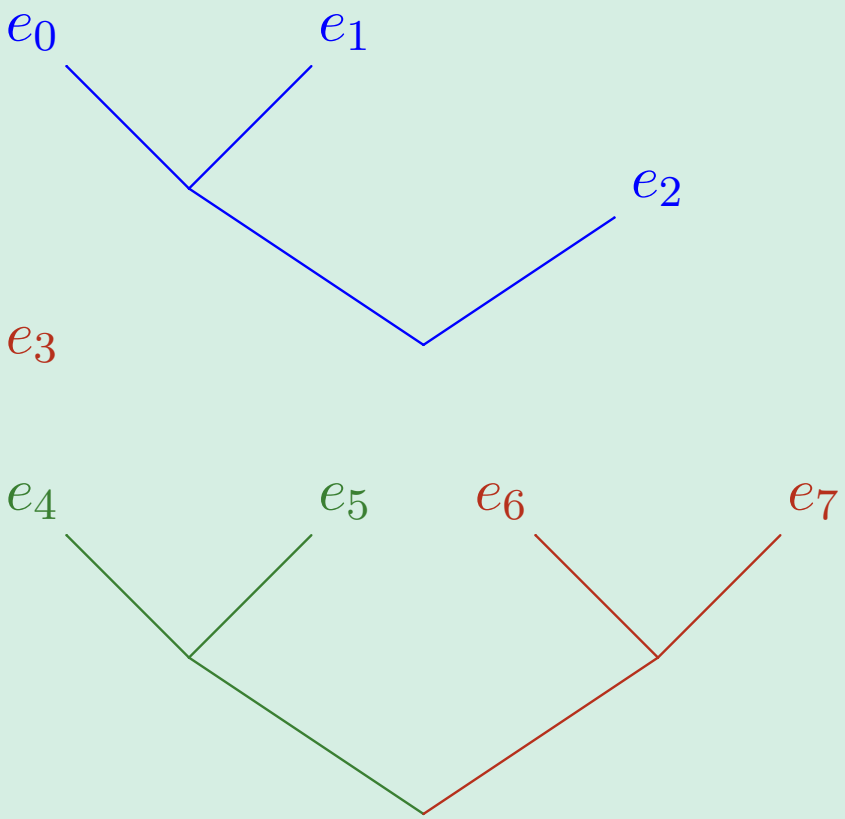


The Kleisli Extension Defined



The Kleisli Extension Defined





The Kleisli Extension Defined

$h^\dagger$ is defined as

$$\pi_1 \circ h^\dagger(M, f) = \bigcup_{w \in M} \mathbf{Min}(\uparrow w \cap \uparrow \pi_1 \circ h \circ f(w))$$

$$((\pi_2 \circ h^\dagger)(M, f))(z) = g(\pi_N(z)) \text{ where } (N, g) = h \circ f \circ \pi_M(z)$$

The Monad

Theorem

RC forms a monad in the category of bounded complete domains.

Proposition

RC shares a distributive law with the Hoare powerdomain.

Relation to Scott's Stochastic Lambda Calculus

Scott has added random variables to his $\mathcal{P}(\mathbb{N})$ model to form a stochastic lambda calculus.

Definition

A **random variable** *is a function* $\mathsf{X} : [0,1] \to \mathcal{P}(\mathbb{N})$ *where* $\{t \in [0,1] \mid n \in \mathsf{X}(t)\}$ *is Lebesgue measurable for all* n *in* $\mathcal{P}(\mathbb{N})$.

Compare this to infinite elements in our monad, which are of the form $\{0,1\}^\omega \to D$.

Relation to Scott's Stochastic Lambda Calculus

The lambda calculus has an application operation that must be lifted to these random variables.

Definition

Given two random variables $\mathsf{X}, \mathsf{Y} : [0,1] \to \mathcal{P}(\mathbb{N})$*, the application operation is defined by*

$$\mathsf{X}(\mathsf{Y})(t) = \mathsf{X}(t)(\mathsf{Y}(t))$$

This coincides with how our Kleisli extension would lift the application operation.

Concluding Remarks

Alterations can be made to the monad to obtain distributive laws with the Smyth and Plotkin powerdomains.

We have used the monad to give a semantics for a randomized PCF.

An interpreter for rPCF has been implemented in SML along with a program to perform the Miller-Rabin test.

For an operational version of our monad, the monad laws and monotonicity of the Kleisli extension have been formally proven in Isabelle.