I think that in Smalltalk a
    Haskell Arrow
    is just a 1 input valuable plus some composition methods

    like

```
>>into: = return a function composition ,
    ( f into: g ) = [ :x | g value: ( f value: x ) ]

>>pair: = return a do both functions and return a size 2 Array ,
    ( f pair: g ) = [ :x | Array with:( f value: x )with:( g value: x ) ]

>>binOp: = return a binary op on Array ,
    ( p binOp: b ) = [ :array | b value:( p value:( array first ) )
                                value:( p value:( array second ) ) ]

>>doAt1 = return a do function on p's Array at:1 and return Array ,
    ( p doAt1 ) = [ :array | array at:1put:( p value: ( array first ) ) ; yourself ]

>>doAt2 = return a do function on Array at:2 and return Array ,
    ( p doAt2 ) = [ :array | array
                             at:2put:( p value: ( array second ) ) ;
                             yourself ]<---------------[ is faster ]
    ( p doAt2 ) = ( p swap into: doAt1 ) swap <---------[ it's shorter it's simpler ]

>>swap = return a do p and return Array transposed
    ( p swap ) = [ :x | ( p value: x ) reverse ]
```

etc (a 1 input valuable is anything that responds to #value:)

Haskell seems to get a lot of mileage out of this
    kind of Category theoretical
    function composition
and no doubt the Haskell type system aids in this
    and practically Maybe demands it
    and makes it really convoluted
        they rebelled against lispy simplicity
            in order to make it simple in order to not to scare the newbies
        they dumped all the parens overboard
        replaced them all with a complicated set of precedent rules
        so now you got to have a parser in your head
        to read the stuff
        and if you don't where you gonna get one
        is that guy going to parse it for you i don't think so
        which far from making it easier
        for anybody but a rank newbie a ranking newbie got to impressem
        this induced parserhead requirement on human
        Haskell code readers
        instead of making it any easier makes it really really hard out on the deep end
        of the pool there is a tremendous drop off a under water cliff
            where newbies who reach puberty begin hanging thenselves out to dry

which doesn't really work cause they keep getting all wet
and this may well be a great resistance to
Haskell uptake
and seems to be an endless source of
confusion contusion and discussion as meetup people's
personal Haskell brain pan parsers shift in and out
of true conformity to the standard
as we speak as we keep on speaking
and
No wait-
s abound
but the people who got a Haskell parser in they head
they say oh no I'm not going through That again
No you get a parser in your head
or you're out
but i regress -digress!
The Smalltalk runtime type system makes it a lot
simpler to see what is going on
so i wonder
what effect this kind of function composition
style could have on Smalltalk code
Where maybe you have a bunch of Methods
that just return 1 input valuables
([:x | ...] , SomeFunctionClass>>value: , etc)
which then get turned into Arrows
and get categorically indubitably functionally composed
who knows what could happen

the Objects themselves in this functional Smalltalk style could be
mostly empty except for accessors
and then stateless Traits
could be the functions
or there could be separate stateless functional Classes
that have the stateless functions
which could be pluggable
on multiple different applicable Object Classes
(kind of a pluggable multiple inheritance thing going on)(?)
because
supposedly
then
you can make new functions
by just using
very simplified
composition expressions
as in the short example above (p doAt2) = ^(p swap into: doAt1) swap
as opposed to having to know a lot about
all the inputs and all about how the sends are supposed to fit together
involving lots of looking things up over and over
the composition combinators do all that
for you
or so the combinator propaganda goes

but is it really true
    or do you have to know just as much
    or more to actually get the
    function combinators to work
    or do you have to finally wise up
        and get smart
        and stop fooling around with writing yet
            another monad
        tutorial(s) just like all the other ranking newbies   and get serious
            and write a brain pan parser compiler
        and become a first class tenderfoot
    And now you're much worse off
        than before
        in the time spent column
        and you don't even know it
        that's the sad part
        because the Maybe she's elegance column is calling   sirens calling
            clouding your mind with foggy fogged up window pain desires
            and you don't even know which end is up anymore
            and you want some of that categorical shit
            they're having at the big people's table

so is it worth it
does it work as advertised
  or is it just to be deride
i wonder

But i would like it if a
    Smalltalk Arrow
    could explain itself
So an Arrow is just an Object that contains a 1 input valuable
    and a BinaryArrow isA Arrow and contains 2 Arrows
    so Arrow>>explainYourself
        returns
        an Array tree of oneInputValuable source code Strings
        which describe how
        the Arrow works because otherwise it's just a great big mystery
    Arrow
        hasA oneInputValuable <---[a one input function]
        BinaryArrow isA Arrow
            hasA firstArrow
                  secondArrow

But the
    BlockClosure>>into: >>pair: >>binOp: >>at1Do >>at2Do >>swap   etc
    and

the
Object>>into: >>pair: >>binOp: >>at1Do >>at2Do >>swap etc
work too
maybe quicker
or
(anArrow asValuable) <---[which gets rid of all the explanation bits]
<---[couldn't the Smalltalk compiler optimize this
functional as it sits]

Haskell is very big on Immutablility
But in Smalltalk exscpecially in the GUI making everything immutable
just doesn't seem right
because you've got all these Objects which are sitting in this web
and they are taking inputs from god knows everywhere
and broadcasting them back out again
so you have the idea of a web of important Objects
with not so important contents which are coming and going
and recording them all all that and saving them all
for posterity just don't seem right
dependent Objects that are interested can note the changes

In Smalltalk when i want an immutable copy of an
(Object o) where o might be changing i just get
(o copy) or (o deepCopy) to take a snapshot of o
but then you have to keep track of when to take a snapshot
it's true

But Smalltalk can have a
Immutable Class
maybe a subClass of ProtoObject?
to make mixins Object Class complete?
where an (Immutable m) is on another (Object o)
Immutable
hasA object <---[object = o]
and m forwards all Messages to o after
making a copy of itself m and o and returning the copy of itself m unless
((o aMessage) ~= o) in which case
((o aMessage) asImmutable) is returned
then you don't have to keep track of when to copy m
but there could be any amount of copying
going on
so hopefully it's not too much

Maybe Object>>immutableCopy might be good which
could be called if it is defined? Maybe not Probably not

so hopefully the programmer can tell when
copy is Immutable enough and

when an Immutable wrapper is ok good