**Technical Report**

# On the Mathematical Properties of Monadic Stream Functions


Manuel Bärenz
University of Bamberg
manuel.baerenz@uni-bamberg.de

Ivan Perez
University of Nottingham
ixp@cs.nottingham.ac.uk

Henrik Nilsson
University of Nottingham
nhn@cs.nottingham.ac.uk


## 1. Monadic Stream Functions

Monadic Stream Functions (MSFs) are defined by the following expression:

$$\begin{aligned}&\textbf{newtype } \mathit{MSF}\ m\ a\ b = \\ &\quad \mathit{MSF}\ \{\mathit{step} :: a \to m\ (b, \mathit{MSF}\ m\ a\ b)\}\end{aligned}$$

In this document:

- We prove that MSFs are Arrows for any monad.
- We prove that, if the monad is commutative, then the MSF is a Commutative Arrow.
- We show that, if the monad is effect-less, then MSFs partially applied to an input and a monad are themselves monads.
- We discuss the relation between Monadic Streams and Monads.


#### Acknowledgements

This work has benefited from discussions with multiple people. The authors would like to thank Graham Hutton, Paolo Capriotti, Baltasar Trancón y Widemann, Neil Sculthorpe, Venanzio Capretta, Jennifer Hackett, Thorsten Altenkirch, Gabe Dijstra, Ambrus Kaposi, Neelakantan R. Krishnaswami and Jeremy Gibbons for their helpful comments.


#### Declaration

The present document has been referenced from an article submitted for review at the Haskell Symposium 2016 (Functional Reactive Programming, Refactored). This document itself has never been submitted to a conference, published in any peer-reviewed publication or submitted to any kind of review. We make it public only to support the our case for the use of MSFs as reactive constructs for FRP and reactive programming in Haskell, but it should not be considered previous work.

## 2. MSF essential interface

The definition of the essential arrow combinators for MSFs are as follows:

$$\begin{aligned}&\mathit{id}_{MSF} = \mathit{MSF}\ \$\ \lambda a \to \mathit{return}\ (a, \mathit{id}_{MSF}) \\ &\mathit{arr}\ f\ \ = \mathit{MSF}\ \$\ \lambda a \to \mathit{return}\ (f\ a, \mathit{arr}\ f) \\ &\mathit{msf1} \ggg \mathit{msf2} = \mathit{MSF}\ \$\ \lambda a \to \textbf{do} \\ &\quad (b, \mathit{msf1}') \leftarrow \mathit{step}\ \mathit{msf1}\ a \\ &\quad (c, \mathit{msf2}') \leftarrow \mathit{step}\ \mathit{msf2}\ b \\ &\quad \mathit{return}\ (c, \mathit{msf1}' \ggg \mathit{msf2}') \\ &\mathit{first}\ \mathit{msf1} = \mathit{MSF}\ \$\ \lambda(a, c) \to \textbf{do} \\ &\quad (b, \mathit{msf1}') \leftarrow \mathit{step}\ \mathit{msf1}\ a \\ &\quad \mathit{return}\ ((b, c), \mathit{first}\ \mathit{msf1}') \\ &\mathit{second}\ \mathit{msf1} = \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ \mathit{msf1} \ggg \mathit{arr}\ \mathit{swap} \\ &\quad \textbf{where}\ \mathit{swap}\ (x, y) = (y, x) \\ &\mathit{msf1} \ast\!\!\ast\!\!\ast\ \mathit{msf2} = \mathit{first}\ \mathit{msf1} \ggg \mathit{second}\ \mathit{msf2}\end{aligned}$$

## 3. MSFs as Arrows

Monad Stream Functions obey the arrow laws, as well as other properties. For reference, we list all the arrow laws:

- $\mathit{arr}\ \mathit{id} \equiv \mathit{id}_{MSF}$
- $\mathit{arr}\ (f \ggg g) \equiv \mathit{arr}\ f \ggg \mathit{arr}\ g$
- $\mathit{first}\ (f \ggg g) \equiv \mathit{first}\ f \ggg \mathit{first}\ g$
- $\mathit{first}\ (\mathit{arr}\ f) \equiv \mathit{arr}\ (\mathit{first}\ f)$
- $\mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} \equiv \mathit{arr}\ \mathit{fst} \ggg f$
- $\mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \ast\!\!\ast\!\!\ast\ g) \equiv \mathit{arr}\ (\mathit{id} \ast\!\!\ast\!\!\ast\ g) \ggg \mathit{first}\ f$
- $\mathit{first}\ (\mathit{first}\ f) \ggg \mathit{arr}\ \mathit{assoc} \equiv \mathit{arr}\ \mathit{assoc} \ggg \mathit{first}\ f$

To prove these laws we model Haskell types as *complete partial orders* (CPOs), functions as continuous maps and use *fixpoint induction* [**?** ].

The application of fixpoint induction means that if $P$ is a predicate about values of some type $a$, and $f : a \to a$, and $P\ \bot$ holds, and for every value $x$ we can infer $P\ x \Rightarrow P\ (f\ x)$, then $P$ holds for $\mathit{fix}\ f\ =_{\text{def}}\ f\ (\mathit{fix}\ f)$.

#### Definitions

All fundamental category and arrow combinators like $\mathit{id}$, $\ggg$, $\mathit{arr}$ and $\mathit{first}$ are a fixpoint of some function. Below we include the definition of these functions as the fixpoint of some other function. For conciseness, we leave out the value constructors, so the type of MSF is simply $\mathit{MSF}\ m\ a\ b = a \to m\ (b, \mathit{MSF}\ m\ a\ b)$.

$$\begin{aligned}&\mathit{arr} &&= \mathit{fix}\ \mathit{arrfix} \\ &\mathit{arrfix}\ \textbf{rec}\ f\ a &&= \mathit{return}\ (f\ a, \textbf{rec}\ f) \\ &(\ggg) &&= \mathit{fix}\ \mathit{compfix} \\ &\mathit{compfix}\ \textbf{rec}\ f\ g\ a &&= \textbf{do} \\ &\quad (b, f') \leftarrow f\ a \\ &\quad (c, g') \leftarrow g\ b \\ &\quad \mathit{return}\ (c, \textbf{rec}\ f'\ g') \\ &\mathit{id}_{MSF} &&= \mathit{fix}\ \mathit{idfix} \\ &\mathit{idfix}\ \textbf{rec}\ a &&= \mathit{return}\ (a, \textbf{rec}) \\ &\mathit{first} &&= \mathit{fix}\ \mathit{firstfix}\end{aligned}$$

$$
\begin{aligned}
&\mathit{firstfix}\ \mathbf{rec}\ \mathit{msf} \quad = \lambda(a, c) \to \mathbf{do} \\
&\quad (b, \mathit{msf}') \leftarrow \mathit{msf}\ a \\
&\quad \mathit{return}\ ((b, c), \mathbf{rec}\ \mathit{msf}')
\end{aligned}
$$

We chose to subscript $id$ with $MSF$ to avoid confusion with the identity function $id\ x = x$.

### 3.1 Proof: $arr\ id \equiv id_{MSF}$

We begin by stating the predicate over which we will perform induction.

$$P\ (x, y) \ =_{\text{def}}\ x\ id \equiv y$$

We now have to prove the start and the step of the induction.
**Proof:**

$$
\begin{aligned}
&P\ (\bot, \bot) \\
&\iff \{\ \textit{By definition of } P\ \} \\
&\qquad \bot\ id \equiv \bot \\
&\iff \{\ \textit{Application to } \bot\ \} \\
&\qquad \bot \quad \equiv \bot \\
&\iff \mathit{true} \\
&P\ (\mathit{arrfix}\ x, \mathit{idfix}\ y) \\
&\iff \{\ \textit{By definition of } P\ \} \\
&\qquad \mathit{arrfix}\ x\ id \qquad\qquad\quad \equiv \mathit{idfix}\ y \\
&\iff \{\ \textit{Definitions of arrfix and idfix}\ \} \\
&\qquad \lambda a \to \mathit{return}\ (id\ a, x\ id) \equiv \lambda a \to \mathit{return}\ (a, y) \\
&\iff \{\ \textit{Reduction of } id\ \} \\
&\qquad \lambda a \to \mathit{return}\ (a, x\ id) \quad \equiv \lambda a \to \mathit{return}\ (a, y) \\
&\Leftarrow \quad x\ id \qquad\qquad\quad \equiv y \\
&\iff \{\ \textit{Definition of } P\ \} \\
&\qquad P\ (x, y)
\end{aligned}
$$

### 3.2 Proof: $arr\ (f \ggg g) \equiv arr\ f \ggg arr\ g$

This method still allows us to prove laws with variables, using universal quantification. We define the predicate $P$ as:

$$P\ (x, y) \ =_{\text{def}}\ \forall\ f\ g\ .\ (x\ f)\ `y`\ (x\ g) \equiv x\ (g \circ f)$$

**Proof**:

$$
\begin{aligned}
&P\ (\bot, \bot) \\
&\iff \{\ \textit{Definition of } P\ \} \\
&\qquad \forall\ f\ g\ .\ (\bot\ f)\ `\bot`\ (\bot\ g) \equiv \bot\ (g \circ f) \\
&\iff \{\ \textit{Application to } \bot\ \} \\
&\qquad \bot \equiv \bot \\
&\iff \mathit{true} \\
&P\ (\mathit{arrfix}\ x, \mathit{compfix}\ y) \\
&\iff \{\ \textit{Definition of } P\ \} \\
&\qquad \forall\ f\ g\ .\ (\mathit{arrfix}\ x\ f)\ `\mathit{compfix}\ y`\ (\mathit{arrfix}\ x\ g) \\
&\qquad\qquad \equiv \mathit{arrfix}\ x\ (g \circ f) \\
&\iff \{\ \textit{Definition of compfix and arrfix and } \beta\textit{-Reduction}\ \} \\
&\qquad \forall\ f\ g\ .\ \mathit{lhs} \equiv \lambda a \to \mathit{return}\ (g\ (f\ a), x\ (g \circ f))\ \mathbf{where} \\
&\qquad\quad \mathit{lhs} = \lambda a \to \mathbf{do} \\
&\qquad\qquad (b, \mathit{msf1}') \leftarrow \mathit{return}\ (f\ a, x\ f) \\
&\qquad\qquad (c, \mathit{msf2}') \leftarrow \mathit{return}\ (g\ b, x\ g) \\
&\qquad\qquad \mathit{return}\ (c, \mathit{msf1}'\ `y`\ \mathit{msf2}') \\
&\iff \{\ \textit{Monad laws}\ \} \\
&\qquad \forall\ f\ g\ .\ \lambda a \to \mathit{return}\ (g\ (f\ a), x\ f\ `y`\ x\ g) \\
&\qquad\qquad \equiv \lambda a \to \mathit{return}\ (g\ (f\ a), x\ (g \circ f)) \\
&\Leftarrow \quad \forall\ f\ g\ .\ x\ f\ `y`\ x\ g \equiv x\ (g \circ f) \\
&\iff P\ (x, y)
\end{aligned}
$$

### 3.3 Proof: $first\ (f \ggg g) \equiv first\ f \ggg first\ g$

The predicate for this proof can be chosen as:

$$
\begin{aligned}
&P\ (x, y) = \forall\ \mathit{msf1}\ \mathit{msf2}\ . \\
&\quad x\ (\mathit{msf1}\ `y`\ \mathit{msf2}) \equiv (x\ \mathit{msf1})\ `y`\ (x\ \mathit{msf2})
\end{aligned}
$$

**Proof**:

$$
\begin{aligned}
&P\ (\bot, \bot) \\
&\quad = \forall\ \mathit{msf1}\ \mathit{msf2}\ . \\
&\qquad\quad \bot\ (\mathit{msf1}\ `\bot`\ \mathit{msf2}) \equiv (\bot\ \mathit{msf1})\ `\bot`\ (\bot\ \mathit{msf2}) \\
&\quad = \bot \equiv \bot \\
&\quad = \mathit{true} \\
&P\ (\mathit{firstfix}\ x, \mathit{compfix}\ y) \\
&\iff \{\ \textit{Definition of } P\ \} \\
&\qquad \forall\ \mathit{msf1}\ \mathit{msf2}\ .\ \mathit{lhs} \equiv \mathit{rhs}\ \mathbf{where} \\
&\qquad\quad \mathit{lhs} = \mathit{firstfix}\ x\ ((\mathit{compfix}\ y)\ \mathit{msf1}\ \mathit{msf2}) \\
&\qquad\quad \mathit{rhs} = (\mathit{compfix}\ y)\ (\mathit{firstfix}\ x\ \mathit{msf1})\ (\mathit{firstfix}\ x\ \mathit{msf2}) \\
&\iff \{\ \textit{Definition of firstfix and compfix, } \beta\textit{-Reduction of } \lambda\textit{-Expressions}\ \} \\
&\qquad \forall\ \mathit{msf1}\ \mathit{msf2}\ .\ \mathit{lhs} \equiv \mathit{rhs}\ \mathbf{where} \\
&\qquad\quad \mathit{lhs} = \lambda(a, d) \to \mathbf{do} \\
&\qquad\qquad (c', \mathit{comp}') \leftarrow \mathbf{do} \\
&\qquad\qquad\quad (b, \mathit{msf1}') \leftarrow \mathit{msf1}\ a \\
&\qquad\qquad\quad (c, \mathit{msf2}') \leftarrow \mathit{msf2}\ b \\
&\qquad\qquad\quad \mathit{return}\ (c, \mathit{msf1}'\ `y`\ \mathit{msf2}') \\
&\qquad\qquad \mathit{return}\ ((c', d), x\ \mathit{comp}') \\
&\qquad\quad \mathit{rhs} = \lambda(a, d) \to \mathbf{do} \\
&\qquad\qquad ((b', d'), \mathit{first1}') \leftarrow \mathbf{do} \\
&\qquad\qquad\quad (b, \mathit{msf1}') \leftarrow \mathit{msf1}\ a \\
&\qquad\qquad\quad \mathit{return}\ ((b, d), x\ \mathit{msf1}') \\
&\qquad\qquad (c', d'') \leftarrow \mathbf{do} \\
&\qquad\qquad\quad (c, \mathit{msf2}') \leftarrow \mathit{msf2}\ b' \\
&\qquad\qquad\quad \mathit{return}\ ((c, d'), x\ \mathit{msf2}') \\
&\qquad\qquad \mathit{return}\ ((c', d''), \mathit{msf1}'\ `y`\ \mathit{msf2}') \\
&\iff \{\ \textit{Monad laws}\ \} \\
&\qquad \forall\ \mathit{msf1}\ \mathit{msf2}\ .\ \mathit{lhs} \equiv \mathit{rhs}\ \mathbf{where} \\
&\qquad\quad \mathit{lhs} = \lambda(a, d) \to \mathbf{do} \\
&\qquad\qquad (b, \mathit{msf1}') \leftarrow \mathit{msf1}\ a \\
&\qquad\qquad (c, \mathit{msf2}') \leftarrow \mathit{msf2}\ b \\
&\qquad\qquad \mathit{return}\ ((c, d), x\ (\mathit{msf1}'\ `y`\ \mathit{msf2}')) \\
&\qquad\quad \mathit{rhs} = \lambda(a, d) \to \mathbf{do} \\
&\qquad\qquad (b, \mathit{msf1}') \leftarrow \mathit{msf1}\ a \\
&\qquad\qquad (c, \mathit{msf2}') \leftarrow \mathit{msf2}\ b \\
&\qquad\qquad \mathit{return}\ ((c, d), (x\ \mathit{msf1}')\ `y`\ (x\ \mathit{msf2}')) \\
&\Leftarrow \quad \forall\ \mathit{msf1}'\ \mathit{msf2}'\ . \\
&\qquad\quad x\ (\mathit{msf1}'\ `y`\ \mathit{msf2}') \equiv x\ \mathit{msf1}'\ `y`\ x\ \mathit{msf2}' \\
&\iff P\ (x, y)
\end{aligned}
$$

### 3.4 Proof: $arr\ f \ggg arr\ g \equiv arr\ (f \ggg g)$

Studying the given examples, it becomes clear how a more informal proof in Haskell could be made rigorous. We will therefore do the remaining proofs directly in Haskell, still simplified by removing the constructors. We will reduce the proofs to seeing that, in each case, both expressions on both sides of the equality reduce to the same program, except for the continuation, which follows the same pattern that we are trying to prove, but applied to the continuations of the internal MSFs in the property. As an example, we re-do one of the properties we have proved by fixpoint induction in normal haskell style, and apply induction in the last step.

**Proof**

Left-hand side:

$$
\begin{aligned}
&arr\ f \ggg arr\ g \\
&= \{\ \textit{definition}\ (\ggg)\ \} \\
&\quad \lambda a \to \mathbf{do} \\
&\qquad (b, f') \leftarrow (\lambda a \to return\ (f\ a, arr\ f))\ a \\
&\qquad (c, g') \leftarrow (\lambda b \to return\ (g\ b, arr\ g))\ b \\
&\qquad return\ (c, f' \ggg g') \\
&= \{\ \textit{left-identity return/bind}\ \} \\
&\quad \lambda a \to \mathbf{do} \\
&\qquad \mathbf{let}\ (b, f') = (f\ a, arr\ f) \\
&\qquad \mathbf{let}\ (c, g') = (g\ b, arr\ g) \\
&\qquad return\ (c, f' \ggg g') \\
&= \{\ \textit{applying variable substitutions}\ \} \\
&\quad \lambda a \to return\ (g\ (f\ a), arr\ f \ggg arr\ g) \\
&= \{\ \textit{function composition}\ \} \\
&\quad \lambda a \to return\ ((g \circ f)\ a, arr\ f \ggg arr\ g) \\
&= \{\ (\ggg) \equiv \textit{flip}\ (\circ)\ \} \\
&\quad \lambda a \to return\ ((f \ggg g)\ a, arr\ f \ggg arr\ g) \\
&= \{\ \textit{by induction}\ \} \\
&\quad \lambda a \to return\ ((f \ggg g)\ a, arr\ (f \ggg g)) \\
&= arr\ (f \ggg g)
\end{aligned}
$$

If we expand the right-hand side, in one step we get to:

$$
\begin{aligned}
&arr\ (f \ggg g) \\
&= \{\ \textit{definition of arr}\ \} \\
&\quad \lambda a \to return\ ((f \ggg g)\ a, arr\ (f \ggg g))
\end{aligned}
$$

which shows that we only need to apply induction in order to be able to prove equality, as the first-component of the pair is the same and the $a$ does not appear in the second component.

### 3.5 Proof: $first\ (arr\ f) \equiv arr\ (first\ f)$

**Definitions**

For pure functions:

$first\ f\ (a, c) = (f\ a, c)$

**Proof**

Left-hand side:

$$
\begin{aligned}
&first\ (arr\ f) \\
&= \{\ \textit{definition first}\ \} \\
&\quad \lambda(a, c) \to \mathbf{do} \\
&\qquad (b, f') \leftarrow arr\ f \\
&\qquad return\ ((b, c), first\ f') \\
&= \{\ \textit{definition arr}\ \} \\
&\quad \lambda(a, c) \to \mathbf{do} \\
&\qquad (b, f') \leftarrow (\lambda a \to return\ (f\ a, arr\ f))\ a \\
&\qquad return\ ((b, c), first\ f') \\
&= \{\ \textit{function application}\ \} \\
&\quad \lambda(a, c) \to \mathbf{do} \\
&\qquad (b, f') \leftarrow return\ (f\ a, arr\ f)) \\
&\qquad return\ ((b, c), first\ f') \\
&= \{\ \textit{left-identity return/bind}\ \} \\
&\quad \lambda(a, c) \to \mathbf{do} \\
&\qquad \mathbf{let}\ (b, f') = (f\ a, arr\ f) \\
&\qquad \mathbf{in}\ return\ ((b, c), first\ f') \\
&= \{\ \textit{variable substitution}\ \} \\
&\quad \lambda(a, c) \to return\ ((f\ a, c), first\ (arr\ f)) \\
&= \{\ \textit{induction}\ \} \\
&\quad arr\ (first\ f)
\end{aligned}
$$

If we expand the right-hand side, we get to:

$$
\begin{aligned}
&arr\ (first\ f) \\
&= \{\ \textit{definition of arr}\ \} \\
&\quad \lambda(a, c) \to return\ ((first\ f)\ (a, c), arr\ (first\ f)) \\
&= \{\ \textit{definition of first for functions}\ \} \\
&\quad \lambda(a, c) \to return\ ((f\ a, c), arr\ (first\ f))
\end{aligned}
$$

which shows that we can apply induction for the second component of the pair (the contiuation MSF).

**3.6 Proof:** $first\ f \ggg arr\ fst \equiv arr\ fst \ggg f$

Left-hand side:

$first\ f \ggg arr\ fst$

$= \{$ *by definition* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b, c'), firstf') \leftarrow first\ f\ (a, c)$
$\quad (b', arrfst') \leftarrow arr\ fst\ (b, c')$
$\quad return\ (b', firstf' \ggg arrfst')$

$= \{$ *insert definition of first and expand* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad (b', arrfst') \leftarrow arr\ fst\ (b, c)$
$\quad return\ (b', first\ f' \ggg arrfst')$

$= \{$ *insert definition of arr and expand* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad return\ (b, first\ f' \ggg arr\ fst)$

Right-hand side:

$arr\ fst \ggg f$

$= \{$ *by definition* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (a', arrfst') \leftarrow arr\ fst\ (a, c)$
$\quad (b, f') \leftarrow f\ a'$
$\quad return\ (b, arrfst' \ggg f')$

$= \{$ *insert definition of arr and expand* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad return\ (b, arr\ fst \ggg f')$

$= \{$ *by induction* $\}$

$first\ f \ggg arr\ fst$

**3.7 Proof:**

$first\ f \ggg arr\ (id \ast\!\!\ast\!\!\ast\ g) \equiv arr\ (id \ast\!\!\ast\!\!\ast\ g) \ggg first\ f$

**Definitions**

On functions:

$(id \ast\!\!\ast\!\!\ast\ g)\ (b, c) = (b, g\ c)$

**Proof**

Left-hand side

$first\ f \ggg arr\ (id \ast\!\!\ast\!\!\ast\ g)$

$= \{$ *definition of* $(\ggg)$ $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b, c'), firstf') \leftarrow first\ f\ (a, c)$
$\quad ((b', d), arridg') \leftarrow arr\ (id \ast\!\!\ast\!\!\ast\ g)\ (b, c')$
$\quad return\ ((b', d), firstf' \ggg arridg')$

$= \{$ *insert definition of first, expand, evaluate let* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad ((b', d), arridg') \leftarrow arr\ (id \ast\!\!\ast\!\!\ast\ g)\ (b, c)$
$\quad return\ ((b', d), first\ f' \ggg arridg')$

$= \{$ *insert definition of arr and expand* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad \mathbf{let}\ ((b', d), arridg') = ((id \ast\!\!\ast\!\!\ast\ g)\ (b, c), arr\ (id \ast\!\!\ast\!\!\ast\ g))$
$\quad return\ ((b', d), first\ f' \ggg arridg')$

$= \{$ *evaluating let* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad (b, f') \leftarrow f\ a$
$\quad return\ ((b, g\ c), first\ f' \ggg arr\ (id \ast\!\!\ast\!\!\ast\ g))$

Right-hand side

$arr\ (id \ast\!\!\ast\!\!\ast\ g) \ggg first\ f$

$= \{$ *definition of* $(\ggg)$ $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b, c'), arridg') \leftarrow arr\ (id \ast\!\!\ast\!\!\ast\ g)\ (a, c)$
$\quad ((b', d), firstf') \leftarrow first\ f\ (b, c')$
$\quad return\ ((b', d), arridg' \ggg firstf')$

$= \{$ *definition of arr* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b, c'), arridg') \leftarrow return\ ((id \ast\!\!\ast\!\!\ast\ g)\ (a, c), arr\ (id \ast\!\!\ast\!\!\ast\ g))$
$\quad ((b', d), firstf') \leftarrow first\ f\ (b, c')$
$\quad return\ ((b', d), arridg' \ggg firstf')$

$= \{$ *definition of* $id \ast\!\!\ast\!\!\ast\ g$ *for functions* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b, c'), arridg') \leftarrow return\ ((a, g\ c), arr\ (id \ast\!\!\ast\!\!\ast\ g))$
$\quad ((b', d), firstf') \leftarrow first\ f\ (b, c')$
$\quad return\ ((b', d), arridg' \ggg firstf')$

$= \{$ *monad laws (return on left of bind)* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad \mathbf{let}\ (b, c') = (a, g\ c)$
$\quad ((b', d), firstf') \leftarrow first\ f\ (b, c')$
$\quad return\ ((b', d), arr\ (id \ast\!\!\ast\!\!\ast\ g) \ggg firstf')$

$= \{$ *let* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b', d), firstf') \leftarrow first\ f\ (a, g\ c)$
$\quad return\ ((b', d), arr\ (id \ast\!\!\ast\!\!\ast\ g) \ggg firstf')$

$= \{$ *first* $\}$

$\lambda(a, c) \to \mathbf{do}$
$\quad ((b', d), firstf') \leftarrow (\lambda(a', c') \to \mathbf{do}$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad (a'', f') \leftarrow f\ a' \\
\qquad\qquad\qquad\qquad \mathit{return}\ ((a'', c'), \mathit{first}\ f') \\
\qquad\qquad\qquad\qquad (a, g\ c) \\
\quad \mathit{return}\ ((b', d), \mathit{arr}\ (id *** g) >>> \mathit{firstf}') \\
= \{\ \mathit{beta\text{-}reduction}\ \} \\
\quad \lambda(a, c) \rightarrow \mathbf{do} \\
\quad\quad ((b', d), \mathit{firstf}') \leftarrow \mathbf{do}\ (a', f'') \leftarrow f\ a \\
\quad\quad\qquad\qquad\qquad\qquad \mathit{return}\ ((a', g\ c), \mathit{first}\ f') \\
\quad\quad \mathit{return}\ ((b', d), \mathit{arr}\ (id *** g) >>> \mathit{firstf}') \\
= \{\ \mathit{associativity\ of\ bind}\ \} \\
\quad \lambda(a, c) \rightarrow \mathbf{do} \\
\quad\quad (a', f') \leftarrow f\ a \\
\quad\quad ((b', d), \mathit{firstf}') \leftarrow \mathit{return}\ ((a', g\ c), \mathit{first}\ f') \\
\quad\quad \mathit{return}\ ((b', d), \mathit{arr}\ (id *** g) >>> \mathit{firstf}') \\
= \{\ \mathit{monad\ laws\ (return\ on\ left\ of\ bind)}\ \} \\
\quad \lambda(a, c) \rightarrow \mathbf{do} \\
\quad\quad (a', f') \leftarrow f\ a \\
\quad\quad \mathbf{let}\ ((b', d), \mathit{firstf}') = ((a', g\ c), \mathit{first}\ f') \\
\quad\quad \mathit{return}\ ((b', d), \mathit{arr}\ (id *** g) >>> \mathit{firstf}') \\
= \{\ \mathit{let}\ \} \\
\quad \lambda(a, c) \rightarrow \mathbf{do} \\
\quad\quad (a', f') \leftarrow f\ a \\
\quad\quad \mathit{return}\ ((a', g\ c), \mathit{arr}\ (id *** g) >>> \mathit{first}\ f') \\
= \{\ \mathit{induction}\ \} \\
\quad \lambda(a, c) \rightarrow \mathbf{do} \\
\quad\quad (a', f') \leftarrow f\ a \\
\quad\quad \mathit{return}\ ((a', g\ c), \mathit{first}\ f' >>> \mathit{arr}\ (id *** g))
\end{array}
$$

**3.8 Proof:**

$\mathit{first}\ (\mathit{first}\ f) >>> \mathit{arr}\ \mathit{assoc} \equiv \mathit{arr}\ \mathit{assoc} >>> \mathit{first}\ f$

**Definitions**

$$\mathit{assoc}\ ((a, b), c) = (a, (b, c))$$

**Proof**

Left-hand side

$$
\begin{array}{l}
\mathit{first}\ (\mathit{first}\ f) >>> \mathit{arr}\ \mathit{assoc} \\
= \{\ \mathit{definition\ of}\ (>>>)\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad (((d, b'), c'), \mathit{firstfirstf}') \leftarrow (\mathit{first}\ (\mathit{first}\ f))\ ((a, b), c) \\
\quad\quad ((d', (b'', c'')), \mathit{arrassoc}') \leftarrow \mathit{arr}\ \mathit{assoc}\ ((d, b'), c') \\
\quad\quad \mathit{return}\ ((d', (b'', c'')), \mathit{firstfirstf}' >>> \mathit{arrassoc}') \\
= \{\ \mathit{definition\ (outer)\ first}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad ((d, b'), \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a, b) \\
\quad\quad ((d', (b'', c')), \mathit{arrassoc}') \leftarrow \mathit{arr}\ \mathit{assoc}\ ((d, b'), c) \\
\quad\quad \mathit{return}\ ((d', (b'', c')), \mathit{first}\ \mathit{firstf}' >>> \mathit{arrassoc}') \\
= \{\ \mathit{definition\ (inner)\ first}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad (d, f') \leftarrow f\ a \\
\quad\quad ((d', (b', c')), \mathit{arrassoc}') \leftarrow \mathit{arr}\ \mathit{assoc}\ ((d, b), c) \\
\quad\quad \mathit{return}\ ((d', (b', c')), \mathit{first}\ (\mathit{first}\ f') >>> \mathit{arrassoc}') \\
= \{\ \mathit{definition\ arr,\ expanded}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad (d, f') \leftarrow f\ a \\
\quad\quad \mathit{return}\ ((d, (b, c)), \mathit{first}\ (\mathit{first}\ f') >>> \mathit{arr}\ \mathit{assoc})
\end{array}
$$

Right-hand side

$$
\begin{array}{l}
\mathit{arr}\ \mathit{assoc} >>> \mathit{first}\ f \\
= \{\ \mathit{definition\ of}\ (>>>)\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad ((a', (b', c')), \mathit{arrassoc}') \leftarrow \mathit{arr}\ \mathit{assoc}\ ((a, b), c) \\
\quad\quad (d, \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a', (b', c')) \\
\quad\quad \mathit{return}\ (d, \mathit{arrassoc}' >>> \mathit{firstf}') \\
= \{\ \mathit{definition\ of\ arr}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad ((a', (b', c')), \mathit{arrassoc}') \leftarrow \mathit{return}\ (\mathit{assoc}\ ((a, b), c), \mathit{arr}\ \mathit{assoc}) \\
\quad\quad (d, \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a', (b', c')) \\
\quad\quad \mathit{return}\ (d, \mathit{arrassoc}' >>> \mathit{firstf}') \\
= \{\ \mathit{left\ identity\ bind\ return}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad \mathbf{let}\ ((a', (b', c')), \mathit{arrassoc}') = (\mathit{assoc}\ ((a, b), c), \mathit{arr}\ \mathit{assoc}) \\
\quad\quad (d, \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a', (b', c')) \\
\quad\quad \mathit{return}\ (d, \mathit{arrassoc}' >>> \mathit{firstf}') \\
= \{\ \mathit{assoc}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad \mathbf{let}\ ((a', (b', c')), \mathit{arrassoc}') = ((a, (b, c)), \mathit{arr}\ \mathit{assoc}) \\
\quad\quad (d, \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a', (b', c')) \\
\quad\quad \mathit{return}\ (d, \mathit{arrassoc}' >>> \mathit{firstf}') \\
= \{\ \mathit{let}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
\quad\quad (d, \mathit{firstf}') \leftarrow \mathit{first}\ f\ (a, (b, c)) \\
\quad\quad \mathit{return}\ (d, \mathit{arr}\ \mathit{assoc} >>> \mathit{firstf}') \\
= \{\ \mathit{first}\ \} \\
\quad \lambda((a, b), c) \rightarrow \mathbf{do}
\end{array}
$$

$$
\begin{aligned}
&\quad (d, \mathit{firstf}') \leftarrow (\lambda(a', b') \rightarrow \mathbf{do} \\
&\qquad\qquad\qquad (a2, f') \leftarrow f\ a1 \\
&\qquad\qquad\qquad \mathit{return}\ ((a2, b'), \mathit{first}\ f') \\
&\qquad\qquad\qquad (a, (b, c)) \\
&\quad \mathit{return}\ (d, arr\ assoc \ggg \mathit{firstf}') \\
&= \{\ \mathit{beta\text{-}reduction}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (d, \mathit{firstf}') \leftarrow \mathbf{do}\ (a2, f') \leftarrow f\ a \\
&\qquad\qquad\qquad\qquad\ \ \mathit{return}\ ((a2, (b, c)), \mathit{first}\ f') \\
&\qquad \mathit{return}\ (d, arr\ assoc \ggg \mathit{firstf}') \\
&= \{\ \mathit{associativity\ of\ bind}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (a2, f') \quad\ \leftarrow f\ a \\
&\qquad (d, \mathit{firstf}') \leftarrow \mathit{return}\ ((a2, (b, c)), \mathit{first}\ f') \\
&\qquad \mathit{return}\ (d, arr\ assoc \ggg \mathit{firstf}') \\
&= \{\ \mathit{left\ identity\ of\ return\ with\ bind}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (a2, f') \quad\ \leftarrow f\ a \\
&\qquad \mathbf{let}\ (d, \mathit{firstf}') = ((a2, (b, c)), \mathit{first}\ f') \\
&\qquad \mathit{return}\ (d, arr\ assoc \ggg \mathit{firstf}') \\
&= \{\ \mathit{let}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (a2, f') \quad\ \leftarrow f\ a \\
&\qquad \mathit{return}\ ((a2, (b, c)), arr\ assoc \ggg \mathit{first}\ f') \\
&= \{\ \mathit{alpha\text{-}equivalence}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (d, f') \leftarrow f\ a \\
&\qquad \mathit{return}\ ((d, (b, c)), arr\ assoc \ggg \mathit{first}\ f') \\
&= \{\ \mathit{induction}\ \} \\
&\quad \lambda((a, b), c) \rightarrow \mathbf{do} \\
&\qquad (d, f') \leftarrow f\ a \\
&\qquad \mathit{return}\ ((d, (b, c)), \mathit{first}\ (\mathit{first}\ f') \ggg arr\ assoc) \\
&= \{\ \mathit{definition}\ \} \\
&\quad \mathit{first}\ (\mathit{first}\ f) \ggg arr\ assoc
\end{aligned}
$$

### 3.9 Commutative arrows

Suppose $m$ is a commutative monad. For convenience, we will recall the commutativity laws:

$$
\begin{aligned}
&doX :: m\ a\\
&doY :: m\ b\\
&f \quad :: a \to b \to m\ c\\
&firstX \equiv firstY\\
&\quad \textbf{where}\\
&\qquad firstX = \textbf{do}\\
&\qquad\quad x \leftarrow doX\\
&\qquad\quad y \leftarrow doY\\
&\qquad\quad return\ \$\ f\ x\ y\\
&\qquad firstY = \textbf{do}\\
&\qquad\quad y \leftarrow doY\\
&\qquad\quad x \leftarrow doX\\
&\qquad\quad return\ \$\ f\ x\ y
\end{aligned}
$$

Monadic stream functions in $m$ are now commutative arrows:

$$
\begin{aligned}
&f :: MSF\ m\ a\ b\\
&g :: MSF\ m\ c\ d\\
&first\ f \ggg second\ g \equiv second\ g \ggg first\ f
\end{aligned}
$$

In arrow notation, the relation to the monad commutativity law is more apparent:

$$
\begin{aligned}
&firstX \equiv firstY\ \textbf{where}\\
&\quad firstX = \textbf{proc}\ (inX, inY) \to \textbf{do}\\
&\qquad outX \leftarrow msfX \prec inX\\
&\qquad outY \leftarrow msfY \prec inY\\
&\qquad returnA \prec (outX, outY)\\
&\quad firstY = \textbf{proc}\ (inX, inY) \to \textbf{do}\\
&\qquad outY \leftarrow msfY \prec inY\\
&\qquad outX \leftarrow msfX \prec inX\\
&\qquad returnA \prec (outX, outY)
\end{aligned}
$$

While the data flow is the same each time, the order of side effects is reversed, but this is guaranteed not to matter since $m$ is commutative.

**Proof**

Left-hand side:

$$
\begin{aligned}
&\quad first\ f \ggg second\ g\\
&= \{\ \textit{definition}\ (\ggg)\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad ((b, c'), firstf') \leftarrow (first\ f)\ (a, c)\\
&\qquad ((b', d), secondg') \leftarrow (second\ g)\ (b, c')\\
&\qquad \textbf{let}\ sf' = firstf' \ggg secondg'\\
&\qquad return\ ((b', d), sf')\\
&= \{\ \textit{Expand first f and use monad laws}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad \textbf{let}\ c' = c\\
&\qquad \textbf{let}\ firstf' = first\ f'\\
&\qquad ((b', d), secondg') \leftarrow (second\ g)\ (b, c)\\
&\qquad \textbf{let}\ sf' = firstf' \ggg secondg'\\
&\qquad return\ ((b', d), sf')\\
&= \{\ \textit{Expand second g and use monad laws}\ \}\\
&\quad \{\ \textit{(using an easy lemma about second)}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad \textbf{let}\ c' = c\\
&\qquad \textbf{let}\ firstf' = first\ f'\\
&\qquad (d, g') \leftarrow g\ c'\\
&\qquad \textbf{let}\ b' = b\\
&\qquad \textbf{let}\ secondg' = second\ g\\
&\qquad \textbf{let}\ sf' = firstf' \ggg secondg'\\
&\qquad return\ ((b', d), sf')\\
&= \{\ \textit{Remove unneccessary lets}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad \textbf{let}\ firstf' = first\ f'\\
&\qquad (d, g') \leftarrow g\ c\\
&\qquad \textbf{let}\ secondg' = second\ g\\
&\qquad \textbf{let}\ sf' = firstf' \ggg secondg'\\
&\qquad return\ ((b, d), sf')\\
&= \{\ \textit{Use commutativity of the monad}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (d, g') \leftarrow g\ c\\
&\qquad \textbf{let}\ secondg' = second\ g\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad \textbf{let}\ firstf' = first\ f'\\
&\qquad \textbf{let}\ sf' = firstf' \ggg secondg'\\
&\qquad return\ ((b, d), sf')\\
&= \{\ \textit{Applyig let several times}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (d, g') \leftarrow g\ c\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad return\ ((b, d), first\ f' \ggg second\ g')
\end{aligned}
$$

Right-hand side:

$$
\begin{aligned}
&second\ g \ggg first\ f\\
&= \{\ \textit{definition of}\ (\ggg)\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad ((a1, c1), secondg') \leftarrow second\ g\ (a, c)\\
&\qquad ((a2, c2), firstf') \quad \leftarrow first\ f\ (a1, c1)\\
&\qquad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ \textit{definition of second}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad ((a1, c1), secondg') \leftarrow (\lambda(a', c') \to \textbf{do}\\
&\qquad\qquad\qquad\qquad (c'', g') \leftarrow g\ c'\\
&\qquad\qquad\qquad\qquad return\ ((a', c''), second\ g'))\\
&\qquad\qquad\qquad\qquad (a, c)\\
&\qquad ((a2, c2), firstf') \quad \leftarrow first\ f\ (a1, c1)\\
&\qquad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ \textit{beta-reduction}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad ((a1, c1), secondg') \leftarrow \textbf{do}\ (c'', g') \leftarrow g\ c\\
&\qquad\qquad\qquad\qquad return\ ((a, c''), second\ g'))\\
&\qquad ((a2, c2), firstf') \quad \leftarrow first\ f\ (a1, c1)\\
&\qquad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ \textit{alpha-equivalence}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad ((a1, c1), secondg') \leftarrow \textbf{do}\ (c', g') \leftarrow g\ c\\
&\qquad\qquad\qquad\qquad return\ ((a, c'), second\ g'))\\
&\qquad ((a2, c2), firstf') \quad \leftarrow first\ f\ (a1, c1)\\
&\qquad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ \textit{associativity of bind}\ \}\\
&\quad \lambda(a, c) \to \textbf{do}\\
&\qquad (c', g') \qquad\qquad\quad \leftarrow g\ c\\
&\qquad ((a1, c1), secondg') \leftarrow return\ ((a, c'), second\ g'))
\end{aligned}
$$

$$
\begin{aligned}
&\quad ((a2, c2), firstf') \quad \leftarrow first\ f\ (a1, c1)\\
&\quad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ return\ as\ left\ identity\ of\ bind\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad \mathbf{let}\ ((a1, c1), secondg') = ((a, c'), second\ g'))\\
&\qquad ((a2, c2), firstf') \leftarrow first\ f\ (a1, c1)\\
&\qquad return\ ((a2, c2), secondg' \ggg firstf')\\
&= \{\ let\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad ((a2, c2), firstf') \leftarrow first\ f\ (a, c')\\
&\qquad return\ ((a2, c2), second\ g' \ggg firstf')\\
&= \{\ definition\ of\ first\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad ((a2, c2), firstf') \leftarrow (\lambda(a'', c'') \rightarrow \mathbf{do}\\
&\qquad\qquad (a''', f') \leftarrow f\ a''\\
&\qquad\qquad return\ ((a''', c''), first\ f')\\
&\qquad\qquad (a, c')\\
&\qquad return\ ((a2, c2), second\ g' \ggg firstf')\\
&= \{\ beta\text{-}reduction\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad ((a2, c2), firstf') \leftarrow \mathbf{do}\ (a''', f') \leftarrow f\ a\\
&\qquad\qquad return\ ((a''', c), first\ f')\\
&\qquad return\ ((a2, c2), second\ g' \ggg firstf')\\
&= \{\ associativity\ of\ bind\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad (a''', f') \leftarrow f\ a\\
&\qquad ((a2, c2), firstf') \leftarrow return\ ((a''', c), first\ f')\\
&\qquad return\ ((a2, c2), second\ g' \ggg firstf')\\
&= \{\ left\text{-}identity\ return/bind\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad (a''', f') \leftarrow f\ a\\
&\qquad \mathbf{let}\ ((a2, c2), firstf') = ((a''', c'), first\ f')\\
&\qquad return\ ((a2, c2), second\ g' \ggg firstf')\\
&= \{\ let\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (c', g') \leftarrow g\ c\\
&\qquad (a''', f') \leftarrow f\ a\\
&\qquad return\ ((a''', c'), second\ g' \ggg first\ f')\\
&= \{\ alpha\text{-}renaming\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (d, g') \leftarrow g\ c\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad return\ ((b, d), second\ g' \ggg first\ f')\\
&= \{\ commutativity\ of\ the\ monad\ \}\\
&\quad \lambda(a, c) \rightarrow \mathbf{do}\\
&\qquad (b, f') \leftarrow f\ a\\
&\qquad (d, g') \leftarrow g\ c\\
&\qquad return\ ((b, d), second\ g' \ggg first\ f')\\
&= \{\ induction\ \}\\
&\quad first\ f \ggg second\ g
\end{aligned}
$$