Arrow Laws and Efficiency in Yampa

Benjamin Lerner

December 11, 2003

Abstract

Yampa is a functional reactive language developed for the specific domain of robotic control. By combining both discrete events and time-varying signals, it incorporates the main types of actions and stimuli a robot likely encounters. To accomplish this, signals and events are represented by a signal function class, which is defined and implemented to be an instance of the Arrow class, which defines the operators associated with that abstraction.

The choice of an arrow as the underlying representation yields two benefits at the programmer's level. First, because of how the operators are defined, arrows do not suffer from the same space-leaks that prior implementations of Yampa did. Second, the patterns in which the operators combine arrows are intuitively similar to those found in circuit diagrams, and hence the correspondence between circuit programming of robots and arrow programming in Yampa is easier to appreciate.

We formally prove that the type SF (the basic signal function class used in Yampa), defined in [2] and implemented in [3], satisfies the arrow laws as defined in [5], for the purposes of showing that Yampa, which is built around the properties of this class, in fact is sound. Two of the nine laws are not satisfied in terms of strict equality; an evaluation function is defined under which those two laws do hold. As the two expressions involved in each law are not the same, some discussion is given as to which expression is more efficient, and its potential as an optimization for Yampa programs.

1 Introduction

The Yampa language has been used to program or model the programming of full- scale robots as mentioned in [2]. Also described in that paper is a model of a simple mobile robot, and examples of how to program it with Yampa. The paper also introduces the notion of an arrow, and explains some of the simple combinators possible using them.

Initial work on Yampa was built on previously developed functional-reactive systems – Fran, FAL and FRP (as noted in [2]) – which used a similar architechture. However, as also mentioned in that paper, and explained in more detail elsewhere, many programs written in Yampa developed space- and time-leaks, degrading the performance of the system to the point where it was no longer a real-time responsive model. The reasons underlying this were nontrivial, but primarily stemmed from the availability of signals as first-class values. To address this issue, the underlying implementation of signals was changed to use an arrow, thus hiding the signals themselves from computation, and only exposing the manipulations permissible on them. This abstraction is valid as long as the signal class in fact is an arrow; that is, that it satisfies the axioms an arrow must uphold.

Arrows were initially defined in [4] as an extension of monads, a standard concept in functional programming. The extension was motivated by noting that certain parsing constructs could not be implemented as monads, which was worrisome to the extent that parsers are rather ubiquitous,

and if they could not be implemented as such, perhaps the definition was too restrictive. Other examples were presented later, showing how the arrow interface defined in this paper can be used with many other common applications. The paper also proves that monads can be simply and almost trivially embedded in an arrow construction, showing that all the functionality of any given monad can be expressed by a suitably defined arrow as well.

Manipulations of arrows are achieved by various combinators, which express formally what is intuitively a "wiring diagram" for the flow of data through a program. For example, the &&& combinator produces the same effect in Yampa code that a two-output *FANOUT* operation achieves in circuits – duplicating the input and routing it to two separate components of the circuit, as illustrated in [2, 6]. There exist minimal sets of three combinators which together are universal, that is, can define all other possible combinators for arrows. The selection of which three should form such a set is a matter of convenience and taste; most often a larger set of combinators is employed, for ease of use [2]. In Yampa, three non-minimal combinators are defined, namely second, &&& and ***, which each can be trivially defined from the minimal three combinators (in fact, they are so defined in the base definition of Arrow, in [3]).

Just as monads satisfy three axiomatic "monad laws", arrows must satisfy a set of nine axioms. (The initial definition in [4] stated eleven such laws, but later papers removed two of them, namely the extensionality laws, which require that if two arrows, when composed with lifted injective or surjective functions, produce identical output, then the original arrows must be identical.) These laws describe the associativity and extensionality of arrows, as well as more abstract proprties such as requiring that various combinators preserve composition of arrows [4]. Satisfying these laws, together with implementations of a universal set of combinators, defines an arrow.

The implementation of SF in the current version of Yampa [3] does satisfy nearly all of the arrow laws. Because SF was defined to optimize two special cases – namely that of a constant arrow, and that of a lifted pure function – and because the various composition operators are defined to respect that optimization as much as possible, SF does not strictly satisfy two of the arrow laws. Both laws deal with the interaction of $\ggg$ and first: the problems arise when one tries to take first constant k, because that cannot be a constant arrow any more, as it depends on its second input. Since this operation changes the optimizations on the arrow, problems arise, and will be more explicitly shown in the relevant two proofs.

Since these two laws are not strictly equal, an embedding function was defined under which the laws could be shown to hold. Intuitively, as long as the arrows on either side of those two laws produce the same output given the same input, it does not matter how the arrow is internally represented (i.e. by a constant arrow or by a lifted constant function, as one possibility). The embedding function formalizes this intuition. This possibility, however, opens the door for a small class of law-based optimizations available to Yampa programs – one side of the equations could be more efficient than the other. This possibility is explored after the proofs are presented.

Yampa redefines three basic combinators, as mentioned above, with the reasoning that specialized redefinitions, written with full knowledge of the implementation of SF, can maintain more of the possible optimizations available than can the default implementation, which has no knowledge and full generality. The proofs presented in this paper do not depend on the redefinitions of the non-minimal arrow combinators described above (namely second, &&& and ***), and concentrate on the axioms that an arrow must satisfy. Further, no proofs are presented that the redefinitions are equivalent to the default ones. This is precisely because, since they are not part of the minimal set of combinators chosen, if SF satisfies the arrow laws as expressed in terms of arr, $\ggg$ and first, then SF is an arrow. If the three redefined combinators are not equivalent to the original default definitions, that does not invalidate SF's status as an arrow, but rather says that the combinators define *new* operations, albeit with rather misleading names. If one wanted to prove that these

three definitions are equivalent to the defaults, it would be straightforward to do so. In each of these three cases, the proof is trivially accomplished by taking the default definitions, which are all defined in terms of arr, $\ggg$ and first, and partially unfolding them. By keeping careful track of each case, one can show an equivalence between them and the cases of the new definition. Some care must be taken, as the new implementation preserves some optimizations which would be lost in the default definitions – that was the whole point of overriding the defauls in the first place. As such, the proper goal is to prove that the evalSF's of each side of the equivalences are in fact equal.

2 Proofs of the Arrow Laws for Yampa

Throughout these proofs, we consistently rename variables to prevent name-clashing, since α-conversion of function definitions in Haskell does not change their meaning. If a line of code is used which has had variables renamed, then those variables will both be defined prior to their appearance in that line of code, and will be consistently used in the rest of the derivation. Whenever a variable is defined in the code as `tf0`, we denote that here as tf_0.

The symbol $\stackrel{\text{def}}{=}$ is used in two related ways in these proofs. When a proof indicates something similar to

$$\text{Let } \mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} \mathsf{arr}\ f'$$

the implied meaning is to unfold the definition of $\mathsf{arr}\ f'$, and bind the resulting value to the variables on the left side of this definition. In this example, we would have

$$tf_f = \lambda a \to (\mathsf{sfArr}\ f', f'a)$$

Alternatively, the following indicates that the operations involved on either side of the equality are definitionally the same, rather than binding to meta-variables, as above:

$$\text{If } f \text{ and } g \text{ are functions, then } f \ggg g \stackrel{\text{def}}{=} f \cdot g$$

Also common through these proofs is the replacement of definitions in the code of the form

$$\mathtt{tf0\ a0} = X$$

with the equivalent λ-abstraction

$$tf_0 = \lambda a_0 \to X$$

except in some cases where doing so would be impossible, due to recursively defined functions.

Where lazy patterns are used in the code, they will be marked as $\lambda \sim a_0$. The exposition of these proofs is influenced both by Haskell's lazy evaluation order and by its nature as a pure functional language – if two expressions both contain a subexpression $g(x)$, for the same g and x, then there is no reason to evaluate that function any further; Haskell won't evaluate it until necessary, and we as the user are ensured that $g(x)$ will produce the same results every time when called with the same arguments.

Finally, as a typographical note, function calls are typeset as arr or $\mathsf{cpAux}\times\mathsf{A}_1$, while datatype constructors and fields are typeset as $\mathtt{SF}\{\}$ or `sfAFun`. This distinction is arbitrarily made to help separate what function or effects are being considered, especially when datatype field names are being used as projection operators. Variables are typeset as g or sf_1, since they are being reasoned about at the meta-level of the proof, and not merely at the syntactic level of the source code.

Practically every proof follows the same format: it proceeds by a case analysis on the data constructors, using a lemma and structural induction to complete the proof when a case involves the

`SFTIVar{}` constructor. Each case unfolds function definitions repeatedly, with all the conventions described above, until it either has two expressions which are equal, or which require an inductively proven lemma to show equality. Those lemmas have exactly as many cases as have already been seen, and are very much identical in spirit to the cases which precede them. In some cases, the details of the proofs of these lemmas will be omitted, as they are clearly inferrable from context and other examples. Deriving the omitted steps is tedious but quite straightforward.

Theorem 2.1. *Let f and g be functions. Then*

$$\mathsf{arr}(f \ggg g) = \mathsf{arr}\, f \ggg \mathsf{arr}\, g$$

Proof. When f and g are functions, $f \ggg g \stackrel{\text{def}}{=} g \cdot f$.

We have from the definition of arr that

$$\mathsf{arr}\, f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\, f, f\ a)\}$$

Let $\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} \mathsf{arr}\, f$. We also have

$$\mathsf{arr}\, g = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\, g, g\ a)\}$$

so let $\mathtt{SF}\{\mathtt{sfTF} = tf_g\} \stackrel{\text{def}}{=} \mathsf{arr}\, g$. Then

$$\begin{aligned}
\mathsf{arr}\, f \ggg \mathsf{arr}\, g &= \mathsf{compPrim}\, \mathsf{arr}\, f\ \ \mathsf{arr}\, g \\
&= \mathsf{compPrim}\, \mathtt{SF}\{\mathtt{sfTF} = tf_f\}\ \mathtt{SF}\{\mathtt{sfTF} = tf_g\} \\
&= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\
&\quad tf_0 = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\quad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\quad\quad\quad\quad = (\mathsf{sfArr}\, f, f\ a_0) \\
&\quad\quad (sf_2, c_0) = tf_g\ b_0 \\
&\quad\quad\quad\quad = (\mathsf{sfArr}\, g, g\ b_0) \\
&\quad\quad\quad\quad = (\mathsf{sfArr}\, g, g(f\ a_0))
\end{aligned}$$

where that last line is obtained by substituting the value of b_0 into the previous line. We now need to evaluate $\mathsf{cpAux}\, sf_1\ sf_2$. We know that

$$\begin{aligned}
sf_1 &= \mathsf{sfArr}\, f \\
&= sf \text{ where} \\
&\quad sf = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda_\ a \to (sf, f\ a), \mathtt{sfAFun} = f\}
\end{aligned}$$

or, ignoring for the moment issues regarding evaluation order,

$$sf_1 = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda\_\ a \to (\mathsf{sfArr}\, f, f\ a), \mathtt{sfAFun} = f\}$$

These two forms are mathematically equivalent, since we explicitly stated that $\mathsf{sfArr}\, f = sf$. We can safely perform this substitution for two reasons: first, since Haskell evaluates lazily, this recursive

declaration will not immediately be evaluated into the infinitely nested structure it can be. Second, we will not actually be needing the $\texttt{sfTF}'$ field for the remainder of this proof. Therefore,

$$\mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_1}(\texttt{sfAFun}\ sf_1)\ sf_2$$

matching the third case of the definition. Now, from above, we have

$$\texttt{sfAFun}\ sf_1 = f$$

and so

$$\mathsf{cpAuxA_1}(\texttt{sfAFun}\ sf_1)\ sf_2 = \mathsf{cpAuxA_1}\ f\ sf_2$$

Since $sf_2 = \mathsf{sfArr}\, g$, we have, by exact analogy with sf_1, that it matches the third case of the definition, and so

$$\mathsf{cpAuxA_1}\ f\ sf_2 = \mathsf{sfArr}(g \cdot f)$$

Substituting these results back into the prior computations yields

$$\begin{aligned} tf_0 &= \lambda a_0 \rightarrow (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \\ &= \lambda a_0 \rightarrow (\mathsf{sfArr}(g \cdot f), g(f\ a_0)) \end{aligned}$$

and therefore

$$\begin{aligned} \mathsf{arr}\, f \ggg \mathsf{arr}\, g &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \rightarrow (\mathsf{sfArr}(g \cdot f), g(f\ a_0))\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \rightarrow (\mathsf{sfArr}(g \cdot f), (g \cdot f)a_0))\} \end{aligned}$$

But this last result is precisely $\mathsf{arr}(g \cdot f)$, and for functions, we know that $f \ggg g \stackrel{\text{def}}{=} g \cdot f$. So we have that

$$\mathsf{arr}\, f \ggg \mathsf{arr}\, g = \mathsf{arr}(f \ggg g)$$

□

Theorem 2.2. *Let f be an arrow. Then*

$$\mathsf{first}(\mathsf{arr}\ f) = \mathsf{arr}(\mathsf{first}\ f)$$

Proof. We have from the definition of arr that

$$\mathsf{arr}\ f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \rightarrow (\mathsf{sfArr}\ f, f\ a)\}$$

Let $\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} \mathsf{arr}\ f$. Then

$$\begin{aligned}
\mathsf{first}(\mathsf{arr}\ f) &= \mathsf{first}(\mathtt{SF}\{\mathtt{sfTF} = tf_f\}) \\
&= \mathsf{firstPrim}\ \mathtt{SF}\{\mathtt{sfTF} = tf_f\} \\
&= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\
&\quad tf_0 = \lambda\ (a_0, c_0) \rightarrow (\mathsf{fpAux}\ sf_1, (b_0, c_0)) \text{ where} \\
&\quad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\quad\quad\quad\quad = (sfArrf, f\ a_0)
\end{aligned}$$

We now need to evaluate $\mathsf{fpAux}\ sf_1$:

$$\begin{aligned}
\mathsf{fpAux}\ sf_1 &= \mathsf{fpAux}\ \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda b \rightarrow (\mathsf{sfArr}\ f, f\ b)\} \\
&= \mathsf{sfArr}(\lambda \sim(a, c) \rightarrow (f\ a, c))
\end{aligned}$$

So, substituting these results in,

$$tf_0 = \lambda \sim(a_0, c_0) \rightarrow (\mathsf{sfArr}(\lambda\ (a, c) \rightarrow (f\ a, c)), (f\ a_0, c_0))$$

and therefore

$$\begin{aligned}
\mathsf{first}(\mathsf{arr}\ f) = \mathtt{SF}\{\mathtt{sfTF} &= \lambda \sim(a_0, c_0) \rightarrow \\
&(\mathsf{sfArr}(\lambda \sim(a, c) \rightarrow (f\ a, c)), (f\ a_0, c_0))\}
\end{aligned}$$

Letting $g_1 \stackrel{\text{def}}{=} \lambda \sim(a_0, c_0) \rightarrow (f\ a_0, c_0)$ for brevity, we have

$$\begin{aligned}
&= \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(a_0, c_0) \rightarrow (\mathsf{sfArr}\ g_1, (f\ a_0, c_0))\} \\
&= \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(a_0, c_0) \rightarrow (\mathsf{sfArr}\ g_1, g_1 \sim(a_0, c_0))\}
\end{aligned}$$

Now we need to evaluate the other side of the expression:

$$\begin{aligned}
\mathsf{arr}(\mathsf{first}\ f) &= \mathsf{arr}(f \mathbin{***} \mathsf{id}) \\
&= \mathsf{arr}(\lambda \sim(a, c) \rightarrow (f\ a, \mathsf{id}\ c)) \\
&= \mathsf{arr}(\lambda \sim(a, c) \rightarrow (f\ a, c)) \\
&= \mathsf{arr}(g_2) \\
&= \mathtt{SF}\{\mathtt{sfTF} = \lambda d \rightarrow (\mathsf{sfArr}\ g_2, g_2\ d)\}
\end{aligned}$$

where id is the identity function $\mathsf{id} \stackrel{\text{def}}{=} \lambda x \rightarrow x$, and $g_2 \stackrel{\text{def}}{=} \lambda \sim(a, c) \rightarrow (f\ a, c)$.

It seems as though this substitution is invalid, as the arguments of g_2 in its definition are $\sim(a, c)$, but in the application are only d and nothing else. However, since g_2 is defined on a lazy pair of values, we can indeed apply g to a single value d, provided d is internally a pair. In fact, according to [1] and "www.haskell.org/tutorial/patterns.html", we have that a pattern $\sim(a, b)$ is equivalent to $\_@(a, b)$, where the underscore implies that we don't care about the variable's name. Therefore, we have that

$$\begin{aligned}\mathsf{arr}(\mathsf{first}\, f) &= \mathtt{SF}\{\mathtt{sfTF} = \lambda d \to (\mathsf{sfArr}\, g_2, g_2\ d)\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(a, c) \to (\mathsf{sfArr}\, g_2, g_2 \sim(a, c))\}\end{aligned}$$

By examination, it is clear that $g_1 = g_2$, since they are identical up to a renaming of a to a_0 and c to c_0. Doing so, we have an obvious equality between

$$\mathsf{first}(\mathsf{arr}\, f) = \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(a_0, c_0) \to (\mathsf{sfArr}\, g_1, g_1 \sim(a_0, c_0))\}$$

and

$$\mathsf{arr}(\mathsf{first}\, f) = \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(a, c) \to (\mathsf{sfArr}\, g_2, g_2 \sim(a, c))\}$$

which gives the desired equiality

$$\mathsf{first}(\mathsf{arr}\, f) = \mathsf{arr}(\mathsf{first}\, f)$$

□

Theorem 2.3. *Let f be an arrow. Then*

$$\mathsf{arr}\,\mathsf{id} \ggg f = f$$

Proof. We proceed by cases on the constructors for $\mathtt{SF}\{\}$.

1. Case $f = \mathsf{constant}\, b$:

   This case is trivial:

   $$\begin{aligned}(\mathsf{arr}\,\mathsf{id}) \ggg f &= g \ggg f \text{ for some arrow } g \\ &= f\end{aligned}$$

   since any arrow passed into a constant arrow yields the constant arrow again.

2. Case $f = \mathsf{arr}\, f' = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\, f', f'\ a)\}$:

   We have from the definition of arr that

   $$\begin{aligned}\mathsf{arr}\,\mathsf{id} &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ a)\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, a)\}\end{aligned}$$

   Let $\mathtt{SF}\{\mathtt{sfTF} = tf_{id}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{id}$. Also, let $\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$. Then

   $$\begin{aligned}\mathsf{arr}\,\mathsf{id} \ggg f &= \mathsf{compPrim}\,\mathsf{arr}\,\mathsf{id}\ \ f \\ &= \mathsf{compPrim}\,\mathtt{SF}\{\mathtt{sfTF} = tf_{id}\}\ \mathtt{SF}\{\mathtt{sfTF} = tf_f\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0 = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\quad\quad (sf_1, b_0) = tf_{id}\ a_0 \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ a_0) \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, a_0) \\ &\quad\quad (sf_2, c_0) = tf_f\ b_0 \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\, f', f'\ b_0) \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\, f', f'\ a_0)\end{aligned}$$

   where that last line is obtained by substituting the value of b_0 into the previous line. We now need to evaluate $\mathsf{cpAux}\, sf_1\ sf_2$. We know that

   $$\begin{aligned}sf_1 &= \mathsf{sfArr}\,\mathsf{id} \\ &= sf \text{ where} \\ &\quad sf = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda\_\ a \to (sf, \mathsf{id}\ \ a), \mathtt{sfAFun} = \mathsf{id}\}\end{aligned}$$

   Substituting the value of sf into this expression, and simplifying $\mathsf{id}\ \ a = a$,

   $$sf_1 = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda\_\ a \to (\mathsf{sfArr}\,\mathsf{id}, a), \mathtt{sfAFun} = \mathsf{id}\}$$

Therefore,

$$\mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_1}(\mathtt{sfAFun}\ sf_1)\ sf_2$$

matching the third case of the definition. Now, from above, we have $\mathtt{sfAFun}\ sf_1 = \mathsf{id}$, and so

$$\mathsf{cpAuxA_1}(\mathtt{sfAFun}\ sf_1)\ sf_2 = \mathsf{cpAuxA_1}\,\mathsf{id}\ \ sf_2$$

Since $sf_2 = \mathsf{sfArr}\ f'$, we have, by exact analogy with sf_1, that it matches the third case of the definition, and so

$$\mathsf{cpAuxA_1}\,\mathsf{id}\ \ sf_2 = \mathsf{sfArr}(f' \cdot \mathsf{id}) = \mathsf{sfArr}\ f'$$

Substituting these results back into the prior computations yields

$$\begin{aligned} tf_0 &= \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \\ &= \lambda a_0 \to (\mathsf{sfArr}\ f', f'\ a_0) \end{aligned}$$

and therefore

$$\begin{aligned} \mathsf{arr}\,\mathsf{id} \ggg \mathsf{arr}\ f' &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathsf{sfArr}\ f', f'\ a_0)\} \\ &= \mathsf{arr}\ f' \end{aligned}$$

Which yields the desired result

$$\mathsf{arr}\,\mathsf{id} \ggg f = f$$

3. Case $f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\ a)\}$ for some initial f':
   We have from the definition of arr that

$$\begin{aligned} \mathsf{arr}\,\mathsf{id} &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ a)\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, a)\} \end{aligned}$$

   Again, let $\mathtt{SF}\{\mathtt{sfTF} = tf_{id}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{id}$ and $\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$. Then

$$\begin{aligned} \mathsf{arr}\,\mathsf{id} \ggg f &= \mathsf{compPrim}\,\mathsf{arr}\,\mathsf{id}\ \ f \\ &= \mathsf{compPrim}\,\mathtt{SF}\{\mathtt{sfTF} = tf_{id}\}\ \mathtt{SF}\{\mathtt{sfTF} = tf_f\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0 = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\quad\quad (sf_1, b_0) = tf_{id}\ a_0 \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ a_0) \\ &\quad\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, a_0) \\ &\quad\quad (sf_2, c_0) = tf_f\ b_0 \\ &\quad\quad\quad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\ b_0) \\ &\quad\quad\quad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\ a_0) \end{aligned}$$

   where that last line is obtained by substituting the value of b_0 into the previous line. We now need to evaluate $\mathsf{cpAux}\, sf_1\ sf_2$. We know that

$$\begin{aligned} sf_1 &= \mathsf{sfArr}\,\mathsf{id} \\ &= sf \text{ where} \\ &\quad sf = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda\_\ a \to (sf, \mathsf{id}\ \ a), \mathtt{sfAFun} = \mathsf{id}\} \end{aligned}$$

Substituting the value of sf into this expression, and simplifying $\mathsf{id}\ \ a = a$,

$$sf_1 = \texttt{SFArr}\{\texttt{sfTF}' = \lambda\_\ a \to (\mathsf{sfArr}\,\mathsf{id}, a), \texttt{sfAFun} = \mathsf{id}\}$$

Therefore,

$$\mathsf{cpAux}\,sf_1\ sf_2 = \mathsf{cpAuxA_1}(\texttt{sfAFun}\ sf_1)\ sf_2$$

matching the third case of the definition. Now, from above, we have $\texttt{sfAFun}\ sf_1 = \mathsf{id}$, and so

$$\begin{aligned}
\mathsf{cpAuxA_1}(\texttt{sfAFun}\ sf_1)\ sf_2 &= \mathsf{cpAuxA_1}\,\mathsf{id}\ \ sf_2 \\
&= \mathsf{cpAuxA_1}\,\mathsf{id}\ \ \texttt{SFTIVar}\{\texttt{sfTF}' = tf\} \\
&= \texttt{SFTIVar}\{\texttt{sfTF}' = tf'\} \text{ where} \\
&\quad tf'\ dt\ a' = (\mathsf{cpAuxA_1}\,\mathsf{id}\ \ sf_2', c) \text{ where} \\
&\quad\quad (sf_2', c) = tf\ dt\ (\mathsf{id}\ \ a') \\
&\quad\quad\quad = tf\ dt\ a'
\end{aligned}$$

We now need a lemma:

Lemma 2.4. $\mathsf{cpAuxA_1}\,\mathsf{id}\ \ s = s$

Proof. This proof proceeds by cases, using structural induction on the nesting depth of $\texttt{SFTIVar}\{\}$ constructions:

(a) Case $s = \mathsf{sfConst}\,c'$: Trivially, $\mathsf{cpAuxA_1}\,\mathsf{id}\ \ s = s$, by the first case of $\mathsf{cpAuxA_1}$.

(b) Case $s = \mathsf{sfArr}\,f'$: Using the second case of $\mathsf{cpAuxA_1}$,

$$\mathsf{cpAuxA_1}\,\mathsf{id}\ \ s = \mathsf{cpAuxA_1}\,\mathsf{id}\ (\mathsf{sfArr}\,f') = \mathsf{sfArr}(f' \cdot \mathsf{id}) = \mathsf{sfArr}\,f' = s$$

(c) Case $s = \texttt{SFTIVar}\{\texttt{sfTF}' = g\}$: Using the third case of $\mathsf{cpAuxA_1}$, we have

$$\begin{aligned}
\mathsf{cpAux}\,\mathsf{id}\ \ s &= \mathsf{cpAux}\,\mathsf{id}\ \ \texttt{SFTIVar}\{\texttt{sfTF}' = g\} \\
&= \texttt{SFTIVar}\{\texttt{sfTF}' = g'\} \text{ where} \\
&\quad g'\ dt\ a_0' = (\mathsf{cpAuxA_1}\,\mathsf{id}\ \ h, c) \text{ where} \\
&\quad\quad (h, c) = g\ dt\ (\mathsf{id}\ \ a_0') \\
&\quad\quad\quad = g\ dt\ a_0'
\end{aligned}$$

Assume, by the induction hypothesis, that $\mathsf{cpAuxA_1}\,\mathsf{id}\ \ h = h$, since h must have a nesting depth of $\texttt{SFTIVar}\{\}$ which is one lower than s. Then

$$\begin{aligned}
g'\ dt\ a_0' &= (\mathsf{cpAuxA_1}\,\mathsf{id}\ \ h, c) \\
&= (h, c) \\
&= g\ dt\ a_0'
\end{aligned}$$

where that last equality is obtained from the line defining h. By η-reduction, we have that $g' = g$, and therefore $\mathsf{cpAuxA_1}\,\mathsf{id}\ \ s = s$.

□

Using this lemma on our previous statement, we have that

$$\mathsf{cpAuxA_1}\ \mathsf{id}\ \ sf_2' = sf_2'$$

and therefore,

$$\begin{aligned} tf'\ dt\ a' &= (\mathsf{cpAuxA_1}\ \mathsf{id}\ \ sf_2', c) \\ &= (sf_2', c) \text{ by the lemma} \\ &= tf\ dt\ a' \end{aligned}$$

By η-reduction, we have that $tf' = tf$, and therefore

$$\begin{aligned} \mathsf{cpAuxA_1}\ \mathsf{id}\ \ \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\} &= \mathtt{SFTIVar}\{\mathtt{sfTF} = tf'\} \\ &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\} \end{aligned}$$

Therefore,

$$\begin{aligned} tf_0 &= \lambda a_0 \to (\mathsf{cpAux}\, sf_1 sf_2, c_0) \\ &= \lambda a_0 \to (\mathsf{cpAux}(\mathsf{sfArr}\ \mathsf{id})(\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}), f'\ a_0) \\ &= \lambda a_0 \to (\mathsf{cpAuxA_1}\ \mathsf{id}\ (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}), f'\ a_0) \\ &= \lambda a_0 \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\ a_0) \text{ by the above } \eta\text{-reduction} \end{aligned}$$

By inspection, it is clear that $tf_0 = tf$, as they differ only in renaming a_0 to a. Substituting this result back in to the original expression for the composition yields

$$\begin{aligned} \mathsf{arr}\ \mathsf{id} \ggg f &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\ a_0)\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf\} \\ &= f \end{aligned}$$

This shows the desired formula, that

$$\mathsf{arr}\ \mathsf{id} \ggg f = f$$

□

Theorem 2.5. *Let f be an arrow. Then*

$$f \ggg \mathsf{arr}\,\mathsf{id} = f$$

Proof. We proceed by cases on the constructors for $\mathtt{SF}\{\}$.

1. Case $f = \mathsf{constant}\,b$:

   We have from the definition of $\mathsf{constant}$ that

   $$\mathsf{constant}\,b = \mathtt{SF}\{\mathtt{sfTF} = \lambda\_ \to (\mathsf{sfConst}\,b, b)\}$$

   We also have from the definition of arr that

   $$\begin{aligned}\mathsf{arr}\,\mathsf{id} &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ a)\}\\ &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\,\mathsf{id}, a)\}\end{aligned}$$

   Let $\mathtt{SF}\{\mathtt{sfTF} = tf_{id}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{id}$, and let let $\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$. Then

   $$\begin{aligned}f \ggg (\mathsf{arr}\,\mathsf{id}) &= \mathsf{compPrim}\ f\ \ \mathsf{arr}\,\mathsf{id}\\ &= \mathsf{compPrim}\,\mathtt{SF}\{\mathtt{sfTF} = tf_f\}\ \mathtt{SF}\{\mathtt{sfTF} = tf_{id}\}\\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where}\\ &\quad tf_0 = \lambda a_0 \to (\mathsf{cpAux}\,sf_1\ sf_2, c_0) \text{ where}\\ &\qquad (sf_1, b_0) = tf_f a_0\\ &\qquad\qquad = (\mathsf{sfConst}\,b, b)\\ &\qquad (sf_2, c_0) = tf_{id} b_0\\ &\qquad\qquad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ b)\\ &\qquad\qquad = (\mathsf{sfArr}\,\mathsf{id}, b)\end{aligned}$$

   where that last line is obtained by substituting the value of b_0 into the previous line. We now need to evaluate $\mathsf{cpAux}\,sf_1\ sf_2$. We know that

   $$\begin{aligned}sf_1 &= \mathsf{sfConst}\,b\\ &= sf \text{ where}\\ &\quad sf = \mathtt{SFConst}\{\mathtt{sfTF}' = \lambda\_\ \_ \to (sf, b), \mathtt{sfCVal} = b\}\end{aligned}$$

   Substituting the value of sf into this expression, we have

   $$sf_1 = \mathtt{SFConst}\{\mathtt{sfTF}' = \lambda\_\ \_ \to (\mathsf{sfConst}\,b, b), \mathtt{sfCVal} = b\}$$

   We also know that

   $$\begin{aligned}sf_2 &= \mathsf{sfArr}\,\mathsf{id}\\ &= sf \text{ where}\\ &\quad sf = \mathtt{SFArr}\{\mathtt{sfTF}' = \lambda\_\ a \to (sf, \mathsf{id}\ \ a), \mathtt{sfAFun} = \mathsf{id}\}\end{aligned}$$

Substituting the value of sf into this expression, and simplifying the application of id by $\mathsf{id}\ \ a = a$,

$$sf_2 = \texttt{SFArr}\{\texttt{sfTF}' = \lambda\_\ a \rightarrow (\mathsf{sfArr}\,\mathsf{id}, a), \texttt{sfAFun} = \mathsf{id}\}$$

Therefore,

$$\mathsf{cpAux}\,sf_1\ sf_2 = \mathsf{cpAuxC}_1(\texttt{sfCVal}\ sf_1)\ sf_2$$

matching the second case of the definition. Now, from above, we have $\texttt{sfCVal}\ sf_1 = b$, and so

$$\mathsf{cpAuxC}_1(\texttt{sfCVal}\ sf_1)\ sf_2 = \mathsf{cpAuxC}_1\,b\ sf_2$$

Since $sf_2 = \mathsf{sfArr}\,\mathsf{id}$, we have that it matches the second case of the definition, and so

$$\mathsf{cpAuxC}_1\,b\ sf_2 = \mathsf{sfConst}(\mathsf{id}\ \ b) = \mathsf{sfConst}\,b$$

Substituting these results back into the prior computations yields

$$\begin{aligned} tf_0 &= \lambda a_0 \rightarrow (\mathsf{cpAux}\,sf_1\ sf_2, c_0) \\ &= \lambda a_0 \rightarrow (\mathsf{sfConst}\,b, b) \\ &= tf_f \end{aligned}$$

and therefore

$$\begin{aligned} f \ggg \mathsf{arr}\,\mathsf{id} &= \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \rightarrow (\mathsf{sfConst}\,b, b)\} \\ &= \texttt{SF}\{\texttt{sfTF} = tf_0\} \\ &= \texttt{SF}\{\texttt{sfTF} = tf_f\} \\ &= f \end{aligned}$$

Which yields the desired result

$$f \ggg \mathsf{arr}\,\mathsf{id} = f$$

2. Case $f = \mathsf{arr}\,f' = \texttt{SF}\{\texttt{sfTF} = \lambda a \rightarrow (\mathsf{sfArr}\,f', f'\ a)\}$:

   Again, let $\texttt{SF}\{\texttt{sfTF} = tf_{id}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{id}$, and let $\texttt{SF}\{\texttt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$. Then

$$\begin{aligned} f \ggg \mathsf{arr}\,\mathsf{id} &= \mathsf{compPrim}\ f\ \ \mathsf{arr}\,\mathsf{id} \\ &= \mathsf{compPrim}\,\texttt{SF}\{\texttt{sfTF} = tf_f\}\ \texttt{SF}\{\texttt{sfTF} = tf_{id}\} \\ &= \texttt{SF}\{\texttt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0 = \lambda a_0 \rightarrow (\mathsf{cpAux}\,sf_1\ sf_2, c_0) \text{ where} \\ &\qquad (sf_1, b_0) = tf_f\ a_0 \\ &\qquad\qquad = (\mathsf{sfArr}\,f', f'\ a_0) \\ &\qquad (sf_2, c_0) = tf_{id}\ b_0 \\ &\qquad\qquad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ b_0) \\ &\qquad\qquad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}(f'\ a_0)) \\ &\qquad\qquad = (\mathsf{sfArr}\,\mathsf{id}, f'\ a_0) \end{aligned}$$

We now need to evaluate $\mathsf{cpAux}\, sf_1\ sf_2$. We know that

$$\begin{aligned} sf_1 &= \mathsf{sfArr}\ f' \\ &= sf \text{ where} \\ &\quad sf = \mathtt{SFArr}\{\mathtt{sfTF'} = \lambda\_\ a \to (sf, f'\ a), \mathtt{sfAFun} = f'\} \end{aligned}$$

Substituting the value of sf into this expression, we obtain

$$sf_1 = \mathtt{SFArr}\{\mathtt{sfTF'} = \lambda\_\ a \to (sf, f'\ a), \mathtt{sfAFun} = f'\}$$

We also know that

$$\begin{aligned} sf_2 &= \mathsf{sfArr}\,\mathsf{id} \\ &= sf \text{ where} \\ &\quad sf = \mathtt{SFArr}\{\mathtt{sfTF'} = \lambda\_\ a \to (sf, \mathsf{id}\ \ a), \mathtt{sfAFun} = \mathsf{id}\} \end{aligned}$$

Substituting the value of sf into this expression, and again simplifying $\mathsf{id}\ \ a = a$,

$$sf_2 = \mathtt{SFArr}\{\mathtt{sfTF'} = \lambda\_\ a \to (\mathsf{sfArr}\,\mathsf{id}, a), \mathtt{sfAFun} = \mathsf{id}\}$$

Therefore,

$$\mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_1}(\mathtt{sfAFun}\ sf_1)\ sf_2$$

matching the third case of the definition. Now, from above, we have $\mathtt{sfAFun}\ sf_1 = f'$, and so

$$\mathsf{cpAuxA_1}(\mathtt{sfAFun}\ sf_1)\ sf_2 = \mathsf{cpAuxA_1}\ f'\ sf_2$$

Since $sf_2 = \mathsf{sfArr}\,\mathsf{id}$, we have, by exact analogy with sf_1, that it matches the second case of the definition, and so

$$\mathsf{cpAuxA_1}\ f'\ sf_2 = \mathsf{sfArr}(\mathsf{id} \cdot f') = \mathsf{sfArr}\ f'$$

Substituting these results back into the prior computations yields

$$\begin{aligned} tf_0 &= \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \\ &= \lambda a_0 \to (\mathsf{sfArr}\ f', f'\ a_0) \end{aligned}$$

and therefore

$$\begin{aligned} \mathsf{arr}\ f' \ggg \mathsf{arr}\,\mathsf{id} &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathsf{sfArr}\ f', f'\ a_0)\} \\ &= \mathsf{arr}\ f' \end{aligned}$$

Which yields the desired result

$$f \ggg \mathsf{arr}\,\mathsf{id} = f$$

3. Case $f = \texttt{SF}\{\texttt{sfTF} = \lambda a \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf\}, f'\ a)\}$ for some initial f':

   Again, let $\texttt{SF}\{\texttt{sfTF} = tf_{id}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{id}$ and $\texttt{SF}\{\texttt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$. Then

$$\begin{aligned}
f \ggg \mathsf{arr}\,\mathsf{id} &= \mathsf{compPrim}\ f\ \ \mathsf{arr}\,\mathsf{id} \\
&= \mathsf{compPrim}\,\texttt{SF}\{\texttt{sfTF} = tf_f\}\ \texttt{SF}\{\texttt{sfTF} = tf_{id}\} \\
&= \texttt{SF}\{\texttt{sfTF} = tf_0\} \text{ where} \\
&\quad tf_0 = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\quad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\quad\quad\quad = (\texttt{SFTIVar}\{\texttt{sfTF}' = tf\}, f'\ a_0) \\
&\quad\quad (sf_2, c_0) = tf_{\mathsf{id}}\ b_0 \\
&\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}\ \ b_0) \\
&\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, \mathsf{id}(f'\ a_0)) \\
&\quad\quad\quad = (\mathsf{sfArr}\,\mathsf{id}, f'\ a_0)
\end{aligned}$$

   where that last line is obtained by substituting the value of b_0 into the previous line. We now need to evaluate $\mathsf{cpAux}\, sf_1\ sf_2$. We know that

$$\begin{aligned}
sf_2 &= \mathsf{sfArr}\,\mathsf{id} \\
&\quad = sf \text{ where} \\
&\quad\quad sf = \texttt{SFArr}\{\texttt{sfTF}' = \lambda_\ a \to (sf, \mathsf{id}\ \ a), \texttt{sfAFun} = \mathsf{id}\}
\end{aligned}$$

   Substituting the value of sf into this expression, and simplifying $\mathsf{id}\ \ a = a$,

$$sf_2 = \texttt{SFArr}\{\texttt{sfTF}' = \lambda\_\ a \to (\mathsf{sfArr}\,\mathsf{id}, a), \texttt{sfAFun} = \mathsf{id}\}$$

   Therefore,

$$\mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA}_2\, sf_1\ (\texttt{sfAFun}\ sf_2)$$

   matching the fourth case of the definition. Now, from above, we have $\texttt{sfAFun}\ sf_2 = \mathsf{id}$, and so

$$\begin{aligned}
\mathsf{cpAuxA}_2\, sf_1\ (\texttt{sfAFun}\ sf_2) &= \mathsf{cpAuxA}_2\, sf_1\ \ \mathsf{id} \\
&= \mathsf{cpAuxA}_2\,\texttt{SFTIVar}\{\texttt{sfTF}' = tf\}\ \ \mathsf{id} \\
&= \texttt{SFTIVar}\{\texttt{sfTF}' = tf'\} \text{ where} \\
&\quad tf'\ dt\ a' = (\mathsf{cpAuxA}_2\, sf_1'\ \ \mathsf{id}, \mathsf{id}\, c) \text{ where} \\
&\quad\quad (sf_1', c) = tf\ dt\ a'
\end{aligned}$$

   We now need a lemma:

Lemma 2.6. $\mathsf{cpAuxA}_2\, s\ \ \mathsf{id} = s$

Proof. This proof proceeds by cases, using structural induction on the nesting depth of $\texttt{SFTIVar}\{\}$ constructions:

(a) Case $s = \mathsf{sfConst}\, c'$: Trivially,

$$\mathsf{cpAuxA_2}\; s\;\; \mathsf{id} = \mathsf{sfConst}(\mathsf{id}\;\; c') = \mathsf{sfConst}\, c' = s$$

by the first case of $\mathsf{cpAuxA_2}$.

(b) Case $s = \mathsf{sfArr}\, f'$: Using the second case of $\mathsf{cpAuxA_2}$,

$$\mathsf{cpAuxA_2}\; s\;\; \mathsf{id} = \mathsf{cpAuxA_2}(\mathsf{sfArr}\; f')\;\; \mathsf{id} = \mathsf{sfArr}(\mathsf{id} \cdot f') = \mathsf{sfArr}\; f' = s$$

(c) Case $s = \mathtt{SFTIVar}\{\mathtt{sfTF}' = g\}$: Using the third case of $\mathsf{cpAuxA_2}$, we have

$$\begin{aligned}
\mathsf{cpAuxA_2}\; s\;\; \mathsf{id} &= \mathsf{cpAuxA_2}\; \mathtt{SFTIVar}\{\mathtt{sfTF}' = g\}\;\; \mathsf{id} \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = g'\} \text{ where} \\
&\quad g'\; dt\; a_0' = (\mathsf{cpAuxA_2}\; h\;\; \mathsf{id}, \mathsf{id}\, c) \text{ where} \\
&\quad\quad (h, c) = g\; dt\; a_0'
\end{aligned}$$

Assume, by the induction hypothesis, that $\mathsf{cpAuxA_2}\; h\;\; \mathsf{id} = h$, since h must have a nesting depth of $\mathtt{SFTIVar}\{\}$ which is one lower than s. Then

$$\begin{aligned}
g'\; dt\; a_0' &= (\mathsf{cpAuxA_2}\; h\;\; \mathsf{id}, \mathsf{id}\, c) \\
&= (h, c) \\
&= g\; dt\; a_0'
\end{aligned}$$

where that last equality is obtained from the line defining h. By η-reduction, we have that $g' = g$, and therefore $\mathsf{cpAuxA_2}\; s\;\; \mathsf{id} = s$.

□

Using this lemma on our previous statement, we have that

$$\mathsf{cpAuxA_2}\; sf_2'\;\; \mathsf{id} = sf_2'$$

and therefore,

$$\begin{aligned}
tf'\; dt\; a' &= (\mathsf{cpAuxA_2}\; sf_2'\;\; \mathsf{id}, c) \\
&= (sf_2', c) \text{ by the lemma} \\
&= tf\; dt\; a'
\end{aligned}$$

By η-reduction, we have that $tf' = tf$, and therefore

$$\begin{aligned}
\mathsf{cpAuxA_2}\; \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}\;\; \mathsf{id} &= \mathtt{SFTIVar}\{\mathtt{sfTF} = tf'\} \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}
\end{aligned}$$

Therefore,

$$\begin{aligned}
tf_0 &= \lambda a_0 \to (\mathsf{cpAux}\, sf_1 sf_2, c_0) \\
&= \lambda a_0 \to (\mathsf{cpAux}(\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\})(\mathsf{sfArr}\, \mathsf{id}), f'\; a_0) \\
&= \lambda a_0 \to (\mathsf{cpAuxA_2}(\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\})\;\; \mathsf{id}, f'\; a_0) \\
&= \lambda a_0 \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf\}, f'\; a_0) \text{ by the above } \eta\text{-reduction}
\end{aligned}$$

By inspection, it is clear that $tf_0 = tf$, as they differ only in renaming a_0 to a. Substituting this result back in to the original expression for the composition yields

$$\begin{aligned} f \ggg \mathsf{arr}\,\mathsf{id}\, f &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf\} \\ &= f \end{aligned}$$

This shows the desired formula, that

$$f \ggg \mathsf{arr}\,\mathsf{id} = f$$

□

Theorem 2.7. *Let f be an arrow, and let g be a function. Then*

$$\mathsf{first}\, f \ggg \mathsf{arr}(\mathsf{id} \times g) = \mathsf{arr}(\mathsf{id} \times g) \ggg \mathsf{first}\, f$$

Proof. We note, to begin with, that this law as stated in [5] is equivalent to the law as stated in [4], provided we have the following result, expressed as a lemma:

Lemma 2.8. *Let g be a function. Then* $\mathsf{second}(\mathsf{arr}\, g) = \mathsf{arr}(\mathsf{id} \times g)$.

Proof. We have from the definition of $(\cdot \times \cdot)$ that

$$\begin{aligned} \mathsf{id} \times g &= \lambda \sim(c, a) \to (\mathsf{id}\ \ c, g\ a) \\ &= (c, g\ a) \end{aligned}$$

and therefore, by simple substitution, that

$$\mathsf{arr}(\mathsf{id} \times g) = \mathsf{arr}(\lambda \sim(c, a) \to (c, g\ a))$$

Approaching the other side of the equation, we have

$$\begin{aligned} \mathsf{second}(\mathsf{arr}\, g) &= \mathsf{second}(\mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}\, g, g\ a)\}) \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0 = \lambda \sim(c_0, a_0) \to (\mathsf{spAux}\, sf_1, (c_0, b_0)) \text{ where} \\ &\quad\quad (sf_1, b_0) = tf_1 a_0 \\ &\quad\quad\quad = (\mathsf{sfArr}\, g, g\ a_0) \end{aligned}$$

and by expanding the definition of spAux, we get

$$\begin{aligned} \mathsf{spAux}\, sf_1 &= \mathsf{spAux}(\mathsf{sfArr}\, g) \\ &= \mathsf{sfArr}(\lambda \sim(c, a) \to (c, g\ a)) \end{aligned}$$

and therefore

$$\begin{aligned} \mathsf{second}(\mathsf{arr}\, g) &= \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(c_0, a_0) \to \\ &\qquad (\mathsf{sfArr}(\lambda \sim(c, a) \to (c, g\ a)), (c_0, g\ a_0))\} \\ &= \mathsf{arr}(\lambda \sim(c_0, a_0) \to (c_0, g\ a_0)) \end{aligned}$$

These two expressions are clearly equal, after renaming a and c to a' and c', and renaming a_0 and c_0 to a and c. Therefore we have

$$\mathsf{second}(\mathsf{arr}\, g) = \mathsf{arr}(\mathsf{id} \times g)$$

□

With that lemma, we can now begin proving the law in earnest. We do so, as usual, by proceeding with a case analysis.

1. Case $f = \mathsf{constant}\, b$:

   We know that $\mathsf{first}\, f = \mathsf{arr}(\lambda \sim(\_, c) \to (b, c))$. Let

   $$\mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} \mathsf{first}\, f$$

   We also know that

   $$\begin{aligned}\mathsf{arr}(\mathsf{id} \times g) = \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim(c, a) \to \\ (\mathsf{sfArr}(\lambda \sim(c', a') \to (c', g\ a')), \\ (c, g\ a))\}\end{aligned}$$

   and therefore will let

   $$\mathtt{SF}\{\mathtt{sfTF} = tf_c\} \stackrel{\text{def}}{=} \mathsf{arr}(\mathsf{id} \times g)$$

   Therefore,

   $$\begin{aligned}\mathsf{first}\, f \ggg \mathsf{arr}(\mathsf{id} \times g) &= \mathsf{compPrim}(\mathsf{first}\, f)(\mathsf{arr}(\mathsf{id} \times g)) \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf\} \text{ where} \\ &\quad tf = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\qquad (sf_1, b_0) = tf_f\ a_0@(a, c) \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(\_, c) \to (b, c)), (b, c)) \\ &\qquad (sf_2, c_0) = tf_c\ b_0 \\ &\qquad\qquad = tf_c(b, c) \\ &\qquad\qquad = (\mathsf{sfArr}(\mathsf{id} \times g), (\mathsf{id} \times g)(b, c)) \\ &\qquad\qquad = (\mathsf{sfArr}(\mathsf{id} \times g), (b, g\ c))\end{aligned}$$

   Simplifying cpAux, we have

   $$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_1(\lambda \sim(\_, c) \to (b, c))\ sf_2 \\ &= \mathsf{sfArr}((\mathsf{id} \times g) \cdot (\lambda \sim(\_, c) \to (b, c))) \\ &= \mathsf{sfArr}(\lambda \sim(\_, c) \to (b, g\ c))\end{aligned}$$

   From the other side of the equation, we have

   $$\begin{aligned}\mathsf{arr}(\mathsf{id} \times g) \ggg \mathsf{first}\, f &= \mathsf{compPrim}(\mathsf{arr}(\mathsf{id} \times g))(\mathsf{first}\, f) \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf\} \text{ where} \\ &\quad tf = \lambda a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\qquad (sf_1, b_0) = tf_c\ a_0@(a, c) \\ &\qquad\qquad = (\mathsf{sfArr}((\mathsf{id} \times g)), (a, g\ c)) \\ &\qquad (sf_2, c_0) = tf_f\ b_0 \\ &\qquad\qquad = tf_f(a, g\ c) \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(\_, c) \to (b, c)), \\ &\qquad\qquad\quad (b, g\ c))\end{aligned}$$

Simplifying cpAux we have

$$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_1}(\mathsf{id} \times g)\ sf_2 \\ &= \mathsf{sfArr}((\lambda \sim(\_, c) \to (b, c)) \cdot (\mathsf{id} \times g)) \\ &= \mathsf{sfArr}(\lambda \sim(\_, c) \to (b, g\ c))\end{aligned}$$

As

$$\mathsf{snd}(tf_0\ a_0) = (b, g\ c) = \mathsf{snd}(tf_1\ a_0)$$

and

$$\mathsf{fst}\, tf_0\ a_0 = \mathsf{sfArr}(\lambda \sim(\_, c) \to (b, g\ c) = \mathsf{fst}\, tf_1\ a_0$$

we have the desired equality.

2. Case $f = \mathsf{arr}\, f'$:

$$\begin{aligned}\mathsf{first}(\mathsf{arr}\, f') \ggg \mathsf{arr}(\mathsf{id} \times g) = \mathsf{arr}(\lambda \sim(a, c) \to (f'\ a, c)) \ggg \\ \mathsf{arr}(\lambda \sim(a, c) \to (a, g\ c))\end{aligned}$$

where the functions are obtained by unfolding $\mathsf{first}(\mathsf{arr}\, f')$ and $\mathsf{id} \times g$. Let $tf_{10} \stackrel{\text{def}}{=} \mathsf{first}(\mathsf{arr}\, f')$ and $tf_{20} \stackrel{\text{def}}{=} \mathsf{arr}(\mathsf{id} \times g)$.

$$\begin{aligned}&= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0\ a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\qquad (sf_1, b_0) = tf_{10}\ a_0@(a, c) \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(a, c) \to (f'\ a, c)), (f'\ a, c)) \\ &\qquad (sf_2, c_0) = tf_{20}\ b_0 \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(a, c) \to (a, g\ c)), (f'\ a, g\ c))\end{aligned}$$

Simplifying cpAux, we have

$$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_1}(\mathtt{sfAFun} sf_1) sf_2 \\ &= \mathsf{cpAuxA_1}(\lambda \sim(a, c) \to (f'\ a, c))\ \ \mathsf{sfArr}(\lambda \sim(a, c) \to (a, g\ c)) \\ &= \mathsf{sfArr}((\lambda \sim(a, c) \to (a, g\ c)) \cdot (\lambda \sim(a, c) \to (f'\ a, c))) \\ &= \mathsf{sfArr}(\lambda \sim(a, c) \to (f'\ a, g\ c))\end{aligned}$$

From the other side of the equation, we have

$$\begin{aligned}\mathsf{arr}(\mathsf{id} \times g) \ggg \mathsf{first}(\mathsf{arr}\, f') &= \mathsf{arr}(\lambda \sim(a, c) \to (a, g\ c)) \ggg \\ &\qquad \mathsf{arr}(\lambda \sim(a, c) \to (f'\ a, c)) \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_1\} \text{ where} \\ &\quad tf_1\ a_0 \to (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\qquad (sf_1, b_0) = tf_{10}\ a_0@(a, c) \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(a, c) \to (a, g\ c)), (a, g\ c)) \\ &\qquad (sf_2, c_0) = tf_{20}\ b_0 \\ &\qquad\qquad = (\mathsf{sfArr}(\lambda \sim(a, c) \to (f'\ a, c)), (f'\ a, g\ c))\end{aligned}$$

and again simplifying cpAux, we have

$$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_1}(\mathtt{sfAFun}sf_1)sf_2\\ &= \mathsf{cpAuxA_1}(\lambda\sim(a,c) \to (a, g\ c))\,\mathsf{sfArr}(\lambda\sim(a,c) \to (f'\ a, c))\\ &= \mathsf{sfArr}((\lambda\sim(a,c) \to (f'\ a, c))\cdot(\lambda\sim(a,c) \to (a, g\ c)))\\ &= \mathsf{sfArr}(\lambda\sim(a,c) \to (f'\ a, g\ c))\end{aligned}$$

As

$$\mathsf{snd}(tf_0\ a_0) = (f'\ a, g\ c) = \mathsf{snd}(tf_1\ a_0)$$

and

$$\mathsf{fst}\, tf_0\ a_0 = \mathsf{sfArr}(\lambda\sim(a,c) \to (f'\ a, g\ c) = \mathsf{fst}\, tf_1\ a_0$$

we have the desired equality.

3. Case $f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'\}, t'\ a)\}$: Let $\mathtt{SF}\{\mathtt{sfTF} = tf_g\} = \mathsf{arr}(\mathsf{id} \times g)$. Expanding $\mathsf{first}\, f$, we obtain

$$\begin{aligned}\mathsf{first}\, f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\}&\text{ where}\\ tf_f \sim(a,c) = (\mathsf{fpAux}\, sf', (b_f, c))&\text{ where}\\ (sf', b_f) = tf'a&\end{aligned}$$

Then

$$\begin{aligned}\mathsf{first}\, f \ggg \mathsf{arr}(\mathsf{id} \times g) = \mathtt{SF}\{\mathtt{sfTF} = tf_1\}&\text{ where}\\ tf_1\ a_0@(a,c) = (\mathsf{cpAux}\, sf_1\ sf_2, c_0)&\text{ where}\\ (sf_1, b_0) = tf_f\ a_0&\\ = (\mathsf{fpAux}\, sf_1', (b_f, c))&\\ (sf_2, c_0) = tf_g\ b_0&\\ = (\mathsf{sfArr}(\mathsf{id} \times g), (b_f, g\ c))&\end{aligned}$$

and unfolding cpAux once yields

$$\mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1')(\mathsf{id} \times g)$$

From the other side of the equation we have

$$\begin{aligned}\mathsf{arr}(\mathsf{id} \times g) \ggg \mathsf{first}\, f = \mathtt{SF}\{\mathtt{sfTF} = tf_2\}&\text{ where}\\ tf_2\ a_0@(a,c) = (\mathsf{cpAux}\, sf_1'\ sf_2', c_0)&\text{ where}\\ (sf_1', b_0) = tf_g\ a_0&\\ = (\mathsf{sfArr}(\mathsf{id} \times g), (a, g\ c))&\\ (sf_2', c_0) = tf_f\ b_0&\\ = (\mathsf{fpAux}\, sf', (b_f, g\ c))&\end{aligned}$$

and again unfolding cpAux once yields

$$\mathsf{cpAux}\, sf_1'\ sf_2' = \mathsf{cpAux}(\mathsf{id} \times g)(\mathsf{fpAux}\, sf')$$

It is clear by inspection that $\mathsf{snd}(tf_1\ a_0) = (b_f, g\ c) = \mathsf{snd}(tf_2\ a_0)$, so we now need to show that $\mathsf{fst}(tf_1\ a_0) = \mathsf{fst}(tf_2\ a_0)$.

Lemma 2.9. *For any sf,*

$$\mathsf{cpAuxA}_1(\mathsf{id} \times g)(\mathsf{fpAux}\, sf) = \mathsf{cpAuxA}_2(\mathsf{fpAux}\, sf)(\mathsf{id} \times g)$$

Proof. We proceed by cases on sf:

(a) Case $sf = \mathsf{sfConst}\, b$:
It is clear from the first case of fpAux that

$$\mathsf{fpAux}\, sf = \mathsf{sfArr}(\lambda \sim(\_, c) \to (b, c))$$

Expanding one side of the desired equality yields

$$\begin{aligned}\mathsf{cpAuxA}_1(\mathsf{id} \times g)(\mathsf{fpAux}\, sf) &= \mathsf{sfArr}((\lambda \sim(\_, c) \to (b, c)) \cdot (\mathsf{id} \times g)) \\ &= \mathsf{sfArr}(\lambda \sim(a, c) \to (b, g\ c))\end{aligned}$$

while the other side yields

$$\begin{aligned}\mathsf{cpAuxA}_2(\mathsf{fpAux}\, sf)(\mathsf{id} \times g) &= \mathsf{sfArr}((\mathsf{id} \times g) \cdot (\lambda \sim(\_, c) \to (b, c))) \\ &= \mathsf{sfArr}(\lambda \sim(a, c) \to (b, g\ c))\end{aligned}$$

(b) Case $sf = \mathsf{sfArr}\, h$: It is clear from the second case of fpAux that

$$\mathsf{fpAux}\, sf = \mathsf{sfArr}(\lambda \sim(a, c) \to (h\ a, c))$$

Expanding one side of the desired equality yields

$$\begin{aligned}\mathsf{cpAuxA}_1(\mathsf{id} \times g)(\mathsf{fpAux}\, sf) &= \mathsf{sfArr}((\lambda \sim(a, c) \to (h\ a, c)) \cdot (\mathsf{id} \times g)) \\ &= \mathsf{sfArr}(\lambda \sim(a, c) \to (h\ a, g\ c))\end{aligned}$$

while the other side yields

$$\begin{aligned}\mathsf{cpAuxA}_2(\mathsf{fpAux}\, sf)(\mathsf{id} \times g) &= \mathsf{sfArr}((\mathsf{id} \times g) \cdot (\lambda \sim(a, c) \to (h\ a, c))) \\ &= \mathsf{sfArr}(\lambda \sim(a, c) \to (h\ a, g\ c))\end{aligned}$$

(c) Case $sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}$: From the third case of fpAux we have

$$\begin{aligned}\mathsf{fpAux}\, sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'\} &\text{ where} \\ tf'dt\ \sim(a, c) = (\mathsf{fpAux}\, sf'_1, b) &\text{ where} \\ (sf'_1, b) = sf'dt\ a&\end{aligned}$$

Expanding one side of the desired equality yields

$$\begin{aligned}\mathsf{cpAuxA}_1(\mathsf{id} \times g)(\mathsf{fpAux}\, sf) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_{A_1}\} &\text{ where} \\ tf_{A_1}dt\ a@(a_0, c_0) = (\mathsf{cpAuxA}_1(\mathsf{id} \times g)sf'_2, c) &\text{ where} \\ (sf'_2, c) = tf'dt((\mathsf{id} \times g)a)& \\ = (\mathsf{fpAux}\, sf'_1, (b, g\ c_0))&\end{aligned}$$

while the other side yields

$$\begin{aligned}
\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf)(\mathsf{id} \times g) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_{A_2}\} \text{ where} \\
tf_{A_2} dt\ a@(a_0, c_0) &= (\mathsf{cpAuxA_2}\ sf'_3(\mathsf{id} \times g), (\mathsf{id} \times g)c) \text{ where} \\
(sf'_3, c) &= tf' dt\ a \\
&= (\mathsf{fpAux}\, sf'_1, (b, c_0)) \\
(\mathsf{id} \times g)c &= (b, g\ c_0)
\end{aligned}$$

Clearly $sf'_3 = \mathsf{fpAux}\, sf'_1 = sf'_2$. Assume by the induction hypothesis that $\mathsf{cpAuxA_1}(\mathsf{id} \times g)sf'_2 = \mathsf{cpAuxA_2}\ sf'_3(\mathsf{id} \times g)$. Then it is obvious that $tf_{A_2} = tf_{A_1}$, and hence

$$\mathsf{cpAuxA_1}(\mathsf{id} \times g)(\mathsf{fpAux}\, sf) = \mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf)(\mathsf{id} \times g)$$

as desired.

□

Using this lemma, clearly $\mathsf{fst}(tf_1\ a_0) = \mathsf{fst}(tf_2\ a_0)$, and therefore $tf_1 = tf_2$, and therefore we have the desired equality

$$\mathsf{first}\, f \ggg \mathsf{arr}(\mathsf{id} \times g) = \mathsf{arr}(\mathsf{id} \times g) \ggg \mathsf{first}\, f$$

□

Theorem 2.10. *Let f be an arrow. Then*

$$\mathsf{first}(\mathsf{first}\, f) \ggg \mathsf{arr}\, \mathsf{assoc} = \mathsf{arr}\, \mathsf{assoc} \ggg \mathsf{first}\, f$$

Proof. As usual, we proceed by cases of f:

1. Case $f = \mathsf{constant}\, b$:

   By the first case of first we have

   $$\mathsf{first}\, f = \mathsf{arr}(\lambda \sim (a_0, c_0) \to (b, c_0))$$

   Let $\mathsf{arr}\, h \stackrel{\text{def}}{=} \mathsf{first}\, f$.

   $$\mathsf{first}(\mathsf{first}\, f) = \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to ((b, c_0), c_1))$$

   Let $\mathsf{arr}\, g \stackrel{\text{def}}{=} \mathsf{first}(\mathsf{first}\, f)$. Then from one side of the equation we have

   $$\begin{aligned} \mathsf{arr}\, g \ggg \mathsf{arr}\, \mathsf{assoc} &= \mathsf{arr}(\mathsf{assoc} \cdot g) \\ &= \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to (b, (c_0, c_1))) \end{aligned}$$

   while from the other we have

   $$\begin{aligned} \mathsf{arr}\, \mathsf{assoc} \ggg \mathsf{first}\, f &= \mathsf{arr}(h \cdot \mathsf{assoc}) \\ &= \mathsf{arr}(\lambda \sim (a_1 @ (a_0, c_0), c_1) \to h(a_0, (c_0, c_1))) \\ &= \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to (b, (c_0, c_1))) \end{aligned}$$

2. Case $f = \mathsf{arr}\, f'$:

   $$\mathsf{first}\, f = \mathsf{arr}(\lambda \sim (a_0, c_0) \to (f'\ a_0, c_0))$$

   Let $\mathsf{arr}\, h \stackrel{\text{def}}{=} \mathsf{first}\, f$.

   $$\mathsf{first}(\mathsf{first}\, f) = \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to ((f'\ a_0, c_0), c_1))$$

   Let $\mathsf{arr}\, g \stackrel{\text{def}}{=} \mathsf{first}(\mathsf{first}\, f)$. Then from one side of the equation we have

   $$\begin{aligned} \mathsf{arr}\, g \ggg \mathsf{arr}\, \mathsf{assoc} &= \mathsf{arr}(\mathsf{assoc} \cdot g) \\ &= \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to (f'\ a_0, (c_0, c_1))) \end{aligned}$$

   while from the other we have

   $$\begin{aligned} \mathsf{arr}\, \mathsf{assoc} \ggg \mathsf{first}\, f &= \mathsf{arr}(h \cdot \mathsf{assoc}) \\ &= \mathsf{arr}(\lambda \sim (a_1 @ (a_0, c_0), c_1) \to h(a_0, (c_0, c_1))) \\ &= \mathsf{arr}(\lambda \sim ((a_0, c_0), c_1) \to (f'\ a_0, (c_0, c_1))) \end{aligned}$$

3. Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} \stackrel{\text{def}}{=} \mathtt{SF}\{\mathtt{sfTF} = \lambda\sim(a_0, c_0) \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0, c_0)\}$:

$$\begin{aligned}\mathsf{first}\, f = \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where}\\ tf_0\ \sim(a_0, c_0) = (\mathsf{fpAux}\, sf'_1, (b_0, c_0)) \text{ where}\\ (sf'_1, b_0) = tf_f\ a_0\end{aligned}$$

and so

$$\begin{aligned}\mathsf{first}(\mathsf{first}\, f) = \mathtt{SF}\{\mathtt{sfTF} = tf_1\} \text{ where}\\ tf_1\ \sim(a_1@(a, c), c_1) = (\mathsf{fpAux}\, sf_2, (b_1, c_1)) \text{ where}\\ (sf_2, b_1) = tf_0\ a_1\\ = (\mathsf{fpAux}\, sf'_1, (b_0, c))\end{aligned}$$

or

$$\begin{aligned}\mathsf{first}(\mathsf{first}\, f) = \mathtt{SF}\{\mathtt{sfTF} = \lambda\sim((a, c), c_1) \to (\mathsf{fpAux}(\mathsf{fpAux}\, sf_1), ((b_0, c), c_1))\} \text{ where}\\ (sf'_1, b_0) = tf_f\ a\end{aligned}$$

Therefore

$$\begin{aligned}\mathsf{first}(\mathsf{first}\, f) \ggg \mathsf{arr}\ \mathsf{assoc} = \mathtt{SF}\{\mathtt{sfTF} = tf_2\} \text{ where}\\ tf_2\ a_0@((a, b), c) = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where}\\ (sf_1, b_0) = (\mathsf{fpAux}(\mathsf{fpAux}\, sf'_1), ((a', b), c)) \text{ where}\\ (sf'_1, a') = tf_f\ a\\ (sf_2, c_0) = (\mathsf{sfArr}\ \mathsf{assoc}, (a', (b, c)))\\ \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fpAux}\, sf'_1))\, \mathsf{assoc}\end{aligned}$$

and

$$\begin{aligned}\mathsf{arr}\ \mathsf{assoc} \ggg \mathsf{first}\, f = \mathtt{SF}\{\mathtt{sfTF} = tf_3\} \text{ where}\\ tf_3\ a_0@((a, b), c) = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where}\\ (sf_1, b_0) = tf_{assoc}\ a_0\\ = (\mathsf{sfArr}\ \mathsf{assoc}, (a, (b, c)))\\ (sf_2, c_0) = tf_0\ b_0\\ = (\mathsf{fpAux}\, sf'_1, (a', (b, c))) \text{ where}\\ (sf'_1, a') = tf_f\ a\\ \mathsf{cpAux}(\mathsf{sfArr}\ \mathsf{assoc})(\mathsf{fpAux}\, sf'_1) = \mathsf{cpAuxA}_1\ \mathsf{assoc}(\mathsf{fpAux}\, sf'_1)\end{aligned}$$

Clearly $\mathsf{snd}(tf_2\ a_0) = (a', (b, c)) = \mathsf{snd}(tf_3\ a_0)$. To get $\mathsf{fst}(tf_2\ a_0) = \mathsf{fst}(tf_3\ a_0)$, we need a lemma:

Lemma 2.11. *With all variables as above,*

$$\mathsf{cpAuxA}_1\ \mathsf{assoc}(\mathsf{fpAux}\, sf'_1) = \mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fpAux}\, sf'_1))\, \mathsf{assoc}$$

Proof. We proceed by induction on cases of sf_1':

(a) Case $sf_1' = \mathsf{sfConst}\, k$:

$$\mathsf{fpAux}\, sf_1' = \mathsf{sfArr}(\lambda \sim(a_0, c_0) \to (k, c_0))$$

Let $\mathsf{sfArr}\, h \stackrel{\text{def}}{=} \mathsf{fpAux}\, f$.

$$\mathsf{fpAux}(\mathsf{fpAux}\, f) = \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to ((k, c_0), c_1))$$

Let $\mathsf{sfArr}\, g \stackrel{\text{def}}{=} \mathsf{fpAux}(\mathsf{fpAux}\, f)$. Then from one side we have

$$\begin{aligned} \mathsf{cpAuxA_1}\, \mathsf{assoc}(\mathsf{fpAux}\, sf_1') &= \mathsf{sfArr}(h \cdot \mathsf{assoc}) \\ &= \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to (k, (c_0, c_1))) \end{aligned}$$

while from the other we have

$$\begin{aligned} \mathsf{cpAuxA_2}(\mathsf{fpAux}(\mathsf{fpAux}\, sf_1'))\, \mathsf{assoc} &= \mathsf{sfArr}(\mathsf{assoc} \cdot g) \\ &= \mathsf{sfArr}(\lambda \sim(a_1 @ (a_0, c_0), c_1) \to h(a_0, (c_0, c_1))) \\ &= \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to (k, (c_0, c_1))) \end{aligned}$$

As these two expressions are equal, we have the desired result.

(b) Case $sf_1' = \mathsf{sfArr}\, f'$:

$$\mathsf{fpAux}\, sf_1' = \mathsf{sfArr}(\lambda \sim(a_0, c_0) \to (f'\ a_0, c_0))$$

Let $\mathsf{sfArr}\, h \stackrel{\text{def}}{=} \mathsf{fpAux}\, f$.

$$\mathsf{fpAux}(\mathsf{fpAux}\, f) = \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to ((f'\ a_0, c_0), c_1))$$

Let $\mathsf{sfArr}\, g \stackrel{\text{def}}{=} \mathsf{fpAux}(\mathsf{fpAux}\, f)$. Then from one side we have

$$\begin{aligned} \mathsf{cpAuxA_1}\, \mathsf{assoc}(\mathsf{fpAux}\, sf_1') &= \mathsf{sfArr}(h \cdot \mathsf{assoc}) \\ &= \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to (f'\ a_0, (c_0, c_1))) \\ \mathsf{cpAuxA_2}(\mathsf{fpAux}(\mathsf{fpAux}\, sf_1'))\, \mathsf{assoc} &= \mathsf{sfArr}(\mathsf{assoc} \cdot g) \end{aligned}$$

while from the other we have

$$\begin{aligned} &= \mathsf{sfArr}(\lambda \sim(a_1 @ (a_0, c_0), c_1) \to h(a_0, (c_0, c_1))) \\ &= \mathsf{sfArr}(\lambda \sim((a_0, c_0), c_1) \to (f'\ a_0, (c_0, c_1))) \end{aligned}$$

As these two expressions are equal, we have the desired result.

(c) Case $sf_1' = \texttt{SFTIVar}\{\texttt{sfTF}' = tf_s'\}$:

$$\begin{aligned}
\mathsf{fpAux}\, sf_1' &= \texttt{SFTIVar}\{\texttt{sfTF}' = tf'\} \text{ where} \\
&\quad tf'\ dt\ \sim(a,c) = (\mathsf{fpAux}\, sf_1'', (b,c)) \text{ where} \\
&\quad\quad (sf_1'', b) = tf_s'\ dt\ a \\
\mathsf{fpAux}(\mathsf{fpAux}\, sf_1') &= \texttt{SFTIVar}\{\texttt{sfTF} = tf''\} \text{ where} \\
&\quad tf''\ dt\ \sim(a@(x,y),c) = (\mathsf{fpAux}\, sf_1''', (b,c)) \text{ where} \\
&\quad\quad (sf_1''', b) = (\texttt{sfTF}'(\mathsf{fpAux}\, sf_1'))\ dt\ a \\
&\quad\quad\quad = tf'\ dt\ a \\
&\quad\quad\quad = (\mathsf{fpAux}\ \ sf_1'', (b', y)) \text{ where} \\
&\quad\quad\quad\quad (sf_1'', b') = tf_s'\ dt\ x
\end{aligned}$$

Then

$$\begin{aligned}
\mathsf{cpAuxA}_1\, \mathsf{assoc}(\mathsf{fpAux}\, sf_1') &= \texttt{SFTIVar}\{\texttt{sfTF}' = tfc'\} \text{ where} \\
&\quad tfc'\ dt\ a@((x,y),z) = (\mathsf{cpAuxA}_1\, \mathsf{assoc}\, sf_2', c) \text{ where} \\
&\quad\quad (sf_2', c) = tf'\ dt\ (\mathsf{assoc}\, a) \\
&\quad\quad\quad = (\mathsf{fpAux}\, sf_1'', (b, (y,z))) \text{ where} \\
&\quad\quad\quad\quad (sf_1'', b) = tf_s'\ dt\ x
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fpAux}\, sf_1'))\, \mathsf{assoc} &= \texttt{SFTIVar}\{\texttt{sfTF}' = tfc''\} \text{ where} \\
&\quad tfc''\ dt\ a@((x,y),z) = (\mathsf{cpAuxA}_2\, sf_2'\, \mathsf{assoc}, \mathsf{assoc}\, a) \text{ where} \\
&\quad\quad (sf_2', c) = tf''\ dt\ a \\
&\quad\quad\quad = (\mathsf{fpAux}(\mathsf{fpAux}\, sf_1''), ((b,y),z)) \text{ where} \\
&\quad\quad\quad\quad (sf_1'', b) = tf_s'\ dt\ x
\end{aligned}$$

We assume

$$\mathsf{cpAuxA}_1\, \mathsf{assoc}(\mathsf{fpAux}\, sf_1'') = \mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fpAux}\, sf_1''))\, \mathsf{assoc}$$

by the induction hypothesis. Therefore, we have the lemma that

$$\mathsf{cpAuxA}_1\, \mathsf{assoc}(\mathsf{fpAux}\, sf_1') = \mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fpAux}\, sf_1'))\, \mathsf{assoc}$$

□

Given this lemma, we have the desired result that

$$\mathsf{first}(\mathsf{first}\, f) \ggg \mathsf{arr}\, \mathsf{assoc} = \mathsf{arr}\, \mathsf{assoc} \ggg \mathsf{first}\, f$$

□

2.1 Invalid laws under strict equality

These two laws break where constant and arr interact. As a result, we need an evaluation function to show equivalence instead of strict equality. That function is defined herein as follows:

$$\begin{aligned}
&\mathsf{evalSF}\ \mathtt{SF}\{\mathtt{sfTF} = tf\}\ []\ _ = []\\
&\mathsf{evalSF}\ \mathtt{SF}\{\mathtt{sfTF} = tf\}\ a : as\ dts = \mathsf{evalSFTF}(tf\ a)\ dts\ as\\
&\text{where}\ \ \mathsf{evalSFTF}(sf', b)\ dts\ as = b : (\mathsf{evalSF}'\ sf'\ dts\ as)\\
&\qquad \mathsf{evalSF}'\ _\ []\ _ = []\\
&\qquad \mathsf{evalSF}'\ _\ _\ [] = []\\
&\qquad \mathsf{evalSF}'\ \mathtt{SFConst}\{\mathtt{sfTF}' = tf', \mathtt{sfCVal} = b\}\ dt : dts\ a : as = \mathsf{evalSFTF}'(tf'\ dt\ a)\ dts\ as\\
&\qquad \mathsf{evalSF}'\ \mathtt{SFArr}\{\mathtt{sfTF}' = tf', \mathtt{sfAFun} = f\}\ dt : dts\ a : as = \mathsf{evalSFTF}'(tf'\ dt\ a)\ dts\ as\\
&\qquad \mathsf{evalSF}'\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'\}\ dt : dts\ a : as = \mathsf{evalSFTF}'(tf'\ dt\ a)\ dts\ as
\end{aligned}$$

This definition ignores cached constant values or functions for now; it can be trivially modified to include them, and the proofs involving it modified accordingly to use the new values.

Theorem 2.12. *Let f and g be arrows. Then*

$$\mathsf{first}(f \ggg g) = \mathsf{first}\ f \ggg \mathsf{first}\ g$$

Proof. We proceed by cases on f and g:

1. Case $g = \mathtt{SF}\{\mathtt{sfTF} = tf_g\} = \mathsf{constant}\ b$:

   In this case, $f \ggg g = g$, and so

   $$\mathsf{first}(f \ggg g) = \mathsf{first}\ g = \mathsf{arr}(\lambda \sim(\_, c) \to (b, c))$$

   which is easily checked with two unfolding steps. Let $\mathsf{arr}\ g' \stackrel{\text{def}}{=} \mathsf{first}\ g$.

   (a) Case $f = \mathsf{constant}\ h$: By the same two unfoldings as above,

   $$\mathsf{first}\ f = \mathsf{arr}(\lambda \sim(\_, c) \to (h, c))$$

   and so

   $$\begin{aligned}
   \mathsf{first}\ f \ggg \mathsf{first}\ g &= \mathsf{arr}(g' \cdot (\lambda \sim(_, c) \to (h, c)))\\
   &= \mathsf{arr}(\lambda \sim(_, c) \to (b, c))\\
   &= \mathsf{arr}\ g'\\
   &= \mathsf{first}(f \ggg g)
   \end{aligned}$$

   (b) Case $f = \mathsf{arr}\ f'$:

   Using the second case of first, we have

   $$\mathsf{first}\ f = \mathsf{arr}(\lambda \sim(a, c) \to (f'\ a, c))$$

   and so

   $$\begin{aligned}
   \mathsf{first}\ f \ggg \mathsf{first}\ g &= \mathsf{arr}(g' \cdot (\lambda \sim(a, c) \to (f'\ a, c)))\\
   &= \mathsf{arr}(\lambda \sim(_, c) \to (b, c))\\
   &= \mathsf{arr}\ g'\\
   &= \mathsf{first}(f \ggg g)
   \end{aligned}$$

(c) Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a)\}$:
Using the third case of first, we have

$$\begin{aligned}
\mathsf{first}\, f &= \mathtt{SF}\{\mathtt{sfTF} = tf_{ffirst}\} \text{ where} \\
&\quad tf_{ffirst} \sim (a_0, c_0) = (\mathsf{fpAux}\, sf'_1, (b_0, c_0)) \text{ where} \\
&\qquad (sf_1, b_0) = tf_f\ a_0 \\
&\qquad\qquad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0) \\
&\quad \mathsf{fpAux}\, sf_1 = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{ffirst}\} \text{ where} \\
&\qquad tf'_{ffirst}\ dt\ \sim (a, c) = (\mathsf{fpAux}\, sf'_1, (b', c)) \text{ where} \\
&\qquad\quad (sf'_1, b') = tf'_f\ dt\ a
\end{aligned}$$

and therefore

$$\begin{aligned}
\mathsf{first}\, f \ggg \mathsf{first}\, g &= \mathtt{SF}\{\mathtt{sfTF} = tf_{fg}\} \text{ where} \\
&\quad tf_{fg}\ a_0 @ (a, c) = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\qquad (sf_1, b_0) = tf_{ffirst}\ a_0 \\
&\qquad\qquad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{ffirst}\}, (f'\ a, c)) \\
&\qquad (sf_2, c_0) = tf_g\ b_0 \\
&\qquad\qquad = (\mathsf{sfArr}(\lambda \sim (_, c) \to (b, c)), (b, c)) \\
&\qquad\qquad = (\mathsf{sfArr}\, g', (b, c)) \\
&\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA}_2\, sf_1\ g' \\
&\qquad\qquad = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg}\} \text{ where} \\
&\qquad\qquad\quad tf'_{fg}\ dt\ (a, c) = (\mathsf{cpAuxA}_2\, sf'_1\ g', g'b') \text{ where} \\
&\qquad\qquad\qquad (sf'_1, b') = tf_{ffirst}\ dt\ (a, c) \\
&\qquad\qquad\qquad\qquad = (\mathsf{fpAux}\, sf''_1, (b'', c)) \text{ where} \\
&\qquad\qquad\qquad\qquad (sf''_1, b'') = tf'_f\ dt\ a
\end{aligned}$$

where that last is obtained by substituting the value for tf_{ffirst} found above, and renaming variables as needed. Simplifying all of this, we have

$$\begin{aligned}
\mathsf{first}(f \ggg g) &= \mathsf{arr}(\lambda \sim (_, c) \to (b, c)) \\
&= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 @ (a, c) \to (\mathsf{sfArr}\, g', (b, c))\} \\
\mathsf{first}\, f \ggg \mathsf{first}\, g &= \mathtt{SF}\{\mathtt{sfTF} = tf_{fg}\} \text{ where} \\
&\quad tf_{fg} = \lambda a_0 @ (a, c) \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg}\}, (b, c)) \text{ where} \\
&\qquad tf'_{fg}\ dt\ (a, c) = (\mathsf{cpAuxA}_2\, sf'_1\ g', (b, c)) \text{ where} \\
&\qquad\quad sf'_1 = \mathsf{fpAux}(\mathsf{fst}(tf'_f\ dt\ a))
\end{aligned}$$

Clearly, these two constructions are not equal. Their second elements are equal, as is obvious by inspection, however their first elements are not. They are equivalent, however, under evalSF, and showing that will give the equality we need:

Lemma 2.13. *With all variables defined as above,*

$$\mathsf{evalSF}(\mathsf{arr}\, g')\ a_0 : as\ dts = \mathsf{evalSF}\, \mathtt{SF}\{\mathtt{sfTF} = tf_{fg}\}\ a_0 : as\ dts$$

Proof. We prove that after the first expansion, both sides contain subexpressions which evaluate to the same value:

$$\mathsf{evalSF}(\mathsf{arr}\, g')a_0 : as\ dts = \mathsf{evalSFTF}(tf_g\ a_0)\ dts\ as$$

Noting that a_0 must be a pair, and noting that g' accepts lazy pairs, let $(a, c) \stackrel{\text{def}}{=} a_0$

$$= (b, c) : (\mathsf{evalSF}'\, \mathsf{sfArr}(g')\ dts\ as)$$

Let $a_1@(a', c') : as' \stackrel{\text{def}}{=} as$, and $dt : dts' \stackrel{\text{def}}{=} dts$

$$\begin{aligned} &= (b, c) : (\mathsf{evalSFTF}(tf'_g\ dt\ a_1)\ dts'\ as') \\ &= (b, c) : (b, c') : (\mathsf{evalSF}'\, \mathsf{sfArr}(g')\ dts'\ as') \end{aligned}$$

Now, by repeated unfolding and substitution,

$$\begin{aligned} &\mathsf{evalSF}\mathtt{SF}\{\mathtt{sfTF} = tf_{fg}\}a_0 : as\ dts = \mathsf{evalSFTF}(tf_{fg}\ a_0)\ dts\ as \\ &\quad = (b, c) : (\mathsf{evalSF}'\, \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg}\}\ dts\ as) \\ &\quad = (b, c) : (\mathsf{evalSFTF}(tf'_{fg}\ dt\ a_1)\ dts'\ as') \\ &\quad = (b, c) : (\mathsf{evalSFTF}(\mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fst}(tf'_f\ dt\ a_1)))g', g'(\mathsf{snd}(tf'_f\ dt\ a_1), c'))\ dts'\ as') \\ &\quad = (b, c) : (\mathsf{evalSFTF}(\mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fst}(tf'_f\ dt\ a_1)))g', (b, c'))\ dts'\ as') \\ &\quad = (b, c) : (b, c') : (\mathsf{evalSF}'(\mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fst}(tf'_f\ dt\ a')))g')\ dts'\ as') \end{aligned}$$

We now need to show that those last two subexpressions are equal. We proceed inductively by cases on the right hand side:

i. Case $\mathsf{fst}(tf'_f\ dt\ a') = \mathsf{sfConst}\, h$:
We immediately have

$$\mathsf{fpAux}\, \mathsf{sfConst}\, h = \mathsf{sfArr}(\lambda \sim(\_, c) \to (h, c))$$

and therefore

$$\mathsf{cpAuxA}_2(\mathsf{fpAux}\, \mathsf{sfConst}\, h)g' = \mathsf{sfArr}\, g'$$

which is equal to the above left hand expression and we are done.

ii. Case $\mathsf{fst}(tf'_f\ dt\ a') = \mathsf{sfArr}\ f''$:
We immediately have

$$\mathsf{fpAux}\, \mathsf{sfArr}\ f'' = \mathsf{sfArr}(\lambda \sim(a, c) \to (f''\ a, c))$$

and therefore

$$\mathsf{cpAuxA}_2(\mathsf{fpAux}\, \mathsf{sfConst}\, h)g' = \mathsf{sfArr}\, g'$$

which is equal to the above left hand expression and we are done.

iii. Case $\mathsf{fst}(tf'_f\ da\ a') = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_f\}$:

$$\begin{aligned}\mathsf{fpAux}\,\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_f\} = \\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{ffirst}\}\ \text{where} \\ tf'_{ffirst}\ dt\ \sim(a,c) = (\mathsf{fpAux}\,sf'_1, (b,c))\ \text{where} \\ (sf'_1, b) = tf''_f\ dt\ a\end{aligned}$$

and therefore

$$\begin{aligned}\mathsf{cpAuxA}_2(\mathsf{fpAux}\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_f\})g' = \\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_{ffirst}\}\ \text{where} \\ tf''_{ffirst}\ dt\ a = (\mathsf{cpAuxA}_2(\mathsf{fpAux}\,sf'_1)g', g'\ b')\ \text{where} \\ (sf'_1, b') = tf''_f\ dt\ a\end{aligned}$$

By the inductive hypothesis,

$$\mathsf{evalSF}(\mathsf{cpAuxA}_2(\mathsf{fpAux}(\mathsf{fst}(tf''_f\ dt\ a)))g')\ dts\ as = \mathsf{evalSF}(\mathsf{sfArr}\,g')\ dts\ as$$

since tf''_f is of lower nesting depth of $\mathtt{SFTIVar}\{\}$ constructions, and so we are done.

□

2. Case $g = \mathtt{SF}\{\mathtt{sfTF} = tf_g\} = \mathsf{arr}\,g'$:

   (a) Case $f = \mathsf{constant}\,h$:
   We know from the first two cases of first that

$$\begin{aligned}\mathsf{first}\,f &= \mathsf{arr}(\lambda\sim(\_,c) \to (b,c)) \\ \mathsf{first}\,g &= \mathsf{arr}(\lambda\sim(a,c) \to (g'\ a,c))\end{aligned}$$

   Therefore,

$$\mathsf{first}(f \ggg g) = \mathsf{arr}(\lambda\sim(\_,c) \to (g'\ b,c))$$

   and

$$\mathsf{first}\,f \ggg \mathsf{first}\,g = \mathsf{arr}(\lambda\sim(\_,c) \to (g'\ b,c))$$

   (b) Case $f = \mathsf{arr}\,f'$: By the exact same logic as above

$$\begin{aligned}\mathsf{first}\,f &= \mathsf{arr}(\lambda\sim(a,c) \to (f'\ a,c)) \\ \mathsf{first}\,g &= \mathsf{arr}(\lambda\sim(a,c) \to (g'\ a,c)) \\ \mathsf{first}(f \ggg g) &= \mathsf{arr}(\lambda\sim(a,c) \to (g'(f'\ a),c)) \\ \mathsf{first}\,f \ggg \mathsf{first}\,g &= \mathsf{arr}(\lambda\sim(\_,c) \to (g'(f'\ a),c))\end{aligned}$$

(c) Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a)\}$:

$$
\begin{aligned}
\mathsf{first}\, f &= \mathtt{SF}\{\mathtt{sfTF} = tf_{f_2}\} \text{ where} \\
&\qquad tf_{f_2}\ \sim(a_0, c_0) = (\mathsf{fpAux}\, sf_1, (b_0, c_0)) \text{ where} \\
&\qquad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\qquad\qquad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a) \\
&\qquad \mathsf{fpAux}\, sf_1 = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_f\} \text{ where} \\
&\qquad\quad tf''_f\ dt\ \sim(a, c) = (\mathsf{fpAux}\, sf'_1, (b, c)) \text{ where} \\
&\qquad\qquad (sf'_1, b) = tf'_f\ dt\ a
\end{aligned}
$$

and

$$
\begin{aligned}
\mathsf{first}\, g &= \mathsf{arr}(\lambda \sim(a, c) \to (g'\ a, c)) \\
&= \mathtt{SF}\{\mathtt{sfTF} = tf_{f_2}\} \text{ where} \\
&\qquad tf_{g_2}\ a_0@(a, c) = (\mathsf{sfArr}(\lambda \sim(a, c) \to (g'\ a, c)), (g'\ a, c))
\end{aligned}
$$

Let $\mathsf{arr}\, g_2 \stackrel{\text{def}}{=} \mathsf{first}\, g$. Then

$$
\begin{aligned}
\mathsf{first}\, f \ggg \mathsf{first}\, g &= \mathtt{SF}\{\mathtt{sfTF} = tf_{fg_1}\} \text{ where} \\
&\qquad tf_{fg_1}\ a@(a_0, c_0) = (\mathsf{cpAux}\, sf_1\ sf_2, c'_0) \text{ where} \\
&\qquad\quad (sf_1, b_0) = tf_{f_2}\ a \\
&\qquad\qquad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_f\}, (f'\ a_0, c_0)) \\
&\qquad\quad (sf_2, c'_0) = tf_{g_2}\ b_0 \\
&\qquad\qquad\quad = (\mathsf{sfArr}\, g_2, (g'(f'\ a_0), c_0)) \\
&\qquad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA}_2\, sf_1\ g_2 \\
&\qquad\qquad\quad = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg_1}\} \text{ where} \\
&\qquad\qquad\quad tf'_{fg_1}\ dt\ (a, c) = (\mathsf{cpAuxA}_2\, sf'_1\ g_2, g_2\ b') \text{ where} \\
&\qquad\qquad\qquad (sf'_1, b') = tf''_f\ dt\ (a, c) \\
&\qquad\qquad\qquad\quad = (\mathsf{fpAux}\, sf''_1, (b, c)) \text{ where} \\
&\qquad\qquad\qquad\quad (sf''_1, b) = tf'_f\ dt\ a
\end{aligned}
$$

On the other side of the equation, we have

$$
\begin{aligned}
f \ggg g = \mathtt{SF}\{\mathtt{sfTF} = tf'_{fg_2}\} \text{ where} \\
tf'_{fg_2}\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_f\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0) \\
(sf_2, c_0) &= tf_g\ b_0 \\
&= (\mathsf{sfArr}\, g', g'(f'\ a_0)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_2}\, sf_1\ g' \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_{fg_2}\} \text{ where} \\
tf''_{fg_2}\ dt\ a &= (\mathsf{cpAuxA_2}\, sf'_1\ g', g'\ b) \text{ where} \\
(sf'_1, b) &= tf'_f\ dt\ a
\end{aligned}
$$

and so

$$
\begin{aligned}
\mathsf{first}(f \ggg g) = \mathtt{SF}\{\mathtt{sfTF} = tf_{fg_2}\} \text{ where} \\
tf_{fg_2} a @ (a_0, c_0) &= (\mathsf{fpAux}\, sf_1, (b_0, c_0)) \text{ where} \\
(sf_1, b_0) &= tf'_{fg_2}\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf''_{fg_2}\}, g'(f'\ a_0)) \\
\mathsf{fpAux}\, sf_1 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'''_{fg_2}\} \text{ where} \\
tf'''_{fg_2}\ dt\ \sim(a, c) &= (\mathsf{fpAux}\, sf'_1, (b', c)) \text{ where} \\
(sf'_1, b') &= tf''_{fg_2}\ dt\ a \\
&= (\mathsf{cpAuxA_2}\, sf''_1\ g', g'\ b) \text{ where} \\
(sf''_1, b) &= tf'_f\ dt\ a
\end{aligned}
$$

Now,

$$\mathsf{snd}(tf_{fg_1}\ (a_0, c_0)) = (g'(f'\ a_0), c_0) = \mathsf{snd}(tf_{fg_2}\ (a_0, c_0))$$

and

$$
\begin{aligned}
\mathsf{fst}(tf_{fg_1}\ (a_0, c_0)) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg_1}\} \text{ where} \\
tf'_{fg_1}\ dt\ \sim(a, c) &= (\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf''_1)\ g_2, (g'\ b, c)) \text{ where} \\
(sf''_1, b) &= tf'_f\ dt\ a
\end{aligned}
$$

while

$$
\begin{aligned}
\mathsf{fst}(tf_{fg_2}\ (a_0, c_0)) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'''_{fg_2}\} \text{ where} \\
tf'''_{fg_2}\ dt\ \sim(a, c) &= (\mathsf{fpAux}(\mathsf{cpAuxA_2}\, sf''_1\ g'), (g'\ b, c)) \text{ where} \\
(sf''_1, b) &= tf'_f\ dt\ a
\end{aligned}
$$

Naturally, we have the following lemma, which proves the above are equal:

Lemma 2.14. *With all variables as above,*

$$\mathsf{fpAux}(\mathsf{cpAuxA_2}\, sf''_1\ g') = \mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf''_1)\ g_2$$

Proof. As usual, proceed by cases on sf_1'':

i. Case $sf_1'' = \mathsf{sfConst}\, k$:

$$\mathsf{fpAux}\, sf_1'' = \mathsf{sfArr}(\lambda \sim(\_, c) \to (k, c))$$

Let $\mathsf{sfArr}\, f_k \stackrel{\text{def}}{=} \mathsf{fpAux}\, sf_1''$

$$\begin{aligned}
\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1'')\ g_2 &= \mathsf{sfArr}(g_2 \cdot f_k) \\
&= \mathsf{sfArr}(\lambda \sim(_, c) \to (g'\ k, c)) \\
\mathsf{cpAuxA_2}\, sf_1''\ g' &= \mathsf{sfConst}(g'\ k) \\
\mathsf{fpAux}(\mathsf{cpAuxA_2}\, sf_1''\ g') &= \mathsf{sfArr}(\lambda \sim(_, c) \to (g'\ k, c))
\end{aligned}$$

ii. Case $sf_1'' = \mathsf{sfArr}\, h$:

$$\mathsf{fpAux}\, sf_1'' = \mathsf{sfArr}(\lambda \sim(a, c) \to (h\ a, c))$$

Let $\mathsf{sfArr}\, f_k \stackrel{\text{def}}{=} \mathsf{fpAux}\, sf_1''$

$$\begin{aligned}
\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1'')\ g_2 &= \mathsf{sfArr}(g_2 \cdot f_k) \\
&= \mathsf{sfArr}(\lambda \sim(a, c) \to (g'(h\ a), c)) \\
\mathsf{cpAuxA_2}\, sf_1''\ g' &= \mathsf{sfArr}(g' \cdot h) \\
\mathsf{fpAux}(\mathsf{cpAuxA_2}\, sf_1''\ g') &= \mathsf{sfArr}(\lambda \sim(a, c) \to ((g' \cdot h)a, c)) \\
&= \mathsf{sfArr}(\lambda \sim(a, c) \to (g'(h\ a), c))
\end{aligned}$$

iii. Case $sf_1'' = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s'\}$:

$$\begin{aligned}
\mathsf{fpAux}\, sf_1'' &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s''\} \text{ where} \\
&\quad tf_s''\ dt\ \sim(a, c) = (\mathsf{fpAux}\, sf_1', (b, c)) \text{ where} \\
&\quad\quad (sf_1', b) = tf_s'\ dt\ a \\
\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1'')\ g_2 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s'''\} \text{ where} \\
&\quad tf_s'''\ dt\ a_0@(a, c) = (\mathsf{cpAuxA_2}\, sf_1'''\ g_2, g_2\ b') \text{ where} \\
&\quad\quad (sf_1''', b') = tf_s''\ dt\ a_0 \\
&\quad\quad\quad = (\mathsf{fpAux}\, sf_1', (b, c)) \text{ where} \\
&\quad\quad\quad\quad (sf_1', b) = tf_s'\ dt\ a
\end{aligned}$$

So

$$\begin{aligned}
\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1'')\ g_2 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s'''\} \text{ where} \\
&\quad tf_s'''\ dt\ a_0@(a, c) = (\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1')\ g_2, (g'\ b, c)) \text{ where} \\
&\quad\quad (sf_1', b) = tf_s'\ dt\ a
\end{aligned}$$

On the other side, we have

$$\begin{aligned}
\mathsf{cpAuxA_2}\ sf_1''\ g' &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s''\} \text{ where} \\
&\quad tf_s''\ dt\ a = (\mathsf{cpAuxA_2}\ sf_1'\ g', g'\ b) \text{ where} \\
&\quad\quad (sf_1', b) = tf_s'\ dt\ a \\
\mathsf{fpAux}(\mathsf{cpAuxA_2}\ sf_1'\ g') &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s'''\} \text{ where} \\
&\quad tf_s'''\ dt\ {\sim}(a, c) = (\mathsf{fpAux}\ sf_1''', (b', c)) \text{ where} \\
&\quad\quad (sf_1''', b') = tf_s''\ dt\ a \\
&\quad\quad\quad = (\mathsf{cpAuxA_2}\ sf_1'\ g', g'\ b) \text{ where} \\
&\quad\quad\quad\quad (sf_1', b) = tf_s'\ dt\ a
\end{aligned}$$

So

$$\begin{aligned}
\mathsf{fpAux}(\mathsf{cpAuxA_2}\ sf_1''\ g') &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_s'''\} \text{ where} \\
&\quad tf_s'''\ dt\ a_0@(a, c) = (\mathsf{fpAux}(\mathsf{cpAuxA_2}\ sf_1'\ g'), (g'\ b, c)) \text{ where} \\
&\quad\quad (sf_1', b) = tf_s'\ dt\ a
\end{aligned}$$

Clearly these two sides are equal, and hence the lemma is proven.

□

With the lemma as above, we have that $tf_{fg_1} = tf_{fg_2}$, and so we have the desired result in this case.

3. Case $g = \mathtt{SF}\{\mathtt{sfTF} = tf_g\} = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_g'\}, g'\ a)\}$:

   (a) Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} = \mathsf{constant}\, h$:

$$\begin{aligned}
f \ggg g &= \mathtt{SF}\{\mathtt{sfTF} = tf_{fg}\} \text{ where} \\
&\quad tf_{fg}\ a_0 = (\mathsf{cpAux}\ sf_1\ sf_2, c_0) \text{ where} \\
&\quad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\quad\quad\quad = (\mathsf{sfConst}\, h, h) \\
&\quad\quad (sf_2, c_0) = tf_g\ b_0 \\
&\quad\quad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_g'\}, g'\ h) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxC_1}\, h\ sf_2 \\
&\quad = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_{fg}'\} \text{ where} \\
&\quad\quad tf_{fg}'\ dt\ _ = (\mathsf{cpAuxC_1}\, h\ sf_2', c) \text{ where} \\
&\quad\quad\quad (sf_2', c) = tf_g'\ dt\ h
\end{aligned}$$

Then

$$\begin{aligned}
\mathsf{first}(f \ggg g) = \mathtt{SF}\{\mathtt{sfTF} = tf_{fg_1}\} \text{ where} \\
tf_{fg_1}\ \sim(a_0, c_0) &= (\mathsf{fpAux}\, sf_1, (b_0, c_0)) \text{ where} \\
(sf_1, b_0) &= tf_{fg}\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg}\}, g'\ h) \\
\mathsf{fpAux}\, sf_1 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg_1}\} \text{ where} \\
tf'_{fg_1}\ dt\ \sim(a, c) &= (\mathsf{fpAux}\, sf'_1, (b, c)) \text{ where} \\
(sf'_1, b) &= tf'_{fg}\ dt\ a \\
&= (\mathsf{cpAuxC_1}\, h\ sf'_2, c') \text{ where} \\
(sf'_2, c') &= tf'_g\ dt\ h
\end{aligned}$$

On the other side, we have

$$\mathsf{first}\, f = \mathsf{arr}(\lambda \sim(\_, c) \to (h, c))$$

Let $\mathtt{SF}\{\mathtt{sfTF} = tf_{firstf}\} \stackrel{\text{def}}{=} \mathsf{arr}\, f' \stackrel{\text{def}}{=} \mathsf{first}\, f$

$$\begin{aligned}
\mathsf{first}\, g = \mathtt{SF}\{\mathtt{sfTF} = tf_{firstg}\} \text{ where} \\
tf_{firstg}\ \sim(a_0, c_0) &= (\mathsf{fpAux}\, sf_1, (b_0, c_0)) \text{ where} \\
(sf_1, b_0) &= tf_g a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_g\}, g'\ a_0) \\
\mathsf{fpAux}\, sf_1 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{firstg}\} \text{ where} \\
tf'_{firstg}\ dt\ \sim(a, c) &= (\mathsf{fpAux}\, sf'_1, (b, c)) \text{ where} \\
(sf'_1, b) &= tf'_g\ dt\ a
\end{aligned}$$

And therefore,

$$\begin{aligned}
\mathsf{first}\, f \ggg \mathsf{first}\, g = \mathtt{SF}\{\mathtt{sfTF} = tf_{fg_2}\} \text{ where} \\
tf_{fg_2}\ a_0@(a, c) &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_{firstf}\ a_0 \\
&= (\mathsf{sfArr}\, f', (h, c)) \\
(sf_2, c_0) &= tf_{firstg}\ b_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{firstg}\}, (g'\ h, c)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_1}\, f'\ sf_2 \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fg_2}\} \text{ where} \\
tf'_{fg_2}\ dt\ a@(a_1, c_1) &= (\mathsf{cpAuxA_1}\, f'\ sf'_2, c') \text{ where} \\
(sf'_2, c') &= tf'_{firstg}\ dt\ (f'\ a) \\
&= (\mathsf{fpAux}\, sf'_1, (b, c)) \text{ where} \\
(sf'_1, b) &= tf'_g\ dt\ h
\end{aligned}$$

Up to renaming, it is clear that

$$\mathsf{snd}(tf_{fg_1}\ (a, c)) = (g'\ h, c) = \mathsf{snd}(tf_{fg_2}\ (a, c))$$

and that

$$\mathsf{snd}(tf'_{fg_1}\ dt\ (a,c)) = (\mathsf{snd}(tf'_g\ dt\ h), c) = \mathsf{snd}(tf'_{fg_2}\ dt\ (a,c))$$

and if $\mathsf{fst}(tf'_{fg_1}\ dt\ (a,c)) = \mathsf{fst}(tf'_{fg_2}\ dt\ (a,c))$, then we can show that $tf_{fg_1} = tf_{fg_2}$, and we are done.

Lemma 2.15. *With all variables as above,*

$$\mathsf{fpAux}(\mathsf{cpAuxC}_1\ h\ (\mathsf{fst}(tf'_g\ dt\ h))) = \mathsf{cpAuxA}_1\ f'\ (\mathsf{fpAux}(\mathsf{fst}(tf'_g\ dt\ h)))$$

Proof. Proceed, as always, by cases on $sf \stackrel{\text{def}}{=} \mathsf{fst}(tf'_g\ dt\ h)$:

i. Case $sf = \mathsf{sfConst}\,k$:

$$\begin{aligned}
\mathsf{cpAuxC}_1\,h\,\mathsf{sfConst}\,k &= \mathsf{sfConst}\,k \\
\mathsf{fpAux}\,\mathsf{sfConst}\,k &= \mathsf{sfArr}(\lambda\sim(_,c) \to (k,c)) \\
\mathsf{cpAuxA}_1\ f'\ (\mathsf{fpAux}\,sf) &= \mathsf{sfArr}(\lambda\sim(_,c) \to (k,c))
\end{aligned}$$

ii. Case $sf = \mathsf{sfArr}\,k'$:

$$\begin{aligned}
\mathsf{cpAuxC}_1\,h\ \ \mathsf{sfArr}\,k' &= \mathsf{sfConst}(k'\ h) \\
\mathsf{fpAux}\,\mathsf{sfConst}(k'h) &= \mathsf{sfArr}(\lambda\sim(_,c) \to (k'\ h,c)) \\
\mathsf{fpAux}\,\mathsf{sfArr}\,k' &= \mathsf{sfArr}(\lambda\sim(a,c) \to (k'\ a,c)) \\
\mathsf{cpAuxA}_1\ f'\ (\mathsf{fpAux}\,sf) &= \mathsf{sfArr}(\lambda\sim(_,c) \to (k'\ h,c))
\end{aligned}$$

iii. Case $sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}$:
This case is similar to the ones above it, details omitted. See other lemmas for similar proof techniques.

□

(b) Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} = \mathsf{arr}\ f'$:
This proof is identical to the one above, replacing the constant k by the function application $f'\ a$.

(c) Case $f = \mathtt{SF}\{\mathtt{sfTF} = tf_f\} = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a)\}$: This case is similar to the ones above it, details omitted. See other lemmas for similar proof techniques.

□

Theorem 2.16. *Let f be an arrow. Then*

$$\mathsf{first}\ f \ggg \mathsf{arr}\ \mathsf{fst} = \mathsf{arr}\ \mathsf{fst} \ggg f$$

Proof. We proceed by cases on the constructors for $\mathtt{SF}\{\}$.

1. Case $f = \mathsf{constant}\, b$: We have from the definition of first that

$$\mathsf{first}\ f = \mathtt{SF}\{\mathtt{sfTF} = \lambda \sim (a_0, c_0) \to (\mathsf{sfArr}(\lambda \sim (\_, c) \to (b, c)), (b, c_0))\}$$

   We also have from the definition of arr that

$$\mathsf{arr}\ \mathsf{fst} = \mathtt{SF}\{\mathtt{sfTF} = \lambda a \to (\mathsf{sfArr}(\lambda \sim (a_0, c_0) \to c_0), c_0)\}$$

   Then

$$\begin{aligned}\mathsf{first}\ f \ggg \mathsf{arr}\ \mathsf{fst} = \mathtt{SF}\{\mathtt{sfTF} = tf_0\}\ &\text{where}\\ tf_0 = \lambda a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0)\ &\text{where}\\ (sf_1, b_0) &= tf_1\ a_0\\ &= (\_, (b, \_))\\ (sf_2, c_0) &= tf_2\ b_0\\ &= (\mathsf{sfArr}\ \mathsf{fst}, b)\end{aligned}$$

   Therefore

$$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_1(\lambda \sim (\_, c) \to (b, c))(\mathsf{sfArr}\ \mathsf{fst})\\ &= \mathsf{sfArr}(\mathsf{fst}(\lambda \sim (\_, c) \to (b, c)))\\ &= \mathsf{sfArr}(\lambda \sim (\_, c) \to b)\end{aligned}$$

   And so

$$\mathsf{first}\ f \ggg \mathsf{arr}\ \mathsf{fst} = \mathsf{arr}(\lambda \sim (b, c) \to b)$$

   But

$$\mathsf{arr}\ \mathsf{fst} \ggg \mathsf{first}\ f = g \ggg f = f = \mathsf{constant}\, b$$

   and so clearly these two values are not equal. However, they *are* equivalent, under evalSF:

   Lemma 2.17.

$$\mathsf{evalSF}(\mathsf{arr}(\lambda \sim a \to b))\ as\ dts = \mathsf{evalSF}(\mathsf{constant}\, b)\ as\ dts$$

   Proof. By induction on the length of as:

   (a) Case $as = []$: Trivially, both sides equal $[]$.

   (b) Case $as = a : as'$: Let $f \stackrel{\text{def}}{=} \lambda \sim a \to b$. Then let $\mathtt{SF}\{\mathtt{sfTF} = tf_{arr}\} \stackrel{\text{def}}{=} \mathsf{arr}\, f$, and let $\mathtt{SF}\{\mathtt{sfTF} = tf_{const}\} \stackrel{\text{def}}{=} \mathsf{constant}\, b$. Then

$$\begin{aligned}\mathsf{evalSF}(\mathtt{SF}\{\mathtt{sfTF} = tf_{arr}\})\ a : as\ dts &= \mathsf{evalSFTF}(tf_{arr} a)\ dts\ as\\ &= \mathsf{evalSFTF}(\mathsf{sfArr}\, f, b)\ dts\ as\\ &= b : \mathsf{evalSF}'(\mathsf{sfArr}\, f)\ dts\ as\end{aligned}$$

Let $\texttt{SFArr}\{\texttt{sfTF}' = tf'\} \stackrel{\text{def}}{=} \mathsf{sfArr}\, f$. Then

$$\begin{aligned} &= b : \mathsf{evalSFTF}(tf'\ \ \mathsf{hd}(dts)\,\mathsf{hd}(as))\ \ \mathsf{tl}(dts)\ \ \mathsf{tl}(as) \\ &= b : \mathsf{evalSFTF}(\mathsf{sfArr}\, f, b)\ \ \mathsf{tl}(dts)\ \ \mathsf{tl}(as) \end{aligned}$$

and

$$\begin{aligned} \mathsf{evalSF}(\texttt{SF}\{\texttt{sfTF} = tf_{const}\})\ a : as\ dts &= \mathsf{evalSFTF}(tf_{const} a)\ dts\ as \\ &= \mathsf{evalSFTF}(\mathsf{sfConst}\, b, b)\ dts\ as \\ &= b : \mathsf{evalSF}'(\mathsf{sfConst}\, b)\ dts\ as \end{aligned}$$

Let $\texttt{SFArr}\{\texttt{sfTF}' = tf'\} \stackrel{\text{def}}{=} \mathsf{sfConst}\, b$. Then

$$\begin{aligned} &= b : \mathsf{evalSFTF}(tf'\ \ \mathsf{hd}(dts)\,\mathsf{hd}(as))\ \ \mathsf{tl}(dts)\ \ \mathsf{tl}(as) \\ &= b : \mathsf{evalSFTF}(\mathsf{sfConst}\, b, b)\ \ \mathsf{tl}(dts)\ \ \mathsf{tl}(as) \end{aligned}$$

By the inductive assumption, these two last values are equal, hence the original values are equal.

□

2. Case $f = \mathsf{arr}\, f' = \texttt{SF}\{\texttt{sfTF} = \lambda a \to (\mathsf{sfArr}\, f', f' a)\}$: We know from the second case of first that

$$\mathsf{first}\, f = \mathsf{arr}(\lambda \sim (a_0, c_0) \to (f'\ a_0, c_0))$$

and so

$$\begin{aligned} \mathsf{first}\, f \ggg \mathsf{arr}\,\mathsf{fst} &= \mathsf{arr}(\mathsf{fst} \cdot (\lambda \sim (a_0, c_0) \to (f'\ a_0, c_0))) \\ &= \mathsf{arr}(\lambda \sim (a_0, c_0) \to f'\ a_0) \end{aligned}$$

Likewise

$$\begin{aligned} \mathsf{arr}\,\mathsf{fst} \ggg f &= \mathsf{arr}(f' \cdot \mathsf{fst}) \\ &= \mathsf{arr}(\lambda \sim (a_0, c_0) \to f'\ a_0) \end{aligned}$$

3. Case $f = \texttt{SF}\{\texttt{sfTF} = \lambda a \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf\}, f'\ a)\}$ for some initial f':

   Let $\texttt{SF}\{\texttt{sfTF} = tf_f\} \stackrel{\text{def}}{=} f$, and $\texttt{SF}\{\texttt{sfTF} = tf_{fst}\} \stackrel{\text{def}}{=} \mathsf{arr}\,\mathsf{fst}$.

$$\begin{aligned} \mathsf{first}\, f = \texttt{SF}\{\texttt{sfTF} = tf_{f_0}\} \text{ where} \\ tf_{f_0} \sim (a_0, c_0) &= (\mathsf{fpAux}\, sf_1, (b_0, c_0)) \text{ where} \\ (sf_1, b_0) &= tf_f a_0 \\ &= (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_f\}, f'\ a_0) \\ \mathsf{fpAux}\, sf_1 = \texttt{SFTIVar}\{\texttt{sfTF}' = tf'_1\} &\text{ where} \\ tf'_1\ dt\ \sim (a, c) &= (\mathsf{fpAux}\, sf'_1, (b, c)) \text{ where} \\ (sf'_1, b) &= tf'_f\ dt\ a \end{aligned}$$

Then from one side we have

$$\begin{aligned}
\mathsf{first}\ f \ggg \mathsf{arr}\ \mathsf{fst} = \mathtt{SF}\{\mathtt{sfTF} = tf_{ffst}\}\ \text{where} \\
tf_{ffst} a_0 @ (a, c) &= (\mathsf{cpAux}\ sf_1\ sf_2, c_0)\ \text{where} \\
(sf_1, b_0) &= tf_{f_0}\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_1'\}, (f'\ a, c)) \\
(sf_2, c_0) &= tf_{fst}\ b_0 \\
&= (\mathsf{sfArr}\ \mathsf{fst}, f'\ a_0)
\end{aligned}$$

where

$$\begin{aligned}
\mathsf{cpAux}\ sf_1\ sf_2 &= \mathsf{cpAuxA}_2\ sf_1\ \ \mathsf{fst} \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{ffst}\}\ \text{where} \\
tf'_{ffst}\ td\ (a, c) &= (\mathsf{cpAuxA}_2\ sf_1'\ \mathsf{fst}, \mathsf{fst}\ b)\ \text{where} \\
(sf_1', b) &= tf_1'\ dt\ (a, c) \\
&= (\mathsf{fpAux}\ sf_1'', (b', c))\ \text{where} \\
(sf_1'', b') &= tf_f'\ dt\ a
\end{aligned}$$

While from the other we have

$$\begin{aligned}
\mathsf{arr}\ \mathsf{fst} \ggg f = \mathtt{SF}\{\mathtt{sfTF} = tf_{fstf}\}\ \text{where} \\
tf_{fstf} a_0 @ (a, c) &= (\mathsf{cpAux}\ sf_1\ sf_2, c_0)\ \text{where} \\
(sf_1, b_0) &= tf_{fst}\ (a, c) \\
&= (\mathsf{sfArr}\ \mathsf{fst}, a) \\
(sf_2, c_0) &= tf_f b_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_f'\}, f'\ a)
\end{aligned}$$

where

$$\begin{aligned}
\mathsf{cpAux}\ sf_1\ sf_2 &= \mathsf{cpAuxA}_1\ \mathsf{fst}\ \ sf_2 \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fstf}\}\ \text{where} \\
tf'_{fstf}\ dt\ (a, c) &= (\mathsf{cpAuxA}_1\ \mathsf{fst}\ \ sf_2', c)\ \text{where} \\
(sf_2', c) &= tf_f'\ dt\ \ \mathsf{fst}(a, c) \\
&= tf_f'\ dt\ a
\end{aligned}$$

Clearly, $\mathsf{snd}(tf_{ffst}\ a_0) = f'\ a_0 = \mathsf{snd}(tf_{fstf}\ a_0)$. Now, we need

$$\mathsf{fst}(tf_{ffst}\ a_0) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{ffst}\}$$

to equal

$$\mathsf{fst}(tf_{fstf}\ a_0) = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_{fstf}\}$$

for the two expressions to be equal. We note that

$$\mathsf{snd}(tf'_{ffst}\ dt\ (a, c)) = \mathsf{snd}(tf_f'\ dt\ a) = \mathsf{snd}(tf'_{fstf}\ dt\ (a, c)$$

and therefore we only need

$$\mathsf{fst}(tf'_{ffst}\ dt\ (a,c)) = \mathsf{cpAuxA}_2(\mathsf{fpAux}\,\mathsf{fst}(tf'_f\ dt\ a))\,\mathsf{fst}$$

to equal

$$\mathsf{fst}(tf'_{fstf}\ dt\ (a,c)) = \mathsf{cpAuxA}_1\,\mathsf{fst}\ (\mathsf{fst}(tf'_f\ dt\ a))$$

to finish this case. We prove this in a lemma:

Lemma 2.18. *For any sf,*

$$\mathsf{cpAuxA}_1\,\mathsf{fst}\,sf = \mathsf{cpAuxA}_2(\mathsf{fpAux}\,sf)\,\mathsf{fst}$$

Proof. We proceed by cases of sf:

(a) Case $sf = \mathsf{sfConst}\,b$:
We have

$$\mathsf{cpAuxA}_1\,\mathsf{fst}\,\mathsf{sfConst}\,b = \mathsf{sfConst}\,b$$

by the first case of cpAuxA_1. Now,

$$\mathsf{fpAux}\,sf = \mathsf{sfArr}(\lambda \sim(\_, c) \to (b, c))$$

and

$$\begin{aligned}\mathsf{cpAuxA}_2(\mathsf{fpAux}\,sf)\,\mathsf{fst} &= \mathsf{sfArr}(\mathsf{fst}\cdot(\lambda \sim(\_, c) \to (b, c))) \\ &= \mathsf{sfArr}(\lambda\_ \to b)\end{aligned}$$

As in the first case of the proof of this law, these two expressions are equivalent under evalSF.

(b) Case $sf = \mathsf{sfArr}\,h$:
We have

$$\begin{aligned}\mathsf{cpAuxA}_1\,\mathsf{fst}\,\mathsf{sfArr}\,h &= \mathsf{sfArr}(f\cdot\mathsf{fst}) \\ &= \mathsf{sfArr}(\lambda a \to h\ a)\end{aligned}$$

by the second case of cpAuxA_1. Now,

$$\mathsf{fpAux}\,sf = \mathsf{sfArr}(\lambda \sim(a, c) \to (h\ a, c))$$

and

$$\begin{aligned}\mathsf{cpAuxA}_2(\mathsf{fpAux}\,sf)\,\mathsf{fst} &= \mathsf{sfArr}(\mathsf{fst}\cdot(\lambda \sim(a, c) \to (h\ a, c))) \\ &= \mathsf{sfArr}(\lambda a \to h\ a)\end{aligned}$$

(c) Case $sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}$:
We have

$$\begin{aligned}\mathsf{cpAuxA}_1\,\mathsf{fst}\,sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_1\}&\text{ where} \\ tf_1\ dt\ a_0@(a,c) &= (\mathsf{cpAuxA}_1\,\mathsf{fst}\,sf'_1, c)\text{ where} \\ (sf'_1, c) &= sf'\ dt\ (\mathsf{fst}\,a_0) \\ &= sf'\ dt\ a\end{aligned}$$

Now,

$$\begin{aligned}\mathsf{fpAux}\, sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_2'\} \text{ where}\\ tf_2'\ dt\ \sim(a,c) = (\mathsf{fpAux}\, sf_2', (b,c)) \text{ where}\\ (sf_2', b) = sf'\ dt\ a\end{aligned}$$

and

$$\begin{aligned}\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf)\, \mathsf{fst} = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_2\} \text{ where}\\ tf_2\ dt\ a_0@(a,c) = (\mathsf{cpAuxA_2}\, sf_2''\, \mathsf{fst}, \mathsf{fst}\, b') \text{ where}\\ (sf_2'', b') = tf_2'\ dt\ a_0\\ = (\mathsf{fpAux}\, sf_2', (b,c)) \text{ where}\\ (sf_2', b) = sf'\ dt\ a\end{aligned}$$

Substituting back yields

$$\begin{aligned}tf_2\ dt\ a_0@(a,c) = (\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_2')\, \mathsf{fst}, b) \text{ where}\\ (sf_2', b) = sf'\ dt\ a\end{aligned}$$

Now, $\mathsf{snd}(tf_1\ dt\ a_0) = \mathsf{snd}(sf'\ dt\ a) = \mathsf{snd}(tf_2\ dt\ a_0)$, by simple inspection. Also, it is clear that $sf_1' = \mathsf{fst}(sf'\ dt\ a) = sf_2'$. Then by our induction hypothesis,

$$\begin{aligned}\mathsf{fst}(tf_1\ dt\ a_0) &= (\mathsf{cpAuxA_1}\, \mathsf{fst}\, sf_1')\\ &= (\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_1')\, \mathsf{fst})\\ &= (\mathsf{cpAuxA_2}(\mathsf{fpAux}\, sf_2')\, \mathsf{fst})\\ &= \mathsf{fst}(tf_2\ dt\ a_0)\end{aligned}$$

Hence $tf_1 = tf_2$, proving the lemma

□

Given this lemma, we can see that $tf'_{ffst} = tf'_{fstf}$, and that therefore $tf_{ffst} = tf_{fstf}$, which finally yields the desired result,

$$\mathsf{first}\, f \ggg \mathsf{arr}\, \mathsf{fst} = \mathsf{arr}\, \mathsf{fst} \ggg f$$

□

2.2 The associativity law

This law receives its own subsection for the single reason that it is incredibly long. Nearly twice as long, in sheer number of cases, as the next longest proof, placing it among the others would result in visual distraction. Also, the proofs involved is extremely similar to many proofs seen already; as such, much will be left unstated. Nearly every case involving an `SFTIVar{}` construction requires a lemma, in a similar vein to each of the preceeding proofs. The first lemma is explicitly proven; the remaining ones are set up and stated, but not proven. Their proofs proceed in analogous and straightforward manners.

Fortunately, no case involves dealing with the evalSF embedding; this is because no case involves the transformation (via first) of an sfConst into an sfArr or an `SFTIVar{}` and back – rather, the transformations proceed in only one direction.

Theorem 2.19. *Let f, g and h be arrows. Then*

$$(f \ggg g) \ggg h = f \ggg (g \ggg h)$$

Proof. Throughout these cases, let

$$\begin{aligned}
\texttt{SF}\{\texttt{sfTF} = tf_f\} &\stackrel{\text{def}}{=} f \\
\texttt{SF}\{\texttt{sfTF} = tf_g\} &\stackrel{\text{def}}{=} g \\
\texttt{SF}\{\texttt{sfTF} = tf_h\} &\stackrel{\text{def}}{=} h
\end{aligned}$$

1. Case $h = \mathsf{constant}\, k$: We show a simpler result, that $f \ggg \mathsf{constant}\, k = \mathsf{constant}\, k$ for any f, and from that, this case follows immediately.

$$\begin{aligned}
f \ggg \mathsf{constant}\, k = \texttt{SF}\{\texttt{sfTF} = tf_0\} \text{ where} \\
tf_0\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_f\ a_0 \\
(sf_2, c_0) &= tf_h\ b_0 \\
&= (\mathsf{sfConst}\, k, k) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{sfConst}\, k
\end{aligned}$$

   Hence

$$\begin{aligned}
f \ggg \mathsf{constant}\, k &= \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\mathsf{sfConst}\, k, k)\} \\
&= \mathsf{constant}\, k
\end{aligned}$$

   From this it immediately follows that

$$f \ggg (g \ggg h) = f \ggg h = h$$

   and

$$(f \ggg g) \ggg h = e \ggg h = h$$

   for some $e = f \ggg g$.

2. Case $h = \mathsf{arr}\, h'$:

(a) Case $g = \mathsf{constant}\, k$:

$$\begin{aligned} g \ggg h &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\quad\quad (sf_1, b_0) = tf_g\ a_0 \\ &\quad\quad\quad = (\mathsf{sfConst}\, k, k) \\ &\quad\quad (sf_2, c_0) = tf_h\ b_0 \\ &\quad\quad\quad = \mathsf{sfArr}\, h', h'\ k) \\ &\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxC_1}\, k\ sf_2 \\ &\quad\quad = \mathsf{sfConst}(h'\ k) \end{aligned}$$

and therefore

$$\begin{aligned} g \ggg h &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathsf{sfConst}(h'\ k), h'\ k)\} \\ &= \mathsf{constant}(h'\ k) \end{aligned}$$

Since $g \ggg h$ is a constant construction, clearly, by the case above, $f \ggg (g \ggg h) = g \ggg h$. But we also have that $f \ggg g = g$, and therefore $(f \ggg g) \ggg h = g \ggg h$.

(b) Case $g = \mathsf{arr}\, g'$:

i. Case $f = \mathsf{constant}\, k$: By the above case, $f \ggg g = \mathsf{constant}(g'\ k)$, and therefore

$$(f \ggg g) \ggg h = \mathsf{constant}(h'(g'\ k))$$

Now,

$$\begin{aligned} g \ggg h &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\ &\quad tf_0\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ &\quad\quad (sf_1, b_0) = tf_g\ a_0 \\ &\quad\quad\quad = (\mathsf{sfArr}\, g', g'\ a_0) \\ &\quad\quad (sf_2, c_0) = tf_h\ b_0 \\ &\quad\quad\quad = (\mathsf{sfArr}\, h', h'(g'\ a_0)) \\ &\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_1}\, g'\ sf_2 \\ &\quad\quad = \mathsf{sfArr}(h' \cdot g') \end{aligned}$$

and therefore

$$\begin{aligned} g \ggg h &= \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathsf{sfArr}(h' \cdot g'), (h'(g'\ a_0)))\} \\ &= \mathsf{arr}(h' \cdot g') \end{aligned}$$

and therefore

$$f \ggg (g \ggg h) = \mathsf{constant}\, k \ggg (\mathsf{arr}(h' \cdot g')) = \mathsf{constant}(h'(g'\ k))$$

ii. Case $f = \mathsf{arr}\, f'$: By the above case,

$$\begin{aligned} f \ggg (g \ggg h) &= \mathsf{arr}\, f' \ggg (\mathsf{arr}\, g' \ggg \mathsf{arr}\, h') \\ &= \mathsf{arr}\, f' \ggg (\mathsf{arr}(h' \cdot g')) \\ &= \mathsf{arr}((h' \cdot g') \cdot f') \end{aligned}$$

and

$$\begin{aligned}(f \ggg g) \ggg h &= (\mathsf{arr}\ f' \ggg \mathsf{arr}\ g') \ggg \mathsf{arr}\ h' \\ &= \mathsf{arr}(g' \cdot f') \ggg \mathsf{arr}\ h' \\ &= \mathsf{arr}(h' \cdot (g' \cdot f'))\end{aligned}$$

and since function composition is indeed associative, these two expressions are equal.

iii. Case $f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0)\}$:

$$\begin{aligned}f \ggg g = \mathtt{SF}\{\mathtt{sfTF} &= tf_0\} \text{ where} \\ tf_0\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ (sf_1, b_0) &= tf_f\ a_0 \\ &= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0) \\ (sf_2, c_0) &= tf_g\ b_0 \\ &= (\mathsf{sfArr}\, g', g'(f'\ a_0)) \\ \mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_2\, sf_1\ g' \\ &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_0\} \text{ where} \\ tf'_0\ dt\ a &= (\mathsf{cpAuxA}_2\, sf'_1\ g', g'\ b) \text{ where} \\ (sf'_1, b) &= tf'_f\ dt\ a\end{aligned}$$

Now,

$$\begin{aligned}(f \ggg g) \ggg h &= \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \ggg h \\ &= \mathtt{SF}\{\mathtt{sfTF} = tf_1\} \text{ where} \\ tf_1\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\ (sf_1, b_0) &= tf_0\ a_0 \\ &= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_0\}, g'(f'\ a_0)) \\ (sf_2, c_0) &= tf_h\ b_0 \\ &= (\mathsf{sfArr}\, h', h'(g'(f'\ a_0))) \\ \mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_2\, sf_1\ h' \\ &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_1\} \text{ where} \\ tf'_1\ dt\ a &= (\mathsf{cpAuxA}_2\, sf'_1\ h', h'\ b') \text{ where} \\ (sf'_1, b') &= tf'_0\ dt\ a \\ &= (\mathsf{cpAuxA}_2\, sf''_1\ g', g'\ b) \text{ where} \\ (sf''_1, b) &= tf'_f\ dt\ a\end{aligned}$$

so

$$\begin{aligned}\mathsf{cpAux}\, sf_1\ sf_2 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_1\} \text{ where} \\ tf'_1\ dt\ a &= (\mathsf{cpAuxA}_2(\mathsf{cpAuxA}_2\, sf''_1\ g')\ h', h'(g'\ b)) \text{ where} \\ (sf''_1, b) &= tf'_f\ dt\ a\end{aligned}$$

On the other side of the equation, we have

$$\begin{aligned}
f \ggg (g \ggg h) &= f \ggg \mathsf{arr}(h' \cdot g') \\
&= \mathtt{SF}\{\mathtt{sfTF} = tf_2\} \text{ where} \\
&\quad tf_2\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\quad\quad (sf_1, b_0) = tf_f\ a_0 \\
&\quad\quad\quad = (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_f'\}, f'\ a_0) \\
&\quad\quad (sf_2, c_0) = tf_g\ b_0 \\
&\quad\quad\quad = (\mathsf{sfArr}(h' \cdot g'), h'(g'(f'\ a_0))) \\
&\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA}_2\, sf_1\ g' \\
&\quad\quad = \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_0'\} \text{ where} \\
&\quad\quad\quad tf_0'\ dt\ a = (\mathsf{cpAuxA}_2\, sf_1'\ (h' \cdot g'), h'(g'\ b)) \text{ where} \\
&\quad\quad\quad\quad (sf_1', b) = tf_f'\ dt\ a
\end{aligned}$$

Clearly, $\mathsf{snd}(tf_1\ a_0) = h'(g'(f'\ a_0)) = \mathsf{snd}(tf_2\ a_0)$. We need $tf_1' = tf_2'$ for $\mathsf{fst}(tf_1\ a_0) = \mathsf{fst}(tf_2\ a_0)$. We have that $\mathsf{snd}(tf_1'\ dt\ a) = h'(g'(\mathsf{snd}(tf_f'\ dt\ a))) = \mathsf{snd}(tf_2'\ dt\ a)$. We now need a lemma, for the remaining part:

Lemma 2.20.

$$\mathsf{cpAuxA}_2(\mathsf{cpAuxA}_2\, sf\ g')\ h' = \mathsf{cpAuxA}_2\, sf\ (h'\ \cdot g')$$

Proof. We proceed by cases on sf, and the proof is markedly similar in nature to this overarching case we are currently in:

A. Case $sf = \mathsf{sfConst}\, k$: We have $\mathsf{cpAuxA}_2(\mathsf{sfConst}\, k)g' = \mathsf{sfConst}(g'\ k)$, and so

$$\mathsf{cpAuxA}_2(\mathsf{cpAuxA}_2(\mathsf{sfConst}\, k)g')h' = \mathsf{sfConst}(h'(g'\ k))$$

while on the other side we have

$$\mathsf{cpAuxA}_2(\mathsf{sfConst}\, k)(h' \cdot g') = \mathsf{sfConst}(h'(g'\ k))$$

B. Case $sf = \mathsf{sfArr}\, k'$: We have $\mathsf{cpAuxA}_2(\mathsf{sfArr}\, k')g' = \mathsf{sfConst}(g' \cdot k')$, and so

$$\mathsf{cpAuxA}_2(\mathsf{cpAuxA}_2(\mathsf{sfArr}\, k')g')h' = \mathsf{sfConst}(h' \cdot (g' \cdot k'))$$

while on the other side we have

$$\mathsf{cpAuxA}_2(\mathsf{sfArr}\, k')(h' \cdot g') = \mathsf{sfConst}((h' \cdot g') \cdot k')$$

C. Case $sf = \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}$:

$$\begin{aligned}
\mathsf{cpAuxA}_2\, \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}\ g' &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf_2'\} \text{ where} \\
&\quad sf_2'\ dt\ a = (\mathsf{cpAuxA}_2\, sf_2''\ g', g'\ b) \text{ where} \\
&\quad\quad (sf_2'', b) = sf'\ dt\ a
\end{aligned}$$

and therefore

$$\begin{aligned}\mathsf{cpAuxA_2}\,\mathtt{SFTIVar}\{\mathtt{sfTF}' = sf_2'\}\ h' = \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf_3'\}\ \text{where}\\ sf_3'\ dt\ a = (\mathsf{cpAuxA_2}\,sf_3''\ h', h'\ b')\ \text{where}\\ (sf_3'', b') = sf_2'\ dt\ a\\ = (\mathsf{cpAuxA_2}\,sf_2''\ g', g'\ b)\ \text{where}\\ (sf_2'', b) = sf'\ dt\ a\end{aligned}$$

and so

$$\begin{aligned}\mathsf{cpAuxA_2}(\mathsf{cpAuxA_2}\,sf\ g')\ h' =\\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_3'\}\ \text{where}\\ sf_3'\ dt\ a = (\mathsf{cpAuxA_2}(\mathsf{cpAuxA_2}\,sf_2''\ g')\ h', h'(g'\ b))\ \text{where}\\ (sf_2'', b) = sf'\ dt\ a\end{aligned}$$

while on the other side we have

$$\begin{aligned}\mathsf{cpAuxA_2}\,\mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}\ (h' \cdot g') =\\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf_4'\}\ \text{where}\\ sf_4'\ dt\ a = (\mathsf{cpAuxA_2}\,sf_4''\ (h' \cdot g'), (h' \cdot g')\ b)\ \text{where}\\ (sf_4'', b) = sf'\ dt\ a\end{aligned}$$

or

$$\begin{aligned}\mathsf{cpAuxA_2}\,\mathtt{SFTIVar}\{\mathtt{sfTF}' = sf'\}\ (h' \cdot g') =\\ \mathtt{SFTIVar}\{\mathtt{sfTF}' = sf_4'\}\ \text{where}\\ sf_4'\ dt\ a = (\mathsf{cpAuxA_2}\,sf_4''\ (h' \cdot g'), h'(g'\ b))\ \text{where}\\ (sf_4'', b) = sf'\ dt\ a\end{aligned}$$

Now, it is obvious that $sf_4'' = sf_2''$, and that the b's on either side are equal, since they all come from $sf'\ dt\ a$. Therefore we have that

$$\mathsf{snd}(sf_4'\ dt\ a) = h'(g'\ b) = \mathsf{snd}(sf_2'\ dt\ a)$$

while

$$\begin{aligned}\mathsf{fst}(sf_2'\ dt\ a) &= \mathsf{cpAuxA_2}(\mathsf{cpAuxA_2}\,sf_2''\ g')\ h'\\ \mathsf{fst}(sf_4'\ dt\ a) &= \mathsf{cpAuxA_2}\,sf_4''(h' \cdot g')\end{aligned}$$

We can assume these expressions are equal by induction over the nesting depth of the $\mathtt{SFTIVar}\{\}$ constructions, and hence we have shown that this depth too satisfies the lemma.

□

By the lemma, we've shown that $tf_1 = tf_2$, and hence have established this case.

(c) Case $g = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_g\}, g'\ a_0)\}$: Each of these cases will require a lemma; the first two will have three cases each, while the third requires seven cases to prove. This is quite simply seen by noting how many arrows are of the $\texttt{SFTIVar}\{\}$ type, and noting that the three cases involving $\mathsf{sfConst}$ as the second argument condense to a single case. This lemma, in fact, will very strongly resemble this entire case 1. As such, the proof will be omitted.

i. Case $f = \mathsf{constant}\, k$:

$$
\begin{aligned}
f \ggg g &= \texttt{SF}\{\texttt{sfTF} = tf_0\} \text{ where} \\
&\quad tf_0\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\qquad (sf_1, b_0) = tf_f\ a_0 \\
&\qquad\quad = (\mathsf{sfConst}\, k, k) \\
&\qquad (sf_2, c_0) = tf_g\ b_0 \\
&\qquad\quad = (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_g\}, g'\ k) \\
&\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxC_1}\, k\ sf_2 \\
&\qquad\quad = \texttt{SFTIVar}\{\texttt{sfTF}' = tf'_0\} \text{ where} \\
&\qquad\quad\ tf'_0\ dt\ a = (\mathsf{cpAuxC_1}\, k\ sf'_2, c) \text{ where} \\
&\qquad\qquad (sf'_2, c) = tf'_g\ dt\ k
\end{aligned}
$$

and so

$$
\begin{aligned}
(f \ggg g) \ggg h &= \texttt{SF}\{\texttt{sfTF} = tf_1\} \text{ where} \\
&\quad tf_1\ a_0 = (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
&\qquad (sf_1, b_0) = tf_0\ a_0 \\
&\qquad\quad = (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_0\}, g'\ k) \\
&\qquad (sf_2, c_0) = tf_h\ b_0 \\
&\qquad\quad = (\mathsf{sfArr}\, h', h'(g'\ k)) \\
&\quad \mathsf{cpAux}\, sf_1\ sf_2 = \mathsf{cpAuxA_2}\, sf_1\ h' \\
&\qquad\quad = \texttt{SFTIVar}\{\texttt{sfTF}' = tf'_1\} \text{ where} \\
&\qquad\quad\ tf'_1\ dt\ a = (\mathsf{cpAuxA_2}\, sf'_1\ h', h'\ b) \text{ where} \\
&\qquad\qquad (sf'_1, b) = tf'_0\ dt\ a \\
&\qquad\qquad\quad = (\mathsf{cpAuxC_1}\, k\ sf'_2, c) \text{ where} \\
&\qquad\qquad\quad (sf'_2, c) = tf'_g\ dt\ k
\end{aligned}
$$

or

$$
\begin{aligned}
&\mathsf{cpAux}\, sf_1\ sf_2 = \texttt{SFTIVar}\{\texttt{sfTF}' = tf'_1\} \text{ where} \\
&\qquad tf'_1\ dt\ a = (\mathsf{cpAuxA_2}(\mathsf{cpAuxC_1}\, k\ sf'_2)\ h', h'\ c) \text{ where} \\
&\qquad\quad (sf'_2, c) = tf'_g\ dt\ k
\end{aligned}
$$

while on the other side we have

$$
\begin{aligned}
g \ggg h = \mathtt{SF}\{\mathtt{sfTF} = tf_2\} \text{ where} \\
tf_2\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_g\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_g'\}, g'\ a_0) \\
(sf_2, c_0) &= tf_h\ b_0 \\
&= (\mathsf{sfArr}\, h', h'(g'\ a_0)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_2}\, sf_1\ h' \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_2'\} \text{ where} \\
tf_2'\ dt\ a &= (\mathsf{cpAuxA_2}\, sf_1'\ h', h'\ b) \text{ where} \\
(sf_1', b) &= tf_g'\ dt\ a
\end{aligned}
$$

and so

$$
\begin{aligned}
f \ggg (g \ggg h) = \mathtt{SF}\{\mathtt{sfTF} = tf_3\} \text{ where} \\
tf_3\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_f\ a_0 \\
&= (\mathsf{sfConst}\, k, k) \\
(sf_2, c_0) &= tf_g\ b_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_2'\}, h'(g'\ k)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxC_1}\, k\ sf_2 \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf_3'\} \text{ where} \\
tf_3'\ dt\ a &= (\mathsf{cpAuxC_1}\, k\ sf_2', c) \text{ where} \\
(sf_2', c) &= tf_2'\ dt\ k \\
&= (\mathsf{cpAuxA_2}\, sf_1'\ h', h'\ b) \text{ where} \\
(sf_1', b) &= tf_g'\ dt\ k
\end{aligned}
$$

Comparing components of the functions above, we see that the second parts of each pair are equal, and the first parts are equal under a lemma, which is stated without proof:

Lemma 2.21.

$$\mathsf{cpAuxA_2}(\mathsf{cpAuxC_1}\, k\ sf_2')\ h' = \mathsf{cpAuxC_1}\, k\ (\mathsf{cpAuxA_2}\, sf_2'\ h')$$

This lemma restates what we are trying to prove, one level deeper, at the SF' level instead of the topmost SF level. As such, its proof is precisely analogous to the proofs of lemmas and this case as developed so far, using structural induction at the last step (instead of repeating the lemma) to show that the result holds.

ii. Case $f = \mathsf{arr}\, f'$: This case is precisely analogous to the preceeding case, with the changes being converting each explicit mention of k into $f'\ a_0$, and changing each $\mathsf{cpAuxC_1}$ into $\mathsf{cpAuxA_1}$. (This "naive" description works because of how very parallel

the code in each case has been written.)

$$
\begin{aligned}
f \ggg g = \mathtt{SF}\{\mathtt{sfTF} = tf_0\} \text{ where} \\
tf_0\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_f\ a_0 \\
&= (\mathsf{sfArr}\, f', f'\ a_0) \\
(sf_2, c_0) &= tf_g\ b_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_g\}, g'(f'\ a_0)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_1}\ f'\ sf_2 \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_0\} \text{ where} \\
tf'_0\ dt\ a &= (\mathsf{cpAuxA_1}\ f'\ sf'_2, c) \text{ where} \\
(sf'_2, c) &= tf'_g\ dt\ (f'\ a)
\end{aligned}
$$

and so

$$
\begin{aligned}
(f \ggg g) \ggg h = \mathtt{SF}\{\mathtt{sfTF} = tf_1\} \text{ where} \\
tf_1\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_0\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_0\}, g'(f'\ a_0)) \\
(sf_2, c_0) &= tf_h\ b_0 \\
&= (\mathsf{sfArr}\, h', h'(g'(f'\ a_0))) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA_2}\, sf_1\ h' \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_1\} \text{ where} \\
tf'_1\ dt\ a &= (\mathsf{cpAuxA_2}\, sf'_1\ h', h'\ b) \text{ where} \\
(sf'_1, b) &= tf'_0\ dt\ a \\
&= (\mathsf{cpAuxA_1}\ f'\ sf'_2, c) \text{ where} \\
(sf'_2, c) &= tf'_g\ dt\ (f'\ a)
\end{aligned}
$$

or

$$
\begin{aligned}
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_1\} \text{ where} \\
tf'_1\ dt\ a &= (\mathsf{cpAuxA_2}(\mathsf{cpAuxA_1}\ f'\ sf'_2)\ h', h'\ c) \text{ where} \\
(sf'_2, c) &= tf'_g\ dt\ (f'\ a)
\end{aligned}
$$

while on the other side we have

$$
\begin{aligned}
g \ggg h = \mathtt{SF}\{\mathtt{sfTF} = tf_2\} \text{ where} \\
tf_2\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_g\ a_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_g\}, g'\ a_0) \\
(sf_2, c_0) &= tf_h\ b_0 \\
&= (\mathsf{sfArr}\, h', h'(g'\ a_0)) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_2\, sf_1\ h' \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_2\} \text{ where} \\
tf'_2\ dt\ a &= (\mathsf{cpAuxA}_2\, sf'_1\ h', h'\ b) \text{ where} \\
(sf'_1, b) &= tf'_g\ dt\ a
\end{aligned}
$$

and so

$$
\begin{aligned}
f \ggg (g \ggg h) = \mathtt{SF}\{\mathtt{sfTF} = tf_3\} \text{ where} \\
tf_3\ a_0 &= (\mathsf{cpAux}\, sf_1\ sf_2, c_0) \text{ where} \\
(sf_1, b_0) &= tf_f\ a_0 \\
&= (\mathsf{sfArr}\, f', f'\ a_0) \\
(sf_2, c_0) &= tf_g\ b_0 \\
&= (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_2\}, h'(g'(f'\ a_0))) \\
\mathsf{cpAux}\, sf_1\ sf_2 &= \mathsf{cpAuxA}_1\, f'\ sf_2 \\
&= \mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_3\} \text{ where} \\
tf'_3\ dt\ a &= (\mathsf{cpAuxA}_1\, f'\ sf'_2, c) \text{ where} \\
(sf'_2, c) &= tf'_2\ dt\ a \\
&= (\mathsf{cpAuxA}_2\, sf'_1\ h', h'\ b) \text{ where} \\
(sf'_1, b) &= tf'_g\ dt\ a
\end{aligned}
$$

Comparing components of the functions above, we see that the second parts of each pair are equal, and the first parts are equal under a lemma, which is stated without proof:

Lemma 2.22.

$$\mathsf{cpAuxA}_2(\mathsf{cpAuxA}_1\, f'\ sf'_2)\ h' = \mathsf{cpAuxA}_1\, f'\ (\mathsf{cpAuxA}_2\, sf'_2\ h')$$

iii. Case $f = \mathtt{SF}\{\mathtt{sfTF} = \lambda a_0 \to (\mathtt{SFTIVar}\{\mathtt{sfTF}' = tf'_f\}, f'\ a_0)\}$: This case again boils down to a similar lemma to those previously seen, and as such its proof is equally similar:

Lemma 2.23.

$$\mathsf{cpAuxA}_2(\mathsf{cpAux}\, sf'_1\ sf'_2)\ h' = \mathsf{cpAux}\, sf'_1\ (\mathsf{cpAuxA}_2(sf'_2\ h'))$$

The only distinguishing feature of this lemma from those previous is that, since it has two general $\mathtt{SFTIVar}\{\}$ constructions, it requires using cpAux itself again; it also involves nine cases instead of three, though three can be reduced to a single case where $sf'_2 = \mathsf{sfConst}\, k$.

3. Case $h = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_h\}, h'\ a_0)\}$: Every one of these cases will require a lemma, since h is of the $\texttt{SFTIVar}\{\}$ type. Many of these lemmas will have three cases; the last lemma of cases a and b will have seven cases, while the last lemma will itself have seventeen cases – and will be practically identical in spirit to this entire proof. As such, these lemmas will only be stated, and not proven; the proofs are all straightforward. The setup work will also be left unstated; it is precisely as formulaic as the preceeding seven cases – all the "work" involved is wrapped within the lemmas, which express what occurs when the cpAux expressions are evaluated.

   (a) Case $g = \mathsf{constant}\, k$:

      i. Case $f = \mathsf{constant}\, l$:
      Lemma 2.24.
      $$\mathsf{cpAuxC_1}\ l\ (\mathsf{cpAuxC_1}\ k\ sf'_3) = \mathsf{cpAuxC_1}\ k\ sf'_3$$

      ii. Case $f = \mathsf{arr}\, f'$:
      Lemma 2.25.
      $$\mathsf{cpAuxA_1}\ f'\ (\mathsf{cpAuxC_1}\ k\ sf'_3) = \mathsf{cpAuxC_1}\ k\ sf'_3$$

      iii. Case $f = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_f\}, f'\ a_0)\}$:
      Lemma 2.26.
      $$\mathsf{cpAux}\, sf'_1\ (\mathsf{cpAuxC_1}\ k\ sf'_3) = \mathsf{cpAuxC_1}\ k\ sf'_3$$

   (b) Case $g = \mathsf{arr}\, g'$:

      i. Case $f = \mathsf{constant}\, k$:
      Lemma 2.27.
      $$\mathsf{cpAuxC_1}\ k\ (\mathsf{cpAuxA_1}\ g'\ sf'_3) = \mathsf{cpAuxC_1}(\mathsf{cpAuxC_1}\ k\ g')\ sf'_3$$

      ii. Case $f = \mathsf{arr}\, f'$:
      Lemma 2.28.
      $$\mathsf{cpAuxA_1}\ f'\ (\mathsf{cpAuxA_1}\ g'\ sf'_3) = \mathsf{cpAuxA_1}(g' \cdot f')\ sf'_3$$

      iii. Case $f = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_f\}, f'\ a_0)\}$:
      Lemma 2.29.
      $$\mathsf{cpAux}\, sf'_1\ (\mathsf{cpAuxA_1}\ g'\ sf'_3) = \mathsf{cpAux}(\mathsf{cpAuxA_2}\, sf'_1\ g')\ sf'_3$$

   (c) Case $g = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_g\}, g'\ a_0)\}$:

      i. Case $f = \mathsf{constant}\, k$:
      Lemma 2.30.
      $$\mathsf{cpAuxC_1}\ k\ (\mathsf{cpAux}\, sf'_2\ sf'_3) = \mathsf{cpAux}(\mathsf{cpAuxC_1}\ k\ sf'_2)\ sf'_3$$

      ii. Case $f = \mathsf{arr}\, f'$:
      Lemma 2.31.
      $$\mathsf{cpAuxA_1}\ f'\ (\mathsf{cpAux}\, sf'_2\ sf'_3) = \mathsf{cpAux}(\mathsf{cpAuxA_1}\ f'\ sf'_2)\ sf'_3$$

      iii. Case $f = \texttt{SF}\{\texttt{sfTF} = \lambda a_0 \to (\texttt{SFTIVar}\{\texttt{sfTF}' = tf'_f\}, f'\ a_0)\}$:
      Lemma 2.32.
      $$\mathsf{cpAux}\, sf'_1\ (\mathsf{cpAux}\, sf'_2\ sf'_3) = \mathsf{cpAux}(\mathsf{cpAux}\, sf'_1\ sf'_2)\ sf'_3$$

□

3 Discussion

As seven of the nine arrow laws are satisfied, little needs to be said about them immediately. Expressions can be written with the various primitive combinators, secure in the knowledge that they will work as expected. The arrow laws serve as a semantics for arrows, in a way – they formalize the intuitive interactions between the combinators, such that a programmer can rely on that behavior. However, two of the laws are not strictly satisfied.

The two laws which break under strict equality involve the interactions of first and $\ggg$, specifically in the optimized cases involving constant. The reason for this is simple, and was seen in greater detail in the proofs above. When one takes first constant k, the result can no longer be a constant, but must be a lifted pure function $\mathsf{arr}(\lambda(\_, c) \to (k, c))$. This in itself is not the problem; after all, the arr case is still an optimized version of the most general `SF{}` arrow. The problem is that arr interacts differently in terms of optimizations, than constant does – as was seen, anything composed with a constant returned the constant, without any computation necessary, whereas a lifted function needed to be applied to its argument, regardless of its actions.

To remedy the situation, as above, we implemented an evaluation function, which applies an arrow to a list of inputs (and delta time values), and returns a list of output values. The rationale for this function, as explained above, is that as long as two arrows behave the same for all inputs, it does not matter what their internal representations are. To give a simplistic example, suppose one defined the following two functions:

$$f(x) = x + 2$$
$$g(x) = h(h(x)) \text{ where } h(x) = x + 1$$

Clearly, if one has meta-knowledge of the internals of these two functions, then one can say

$$g(x) = h(h(x)) = h(x + 1) = x + 2 = f(x)$$

and by η-conversion state that $f = g$. But if the functions are opaque, that meta-knowledge is not accessible, and so the argument breaks. This is the standard case, since functions are λ-abstractions, and abstractions by their nature are opaque as to how their internals work.

Returning to our `SF{}` arrows, a similar argument holds. If we could know, for example, that the f in arr f was in fact the result of first constant k, then we could use that meta-knowledge to maintain the optimization of constant, and would not need to resort to the evaluation function. But we cannot know this information *a priori*, and so we are restricted.

This now raises an interesting opportunity. Since the laws are not satisfied under equality (though they are satisfied under equivalence), it follows that one expression may be more efficient than the other, in terms of computations necessary to arrive at the same result as the other side. This means that conceivably, one could write an optimizer for Yampa programs, which could take into account this advantage. Let us examine, then, what opportunities there are.

The first law is the distributive property of first over composition:

$$\mathsf{first}(f \ggg g) = \mathsf{first}\, f \ggg \mathsf{first}\, g$$

Without giving a formal exposition, it is intuitively obvious that the left side should be more efficient than the right side – in terms of combinators called, the left side has one fewer call to first. This in itself is a decent savings, since first is a recursive function which must map itself (via fpAux) down all future `SFTIVar{}` constructions. Halving the number of times this function is needed is a substantial gain, perhaps especially in larger programs.

More importantly than this optimization, is the observation that the left side maintains more optimizations than the right side does. In the case where $g = \mathsf{constant}\, k$, the left side will always be a lifted pure function, since $f \ggg g = g$. The right hand side cannot preserve this optimization for the exact reasons stated above – $\mathsf{first}\, g$ is now a lifted function, and once there, there is no opportunity to maintain the optimizations possible for a constant construction. A similar, if less substantial, gain is available when $f = \mathsf{constant}\, h$, and $g = \mathsf{arr}\, g'$ – in this case, $f \ggg g = \mathsf{constant}\, g'(h)$. While taking first of this will result in an arr construction, which would also have resulted from the right side, the right side would involve computing two functions and composing their outputs, rather than this left side which would only require one function call.

The other law which breaks under strict equality is the law dealing with first and fst:

$$\mathsf{first}\, f \ggg \mathsf{arr}\, \mathsf{fst} = \mathsf{arr}\, \mathsf{fst} \ggg f$$

Again, informally, it should be obvious that the right side is more efficient than the left, since it saves a computation of $\mathsf{first}\, f$. Again, while this is a good savings, it is not as important as the other optimization, namely that when $f = \mathsf{constant}\, k$, the right side immediately reduces to $\mathsf{constant}\, k$, while the left side reduces to $\mathsf{arr}(\lambda\_ \to k)$. These two expressions are equivalent as proven above, but the right side involves zero function calls, which is a savings over any function calls at all.

One other optimization is possible, and is based on the law above that composition is associative. Since composition is associative, let us choose to make it left-associative. This choice is valid, but can have a useful effect in optimizations, again involving the constant case. When the rightmost arguement to $f \ggg g \ggg h \ggg \ldots \ggg z$ is a constant, none of the preceeding arguments need be evaluated (especially in a lazy-evaluation language such as Haskell), since the first case of the implementation of composition will immediately return the constant. To wit, if composition is left-associative, then

$$\begin{aligned} f \ggg g \ggg h \ggg \mathsf{constant}\, k &= ((f \ggg g) \ggg h) \ggg \mathsf{constant}\, k \\ &= \mathsf{constant}\, k \end{aligned}$$

whereas if it were right-associative, the reduction sequence would be

$$\begin{aligned} f \ggg g \ggg h \ggg \mathsf{constant}\, k &= f \ggg (g \ggg (h \ggg \mathsf{constant}\, k)) \\ &= f \ggg (g \ggg \mathsf{constant}\, k) \\ &= f \ggg \mathsf{constant}\, k \\ &= \mathsf{constant}\, k \end{aligned}$$

which yields the same result, but in a time linear in the length of the chain of compositions.

Each of these three optimizations are useful on their own, but even more benefit can be derived by considering their interaction. For example, the statement

$$\mathsf{first}\, f \ggg \mathsf{first}\, g \ggg \mathsf{first}\, h = \mathsf{first}(f \ggg g \ggg h)$$

is obvious, as are the generalizations to longer chains. (This one is derivable in three steps via

$$\begin{aligned} \mathsf{first}\, f \ggg \mathsf{first}\, g \ggg \mathsf{first}\, h &= \mathsf{first}(f \ggg g) \ggg \mathsf{first}\, h \\ &= \mathsf{first}((f \ggg g \ggg)\, \mathsf{first}\, h) \\ &= \mathsf{first}(f \ggg g \ggg h) \end{aligned}$$

where the insistence on parentheses in the second step can be dropped by associativity, and the distribution of first comes from the first rule discussed above.) This has the potential for drastically

reducing the computations needed if a long chain of arrows are applied to one input, while the other is carried through (as might happen, say in some of the looping examples). If at any point in that chain, a constant is encountered, all prior elements of the computation can be dropped, which could result in large savings. Further, any pure arrows encountered can just be "wrapped" around earlier ones, thus shortening the chain still further. For more on this particular optimization, see below.

The second law above lets you discard extraneous data in the second argument; this law is less useful for optimizations, though if a program is constructed to carry along and then discard its second argument, it can easily just ignore it in the first place. As such, while it is an optimization, it seems less likely to arise.

These two laws can interact with the seven strictly satisfied laws, to produce other optimizations as well. While the seven laws themselves have no one side more efficient than the other, some do have one side which interacts better with these laws, which do. For example, the law

$$\mathsf{arr}(f \cdot g) = \mathsf{arr}\, f \ggg \mathsf{arr}\, g$$

is satisfied with strict equality, so neither side is more efficient. However, consider the following scenario:

$$f \ggg \mathsf{arr}\, g \ggg \mathsf{arr}\, h$$

where f could be anything. If f is a constant, then the whole expression evaluates to a constant, but in one fewer steps if $(\mathsf{arr}\, g \ggg \mathsf{arr}\, h)$ is reduced first. If f is a pure arrow, no benefit ensues. But if f is a general arrow, this does have some effect, because rather than compute the recursive cpAux twice (once with g and once with h), it can be computed once (with $h \cdot g$).

Similarly, $\mathsf{first}\, \mathsf{arr}\, f$ is less useful than $\mathsf{arr}(f \times \mathsf{id})$, since the latter form is amenable to some of the optimizations mentioned above.

Other transformations among the arrow laws could be useful, working towards the goal of minimizing the number of applications of first present, since of the three primitive combinators, this is the only one which reduces the possible optimizations.

4 Conclusions and Future Work

The above proofs have shown that SF, the class upon which Yampa is based, in fact does behave as an arrow ought to, thus providing a strong support for reasoning about programs written in Yampa – these proofs constitute an assurance that the the dataflow in the program will behave correctly (though not necessarily as intended – these proofs do not eliminate the existence of logic bugs!) as arrows should. Further, since two of the laws are not satisfied by equality, some discussion was given as to possible optimization strategies based on the arrow laws.

This is not the complete semantics, however, of SF. SF is also declared to be an ArrowLoop – an arrow which has another primitive combinator loop, and which satisfies another set of six identities, also defined in [5]. These laws describe how loop interacts with the three primitive combinators shown above, namely first, $\ggg$ and arr. Looping arrows correspond very closely to circuits with feedback – one or more outputs of the circuit (arrow) are looped and fed back into corresponding inputs of the circuit (arrow). To prove the ArrowLoop laws for SF requires an excursion into domain theory, to understand the semantics of, and to reason about, divergent or unstable loops. One subtle and intriguiging aspect of the loop operator is that the second (feedback) argument of the arrow need not stabilize, provided that the function returns a value for its first component of output which does not depend on the feedback. Another is that reasoning about such loops leads

directly to the notions of fixpoints and least fixpoints, the latter of which corresponds directly to reasoning about what a feedback circuit will produce given no *a priori* knowledge of the current state of the circuit.

With a complete treatment of these ArrowLoop laws, the semantics of SF in Yampa will be shown to be consistent with its expected behavior, and can then be used with no further consideration to the implementation.

References

[1] Paul Hudak, *The haskell school of expression; learning functional programming through multimedia*, Cambridge University Press, 2000.

[2] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson, *Arrows, robots, and functional reactive programming*, Summer School on Advanced Functional Programming 2002, Oxford University, Lecture Notes in Computer Science, Springer-Verlag, 2003, To Appear.

[3] ———, *Yampa 0.9.1 source code*, http://haskell.org/yampa/, 2003.

[4] John Hughes, *Generalising monads to arrows*, Science of Computer Programming **37** (2000), no. 1–3, 67–111.

[5] Ross Paterson, *A new notation for arrows*, International Conference on Functional Programming, ACM Press, September 2001, pp. 229–240.

[6] ———, *Arrows and computation*, The Fun of Programming (Jeremy Gibbons and Oege de Moor, eds.), Palgrave, 2003, pp. 201–222.