Applicative Archery (supplement)

by Daniel Mlot (duplode.github.io)

Every applicative functor in Haskell corresponds to a category, in which the arrows are the "effectful functions" of general type `(Applicative f) => f (a -> b)`. In these notes, we will see how identity and composition for these categories can be defined in terms of `pure` and `(<*>)`, and how to deduce the applicative laws in their usual guise from the category laws.

Definitions

T is some Haskell (Applicative) Functor. The functor laws hold:

$fmap :: (a \to b) \to (T\,a \to T\,b)$
$fmap\ id = id$ [F-1st]
$fmap\ (g . f) = fmap\ g . fmap\ f$ [F-2nd]

fmap will be instantiated at T unless specified otherwise.

T' is the composite functor $T \circ (\to)(\forall b. b \to b)$. Lifting to and from it is done through the sta/fin isomorphism defined below and through $(.)$ (fmap for Reader-like Functors).

$sta = const :: a \to ((b \to b) \to a)$ [sta-def]
$fin = (\$ id) :: ((b \to b) \to a) \to a$ [fin-def]
$fin\ (sta\ x) = x\ ;\ sta\ (fin\ k) = k$ [sta-fin]
$fmap_{T'} = fmap_T . (.)\ ;\ sta\ x\ id = x$

Though the signatures use $(b \to b)$ rather than $(\forall b. b \to b)$ to avoid impredicativity, the proofs will be done as if the quantifier was there.

We postulate

$idA :: T\,(a \to a)$ | $(.*) :: T\,(b \to c) \to T\,(a \to b) \to T\,(a \to c)$

such that the "effectful functions" $T\,(a \to b)$ are arrows in a category, which we will call T-A.

$idA .* u = u$ [A-lid]
$v .* idA = v$ [A-rid]
$w .* v .* u = w .* (v .* u)$ [A-assoc] $(.*)$ is left-associative.

By $I_{T\text{-}A}$ we will refer to the identity functor in T-A. An arrow under $I_{T\text{-}A}$ is a value under the T HASK endofunctor.

Given the usual Applicative operations,

```
pure :: a -> T a     |  (<*>) :: T (a -> b) -> (T a -> T b)
```

our goal is proving the Applicative laws,

```
pure id <*> u = u                              [A-1st]
pure f <*> pure x = pure (f x)                 [A-2nd]
v <*> pure x = pure ($ x) <*> v                [A-3rd]
pure (.) <*> w <*> v <*> u = w <*> (v <*> u)   [A-4th]
pure f <*> u = fmap f u                        [A-F]
```

from the category laws for T-A, the functor laws, relevant naturality properties/free theorems and sensible specifications for the relationships between idA/(.*) and pure/(<*>).

Functors to and from T-A

(.*) and (<*>) should be the arrow mappings of functors from T-A to Hask; also, pure, when specialised to (a -> b) -> T (a -> b) should be the arrow mapping of a functor from Hask to T-A. That (.*) is such a mapping is clear, as the functor laws in this case are equivalent to [A-lid] and [A-assoc]. For pure and (<*>), we would have:

```
pure id = idA                     [pure-id]
pure (g . f) = pure g .* pure f   [pure-comp]

idA <*> u = u                     [<*>-id]
w .* v <*> u = w <*> (v <*> u)    [<*>-comp]
```

We take [pure-id] as an specification for idA. The other properties shall be proved in due course.

Naturality properties

pure as natural transformation (n.t.) from the identity functor to T in Hask:

pure . f = fmap f . pure [pure-nat]   pure (f x) = fmap f (pure x)

A way to define pure in terms of idA follows immediately:

pure (f x) = fmap f (pure x) [pure-nat]
pure (sta x id) = fmap (sta x) (pure id) [f/sta x] [x/id]
pure x = fmap (sta x) idA [pure-spec] [sta-def] [pure-id]

(.*) as n.t. from T(a -> b) to T(((->) r a) -> ((->) r b)), for both the covariant and the contravariant parts:

fmap (f.) v.* u = fmap (f.) (v.* u) [.*-nat]
fmap (.f) v.* u = v.* fmap (f.) u [.*-cnat]

From these follow properties analogous to [A-F] and [A-2nd], only for (.*) instead of (<*>), as well as confirmation that pure is an arrow mapping of a functor (a.m.f).

fmap (f.) idA .* u = fmap (f.) (idA .* u) [.*-nat] [v/idA]
fmap (f.) (pure id) .* u = fmap (f.) u [pure-id] [A-lid]
pure (f.id) .* u = fmap (f.) u [pure-nat]
pure f .* u = fmap (f.) u [A-F-l]

$\text{fmap } (.f)\ v \mathbin{.*} idA = v \mathbin{.*} \text{fmap } (f.)\ idA$ [.*-cnat] [u/idA]
$\text{fmap } (.f)\ v = v \mathbin{.*} \text{fmap } (f.)\ (\text{pure } id)$ [pure-id] [A-rid]
$\text{fmap } (.f)\ v = v \mathbin{.*} \text{pure } (f . id)$ [pure-nat]
$v \mathbin{.*} \text{pure } f = \text{fmap } (.f)\ v$ [A-F-r]

$\text{pure } g \mathbin{.*} \text{pure } f = \text{fmap } (.f)\ (\text{pure } g)$ [A-F-r] [v/pure g]
$\text{pure } g \mathbin{.*} \text{pure } f = \text{pure } ((.f)\ g)$ [pure-nat]
$\text{pure } g \mathbin{.*} \text{pure } f = \text{pure } (g . f)$ [pure-comp]

$(\langle * \rangle)$ as n.t. from $T (a \to b)$ to $T a \to T b$, for both the covariant and the contravariant parts:

$\text{fmap } (f.)\ v \langle * \rangle u = \text{fmap } f\ (v \langle * \rangle u)$ [<*>-nat]
$\text{fmap } (.f)\ v \langle * \rangle u = v \langle * \rangle \text{fmap } f\ u$ [<*>-cnat]

We will make use of these results for $(\langle * \rangle)$ in a little while.

sta and fin as n.t.s between the (.*)-functors (that is, the functors from "A-categories" to HASK with (.*) as a.m.f.) to $T ((\to) r) a)$ and $T' ((\to)\ r\ a)$:

$v \mathbin{.*} u = \text{fmap fin } (\text{fmap } (.)\ v \mathbin{.*} \text{fmap sta } u)$ [.*-iso]

An analogous result holds for the $(\langle * \rangle)$-functors to $T a$ and $T' a$, hinging on the supposition that $(\langle * \rangle)$ is indeed an a.m.f.

$v \langle * \rangle u = \text{fmap fin } (\text{fmap } (.)\ v \langle * \rangle \text{fmap sta } u)$ [<*>-iso] sta and fin as n.t.s between the $(\langle * \rangle)$-functor to $T a$ and the $(.*)$-functor to $T' a = T((\to)\ (b \to b)\ a)$:

$$v \langle * \rangle u = \text{fmap fin } (v .* \text{ fmap sta } u) \quad [\langle * \rangle\text{-spec}]$$

We will use this result as an specification for $(\langle * \rangle)$. Note that this property only follows from naturality if $(\langle * \rangle)$ is an a.m.f; that is, if $[\langle * \rangle\text{-id}]$ and $[\langle * \rangle\text{-comp}]$ hold. For that reason, we will not use any other consequences of that hypothesis (such as $[\langle * \rangle\text{-iso}]$) until we prove $[\langle * \rangle\text{-id}]$ and $[\langle * \rangle\text{-comp}]$, and therefore that, given our other assumptions, $(\langle * \rangle)$ is an a.m.f iff $[\langle * \rangle\text{-spec}]$.

Intuitively, sta is used in $[\langle * \rangle\text{-spec}]$ to "functionalise" the second argument of $(\langle * \rangle)$, making it an arrow that can be composed through $(.*)$. fin is then used to reverse the transformation by supplying a dummy argument.

We can also get a definition of $(.*)$ in terms of $(\langle * \rangle)$:

$$v .* u = \text{fmap fin } (\text{fmap } (.)\ v .* \text{ fmap sta } u) \quad [.*\text{-iso}]$$
$$v .* u = \text{fmap } (.)\ v \langle * \rangle u \quad [.*\text{-spec}] \quad [\langle * \rangle\text{-spec}]\ [v / \text{fmap } (.)\ v]$$

First and second laws

Getting from [<*>-spec] to the first law is straightforward:

idA <*> u = fmap fim (idA .* fmap sta u) [<*>-spec] [v/idA]
idA <*> u = fmap fim (fmap sta u) [A-lid]
idA <*> u = fmap (fim . sta) u [F-2nd]
idA <*> u = fmap id u [sta-fim]
idA <*> u = u [<*>-id] [F-1st]
pure id <*> u = u [A-1st]

At this point, there are multiple ways to get to [A-F] and the second law. Here, we will start from [<*>-nat]:

fmap (f.) v <*> u = fmap f (v <*> u) [<*>-nat]
fmap (f.) idA <*> u = fmap f (idA <*> u) [v/idA]
fmap (f.) (pure id) <*> u = fmap f u [pure-id] [<*>-id]
pure (f . id) <*> u = fmap f u [pure-nat]
pure f <*> u = fmap f u [A-F]

pure f <*> pure x = fmap f (pure x) [A-F] [u/pure x]
pure f <*> pure x = pure (f x) [A-2nd] [pure-nat]

Another way of stating [A-F] is

fmap = (<*>) . pure [A-F]

It suggests that the functor T can be obtained by composing the corresponding pure-functor and (<*>)-functor. We will be able to say that once we prove [<*>-comp].

Third law

Having used [<*>-nat] to prove [A-F], it is time to switch to [<*>-cnat].

```
v <*> pure x
= v <*> fmap (sta x) idA                                   [pure-spec]
= fmap (.sta x) v <*> idA                                  [<*>-cnat]
= fmap fin (fmap (.sta x) v .* fmap sta idA)               [<*>-spec]
= fmap fin (fmap (.sta x) v .* fmap sta (pure id))         [pure-id]
= fmap fin (fmap (.sta x) v .* pure (sta id))              [pure-nat]
= fmap fin (fmap (.sta id) (fmap (.sta x) v))              [A-F-r]
= fmap fin (fmap ((.sta id) . (.sta x)) v)                 [F-2nd]
= fmap fin (fmap (.(sta x . sta id)) v)
= fmap (fin . (.(sta x . sta id))) v                       [F-2nd]
= fmap (fin . (\f -> const (f x))) v                       [sta-def] [fin-def] [*: see note]
= fmap (($id) . (\f -> const (f x))) v                     [fin-def]
= fmap (\f -> f x) v
= fmap ($x) v
= pure ($x) <*> v                                          [A-F]
v <*> pure x = pure ($x) <*> v                             [A-3rd]
```

```
[*]: (.(sta x . sta id)) = \f -> f . sta x . sta id
                         = \f -> \y -> (f . const x) (sta id y)
                         = \f -> f . const x
                         = \f -> const (f x)
```

[A-3rd] is to [A-rid] what [A-1st] is to [A-lid], even though the asymmetry of (<*>) makes the parallel unobvious. Note that the derivation of [A-3rd] requires [A-rid] (via [A-F-r]) but only calls for [A-lid] (via [A-F]) in the final, cosmetical step.

Fourth law (and the $(\langle * \rangle)$-functor)

Now it is time to clear up our debt by proving [<*>-comp]:

$$
\begin{aligned}
& w \langle * \rangle (v \langle * \rangle u) \\
&= \text{fmap fin } (w .\!* \text{ fmap sta } (v \langle * \rangle u)) \quad [\langle * \rangle\text{-spec}] \\
&= \text{fmap fin } (w .\!* \text{ fmap sta } (\text{fmap fin } (v .\!* \text{ fmap sta } u))) \quad [\langle * \rangle\text{-spec}] \\
&= \text{fmap fin } (w .\!* \text{ fmap } (\text{sta} . \text{fin}) \ (v .\!* \text{ fmap sta } u)) \quad [\text{F-2nd}] \\
&= \text{fmap fin } (w .\!* \text{ fmap id } (v .\!* \text{ fmap sta } u)) \\
&= \text{fmap fin } (w .\!* (v .\!* \text{ fmap sta } u)) \quad [\text{F-1st}] \\
&= \text{fmap fin } (w .\!* v .\!* \text{ fmap sta } u) \quad [\text{A-assoc}] \\
&= w .\!* v \langle * \rangle u \quad [\langle * \rangle\text{-spec}] \\
& w .\!* v \langle * \rangle u = w \langle * \rangle (v \langle * \rangle u) \quad [\langle * \rangle\text{-comp}]
\end{aligned}
$$

The proof ensures that $(\langle * \rangle)$ is an a.m.f (iff [<*>-spec]), thus justifying talk about the $(\langle * \rangle)$-functor! In particular, [A-F] now tells us that the T functor is obtained by composing the pure-functor and the $(\langle * \rangle)$-functor. Additionally, [<*>-iso] holds:

$$v \langle * \rangle u = \text{fmap fin } (\text{fmap } (.) \ v \langle * \rangle \text{ fmap sta } u) \quad [\langle * \rangle\text{-iso}]$$

The fourth law readily follows from [<*>-comp]:

$$
\begin{aligned}
& w .\!* v \langle * \rangle u = w \langle * \rangle (v \langle * \rangle u) \quad [\langle * \rangle\text{-comp}] \\
& \text{fmap } (.) \ w \langle * \rangle v \langle * \rangle u = w \langle * \rangle (v \langle * \rangle u) \quad [.\!*\text{-spec}] \\
& \text{pure } (.) \langle * \rangle w \langle * \rangle v \langle * \rangle u = w \langle * \rangle (v \langle * \rangle u) \quad [\text{A-4th}] \ [\text{A-F}]
\end{aligned}
$$

[A-4th] is the law corresponding to [A-assoc]. Thus our task is done, as [A-F] and [A-2nd] are consequences of [A-1st] and naturality conditions, and each of the other laws follows from a category law for T-A.