

Pisa

29 February 2024

# EFFECTFUL TRACE SEMANTICS VIA EFFECTFUL STREAMS

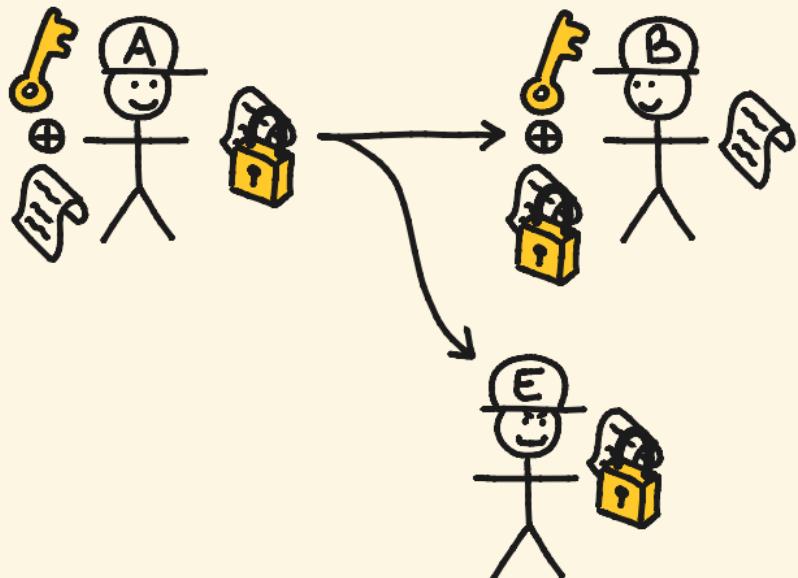
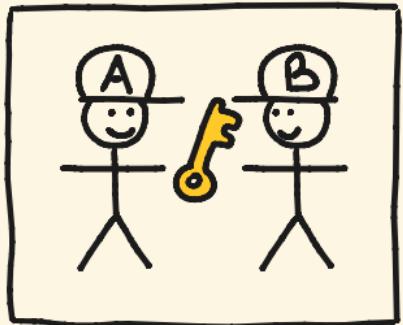
Filippo Bonchi  
Università di Pisa

Elena Di Stefano  
Università di Pisa

Mario Román  
University of Oxford

# ONE-TIME PAD PROTOCOL

1. share a key through a secure channel
2. send an encrypted message through a public channel



# REPEATING THE ONE-TIME PAD

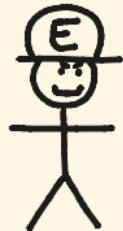
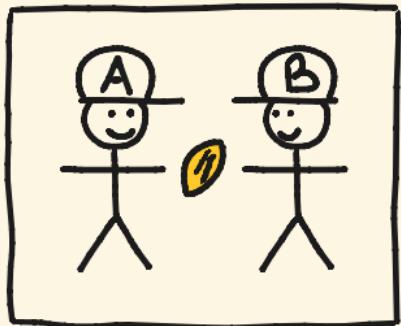
Sending  $n$  messages securely requires  $n$  private keys  
↳ not very useful

- ⇒ • privately share a seed 
- use identical pseudorandom number generator to obtain a new key for each message



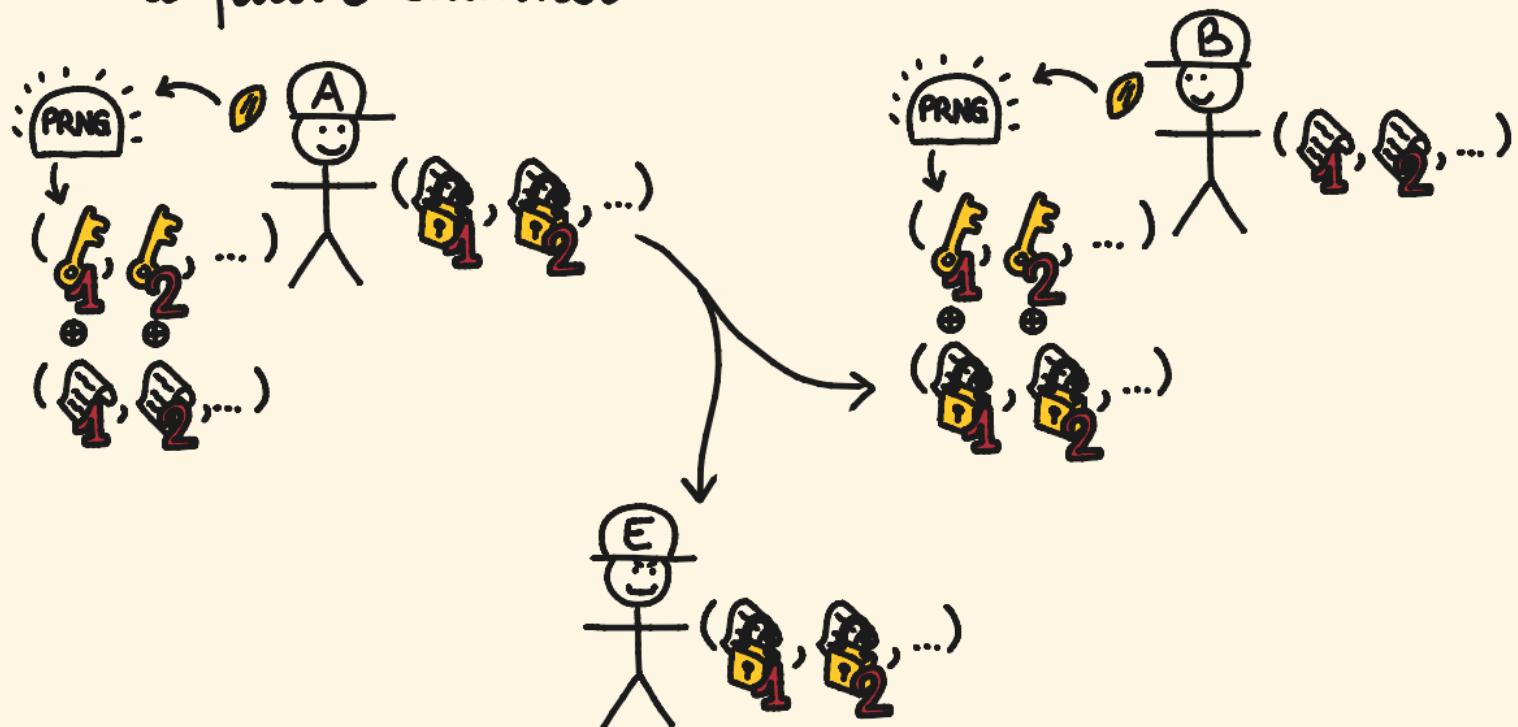
# STREAM CIPHER PROTOCOL (1)

1. share a seed through a secure channel
2. share a pseudorandom number generator



# STREAM CIPHER PROTOCOL (2)

- send a stream of encrypted messages through a public channel



## (CO)INDUCTION

- inductive data types are initial algebras  
(of polynomial functors)

**data** Nat = 0 | succ Nat

N : X → 1 + X

**data** ListNat = [] | cons Nat ListNat

L : X → 1 + Nat × X

## (CO)INDUCTION

- coinductive data types are final coalgebras  
(of polynomial functors)

codata CoNat = 0 | succ CoNat

N : X → 1 + X

codata CoListNat = [] | cons Nat CoListNat

L : X → 1 + Nat × X

codata StreamNat = cons Nat StreamNat

S : X → Nat × X

# PRACTICAL CONDUCTION

- $(-)$  :  $\text{Nat} \rightarrow \text{StreamNat}$

$$(n)^\circ := n$$

$$(n)^+ := (n)$$

- $- \cdot -$  :  $\text{Nat} \times \text{StreamNat} \rightarrow \text{StreamNat}$

$$(n \cdot x)^\circ := n \cdot x^\circ$$

$$(n \cdot x)^+ := x^+$$

- $\subseteq \subseteq \text{StreamNat} \times \text{StreamNat}$

$\rightsquigarrow$  lexicographic order

$$x \subseteq y \text{ iff } x^\circ \leq y^\circ \quad (1)$$

$$\text{and, if } x^\circ = y^\circ, \text{ then } x^+ \subseteq y^+ \quad (2)$$

# PRACTICAL COINDUCTION

CLAIM

The lexicographic order  $\leq$  on StreamNat is transitive.

COINDUCTIVE PROOF

Suppose  $x \leq y \leq z$ .

By (1),  $x^0 \leq y^0 \leq z^0$ .

Then,  $x^0 \leq z^0$  by transitivity in Nat.

Suppose  $x^0 = z^0$ . Then  $x^0 = y^0$  and  $y^0 = z^0$ .

By (2),  $x^+ \leq y^+$  and  $y^+ \leq z^+$ .

By coinduction,  $x^+ \leq z^+$ .

RECALL

$x \leq y$  iff

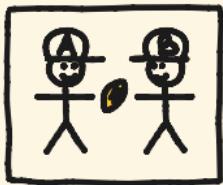
(1)  $x^0 \leq y^0$

and (2), if  $x^0 = y^0$ , then  $x^+ \leq y^+$



# STREAM CIPHER PROTOCOL: COMPONENTS

1.



$\text{rand} : I \rightarrow S$

$\text{setSeed} : S \rightsquigarrow I$

$\text{getSeedA} : I \rightsquigarrow S$

$\text{getSeedB} : I \rightsquigarrow S$

~ generate the seed

~ share the seed

~ A and B get  
the seed

2.



$\text{prng} : S \rightarrow S \otimes M$

~ pseudorandom  
number  
generator

3.

$$\begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{pmatrix} = \begin{pmatrix} \text{key}_1 \\ \text{key}_2 \\ \vdots \\ \text{key}_n \end{pmatrix}$$

$- \oplus - : M \otimes M \rightarrow M$

~ encrypt / decrypt  
messages

# STREAM CIPHER PROTOCOL CONDUCTIVELY

seedGen<sup>0</sup> = do

rand() → s

setSeed(s) ↳ ()

return()

alice(m)<sup>0</sup> = do

getSeedA() ↳ (s)

prng(s) → (s', k)

return(s', m ⊕ k)

seedGen<sup>+0</sup> = do

return()

seedGen<sup>++</sup> = seedGen<sup>+</sup>



alice(s, m)<sup>+0</sup> = do

prng(s) → (s', k)

return(s', m ⊕ k)

alice(s, m)<sup>++</sup> = alice(s, m)<sup>+</sup>

# STREAM CIPHER PROTOCOL CONDUCTIVELY



$\text{bob}(m)^\circ = \text{do}$

$\lceil \text{getSeedB}() \rightsquigarrow (s)$

$\text{prng}(s) \rightarrow (s', k)$

$\lfloor \text{return } (s', m \oplus k)$

$\text{bob}(s, m)^{+0} = \text{do}$

$\lceil \text{prng}(s) \rightarrow (s', k)$

$\lfloor \text{return } (s', m \oplus k)$

$\text{bob}(s, m)^{++} = \text{bob}(s, m)^+$



$\text{eve}(m)^\circ = \text{do}$

$\lfloor \text{return } (m)$

$\text{eve}(m)^+ = \text{eve}(m)$

# OUTLINE

- effectful categories
- effectful streams
- effectful trace semantics
- causal processes

# (PRO) MONADS FOR NOTIONS OF COMPUTATIONS

- Monads add effects to computations

↳ failure and exceptions

probability

reading/writing to memory

input/output

# EXAMPLES

- objects are sets  $A, B, \dots$
- parallel composition is cartesian product  $A \times B$
- unit object is  $1 := \{*\}$

	MORPHISMS	MONAD	
cSet	functions	identity	$A \rightarrow B$
Rel	relations	powerset	$A \rightarrow P(B)$
rStoch	stochastic partial functions	subdistributions	$A \rightarrow \mathcal{D}_{\leq 1}(B)$
State	functions with global state	state	$S \times A \rightarrow S \times B$

# Set : FUNCTIONS

Semantics for deterministic and total computations.

- morphisms of type  $A \rightarrow B$  are functions  $f: A \rightarrow B$
- composition is function composition

$$f; g : A \rightarrow B \rightarrow C$$
$$a \mapsto f(a) \mapsto g(f(a))$$

# Rel : RELATIONS

Semantics for non-deterministic computations.

The powerset monad adds non-determinism.

$$\begin{aligned} P: \text{Set} &\rightarrow \text{Set} \\ A &\mapsto \{X \subseteq A\} \end{aligned}$$

- morphisms of type  $A \rightarrow B$  are relations  $f \subseteq A \times B$ ,  
i.e. functions  $f: A \rightarrow P(B)$ .
- composition is relational composition

$$f;g(a) := \{c \in C \mid \exists b \in g(a) \wedge c \in g(b)\}$$

# p<sub>stoch</sub>: PARTIAL STOCHASTIC FUNCTIONS

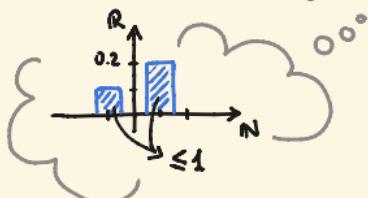
Semantics for partial stochastic computations.

The subdistribution monad adds failure and probability.

$\mathcal{D}_{\leq 1}$ : Set  $\rightarrow$  Set

$A \mapsto \{\sigma : A \rightarrow [0,1] \mid \text{supp } \sigma \text{ is finite}$

$$\wedge \sum_{a \in A} \sigma(a) \leq 1\}$$



- morphisms of type  $A \rightarrow B$  are functions  $f : A \rightarrow \mathcal{D}_{\leq 1}(B)$ .  
 $\begin{cases} f(b|a) \\ f(\perp|a) := 1 - \sum_{b \in B} f(b|a) \end{cases}$   $\rightsquigarrow$  probability of  $b$  given  $a$   
 $\rightsquigarrow$  probability of failure
- composition is  $f; g(c|a) := \sum_{b \in B} f(b|a) \cdot g(c|b)$ .

## State : STATEFUL FUNCTIONS

Semantics for computations with a global state or memory.  
The state monad adds a global state  $S$ .

$$\text{St} : \text{Set}^{\text{op}} \times \text{Set} \rightarrow \text{Set}$$
$$(A, B) \longmapsto \{ f : S \times A \rightarrow S \times B \}$$

- morphisms of type  $A \rightarrow B$  are functions  $f : S \times A \rightarrow S \times B$ .



- composition is function composition

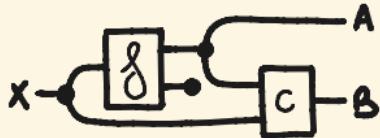
$$s \xrightarrow{\quad g \quad} s \quad (s, a) = g(f(s, a))$$

A diagram illustrating the composition of two stateful functions  $f$  and  $g$ . It shows three horizontal lines. The top line has a square box containing the letter  $f$ , with  $s$  written above it and below it. The middle line has a square box containing the letter  $g$ , with  $s$  written above it and below it. The bottom line has two labels,  $A$  on the left and  $c$  on the right, connected by a horizontal line passing through the centers of both boxes. This represents the function  $g(f(s, a))$ .

# STRING DIAGRAMS & DO-NOTATION

- Symmetric monoidal categories are theories of processes
- String diagrams and do-notation are convenient syntax

$$\nu_x ; ((f; (\nu_A \otimes \varepsilon_B)) \otimes 1_x) ; (1_A \otimes c)$$



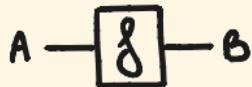
$\text{cond}(x) = \text{do}$

$f(x) \rightarrow (a, b)$   
 $c(a, x) \rightarrow b'$   
 $\text{return}(a, b')$

[Gadducci, Montanari 1998; Bonchi, Lanese, Montanari 2005;  
Corradini, Gadducci 1997; Bonchi, Sobociński, Zanasi 2015]

# STRING DIAGRAMS & DO-NOTATION

- resources  $A, B, \dots$  and processes  $f: A \rightarrow B$ , with possibly multiple inputs/outputs  $h: A \rightarrow B \otimes B'$ ,  $s: I \rightarrow B$



- sequential composition  $f; g: A \rightarrow C$  and identities  $\textcircled{1}_A: A \rightarrow A$

$\text{comp } f; g(a) = \text{do}$

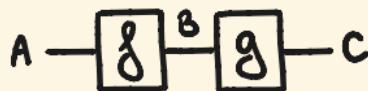
$| f(a) \rightarrow b$

$| g(b) \rightarrow c$

$| \text{return}(c)$

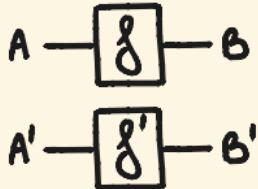
$\text{id}(a) = \text{do}$

$| \text{return}(a)$



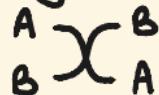
# STRING DIAGRAMS & DO-NOTATION

- parallel composition  $f \otimes f': A \otimes A' \rightarrow B \otimes B'$



tensor  $ff'(a, a') = \text{do}$   
 $f(a) \rightarrow b$   
 $f'(a') \rightarrow b'$   
return ( $b, b'$ )

- permuting resources  $\sigma_{A,B} : A \otimes B \rightarrow B \otimes A$

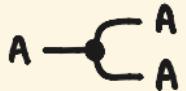


swap( $a, b$ ) = do  
return ( $b, a$ )

$$\begin{array}{c} A \\ B \end{array} \xrightarrow{\quad} \begin{array}{c} A \\ B \end{array} = \begin{array}{c} A \\ B \end{array} \xrightarrow{\quad} \begin{array}{c} B \\ A' \end{array}$$

swapNat( $a, b$ ) = do  
 $f(a) \rightarrow a'$   
return ( $b, a'$ )

# COPY AND DISCARD



A diagram illustrating the composition of two copy operations. It shows a sequence of three lines: an input line, a junction point where it splits into two lines, and another junction point where those two lines merge back into a single line. This is followed by an equals sign and another sequence of three lines: an input line, a junction point where it splits into two lines, and another junction point where those two lines merge back into a single line.

A diagram illustrating the composition of a copy operation followed by a discard operation. It shows a sequence of three lines: an input line, a junction point where it splits into two lines, and another junction point where those two lines merge back into a single line. This is followed by an equals sign and a sequence of two lines: an input line and a junction point where it splits into two lines.

A diagram illustrating the composition of two copy operations. It shows a sequence of three lines: an input line, a junction point where it splits into two lines, and another junction point where those two lines merge back into a single line. This is followed by an equals sign and a sequence of four lines: an input line, a junction point where it splits into two lines, a junction point where those two lines merge into a single line, and finally a junction point where that single line splits into two lines again.

copy(a) = do  
| return(a,a)

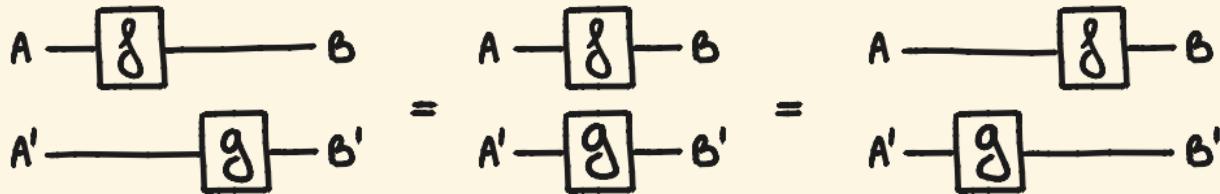
discard(a) = do  
| return()

copyAssoc(a) = do  
| return(a,a,a)

copyUnit(a) = do  
| return(a)

copyCommut(a) = do  
| return(a,a)

# THE INTERCHANGE LAW



$\text{par } f g (a, a') = \text{do}$   
 $f(a) \rightarrow b$   
 $g(a') \rightarrow b'$   
 $\text{return } (b, b')$

=

$\text{par } f g (a, a') = \text{do}$   
 $g(a') \rightarrow b'$   
 $f(a) \rightarrow b$   
 $\text{return } (b, b')$

# PREMONOIDAL CATEGORIES

Some processes do not interchange.

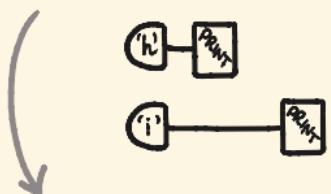
`printHI()` = do

`print('h')`  $\rightsquigarrow ()$   
`print('i')`  $\rightsquigarrow ()$   
`return()`

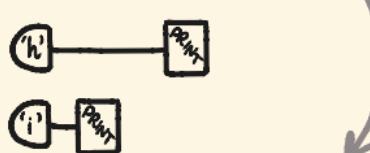
`printIH()` = do

`print('i')`  $\rightsquigarrow ()$   
`print('h')`  $\rightsquigarrow ()$   
`return()`

$\neq$



$\neq$



prints "hi"

prints "ih"

$\rightsquigarrow$  state promonads

$\rightsquigarrow$  IO monad

# STREAM CIPHER PROTOCOL (AGAIN)

seedGen<sup>(0)</sup> = do

rand() → s

setSeed(s) ↳ ()

return()

alice(m)<sup>0</sup> = do

getSeedA() ↳ (s)

prng(s) → (s', k)

return(s', m ⊕ k)

eve(m)<sup>0</sup> = do

return(m)

seedGen<sup>(+0)</sup> = do

return()

seedGen<sup>++</sup> = seedGen<sup>+</sup>

alice(s, m)<sup>+0</sup> = do

prng(s) → (s', k)

return(s', m ⊕ k)

alice(s, m)<sup>++</sup> = alice(s, m)<sup>+</sup>

eve(m)<sup>+</sup> = eve(m)

# PURE COMPUTATIONS

Pure computations are both :

- deterministic

$$\text{copy } \boxed{g} = \begin{array}{c} \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \boxed{g} \\ \text{---} \end{array}$$

$\text{copy } F(x) = \text{do}$   
 $\text{return } (\boxed{g}(x), \boxed{g}(x))$

- and total .

$$\text{discard } \boxed{g} = \text{---}$$

$\text{discard } F(x) = \text{do}$   
 $\text{return } ()$

ex  $(3 \cdot -) : \mathbb{R} \rightarrow \mathbb{R}$

non-ex Flip :  $1 \rightarrow \mathcal{D}(\{\text{H}, \text{T}\})$

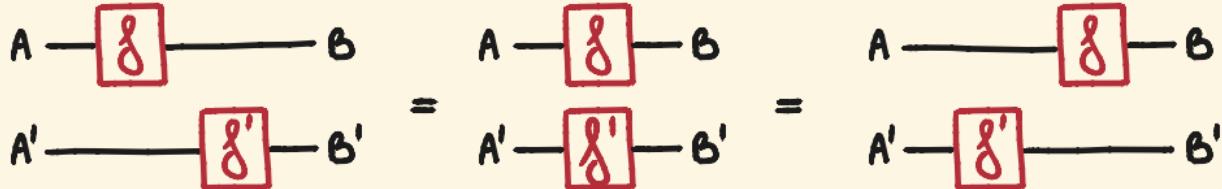
$(3/-) : \mathbb{R} \rightarrow \mathbb{R}$

$$\text{copy } \boxed{g} \neq \begin{array}{c} \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \boxed{g} \\ \text{---} \end{array}$$

$$\text{discard } \boxed{g} \neq \text{---}$$

# LOCAL COMPUTATIONS

Local computations interchange,



$\text{localF}(a, a') = \text{do}$   
 $g(a) \rightarrow b$   
 $g'(a') \rightarrow b'$   
 $\text{return } (b, b')$

$\text{localF}(a, a') = \text{do}$   
 $g'(a') \rightarrow b'$   
 $g(a) \rightarrow b$   
 $\text{return } (b, b')$



but



# EFFECTFUL COMPUTATIONS

Effectful computations may have global effects.



`printHI() = do`

`'h'() → C1`  
`'i'() → C2`  
`print(C1) ↳ ()`  
`print(C2) ↳ ()`  
`return()`

`≠`

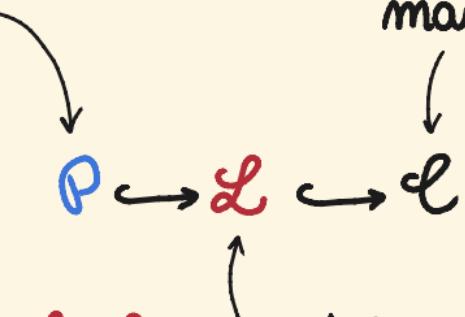
`printIH() = do`

`'h'() → C1`  
`'i'() → C2`  
`print(C2) ↳ ()`  
`print(C1) ↳ ()`  
`return()`

# EFFECTFUL COPY-DISCARD CATEGORIES

Pure computations  
can be copied  
and discarded

Effectful computations  
may have global  
effects



local computations  
interchange

# OUTLINE

- effectful categories
- effectful streams
- effectful trace semantics
- causal processes

# EFFECTFUL STREAMS

An effectful stream  $f: A \rightarrow B$  is

- a memory  $M_g \in \mathcal{L}$
- a first action  $g^\circ: A^\circ \rightarrow M_g \otimes B^\circ$  in  $\mathcal{C}$
- the rest of the action  $f^+: M_g \cdot A^+ \rightarrow B^+$

$$A - \boxed{f} - B = A^\circ - \boxed{g^\circ} - B^\circ \xrightarrow{M_g} A^+ - \boxed{f^+} - B^+$$

quotiented by the equivalence relation generated by

$$\begin{cases} g^\circ; (\pi \otimes 1) = g^\circ \\ f^+ = \pi \cdot g^+ \end{cases} \quad \text{for } \pi: M_g \rightarrow M_g \text{ in } \mathcal{L}$$

$$-\boxed{g^\circ} - \boxed{f^+} - = -\boxed{g^\circ} - \boxed{\pi} - \boxed{f^+} - \sim -\boxed{g^\circ} - \boxed{\pi} - \boxed{f^+} - = -\boxed{g^\circ} - \boxed{g^+} -$$

# COMPOSITIONAL STRUCTURE OF STREAMS

## THEOREM

- Effectful streams form an effectful category Stream.
- composition and monoidal actions are defined coinductively:  
for  $f: N_g \cdot A \rightarrow B$  and  $g: N_g \cdot B \rightarrow C$ ,

$$\begin{cases} (f;_N g)^\circ := \begin{array}{c} N_g \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} g^\circ \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} f^\circ \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} N_g \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \\ (f;_N g)^+ := f^+;_M g^+ \end{cases}$$

$$\begin{cases} (\otimes_N f)^\circ := \begin{array}{c} N_g \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} g^\circ \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} M_g \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \\ (\otimes_N f)^+ := \begin{array}{c} X^+ \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \otimes_M f^+ \end{cases}$$

# SEMANTICS FOR THE STREAM CIPHER PROTOCOL

Six two finite sets Char  $\rightsquigarrow$  messages  
Seed  $\rightsquigarrow$  seeds

Pure computations are functions

Local computations are stochastic functions

Effectful computations read from and write to a global state  
( $\text{Set}$ ,  $\text{Stoch}$ ,  $\text{SeedStoch}$ )

$\text{SeedStoch}$  is the Kleisli category of a promonad  
that adds a global state  $\text{Seed} \otimes \text{Seed}$ :

$\text{SeedStoch}(A, B) := \text{Stoch}(\text{Seed} \otimes \text{Seed} \otimes A, \text{Seed} \otimes \text{Seed} \otimes B)$

# SEMANTICS FOR THE STREAM CIPHER PROTOCOL

Semantics for pure and local computations.

$\llbracket - \oplus - \rrbracket := \text{xor} : \text{Char} \otimes \text{Char} \rightarrow \text{Char}$   $\rightsquigarrow$  bitwise xor

Char  $\xrightarrow{\oplus}$  Char in  $\text{cSet}$



$\llbracket \text{rand} \rrbracket := \text{unif} : 1 \rightarrow \mathcal{D}(\text{Seed})$   $\rightsquigarrow$  uniform distribution

Seed  $\xrightarrow{\text{U}}$  in  $\text{cStoch}$

$\llbracket \text{prng} \rrbracket : \text{Seed} \rightarrow \text{Seed} \times \text{Char}$   $\rightsquigarrow$  use the seed to generate a key

Seed  $\xrightarrow{\text{PR}}$  Seed, Char in  $\text{cStoch}$



# SEMANTICS FOR THE STREAM CIPHER PROTOCOL

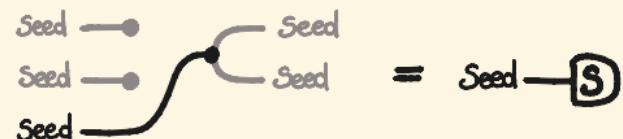
Semantics for effectful computations.

[setSeed] : Seed<sup>3</sup> → Seed<sup>2</sup>

Seed ↼ 1

↝ copy the seed to  
the global state

[setSeed] :=



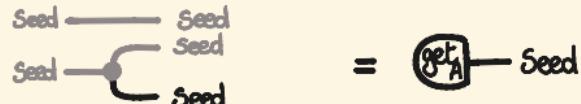
in `SeedStock`

[getSeedA], [getSeedB] : Seed<sup>2</sup> → Seed<sup>3</sup>

1 ↼ Seed

↝ alice and bob  
get their seeds

[getSeedA] :=



[getSeedB] :=

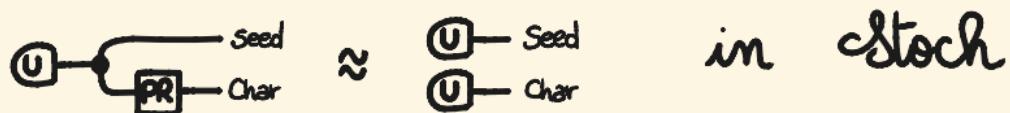


# ON PSEUDORANDOM NUMBERS

There are no perfect pseudorandom number generators in `cStoch`.

## ASSUMPTION

There is an approximate pseudorandom number generator,  
 $\text{PR} : \text{Seed} \rightarrow \mathcal{D}(\text{Char})$  such that :



# SEMANTICS FOR THE STREAM CIPHER PROTOCOL

- $\text{seedGen} = \text{SEED} : \mathbb{I} \rightarrow \mathbb{I}$  in Stream

$$\text{seedGen}^{\circ} := \begin{array}{c} \text{Seed} \\ \text{Seed} \end{array} \xrightarrow{\quad} \text{U} \xrightarrow{\quad} \begin{array}{c} \text{Seed} \\ \text{Seed} \end{array} = \text{U} \text{---} \text{S}$$

$$\text{seedGen}^{+0} := \begin{array}{c} \text{Seed} \\ \text{Seed} \end{array} \xlongequal{\quad} \begin{array}{c} \text{Seed} \\ \text{Seed} \end{array} = \square$$

$$\text{seedGen}^{++} = \text{seedGen}^+$$

- $\text{eve} = \text{Char} \xrightarrow{\text{eve}} \text{Char} : \text{Char} \rightarrow \text{Char}$  in Stream

$$\text{eve}^{\circ} := \begin{array}{c} \text{Seed} \\ \text{Seed} \end{array} \xlongequal{\quad} \begin{array}{c} \text{Char} \\ \text{Char} \end{array} = \text{Char} \text{---} \text{Char}$$

$$\text{eve}^+ = \text{eve}$$

# SEMANTICS FOR THE STREAM CIPHER PROTOCOL

•  $\text{alice} = \text{Char} \xrightarrow{\text{ALICE}} \text{Char} : \text{Char} \rightarrow \text{Char}$  in Stream

•  $\text{bob} = \text{Char} \xrightarrow{\text{Bob}} \text{Char} : \text{Char} \rightarrow \text{Char}$  in Stream

$$\text{alice}^o := \begin{array}{c} \text{Seed} \\ \text{Seed} \\ \text{Seed} \end{array} \xrightarrow{\text{PR}} \text{Char} = \text{get}_A \xrightarrow{\text{PR}} \text{Char}$$

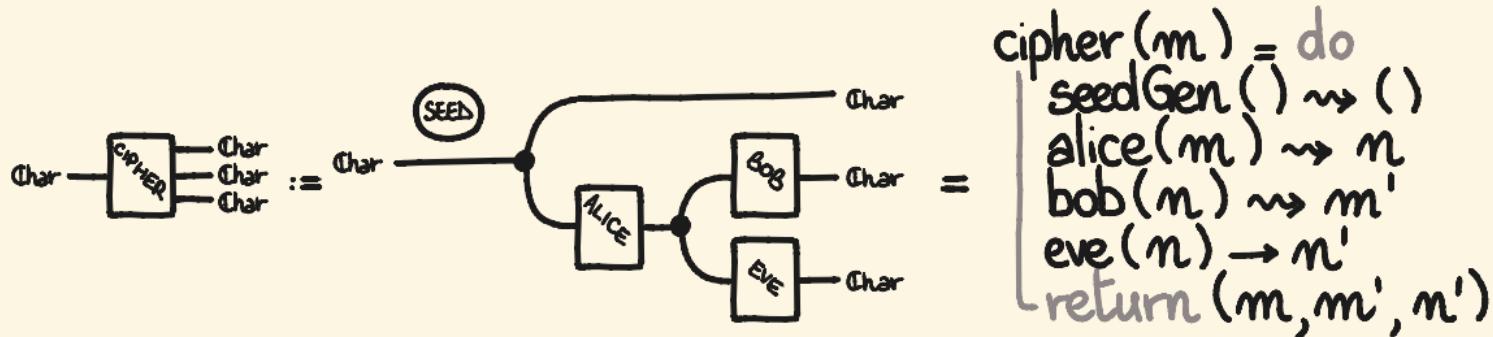
$$\text{bob}^o := \begin{array}{c} \text{Seed} \\ \text{Seed} \\ \text{Seed} \\ \text{Seed} \end{array} \xrightarrow{\text{PR}} \text{Char} = \text{get}_B \xrightarrow{\text{PR}} \text{Char}$$

$$\text{alice}^{+o} = \text{bob}^{+o} := \begin{array}{c} \text{Seed} \\ \text{Seed} \\ \text{Seed} \end{array} \xrightarrow{\text{PR}} \text{Char} = \text{Seed} \xrightarrow{\text{PR}} \text{Char}$$

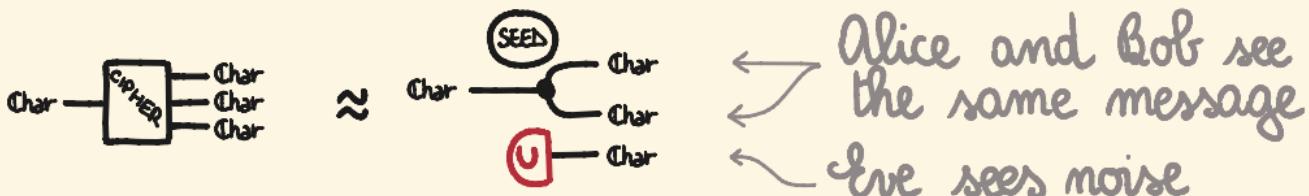
$$\text{alice}^{++} = \text{alice}^+$$

$$\text{bob}^{++} = \text{bob}^+$$

# SEMANTICS FOR THE STREAM CIPHER PROTOCOL



## SECURITY OF THE PROTOCOL



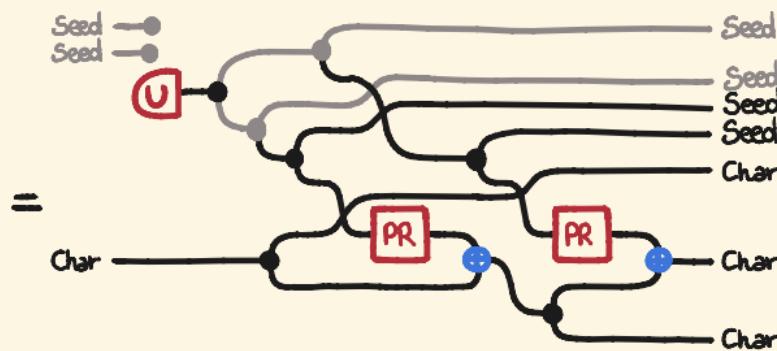
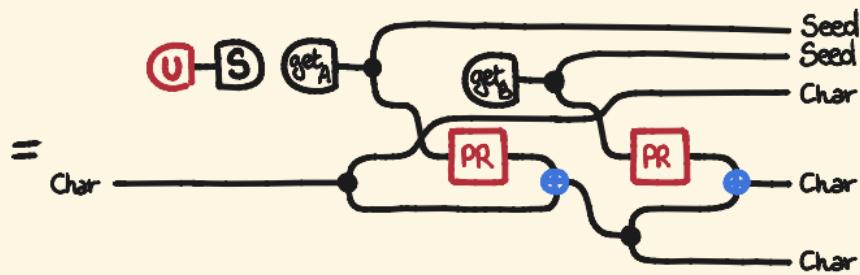
# STREAM CIPHER IS SECURE

Proceed by coinduction.

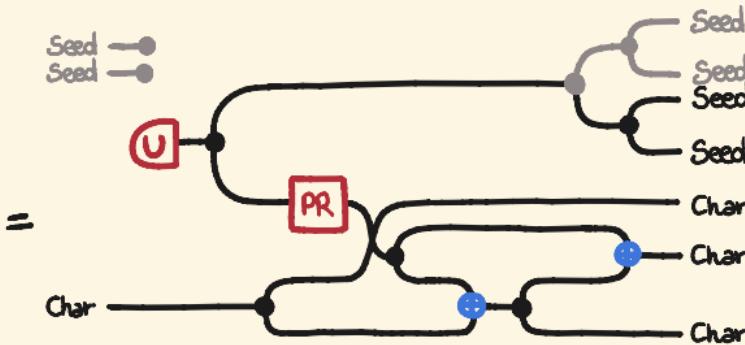
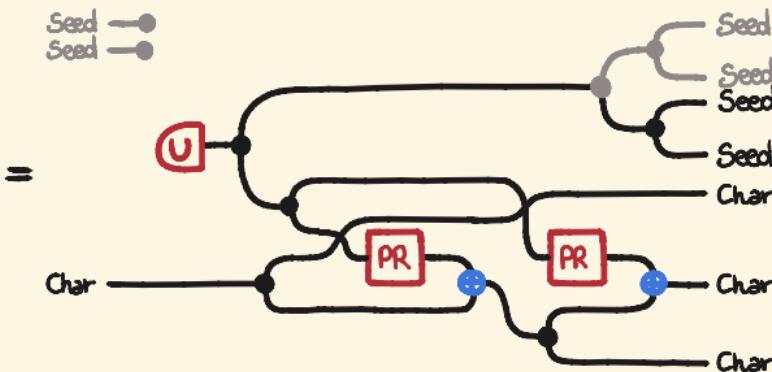
cipher<sup>0</sup>

=  $\begin{cases} \text{cipher}(m)^0 = \text{do} \\ \quad \text{rand}() \rightarrow s \\ \quad \text{setSeed}(s) \rightsquigarrow () \\ \quad \text{getSeedA}() \rightsquigarrow s_A \\ \quad \text{prng}(s_A) \rightarrow (s'_A, k_A) \\ \quad \text{getSeedB}() \rightsquigarrow s_B \\ \quad \text{prng}(s_B) \rightarrow (s'_B, k_B) \\ \quad \text{return } (m, s'_A, m \oplus k_A \oplus k_B, s'_B, m \oplus k_A) \end{cases}$

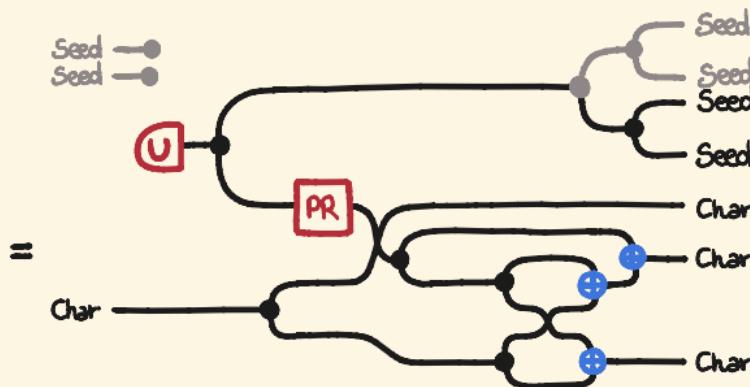
# STREAM CIPHER IS SECURE



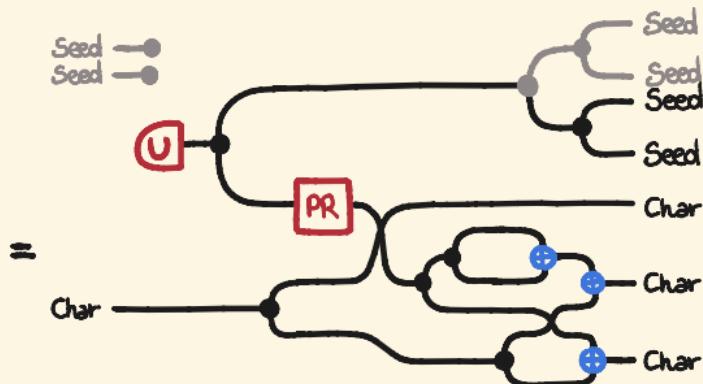
# STREAM CIPHER IS SECURE



# STREAM CIPHER IS SECURE

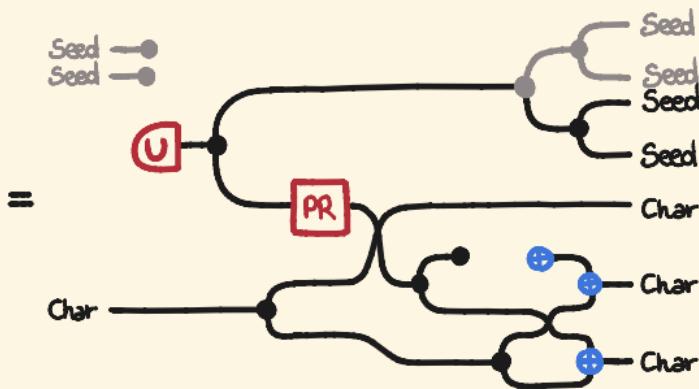


xor is deterministic

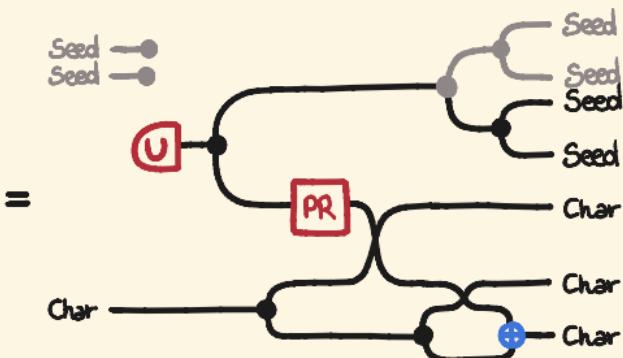


by associativity of copy  
and xor

# STREAM CIPHER IS SECURE

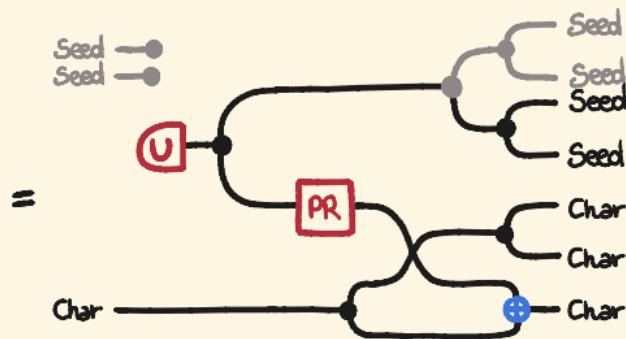


xor is nihilpotent

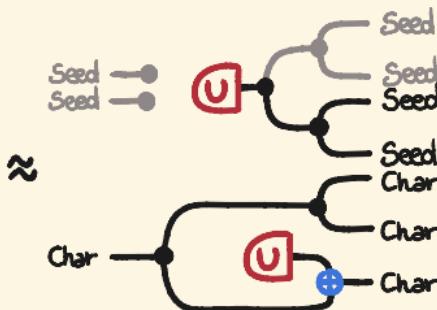


by unitality of copy  
and xor

# STREAM CIPHER IS SECURE

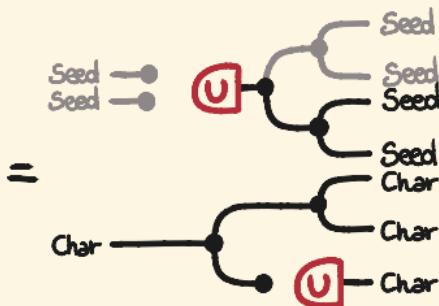


by associativity of copy

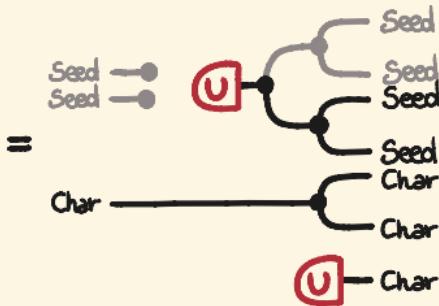


by assumption

# STREAM CIPHER IS SECURE



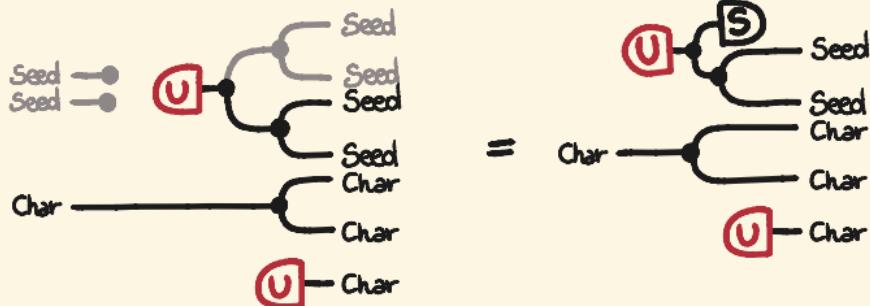
the uniform distribution is a  
Sweedler integral for xor



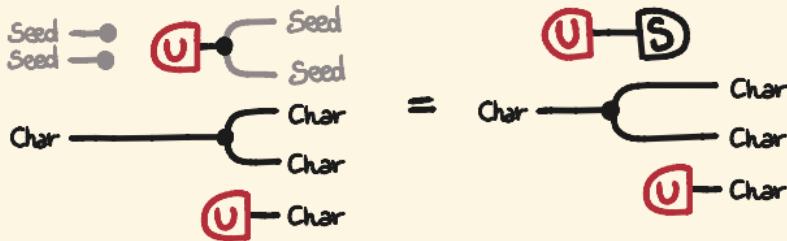
by unitality of copy

# STREAM CIPHER IS SECURE

$\text{cipher}^\circ \approx$



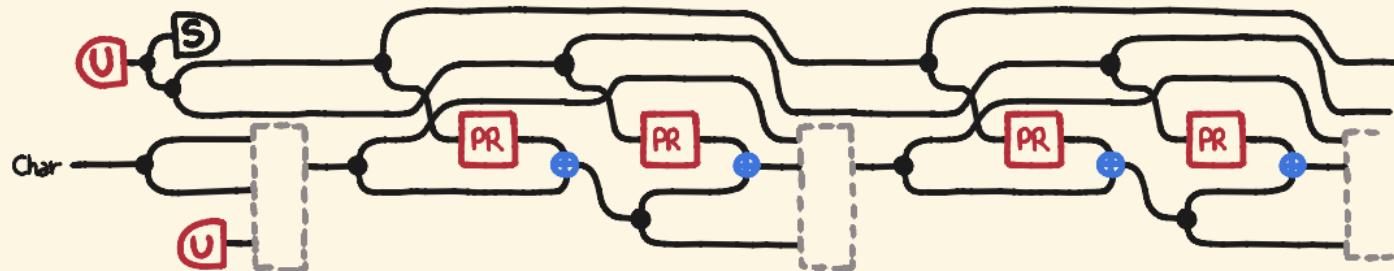
$\text{secure}^\circ :=$



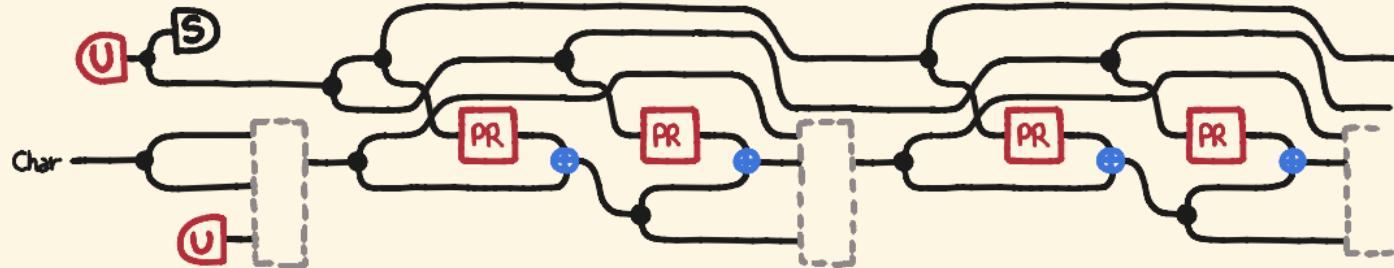
# STREAM CIPHER IS SECURE

cipher

=

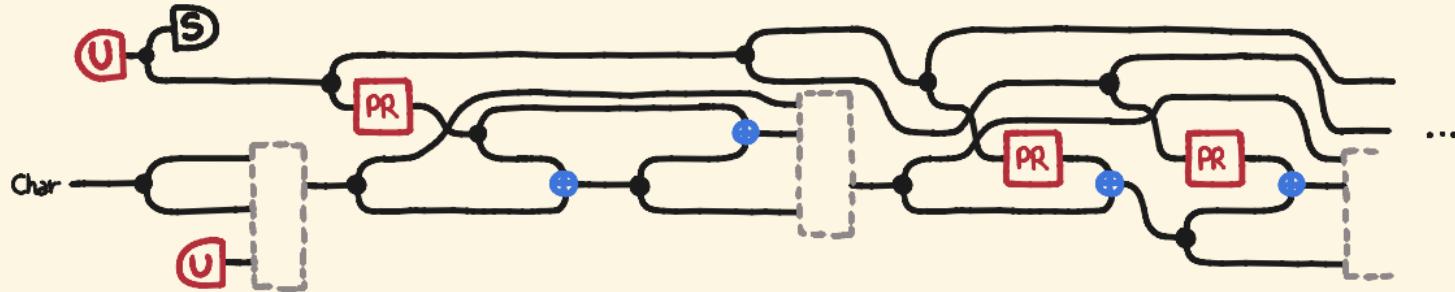


= (by sliding)

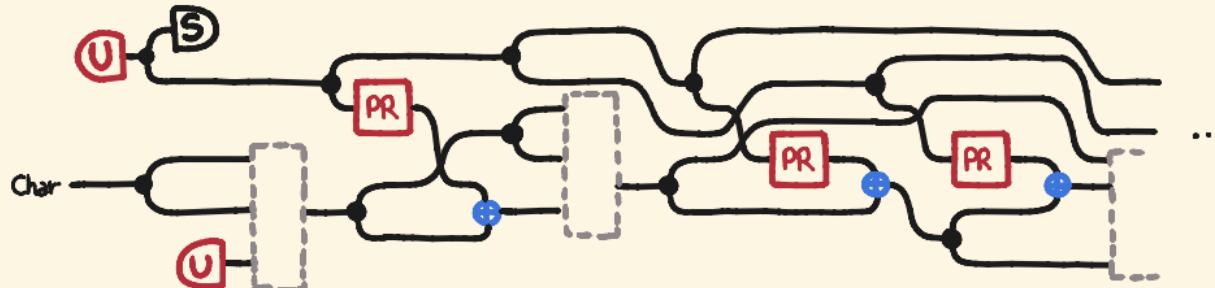


# STREAM CIPHER IS SECURE

= (pseudorandom is deterministic)

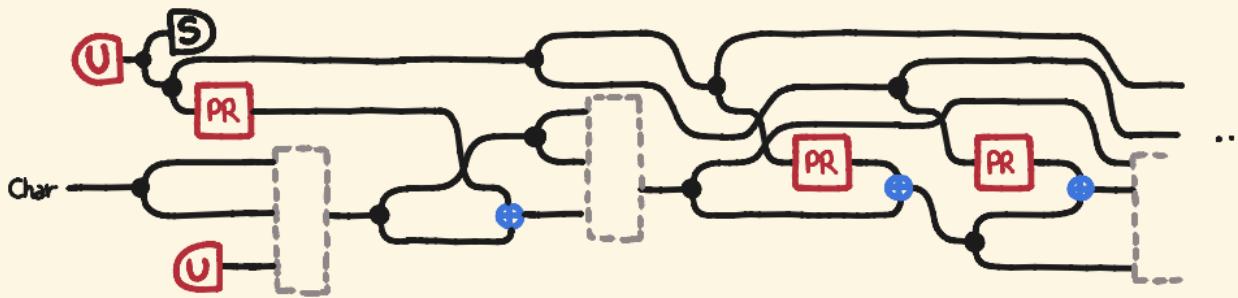


= (xor is deterministic and nihilpotent)

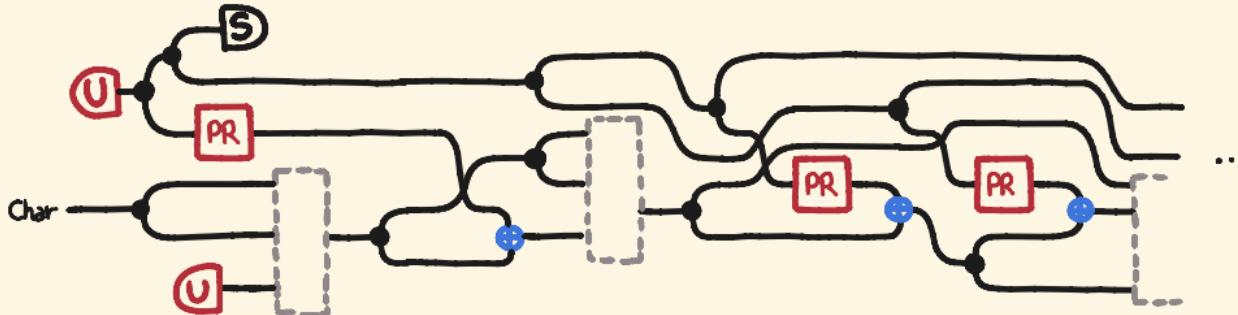


# STREAM CIPHER IS SECURE

= (by sliding)

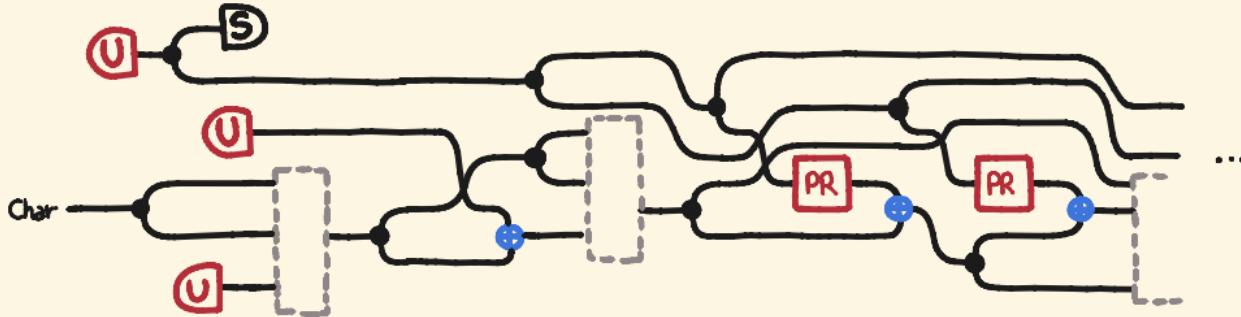


= (by associativity)

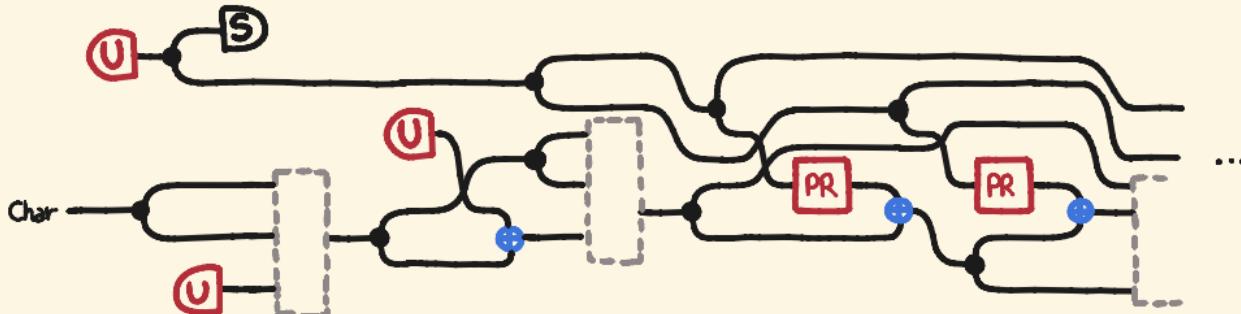


# STREAM CIPHER IS SECURE

≈ (by assumption on pseudorandom)

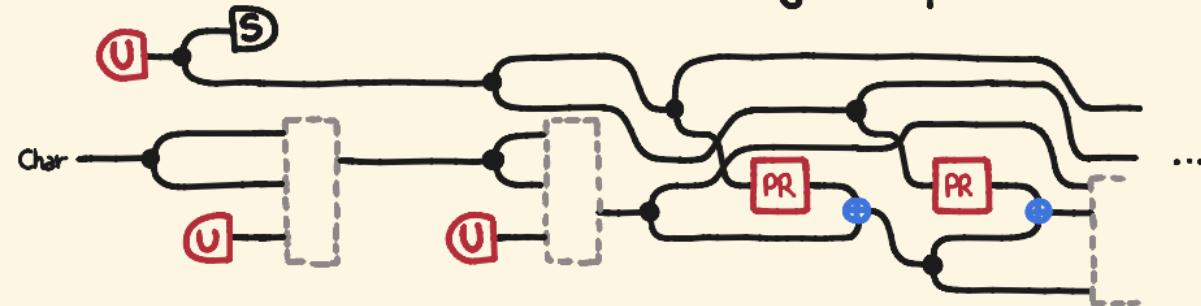


= (by sliding)

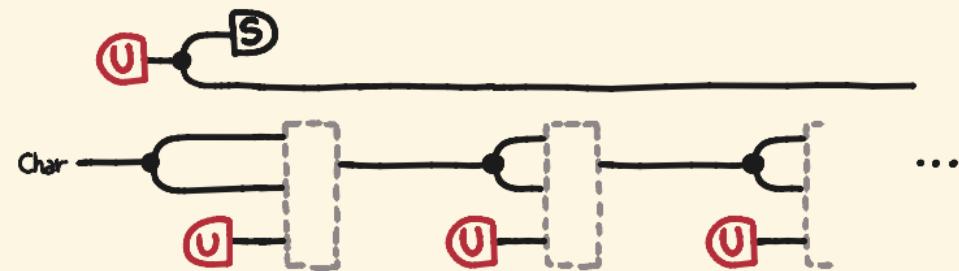


# STREAM CIPHER IS SECURE

= ( unif is a Sweedler integral for xor )

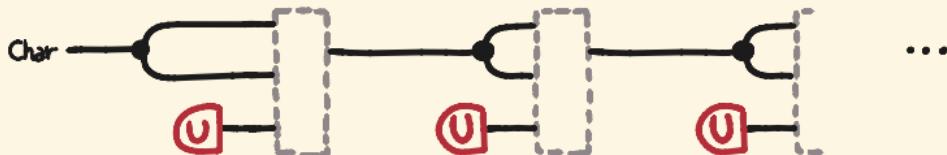


≈ ( by coinduction )

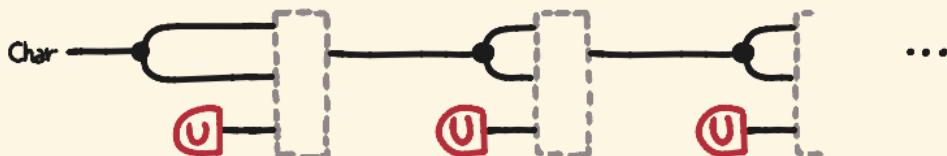


# STREAM CIPHER IS SECURE

= (by coinduction)



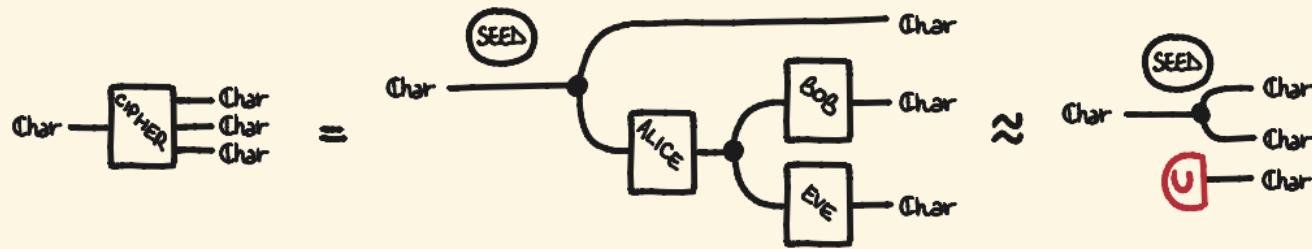
= (by unitality)



= secure

# STREAM CIPHER IS SECURE

We have shown



using sliding and coinduction.

# OUTLINE

- effectful categories
- effectful streams
- effectful trace semantics
- causal processes

# SYSTEMS THAT KEEP OPERATING

$\text{alice}(m)^\circ = \text{do}$

$\text{getSeedA}() \rightsquigarrow (s)$

$\text{prng}(s) \rightarrow (s', k)$

$\text{return } (s', m \oplus k)$

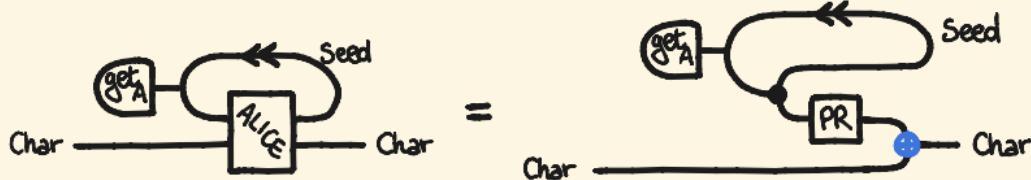


$\text{alice}(s, m)^{\circ 0} = \text{do}$

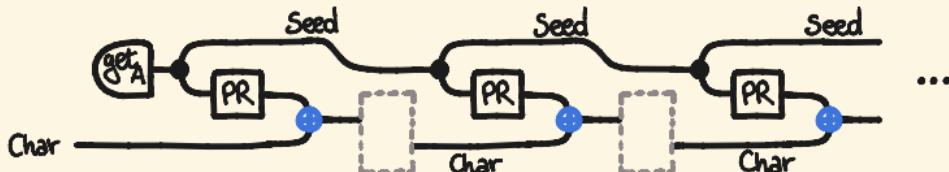
$\text{prng}(s) \rightarrow (s', k)$

$\text{return } (s', m \oplus k)$

$\text{alice}(s, m)^{++} = \text{alice}(s, m)^+$



unrolling  
 $\rightsquigarrow$



# EFFECTFUL MEALY MACHINES

A Mealy machine  $(f, S, s_0) : A \rightarrow B$  in  $\mathcal{C}$  with state space  $S$ , inputs  $A$  and outputs  $B$  is a morphism

$$f : S \otimes A \rightarrow S \otimes B$$

$$\begin{array}{c} S \\ A \xrightarrow{f} B \end{array}$$

with an initial state

$$s_0 : I \rightarrow S$$

$$\begin{array}{c} \textcircled{A} \\ \dashv \end{array} S$$

A morphism of Mealy machines  $u : (f, S, s_0) \rightarrow (g, T, t_0)$   
is a pure morphism  $u : S \rightarrow T$  in  $\mathcal{P}$

such that

$$\begin{array}{c} S \\ A \xrightarrow{f} B \end{array} \xrightarrow{u} \begin{array}{c} T \\ A \xrightarrow{g} B \end{array}$$

$$\begin{array}{c} \textcircled{s_0} \\ \dashv \end{array} \xrightarrow{u} \begin{array}{c} \textcircled{t_0} \\ \dashv \end{array} T$$

[cf. Katis, Sabadini, Walters 1997]

# EFFECTFUL CATEGORY OF MEALY MACHINES

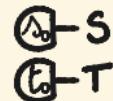
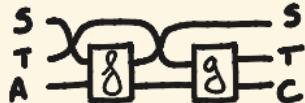
Mealy is an effectful category where

- objects are the objects of  $\mathcal{C}$
- morphisms  $(f, S, s) : A \rightarrow B$  are Mealy machines quotiented by pure isomorphisms  $u : S \xrightarrow{\cong} T$

$$\begin{array}{c} S \\ \text{---} \\ A \end{array} \xrightarrow{\quad f \quad} \begin{array}{c} T \\ \text{---} \\ B \end{array} = \begin{array}{c} S - u \\ \text{---} \\ A \end{array} \xrightarrow{\quad g \quad} \begin{array}{c} T \\ \text{---} \\ B \end{array}$$

$$\begin{array}{c} \text{---} \\ \text{---} \end{array} \xrightarrow{\quad u \quad} \begin{array}{c} T \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \end{array} \xrightarrow{\quad t_0 \quad} T$$

- composition tensors the state spaces



# COMPOSITIONAL TRACE SEMANTICS

## THEOREM

There is an effectful functor

$$\text{Tr} : \text{Mealy} \rightarrow \text{Stream}$$

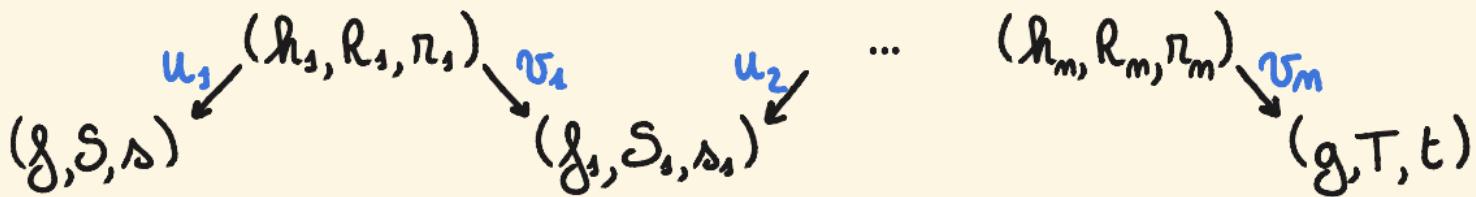
$$A \mapsto (A) = (A, A, \dots)$$

$$\begin{aligned} S_A &= \boxed{A} \xrightarrow{S_B} \boxed{B} \mapsto A \xrightarrow{\quad} \boxed{A} \xrightarrow{S_B} \boxed{(A)} \xrightarrow{S_B} \boxed{(B)} \\ &= A \xrightarrow{\quad} \boxed{A} \xrightarrow{S_B} \boxed{B} \xrightarrow{A} \boxed{A} \xrightarrow{S_B} \boxed{B} \xrightarrow{A} \boxed{B} \dots \end{aligned}$$

→ in Rel these traces coincide with the classical traces

# BISIMULATION

Two effectful Mealy machines  $(f, S, s), (g, T, t) : A \rightarrow B$  are bisimilar if they belong to the same connected component in  $\text{Mealy}(A, B)$ :



# COALGEBRAIC BISIMULATION

## PROPOSITION

When  $\mathcal{C} = \text{Kl}(M)$ , for a commutative monad preserving weak pullbacks,  
then  $(f, S, s)$  and  $(g, T, t)$  are bisimilar iff  
they have the same bisimulation quotient,  
i.e. there is  $(h, Q, q)$  with morphisms

$$(f, S, s) \xrightarrow{u} (h, Q, q) \xleftarrow{v} (g, T, t) .$$

## EXAMPLES

- $\text{Set}$
- $\text{Rel}$
- $\text{probstoch}$

# BISIMULATION $\Rightarrow$ TRACE EQUIVALENCE

## THEOREM

If two effectful Mealy machines are bisimilar,  
then they are trace equivalent.

## PROOF SKETCH

$$u : (f, S, s_0) \rightarrow (g, T, t_0)$$

$$\begin{aligned} \Rightarrow (\text{tr}(g, T, t_0))^{\circ} &= (t_0 \otimes \mathbb{1}) ; g \\ &= ((s_0 ; u) \otimes \mathbb{1}) ; g \\ &= (s_0 \otimes \mathbb{1}) ; f ; (u \otimes \mathbb{1}) \\ &= (\text{tr}(f, S, s_0))^{\circ} ; (u \otimes \mathbb{1}) \end{aligned}$$

by coinduction,  $u \cdot (\text{tr}(g, T, t_0))^+ \sim (\text{tr}(f, S, s_0))^+$

□

# OUTLINE

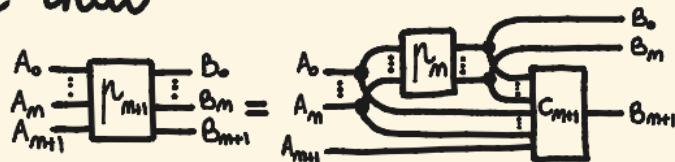
- effectful categories
- effectful streams
- effectful trace semantics
- causal processes

# CAUSAL PROCESSES

A causal process  $p: A \rightarrow B$  in a copy-discard category  $\mathcal{C}$  is a family of morphisms

$$p_m : A_0 \otimes \cdots \otimes A_m \rightarrow B_0 \otimes \cdots \otimes B_m$$

such that



for some  $C_{m+1}: B_0 \otimes \cdots \otimes B_m \otimes A_0 \otimes \cdots \otimes A_m \otimes A_{m+1} \rightarrow B_{m+1}$

## THEOREM

causal processes form a monoidal category  $\text{Proc}$   
when  $\mathcal{C}$  has quasi-total conditionals.

[cf. Ramey 1958; Sprunger & Katsumata 2019]

# CAUSAL PROCESSES ARE STREAMS

## THEOREM

Consider  $(\text{funcl}, \text{tot cl}, cl)$ .

If  $cl$  has quasi-total conditionals and ranges,  
 $\text{Proc} \simeq \text{Stream}$ .

## EXAMPLES

- Set
- Rel
- pStoch
- Par
- Stoch

[ cf. Löecke & Spekkens 2012 ;  
Elho & Jacobs 2019 ;  
Fritz 2020 ;  
Corradini & Gadducci 1999 ;  
Fritz, Gadducci, Perrone, Trotta 2023 ]

# TRACES ARE EFFECTFUL TRACES

Compute the traces of a Mealy machine

$$(f, S, s) : A \rightarrow B$$

in some known cases.

$(b_0, \dots, b_m)$  is a trace of  $(a_0, \dots, a_m)$

Set if  $s_0 = s$  and  $\forall k \leq m \quad (s_{k+1}, b_k) = f(s_k, a_k)$

Rel if  $\exists (s_0, \dots, s_{m+1}) \quad s_0 \in S$   
and  $\forall k \leq m \quad (s_{k+1}, b_k) \in f(s_k, a_k)$

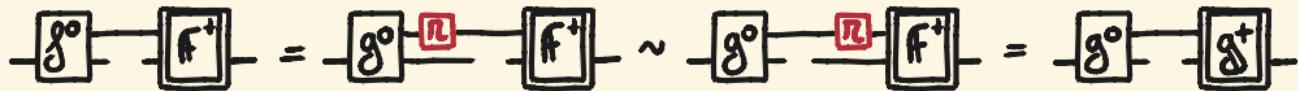
pStoch with probability  $\sum_{(s_0, \dots, s_{m+1})} s(s_0 | *) \cdot \prod_{k \leq m} f(s_{k+1}, b_k | s_k, a_k)$

# SUMMARY

- formal compositional semantics for effectful stream computations
- trace equivalence and bisimulation of effectful Mealy machines
- characterisation as causal stream processes

# FUTURE WORK

- coinduction up-to dinaturality



- Rel with explicit failure
- equality in cStL implies bisimulation



- distance instead of equivalence relation for security

