Friendly Corecursion in Isabelle/HOL

Andrei Popescu
Middlesex University London

Joint work with Jasmin Blanchette, Aymeric Bouzy,
Andreas Lochbihler and Dmitriy Traytel

CALCO/MFPS 2019, London

Overview

Part 1: What Is

Part 2: What We Do

Part 1: What Is

Running Example: Streams of Numbers

$\mathsf{Str} = \mathbb{N}^{\mathbb{N}}$

Use x, y, z for numbers and xs, ys, zs for streams (of numbers)

Each stream xs has the form $xs_0, xs_1, xs_2, \ldots$

Constructor $\# : \mathbb{N} \times \mathsf{Str} \to \mathsf{Str}$

Selectors $\mathsf{hd} : \mathsf{Str} \to \mathbb{N}$, $\mathsf{tl} : \mathsf{Str} \to \mathsf{Str}$

Destructor $\$ = \langle \mathsf{hd}, \mathsf{tl} \rangle : \mathsf{Str} \to \mathbb{N} \times \mathsf{Str}$

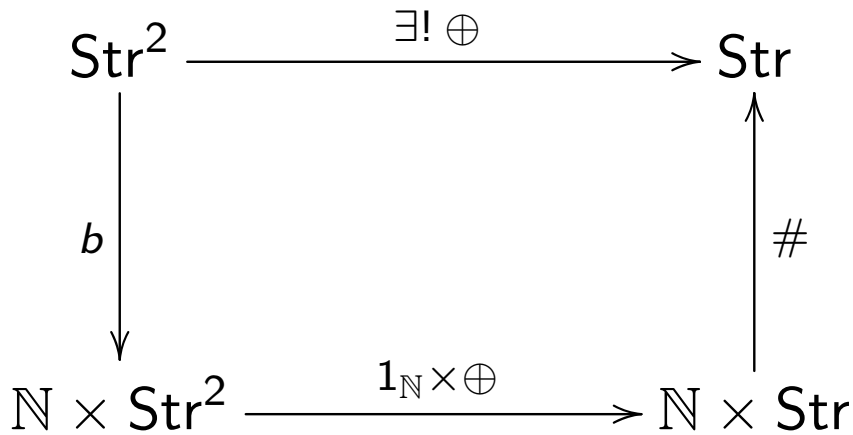
$\#$ and $\$$ are mutually inverse bijections.

$$\oplus : \mathsf{Str}^2 \to \mathsf{Str}$$

$$xs \oplus ys = (\mathsf{hd}(xs) + \mathsf{hd}(ys)) \# (\mathsf{tl}(xs) \oplus \mathsf{tl}(ys))$$

$$\oplus : \mathsf{Str}^2 \to \mathsf{Str}$$

$$xs \oplus ys = (\mathsf{hd}(xs) + \mathsf{hd}(ys)) \# (\mathsf{tl}(xs) \oplus \mathsf{tl}(ys))$$

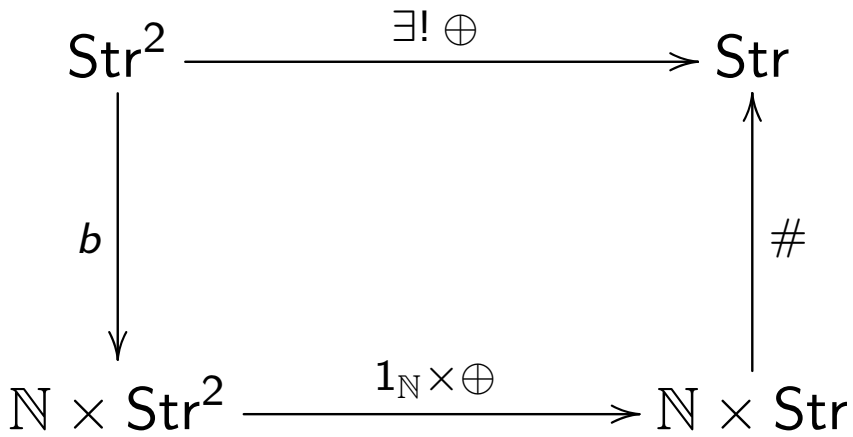


where $b = \lambda(xs, ys).\ (\mathsf{hd}(xs) + \mathsf{hd}(ys),\ (\mathsf{tl}(xs), \mathsf{tl}(ys)))$
is $\oplus$'s "blueprint"

Primitive Corecursion

$$\oplus : \mathsf{Str}^2 \to \mathsf{Str}$$

$$xs \oplus ys = (\mathsf{hd}(xs) + \mathsf{hd}(ys)) \# (\mathsf{tl}(xs) \oplus \mathsf{tl}(ys))$$



where $b = \lambda(xs, ys).\ (\mathsf{hd}(xs) + \mathsf{hd}(ys),\ (\mathsf{tl}(xs), \mathsf{tl}(ys)))$
is $\oplus$'s "blueprint"

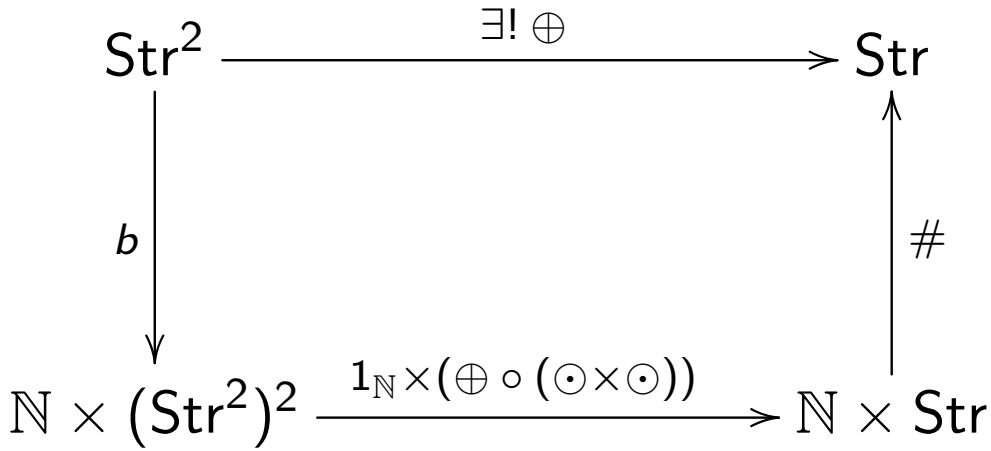
$$\odot : \mathsf{Str}^2 \to \mathsf{Str}$$

$$\begin{array}{lcl} xs \odot ys & = & (\mathsf{hd}(xs) \cdot \mathsf{hd}(ys)) \,\# \\ & & (xs \odot \mathsf{tl}(ys)) \oplus (\mathsf{tl}(xs) \odot ys) \end{array}$$

Corecursion Up To $\oplus$

$$\odot : \mathsf{Str}^2 \to \mathsf{Str}$$

$$\begin{aligned} xs \odot ys &= (\mathsf{hd}(xs) \cdot \mathsf{hd}(ys)) \,\#\, \\ &\quad (xs \odot \mathsf{tl}(ys)) \oplus (\mathsf{tl}(xs) \odot ys) \end{aligned}$$



$$\begin{aligned} \text{where } b &= \lambda(xs, ys).\ \ (\mathsf{hd}(xs) \cdot \mathsf{hd}(ys), \\ &\qquad\qquad ((xs, \mathsf{tl}(ys)), (\mathsf{tl}(xs), ys))) \end{aligned}$$

$$f : \mathsf{Str}^3 \to \mathsf{Str}$$

$$f(xs, ys, zs) = \text{let } us = \mathsf{tl}(\mathsf{tl}(\mathsf{hd}(xs)\#\mathsf{zip}(zs, \mathsf{tl}(\mathsf{tl}(xs))))) \text{ in}$$
$$(\mathsf{hd}(us) \cdot \mathsf{hd}(\mathsf{tl}(ys))) \#$$
$$((f(xs, zs, us) \odot us) \oplus (f(xs, xs, ys) \oplus f(us, ys, zs)))$$

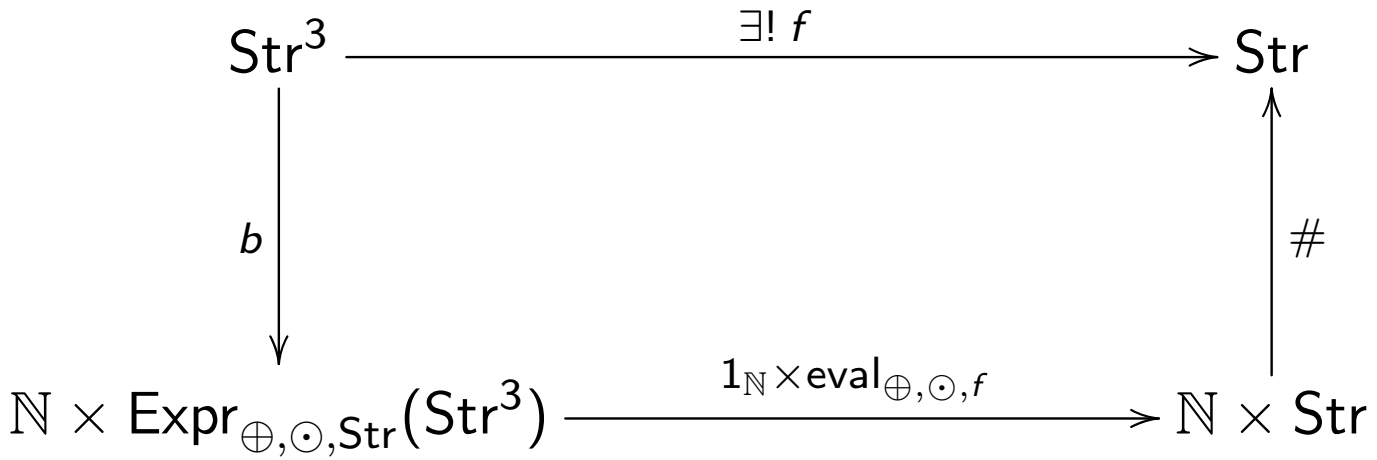
$$f : \mathsf{Str}^3 \to \mathsf{Str}$$

$$f(xs, ys, zs) \;=\; \text{let } us = \quad \ldots \quad \text{in}$$
$$(\mathsf{hd}(us) \cdot \mathsf{hd}(\mathsf{tl}(ys))) \,\#$$
$$((f(xs, zs, us) \odot us) \oplus (f(xs, xs, ys) \oplus f(us, ys, zs)))$$

Corecursion Up To $\{\oplus, \odot\}^*$

$$f : \mathrm{Str}^3 \to \mathrm{Str}$$

$$f(xs, ys, zs) = \text{let } us = \quad \ldots \quad \text{in}$$
$$(\mathrm{hd}(us) \cdot \mathrm{hd}(\mathrm{tl}(ys))) \,\#$$
$$((f(xs, zs, us) \odot us) \oplus (f(xs, xs, ys) \oplus f(us, ys, zs)))$$



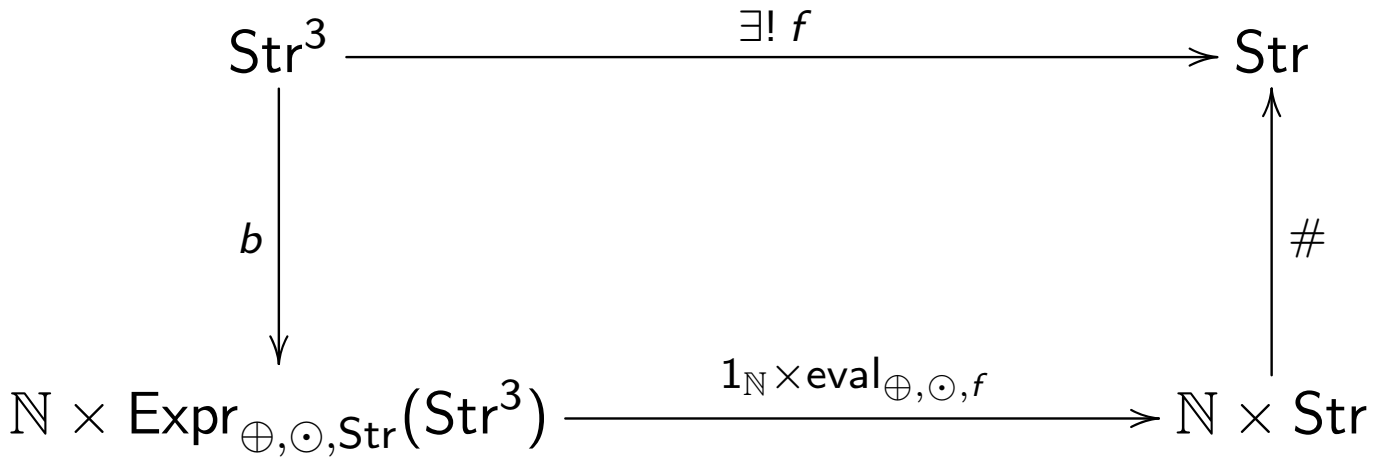
$\mathrm{Expr}_{\oplus,\odot,\mathrm{Str}}(X)$ formal expressions with $\oplus$, $\odot$, consts Str, vars X

$\mathrm{eval}_{\oplus,\odot,f} : \mathrm{Expr}_{\oplus,\odot,\mathrm{Str}}(\mathrm{Str}^3) \to \mathrm{Str}$ evaluator

Corecursion Up To $\{\oplus, \odot\}^*$

$$f : \mathrm{Str}^3 \to \mathrm{Str}$$

$$f(xs, ys, zs) = \text{let } us = \quad \ldots \quad \text{in}$$
$$(\mathrm{hd}(us) \cdot \mathrm{hd}(\mathrm{tl}(ys))) \,\#$$
$$((f(xs, zs, us) \odot us) \oplus (f(xs, xs, ys) \oplus f(us, ys, zs)))$$

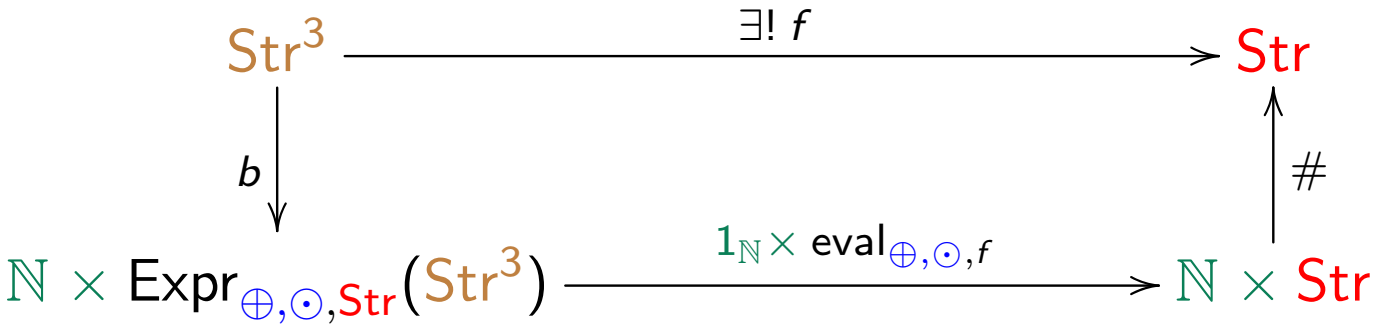


$\mathrm{Expr}_{\oplus,\odot,\mathrm{Str}}(X)$ formal expressions with $\oplus$, $\odot$, consts Str, vars X
$\mathrm{eval}_{\oplus,\odot,f} : \mathrm{Expr}_{\oplus,\odot,\mathrm{Str}}(\mathrm{Str}^3) \to \mathrm{Str}$ evaluator

$$(\mathrm{Var}(xs, zs, us) \boxed{\odot} \mathrm{Ct}(us)) \boxed{\oplus} (\mathrm{Var}(xs, xs, ys) \boxed{\oplus} \mathrm{Var}(us, ys, zs))$$

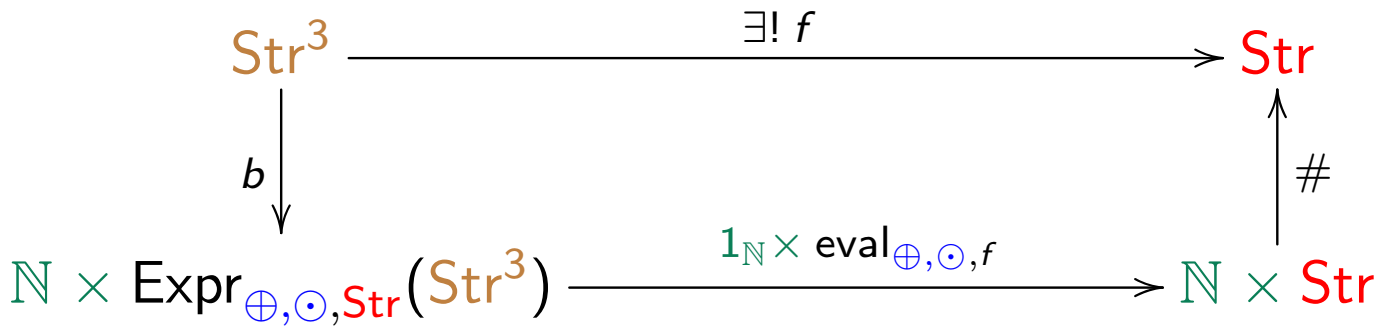
Corecursion Up To $\{\oplus, \odot\}^*$: Generalization

Given up-to-friendly operators $\oplus : \mathrm{Str}^2 \to \mathrm{Str}$, $\odot : \mathrm{Str}^2 \to \mathrm{Str}$
... can define $f : \mathrm{Str}^3 \to \mathrm{Str}$ by choosing a blueprint b:

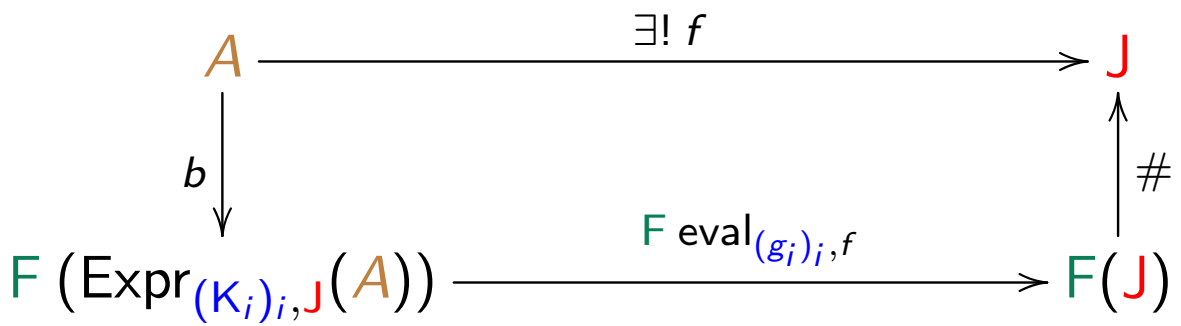


Corecursion Up To $\{\oplus, \odot\}^*$: Generalization

Given up-to-friendly operators $\oplus : \mathrm{Str}^2 \to \mathrm{Str}$, $\odot : \mathrm{Str}^2 \to \mathrm{Str}$
... can define $f : \mathrm{Str}^3 \to \mathrm{Str}$ by choosing a blueprint b:

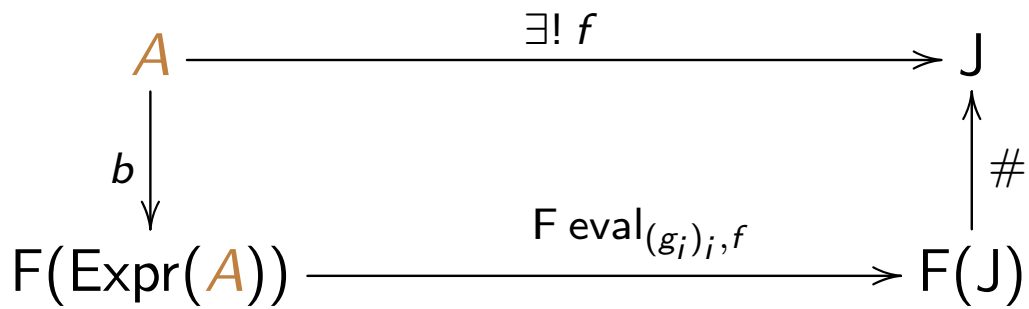


Given n up-to-friendly operators $g_i : \mathrm{K}_i(\mathrm{J}) \to \mathrm{J}$ for $i \in \{1, \ldots, n\}$
... can define $f : A \to \mathrm{J}$ by choosing a blueprint b:



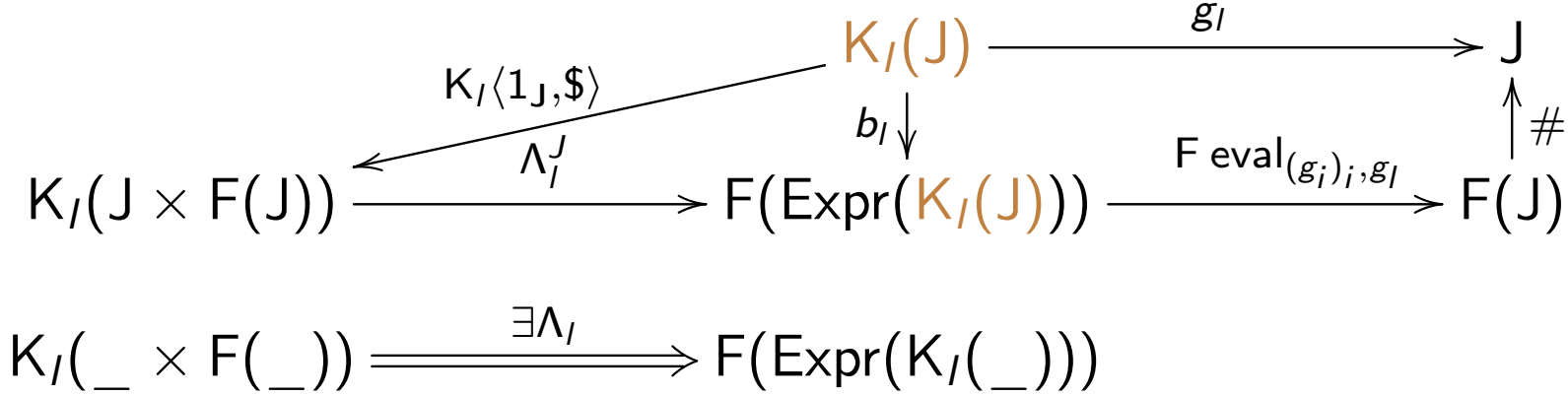
where F functor, $\mathrm{J} \xrightarrow{\$} \mathrm{F}(\mathrm{J}) \xrightarrow{\#} \mathrm{J}$ its final coalgebra, K_i functors, $\mathrm{Expr}_{(\mathrm{K}_i)_i,\mathrm{J}}$ the free monad on $\mathrm{Constant}_{\mathrm{J}} + \sum_{i=1}^{n} K_i$.

Given n up-to-friendly operators $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$ for $i \in \{1, \ldots, n\}$
... can define $f : A \to \mathsf{J}$ by choosing a blueprint b:

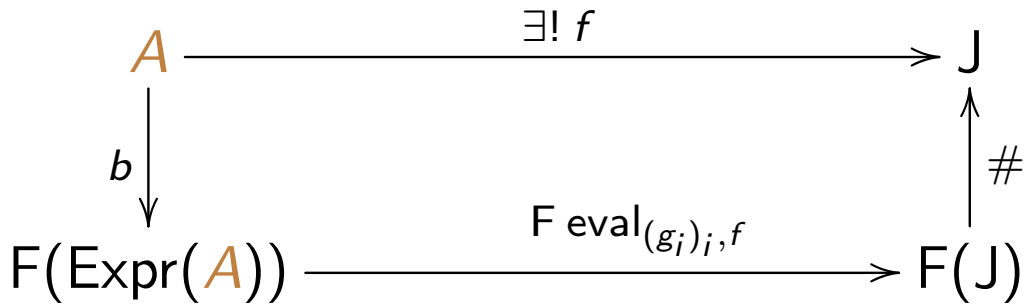


where F functor, $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J}) \xrightarrow{\#} \mathsf{J}$ its final coalgebra, K_i functors, $\mathsf{Expr}_{(\mathsf{K}_i)_i, \mathsf{J}}$ the free monad over $\mathsf{Constant}_{\mathsf{J}} + \sum_{i=1}^{n} K_i$.

Up-To-Friendly Operators: Consume ≤ 1, Produce ≥ 1



Given n up-to-friendly operators $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$ for $i \in \{1, \ldots, n\}$ ... can define $f : A \to \mathsf{J}$ by choosing a blueprint b:



where F functor, $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J}) \xrightarrow{\#} \mathsf{J}$ its final coalgebra, K_i functors, $\mathsf{Expr}_{(\mathsf{K}_i)_i, \mathsf{J}}$ the free monad over $\mathsf{Constant}_\mathsf{J} + \sum_{i=1}^{n} K_i$.

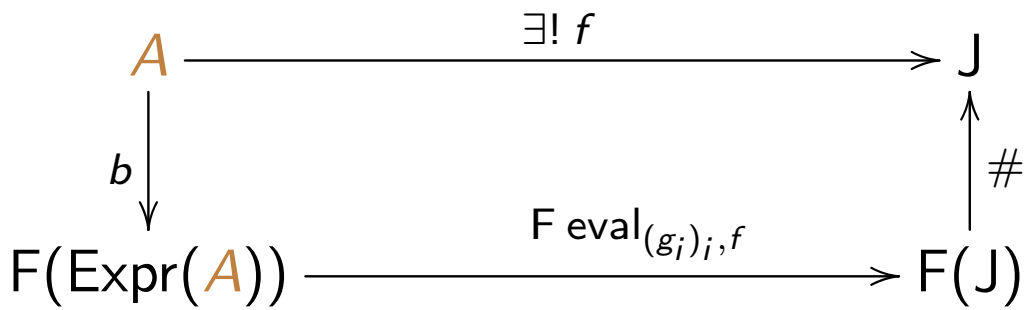
Up-To-Friendly Operators: Consume ≤ 1, Produce ≥ 1

$\# : \mathsf{F}\ \mathsf{J} \to \mathsf{J}$ is the first up-to-friendly operator: $\mathsf{K}_1 = \mathsf{F}$, $g_1 = \#$
Hence could replace F Expr with Expr

F is important though, since it acts as a guard. But the guard needs not be at the top: Define GExpr = Expr F Expr

---

Given n up-to-friendly operators $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$ for $i \in \{1, \ldots, n\}$
... can define $f : A \to \mathsf{J}$ by choosing a blueprint b:



where F functor, $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J}) \xrightarrow{\#} \mathsf{J}$ its final coalgebra, K_i functors, $\mathsf{Expr}_{(\mathsf{K}_i)_i, \mathsf{J}}$ the free monad over $\mathsf{Constant}_{\mathsf{J}} + \sum_{i=1}^{n} K_i$.

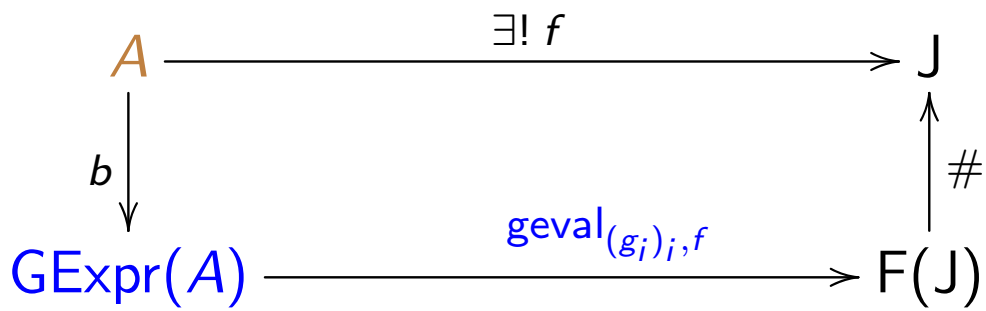
Up-To-Friendly Operators: Consume ≤ 1, Produce ≥ 1

$\# : \mathsf{F}\ \mathsf{J} \to \mathsf{J}$ is the first up-to-friendly operator: $\mathsf{K}_1 = \mathsf{F}$, $g_1 = \#$
Hence could replace F Expr with Expr

F is important though, since it acts as a guard. But the guard needs not be at the top: Define GExpr = Expr F Expr

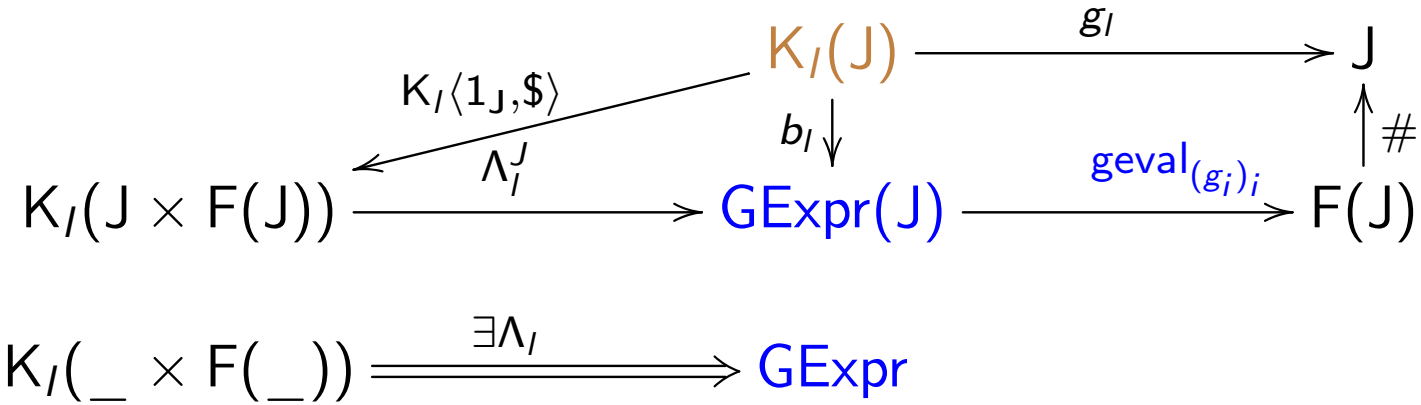
---

Given n up-to-friendly operators $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$ for $i \in \{1, \ldots, n\}$
... can define $f : A \to \mathsf{J}$ by choosing a blueprint b:

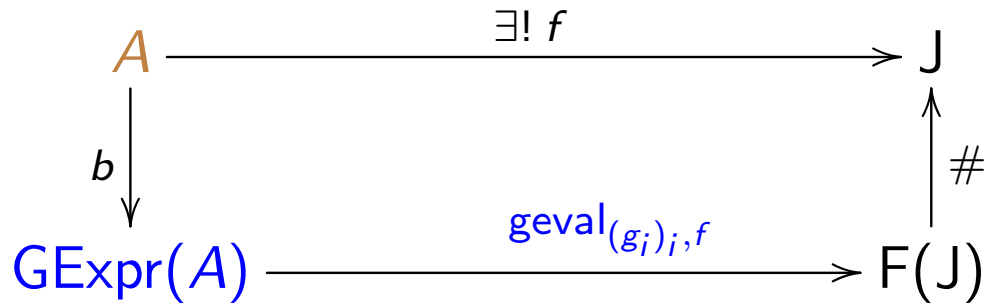


where F functor, $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J}) \xrightarrow{\#} \mathsf{J}$ its final coalgebra, K_i functors, $\mathsf{Expr}_{(\mathsf{K}_i)_i, \mathsf{J}}$ the free monad over $\mathsf{Constant}_{\mathsf{J}} + \sum_{i=1}^{n} K_i$.

Up-To-Friendly Operators: Consume ≤ 1, Produce ≥ 1



Given n up-to-friendly operators $g_i : K_i(J) \to J$ for $i \in \{1, \ldots, n\}$ ... can define $f : A \to J$ by choosing a blueprint b:



where F functor, $J \xrightarrow{\$} F(J) \xrightarrow{\#} J$ its final coalgebra, K_i functors, $\text{Expr}_{(K_i)_i, J}$ the free monad over $\text{Constant}_J + \sum_{i=1}^{n} K_i$.

Part 2: What We Do

What We Do in Isabelle/HOL

For each defined codatatype J, remember its defining functor F (s.t. J is its final coalgebra).

codatatype stream = Cons (hd : nat) (tl : stream)
Remember functor nat $\times$ α

Store friendly operators and their distributive laws – forming the current "corecursion state". Start with the constructor(s).

Register Cons as friendly
Prove up-to-Cons corecursion theorem

Functions targeting J can be defined by the corecursion theorem.

corec $\oplus$: stream $\to$ stream $\to$ stream
where *xs* $\oplus$ *ys* = Cons (hd *xs* + hd *ys*) (tl *xs* $\oplus$ tl *ys*)
Synthesize suitable blueprint and use it to define $\oplus$

Some functions K(J) $\to$ J can be registered as friendly

friend_of_corec $\oplus$
Synthesize distributive law and prove naturality and friendliness
Prove stronger corecursion theorem, factoring in the new friend

What We Do in Isabelle/HOL: Reasoning

Isabelle keeps track of Bounded Natural Functors (BNFs) – k-accessible functors that preserve weak pullbacks.

Hence are relators: Every $R \subseteq A \times B$ lifted to $\mathsf{Frel}(R) \subseteq \mathsf{F}\ A \times \mathsf{F}\ B$.

Full abstraction: If $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J})$ is F's final coalgebra, then equality on J is the largest F-bisimulation.

$$\Downarrow$$

Proof principle: Given $R \subseteq J \times J$,

if $\forall a, b.\ (a, b) \in R \longrightarrow (\$\ a, \$\ b) \in \mathsf{Frel}(R)$

then $\forall a, b.\ (a, b) \in R \longrightarrow a = b$.

What We Do in Isabelle/HOL: Reasoning

Isabelle keeps track of Bounded Natural Functors (BNFs) – k-accessible functors that preserve weak pullbacks.

Hence are relators: Every $R \subseteq A \times B$ lifted to $\mathsf{Frel}(R) \subseteq \mathsf{F}\ A \times \mathsf{F}\ B$.

Full abstraction: If $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J})$ is F's final coalgebra, then equality on J is the largest F-bisimulation up to the friends $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$.

$$\Downarrow$$

Proof principle: Given $R \subseteq J \times J$,

if $\forall a, b.\ (a, b) \in R \longrightarrow (\$\ a, \$\ b) \in \mathsf{Frel}(\mathsf{Cong}_{(g_i)_i}(R))$

then $\forall a, b.\ (a, b) \in R \longrightarrow a = b$.

$\mathsf{Cong}_{(g_i)_i}(X)$ defined (inductively) as the smallest equivalenvce that includes X and is friend-compatible:

$(k, k') \in \mathsf{Krel}_I(\mathsf{Cong}_{(g_i)_i}(X)) \longrightarrow (g_I(k), g_I(k')) \in \mathsf{Cong}_{(g_i)_i}(X)$

What We Do in Isabelle/HOL: Reasoning

Isabelle keeps track of Bounded Natural Functors (BNFs) – k-accessible functors that preserve weak pullbacks.

Hence are relators: Every $R \subseteq A \times B$ lifted to $\mathsf{Frel}(R) \subseteq \mathsf{F}\ A \times \mathsf{F}\ B$.

Full abstraction: If $\mathsf{J} \xrightarrow{\$} \mathsf{F}(\mathsf{J})$ is F's final coalgebra, then equality on J is the largest F-bisimulation up to the friends $g_i : \mathsf{K}_i(\mathsf{J}) \to \mathsf{J}$.

$$\Downarrow$$

Proof principle: Given $R \subseteq J \times J$,

if $\forall a, b.\ (a, b) \in R \longrightarrow (\$\ a, \$\ b) \in \mathsf{Frel}(\mathsf{Cong}_{(g_i)_i}(R))$

then $\forall a, b.\ (a, b) \in R \longrightarrow a = b$.

$\mathsf{Cong}_{(g_i)_i}(X)$ defined (inductively) as the smallest equivalenvce that includes X and is friend-compatible:

$(k, k') \in \mathsf{Krel}_I(\mathsf{Cong}_{(g_i)_i}(X)) \longrightarrow (g_I(k), g_I(k')) \in \mathsf{Cong}_{(g_i)_i}(X)$

Coinduction principle updated together with the corecursion state.

Things We Don't Do in Isabelle/HOL

We don't expose the regular users to the underlying category theory. Why?

Things We Don't Do in Isabelle/HOL

We don't expose the regular users to the underlying category theory. Why?

Question by user on a mailing list: "I have a coinductive predicate P and I want to show that "P s" holds, where "s" is a corecursively-defined stream ... How do I do something like this?"

My answer: [Category-theory based suggestion, comprising three steps.]

Things We Don't Do in Isabelle/HOL

We don't expose the regular users to the underlying category theory. Why?

Question by user on a mailing list: "I have a coinductive predicate P and I want to show that "P s" holds, where "s" is a corecursively-defined stream . . . How do I do something like this?"

My answer: [Category-theory based suggestion, comprising three steps.]

His follow-up, also making three points: "I have decided not to look into this any further at the moment, seeing as

1. it gives me a headache,
2. I have working proofs for now and
3. I'm quite eager to continue my formalisation elsewhere."

Things We Don't Do in Isabelle/HOL

While benefiting from category theory, users are not exposed to it:

Blueprints and distributive laws are inferred automatically and disappear after doing their jobs.

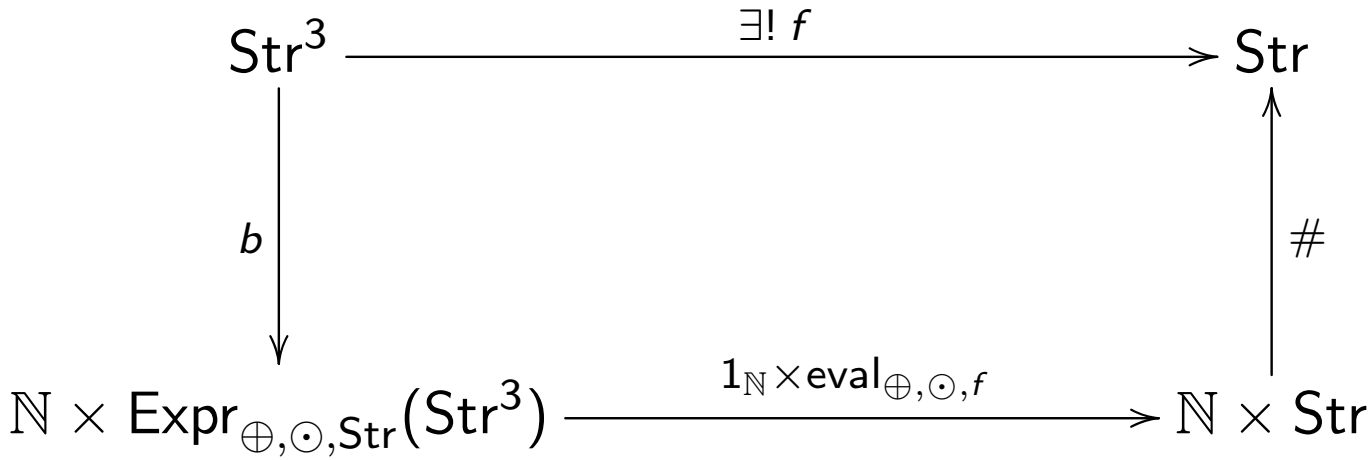
Reasoning infrastructure does not need them either: It works directly with the concrete operators.

Example

User writes equation, counting on guarded computation intuition:

$$
\begin{aligned}
&f(xs, ys, zs) \;=\\
&\quad (\mathsf{hd}(xs) \cdot \mathsf{hd}(\mathsf{tl}(ys)))\,\#\\
&\quad ((f(xs, zs, xs) \odot xs) \oplus (f(xs, xs, ys) \oplus f(zs, ys, zs)))
\end{aligned}
$$

In response, system silently does the following work:



. . . and comes back accepting that equation as a definition.

Some Literature

Rutten 1992. Processes as Terms: Non-Well-Founded Models for Bisimulation

Bartels 2003. Generalised Coinduction

Milius, Moss and Schwencke 2013. Abstract GSOS rules and a modular treatment of recursive definitions

Blanchette, Popescu and Traytel 2014. Foundational extensible corecursion: A proof assistant perspective

Blanchette, Bouzy, Lochbihler, Popescu and Traytel, 2017. Friends with Benefits – Implementing Corecursion in Foundational Proof Assistants

Pous and Rot, 2017. Companions, Codensity and Causality

Basold, Pous and Rot, 2017. Monoidal Company for Accessible Functors

Abel, 2014. Programming and Reasoning with Infinite Structures Using Copatterns and Sized Types

In Case People Are Not Too Eager To Go To Lunch...

$$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$$

$$\mathsf{f}\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ (n \,\#\, \mathsf{f}\,(n\,\mathsf{DIV}\,2)) \odot (n \,\#\, \mathsf{f}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

In Case People Are Not Too Eager To Go To Lunch...

$$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$$

$$\mathsf{f}\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ (n \,\#\, \mathsf{f}\,(n\,\mathsf{DIV}\,2)) \odot (n \,\#\, \mathsf{f}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Blueprint synthesis succeeds:

- Friendly context around the calls
- All calls are guarded

$$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$$
$$f\ (n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\ (n\ \mathsf{DIV}\ 2) \odot (n \# f\ (3 * n + 1)) & \text{if } n > 1 \end{cases}$$

$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$

$$f\ (n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\ (n\ \mathsf{DIV}\ 2) \odot (n \# f\ (3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Standard blueprint synthesis fails:

- Friendly context around the calls
- Not all calls are guarded

$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$

$$f\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\,(n\,\mathsf{DIV}\,2) \odot (n \,\#\, f\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Standard blueprint synthesis fails:

Friendly context around the calls

Not all calls are guarded

We don't give up: Try to define a recursive blueprint. Is $b : \mathbb{N} \to \mathsf{GExpr}(\mathbb{N})$ is correctly defined?

$$b\,(n) = \begin{cases} \mathsf{Ct}\,(\mathsf{zeros}) & \text{if } n \leq 1 \\ b\,(n\,\mathsf{DIV}\,2) \boxed{\odot} (n \boxed{\#} \mathsf{Var}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$

$$f\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\,(n\,\mathsf{DIV}\,2) \odot (n \,\#\, f\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Standard blueprint synthesis fails:

Friendly context around the calls

Not all calls are guarded

We don't give up: Try to define a recursive blueprint. Is $b : \mathbb{N} \to \mathsf{GExpr}(\mathbb{N})$ is correctly defined?

$$b\,(n) = \begin{cases} \mathsf{Ct}\,(\mathsf{zeros}) & \text{if } n \leq 1 \\ b\,(n\,\mathsf{DIV}\,2) \boxed{\odot} (n \boxed{\#} \mathsf{Var}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

It is! By terminating recursion on n. And the function defined from blueprint b turns out to satisfy the desired equation for f.

Eventual Up-To Corecursion

$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$

$$f\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\,(n\,\mathsf{DIV}\,2) \odot (n \,\#\, f\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Standard blueprint synthesis fails:

Friendly context around the calls

Not all calls are guarded

We don't give up: Try to define a recursive blueprint. Is $b : \mathbb{N} \to \mathsf{GExpr}(\mathbb{N})$ is correctly defined?

$$b\,(n) = \begin{cases} \mathsf{Ct}\,(\mathsf{zeros}) & \text{if } n \leq 1 \\ b\,(n\,\mathsf{DIV}\,2)\ \boxed{\odot}\ (n\ \boxed{\#}\ \mathsf{Var}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

It is! By terminating recursion on n. And the function defined from blueprint b turns out to satisfy the desired equation for f.

Eventual Up-To Corecursion

$\mathsf{f} : \mathbb{N} \to \mathsf{Str}$

$$f\,(n) = \begin{cases} \mathsf{zeros} & \text{if } n \leq 1 \\ f\,(n\,\mathsf{DIV}\,2) \odot (n \# f\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

Standard blueprint synthesis fails:

- Friendly context around the calls
- Not all calls are guarded

We don't give up: Try to define a recursive blueprint. Is $b : \mathbb{N} \to \mathsf{GExpr}(\mathbb{N})$ is correctly defined?

$$b\,(n) = \begin{cases} \mathsf{Ct}\,(\mathsf{zeros}) & \text{if } n \leq 1 \\ b\,(n\,\mathsf{DIV}\,2)\ \boxed{\odot}\ (n\ \boxed{\#}\ \mathsf{Var}\,(3 * n + 1)) & \text{if } n > 1 \end{cases}$$

It is! By terminating recursion on n. And the function defined from blueprint b turns out to satisfy the desired equation for f.

Corecursion tool synthesizes equation for b
Recursive function package (Krauss et al.) validates this equation
Corecursion tool does the rest

Summary

Isabelle/HOL is one of the corecursion-friendliest proof assistants.
It supports:

- Functor-based (co)datatypes
- Incremental, up-to corecursion based on friendly operators
- Corresponding up-to coinduction (for equality)
- Mixed recursion-corecursion