Equivalence of Applicative Functors and Multifunctors

Andreas Abel
Department of Computer Science, Gothenburg University, Sweden

February 2022, January 2024

McBride and Paterson [2008] introduced `Applicative` functors to Haskell, which are equivalent to the lax monoidal functors (with strength) of category theory. Applicative functors F are presented via idiomatic application $\_\circledast\_ \,:\, F\,(A \to B) \to F\,A \to F\,B$ and laws that are a bit hard to remember. Capriotti and Kaposi [2014] observed that applicative functors can be conceived as multifunctors, i. e., by a family $\mathsf{liftA}_n \,:\, (A_1 \to \ldots \to A_n \to C) \to F\,A_1 \to \ldots \to F\,A_n \to F\,C$ of `zipWith`-like functions that generalize `pure` ($n = 0$), `fmap` ($n = 1$) and `liftA2` ($n = 2$). This reduces the associated laws to just the first functor law and a uniform scheme of second (multi)functor laws, i. e., a composition law for liftA. In this note, we rigorously prove that applicative functors are in fact equivalent to multifunctors, by interderiving their laws.

1 Introduction

McBride and Paterson [2008] introduce applicative functors as a Haskell type constructor class Applicative F with two methods

pure	:	$A \to F\,A$	embedding
$\_\circledast\_$	:	$F\,(A \to B) \to F\,A \to F\,B$	idiomatic application, left associative

satisfying four laws:

identity	$\mathsf{pure}\,(\lambda x \to x) \circledast u$	$=$	u
composition	$\mathsf{pure}\,(\lambda f\,g\,x \to f\,(g\,x)) \circledast u \circledast v \circledast w$	$=$	$u \circledast (v \circledast w)$
interchange	$\mathsf{pure}\,(\lambda f \to f\,x) \circledast u$	$=$	$u \circledast \mathsf{pure}\,x$
homomorphism	$\mathsf{pure}\,(f\,x)$	$=$	$\mathsf{pure}\,f \circledast \mathsf{pure}\,x$

Using the usual definitions of identity id and composition $(\_\circ\_)$ the first two laws can be presented as follows:

identity	$\mathsf{pure}\,\mathsf{id} \circledast u$	$=$	u
composition	$\mathsf{pure}\,(\_\circ\_) \circledast u \circledast v \circledast w$	$=$	$u \circledast (v \circledast w)$

Functoriality of F is recovered via $\mathsf{fmap}\ f\ u = \mathsf{pure}\ f \circledast u$ where identity acts as the first functor law. The second functor law can be derived via composition and homomorphism as follows:

$$\begin{array}{lclcl}
\mathsf{fmap}\ f\ (\mathsf{fmap}\ g\ u) & = & \mathsf{pure}\ f \circledast (\mathsf{pure}\ g \circledast u) & = & \mathsf{pure}\ (_\circ_) \circledast \mathsf{pure}\ f \circledast \mathsf{pure}\ g \circledast u \\
 & = & \mathsf{pure}\ (f \circ _) \circledast \mathsf{pure}\ g \circledast u & = & \mathsf{pure}\ (f \circ g) \circledast u \\
 & = & \mathsf{fmap}\ (f \circ g)\ u & &
\end{array}$$

Unfortunately, McBride and Paterson's laws are not easy to remember, especially the composition and interchange laws. They do not follow simple patterns like the functor laws which can be seen as actions of the function category, or the monad laws, which can be conceived as generalization of the monoid laws. It is also not intuitively clear at a glance that these laws are complete.

Starting with GHC 8.2 (2017), Applicatives can also be given via $\mathsf{liftA}_2 : (A \to B \to C) \to F\,A \to F\,B \to F\,C$ rather than idiomatic application, which are interdefinable:

$$\begin{array}{lcl}
\mathsf{liftA}_2\ f\ u\ v & = & \mathsf{pure}\ f \circledast u \circledast v \\
h \circledast u & = & \mathsf{liftA}_2\ (\lambda f\,x \to f\,x)\ h\ u
\end{array}$$

However, to this date (2024-01-24) the documentation[1] of Applicative does not spell out the type class laws in terms of liftA_2.

Note that liftA_2 appears to be the binary generalization of the unary $\mathsf{fmap} = \mathsf{liftA}_1 : (A \to B) \to F\,A \to F\,B$. In the same way, we get the nullary $\mathsf{pure} = \mathsf{liftA}_0 : A \to F\,A$.

In this note, we show that the further generalization to arbitrary arities liftA_n gives very elegant laws for the family liftA_n, which are just generalizations of the two functor laws.

The infinite family liftA_n can be truncated to $n \le 2$, yielding the following composition laws in addition to the functor laws (for liftA_1):

$$\begin{array}{lcll}
\mathsf{liftA}_1\ f\ (\mathsf{liftA}_0\ x) & = & \mathsf{liftA}_0\ (f\ x) & \text{homomorphism} \\
\mathsf{liftA}_2\ f\ (\mathsf{liftA}_0\ x) & = & \mathsf{liftA}_1\ (f\ x) & \text{homomorphism} \\
\mathsf{liftA}_2\ f\ u\ (\mathsf{liftA}_0\ y) & = & \mathsf{liftA}_1\ (\lambda x \to f\ x\ y)\ u & \text{exchange} \\
\mathsf{liftA}_2\ f\ (\mathsf{liftA}_1\ g\ u) & = & \mathsf{liftA}_2\ (f \circ g)\ u & \text{2nd functor law} \\
\mathsf{liftA}_2\ f\ u\ (\mathsf{liftA}_1\ h\ v) & = & \mathsf{liftA}_2\ (\lambda x \to f\ x \circ h)\ u\ v & \\
\mathsf{liftA}_1\ f\ (\mathsf{liftA}_2\ g\ u\ v) & = & \mathsf{liftA}_2\ (\lambda x \to f \circ g\ x)\ u\ v & \\
\mathsf{liftA}_2\ f\ (\mathsf{liftA}_2\ g\ u\ v)\ w & = & \mathsf{liftA}_2\ (\lambda x(y,z) \to f\ (g\ x\ y)\ z)\ u\ (\mathsf{liftA}_2\ (_,_)\ v\ w) & \\
\mathsf{liftA}_2\ f\ u\ (\mathsf{liftA}_2\ g\ v\ w) & = & \mathsf{liftA}_2\ (\lambda(x,y) \to f\ x \circ g\ y)\ (\mathsf{liftA}_2\ (_,_)\ u\ v)\ w &
\end{array}$$

2 Applicative Functors as Multifunctors

Preliminaries: generalized composition. If $f : A_{1..n} \to C \to D$ and $g : B_{1..m} \to C$ then let $f \circ^m_n g : A_{1..n} \to B_{1..m} \to D$ be defined by

$$(f \circ^m_n g)\ a_{1..n}\ b_{1..m} = f\ a_{1..n}\ (g\ b_{1..m}).$$

[1] `https://hackage.haskell.org/package/base-4.19.0.0/docs/Control-Applicative.html`

Herein, $h\, x_{1..n}$ is to be understood as curried application $h\, x_1 \,\ldots\, x_n$. A sequence $x_{1..n}$ may more succinctly be written as $\vec{x}^n$ or just $\vec{x}$.

Note that $(f \circ^1_0 g)\, x = f\,(g\, x)$ is ordinary unary function composition. Further, $(f \circ^0_n y)\, a_{1..n} = f\; a_{1..n}\; y$ is partial application of f to its $n{+}1$st argument, which for $n = 0$ is just plain application: $f \circ^0_0 y = f\; y$.

Multifunctors. To distinguish our concept of applicative functors from that of McBride and Paterson, we temporarily call them multifunctor.[2]

A multifunctor F shall be witnessed by a family of functions ($n \geq 0$)

$$\mathsf{liftA}_n : (A_1 \to \ldots \to A_n \to C) \to F A_1 \to \ldots \to F A_n \to F C$$

satisfying the following laws:

$$\begin{array}{lrcl} \text{identity} & \mathsf{liftA}_1\;\mathsf{id} & = & \mathsf{id} \\ \text{composition} & \mathsf{liftA}_{n+1+m}\; f\; \vec{u}^n\; (\mathsf{liftA}_k\; g\; \vec{v}^k) & = & \mathsf{liftA}_{n+k+m}\; (f \circ^k_n g)\; \vec{u}\; \vec{v} \end{array}$$

We may drop the index to liftA when it is generic or clear from the context of discourse.

Just functoriality of F can be recovered by $\mathsf{fmap} = \mathsf{liftA}_1$ with identity being the first functor law and composition specializing to the second functor law with $n = m = 0$ and $k = 1$:

$$\mathsf{liftA}_1\; f\; (\mathsf{liftA}_1\; g\; v) = \mathsf{liftA}_1\; (f \circ^1_0 g)\; v$$

Pure computations are represented via $\mathsf{pure} = \mathsf{liftA}_0$, with the composition law specializing to:

$$\mathsf{liftA}\; f\; \vec{u}^n\; (\mathsf{pure}\, x) = \mathsf{liftA}\; (f \circ^0_n x)\; \vec{u}$$

For fmap ($n = m = 0$) this yields $\mathsf{fmap}\, f\, (\mathsf{pure}\, x) = \mathsf{pure}\,(f\; x)$. For just $n = 0$ we get law $\mathsf{liftA}_{1+m}\; f\; (\mathsf{pure}\, x)\; \vec{w} = \mathsf{liftA}_m\; (f\; x)\; \vec{w}$. This can be iterated to $\mathsf{liftA}_n\; f\; (\mathsf{pure}\, x_1)\; \ldots\; (\mathsf{pure}\, x_n) = \mathsf{pure}\,(f\; x_{1..n})$ corresponding to the intuition that composition of effect-free computations is again an effect-free computation.

2.1 Multifunctors are applicative

Idiomatic application can be obtained as a special case of liftA_2:

$$\begin{array}{rcl} \_ \circledast \_ & : & F(A \to B) \to F A \to F B \\ u \circledast v & = & \mathsf{liftA}_2\; \mathsf{id}\; u\; v \end{array}$$

We easily derive its laws:

1. identity:

$$\begin{array}{rcllcll} \mathsf{pure\,id} \circledast u & = & \mathsf{liftA}_2\;\mathsf{id}\;(\mathsf{pure\,id})\; u & = & \mathsf{liftA}_1\;(\mathsf{id} \circ^0_0 \mathsf{id})\; u & & \text{by composition} \\ & = & \mathsf{liftA}_1\;(\mathsf{id\,id})\; u & = & \mathsf{liftA}_1\;\mathsf{id}\; u & = u & \text{by identity} \end{array}$$

[2] The name multi-functor is taken from Capriotti and Kaposi [2014] and already motivated there as means to "naturally arrive at the definition of the Applicative clase via an obvious generalization of the notion of functor."

2. composition.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{pure}\,(\_\circ\_) \circledast u \circledast v \circledast w} \\
= \mathsf{liftA}_2\,\mathsf{id}\,(\mathsf{pure}\,(\_\circ\_)\,u \circledast v \circledast w & = \mathsf{liftA}_1\,(\_\circ\_)\,u \circledast v \circledast w & \text{by composition} \\
= \mathsf{liftA}_2\,\mathsf{id}\,(\mathsf{liftA}_1\,(\_\circ\_)\,u)\,v \circledast w & = \mathsf{liftA}_2\,(\_\circ\_)\,u\,v \circledast w & \text{by composition} \\
= \mathsf{liftA}_2\,\mathsf{id}\,(\mathsf{liftA}_2\,(\_\circ\_)\,u\,v)\,w & = \mathsf{liftA}_3\,(\_\circ\_)\,u\,v\,w & \text{by composition} \\
= \mathsf{liftA}_3\,(\lambda f\,g\,x \to f\,(g\,x))\,u\,v\,w & = \mathsf{liftA}_3\,(\lambda f\,g\,x \to \mathsf{id}\,f\,(\mathsf{id}\,g\,x))\,u\,v\,w & \\
= \mathsf{liftA}_3\,(\mathsf{id} \circ^2_1 \mathsf{id})\,u\,v\,w & = \mathsf{liftA}_2\,\mathsf{id}\,u\,(\mathsf{liftA}_2\,\mathsf{id}\,v\,w) & \text{by composition} \\
= u \circledast (v \circledast w) & &
\end{array}
$$

3. homomorphism. This has been shown before, here again step-by-step:

$$
\begin{array}{llll}
\mathsf{pure}\,f \circledast \mathsf{pure}\,x & = \mathsf{liftA}_2\,\mathsf{id}\,(\mathsf{pure}\,f)\,(\mathsf{pure}\,x) & = \mathsf{liftA}_1\,(\mathsf{id}\,f)\,(\mathsf{pure}\,x) & \text{by composition} \\
& = \mathsf{liftA}_1\,f\,(\mathsf{pure}\,x) & = \mathsf{liftA}_0\,(f\ x) & \text{by composition} \\
& = \mathsf{pure}\ (f\ x) & &
\end{array}
$$

4. interchange:

$$
\begin{array}{llll}
u \circledast (\mathsf{pure}\,x) & = \mathsf{liftA}_2\,\mathsf{id}\,u\,(\mathsf{liftA}_0\,x) & = \mathsf{liftA}_1\,(\mathsf{id} \circ^0_1 x)\,u & \text{by composition} \\
& = \mathsf{liftA}_1\,(\lambda f \to f\ x)\,u & = \mathsf{pure}\,(\lambda f \to f\,x) \circledast u & \text{by composition}
\end{array}
$$

2.2 Applicative functors are multifunctors

Following McBride and Paterson [2008], the family liftA_n can be defined for each applicative functor:

$$
\begin{array}{lll}
\mathsf{liftA}_0\,x & = & \mathsf{pure}\,x \\
\mathsf{liftA}_{n+1}\,f\,\vec{u}\,v & = & \mathsf{liftA}_n\,f\,\vec{u} \circledast v
\end{array}
$$

The identity is just identity. We establish composition by a series of inductions.

Lemma 1 (Frame). If $\mathsf{liftA}_n\,f\,\vec{u}^n = \mathsf{liftA}_k\,g\,\vec{v}^k$ then $\mathsf{liftA}_{n+m}\,f\,\vec{u}\,\vec{w}^m = \mathsf{liftA}_{k+m}\,g\,\vec{v}\,\vec{w}$.

Proof. By induction on m. □

As a consequence of Lemma 1, we only need to show the composition law for $m = 0$:

$$\mathsf{liftA}_{n+1}\,f\,\vec{u}^n\,(\mathsf{liftA}_k\,g\,\vec{v}^k) = \mathsf{liftA}_{n+k}\,(f \circ^k_n g)\,\vec{u}\,\vec{v}$$

We first show the case $n = 0$:

Lemma 2 (Composition for $n = 0$).

$$\mathsf{liftA}_1\,f\,(\mathsf{liftA}_k\,g\,\vec{v}) = \mathsf{liftA}_k\,(f \circ^k_0 g)\,\vec{v}$$

Proof. By induction on k.

Case $k = 0$: This is homomorphism.

Case $k \to k + 1$.

$$
\begin{array}{lll}
\multicolumn{2}{l}{\mathsf{liftA}_1\, f\, (\mathsf{liftA}_{k+1}\, g\, \vec{v}\, w)} & \\
= & \mathsf{pure}\, f \circledast (\mathsf{liftA}_k\, g\, \vec{v} \circledast w) & \\
= & \mathsf{pure}\, (\_\circ\_) \circledast \mathsf{pure}\, f \circledast \mathsf{liftA}_k\, g\, \vec{v} \circledast w & \text{by composition} \\
= & \mathsf{pure}\, (f \circ \_) \circledast \mathsf{liftA}_k\, g\, \vec{v} \circledast w & \text{by homomorphism} \\
= & \mathsf{liftA}_k\, ((f \circ \_) \circ^k_0\, g)\, \vec{v} \circledast w & \text{by ind.hyp.} \\
= & \mathsf{liftA}_{k+1}\, (f \circ^{k+1}_0\, g)\, \vec{v}\, w &
\end{array}
$$

For the last step, note that $(f \circ \_) \circ^k_0 g = \lambda \vec{x}^k \to (f \circ \_)(g\, \vec{x}) = \lambda \vec{x} \to f \circ (g\, \vec{x}) = \lambda \vec{x} y \to f(g\, \vec{x}\, y) = f \circ^{k+1}_0 g$. □

Corollary 3 (Composition for $k = 0$).

$$\mathsf{liftA}_{n+1}\, f\, \vec{u}\, (\mathsf{pure}\, x) = \mathsf{liftA}_n\, (f \circ^0_n x)\, \vec{u}$$

Proof.

$$
\begin{array}{lllll}
\multicolumn{5}{l}{\mathsf{liftA}_{n+1}\, f\, \vec{u}\, (\mathsf{pure}\, x)} \\
= & \mathsf{liftA}_n\, f\, \vec{u} \circledast \mathsf{pure}\, x & = & \mathsf{pure}\, (\lambda k \to k\, x) \circledast \mathsf{liftA}_n\, f\, \vec{u} & \text{by exchange} \\
= & \mathsf{liftA}_n\, ((\lambda k \to k\, x) \circ^n_0 f)\, \vec{u} & = & \mathsf{liftA}_n\, (f \circ^0_n x)\, \vec{u} &
\end{array}
$$

The last step is justified by $(\lambda k \to k\, x) \circ^n_0 f = \lambda \vec{y}^k \to (\lambda k \to k\, x)(f\, \vec{y}) = \lambda \vec{y}^k \to f\, \vec{y}\, x = f \circ^0_n x$. □

Theorem 4 (Composition).

$$\mathsf{liftA}_{n+1}\, f\, \vec{u}^n\, (\mathsf{liftA}_k\, g\, \vec{v}^k) = \mathsf{liftA}_{n+k}\, (f \circ^k_n g)\, \vec{u}\, \vec{v}$$

Proof. By induction on k.

Case $k = 0$: This is Corollary 3.

Case $k \to k + 1$:

$$
\begin{array}{lll}
\multicolumn{2}{l}{\mathsf{liftA}_{n+1}\, f\, \vec{u}^n\, (\mathsf{liftA}_{k+1}\, g\, \vec{v}^k\, w)} & \\
= & \mathsf{liftA}_n\, f\, \vec{u} \circledast (\mathsf{liftA}_k\, g\, \vec{v} \circledast w) & \\
= & \mathsf{pure}\, (\_\circ\_) \circledast \mathsf{liftA}_n\, f\, \vec{u} \circledast \mathsf{liftA}_k\, g\, \vec{v} \circledast w & \text{by composition} \\
= & \mathsf{liftA}_n\, ((\_\circ\_) \circ^n_0 f)\, \vec{u} \circledast \mathsf{liftA}_k\, g\, \vec{v} \circledast w & \text{by Lemma 2} \\
= & \mathsf{liftA}_{n+k}\, (((\_\circ\_) \circ^n_0 f) \circ^k_n g)\, \vec{u}\, \vec{v} \circledast w & \text{by ind.hyp.} \\
= & \mathsf{liftA}_{n+k+1}\, (f \circ^{k+1}_n g)\, \vec{u}\, \vec{v}\, w &
\end{array}
$$

For the last step, we calculate $((\_\circ\_) \circ^n_0 f) \circ^k_n g = \lambda \vec{x}^n \vec{y}^k \to (\lambda \vec{x}^n \to (f\, \vec{x}) \circ \_))\, \vec{x}\, (g\, \vec{y}) = \lambda \vec{x}^n \vec{y}^k \to (f\, \vec{x}) \circ (g\, \vec{y}) = \lambda \vec{x}^n \vec{y}^k z \to f\, \vec{x}\, (g\, \vec{y}\, z) = f \circ^{k+1}_n g$.

□

Q.E.D.

References

P. Capriotti and A. Kaposi. Free applicative functors. In P. B. Levy and N. Krishnaswami, editors, Proc. 5th Wksh. on Mathematically Structured Functional Programming, MSFP 2014, volume 153 of Electr. Proc. in Theor. Comp. Sci., pages 2–30, 2014. URL https://doi.org/10.4204/EPTCS.153.2.

C. McBride and R. Paterson. Applicative programming with effects. J. Func. Program., 18(1): 1–13, 2008. URL https://doi.org/10.1017/S0956796807006326.