Arrows and Computations
2. Arrow notation

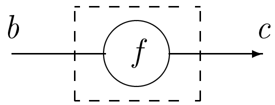
Ross Paterson
City University, London

16th November 2001

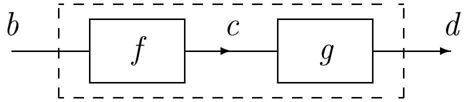
The arrow class

class $Arrow\ \mathsf{a}$ **where**

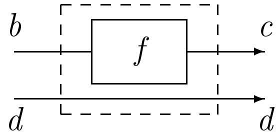
$\mathsf{pure} :: (\mathsf{b} \to \mathsf{c}) \to \mathsf{a\ b\ c}$



$(\ggg) :: \mathsf{a\ b\ c} \to \mathsf{a\ c\ d} \to \mathsf{a\ b\ d}$



$\mathsf{first} :: \mathsf{a\ b\ c} \to \mathsf{a\ (b, d)\ (c, d)}$



Point-free programming

Arrows present a usefully general interface to computation, but they force a point-free style:

$\mathsf{fetch} :: \mathsf{StateTrans\ s\ ()\ s}$
$\mathsf{fetch} = \mathsf{ST}\ (\lambda(\mathsf{s}, \_) \to (\mathsf{s}, \mathsf{s}))$

$\mathsf{store} :: \mathsf{StateTrans\ s\ s\ ()}$
$\mathsf{store} = \mathsf{ST}\ (\lambda(\_, \mathsf{s}) \to (\mathsf{s}, ()))$

$\mathsf{genSym} :: \mathsf{StateTrans\ Int\ ()\ Int}$
$\mathsf{genSym} = \mathsf{fetch} \ggg \mathsf{pure}\ (\lambda \mathsf{n} \to (\mathsf{n}+1, \mathsf{n})) \ggg$
$\qquad \mathsf{first\ store} \ggg \mathsf{pure\ snd}$

Solution: a term notation for arrows.

A special case

We could bind variables with λ if the arrow satisfies

$$\mathsf{a\ (b, c)\ d} \quad \cong \quad \mathsf{b} \to \mathsf{a\ c\ d}$$

Some (but not all) arrows have this isomorphism:

newtype $\mathsf{StateTrans\ s\ b\ c} = \mathsf{ST}\ ((\mathsf{s}, \mathsf{b}) \to (\mathsf{s}, \mathsf{c}))$
newtype $Writer\ \mathsf{b\ c} = W\ (\mathsf{b} \to (\mathsf{String}, \mathsf{c}))$
newtype $\mathsf{CPS\ r\ b\ c} = \mathsf{CPS}\ ((\mathsf{c} \to \mathsf{r}) \to (\mathsf{b} \to \mathsf{r}))$

Arrows with this isomorphism can be factored as $\mathsf{b} \to \mathsf{m\ c}$:

$$\mathsf{a\ b\ d} \quad \cong \quad \mathsf{a\ (b, ())\ d} \quad \cong \quad \mathsf{b} \to \mathsf{a\ ()\ d}$$

(m is a monad and a is its Kliesli arrow.)

An apply arrow

If an arrow satisfies

$$\mathsf{a}\ (\mathsf{b},\mathsf{c})\ \mathsf{d} \quad \cong \quad \mathsf{b} \to \mathsf{a}\ \mathsf{c}\ \mathsf{d}$$

then applying this isomorphism to $\mathsf{id} :: \mathsf{a}\ \mathsf{c}\ \mathsf{d} \to \mathsf{a}\ \mathsf{c}\ \mathsf{d}$ yields a combinator

$$\begin{array}{l} \textbf{class}\ \mathit{Arrow}\ \mathsf{a} \Rightarrow \mathit{ArrowApply}\ \mathsf{a}\ \textbf{where} \\ \qquad \mathsf{app} :: \mathsf{a}\ (\mathsf{a}\ \mathsf{c}\ \mathsf{d}, \mathsf{c})\ \mathsf{d} \end{array}$$

which (with appropriate axioms) implies the isomorphism.

Some arrows (e.g. $\mathsf{StateTrans}$, $\mathsf{Backtrack}$, Writer, CPS) are instances. Others (e.g. $\mathsf{StreamProc}$, Count, $\mathsf{MapTrans}$, Auto, Hyper) are not.

New syntax: arrow expressions

$$\boxed{\begin{array}{lcl} \mathit{exp} & = & \ldots \\ & | & \textbf{proc}\ \mathit{pat} \to \textbf{do}\ \{\ \mathit{stmt}; \ldots; \mathit{stmt}; \mathit{exp} \prec \mathit{exp}\ \} \\ \mathit{stmt} & = & \mathit{pat} \leftarrow \mathit{exp} \prec \mathit{exp} \\ & | & \mathit{exp} \prec \mathit{exp} \\ & | & \textbf{rec}\ \{\ \mathit{stmt}; \ldots; \mathit{stmt}\ \} \end{array}}$$

For example:

$$\begin{array}{l} \mathsf{genSym} :: \mathsf{StateTrans\ Int}\ ()\ \mathsf{Int} \\ \mathsf{genSym} = \textbf{proc}\ () \to \textbf{do} \\ \qquad\qquad \mathsf{n} \leftarrow \mathsf{fetch} \prec () \\ \qquad\qquad \mathsf{store} \prec \mathsf{n} + 1 \\ \qquad\qquad \mathsf{returnA} \prec \mathsf{n} \end{array}$$

Arrow application

$$\boxed{\textbf{proc}\ p \to \textbf{do}\ \{\ f \prec a\ \} \triangleq \begin{cases} \mathsf{pure}\ (\lambda\ p \to a) \ggg f \\ \qquad\qquad \text{if } FV(p) \cap FV(f) = \varnothing \\ \mathsf{pure}\ (\lambda\ p \to (f, a)) \ggg \mathsf{app} \quad \text{otherwise} \end{cases}}$$

An example arrow:

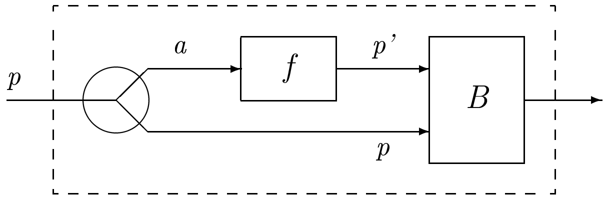
$$\begin{array}{l} \mathsf{returnA} :: \mathit{Arrow}\ \mathsf{a} \Rightarrow \mathsf{a}\ \mathsf{b}\ \mathsf{b} \\ \mathsf{returnA} = \mathsf{pure\ id} \end{array}$$

So that

$$\textbf{proc}\ p \to \textbf{do}\ \mathsf{returnA} \prec e \quad = \quad \mathsf{pure}\ (\lambda\ p \to e)$$

Sequencing and binding

$$\boxed{\begin{array}{rl} \textbf{proc}\ p \to \textbf{do}\ \{\ p' \leftarrow f \prec a; B\ \} \triangleq & \mathsf{pure}\ (\lambda\ p \to (a, p)) \ggg \\ & \mathsf{first}\ f \ggg \\ & \textbf{proc}\ (p', p) \to \textbf{do}\ \{\ B\ \} \end{array}}$$



A special case:

$$\boxed{\textbf{proc}\ p \to \textbf{do}\ \{\ f \prec a; B\ \} \triangleq \textbf{proc}\ p \to \textbf{do}\ \{\ \_ \leftarrow f \prec a; B\ \}}$$

Example: genSym

$$
\begin{array}{rl}
 & \textbf{proc}\ () \to \textbf{do}\ \mathsf{n} \leftarrow \mathsf{fetch} \prec ()\\
 & \qquad\qquad\quad \mathsf{store} \prec \mathsf{n}+1\\
 & \qquad\qquad\quad \mathsf{returnA} \prec \mathsf{n}\\
= & \mathsf{pure}\ (\lambda() \to ((),())) \ggg \mathsf{first\ fetch} \ggg\\
 & \textbf{proc}\ ((),\mathsf{n}) \to \textbf{do}\ \mathsf{store} \prec \mathsf{n}+1\\
 & \qquad\qquad\qquad\quad \mathsf{returnA} \prec \mathsf{n}\\
= & \mathsf{pure}\ (\lambda() \to ((),())) \ggg \mathsf{first\ fetch} \ggg\\
 & \mathsf{pure}\ (\lambda((),\mathsf{n}) \to (\mathsf{n}+1,((),\mathsf{n}))) \ggg \mathsf{first\ store} \ggg\\
 & \textbf{proc}\ ((),((),\mathsf{n})) \to \textbf{do}\ \mathsf{returnA} \prec \mathsf{n}\\
= & \mathsf{pure}\ (\lambda() \to ((),())) \ggg \mathsf{first\ fetch} \ggg\\
 & \mathsf{pure}\ (\lambda((),\mathsf{n}) \to (\mathsf{n}+1,((),\mathsf{n}))) \ggg \mathsf{first\ store} \ggg\\
 & \mathsf{pure}\ (\lambda((),((),\mathsf{n})) \to \mathsf{n}) \ggg \mathsf{returnA}
\end{array}
$$

Example: homogeneous functions

type $\mathsf{Pair\ a} = (\mathsf{a},\mathsf{a})$
data $\mathsf{Hom\ b\ c} = (\mathsf{b} \to \mathsf{c})\ {:}\&{:}\ \mathsf{Hom\ (Pair\ b)\ (Pair\ c)}$

That is, sequences of functions:

$$\mathsf{Hom\ b\ c} \;\cong\; \prod_{n=0}^{\infty} \mathsf{Pair}^n\ \mathsf{b} \to \mathsf{Pair}^n\ \mathsf{c}$$

Arrow operations in terms of sequences:

$$
\begin{array}{rcl}
(\mathsf{pure\ f})_n & = & \mathsf{Pair}^n\ (\mathsf{f} :: \mathsf{b} \to \mathsf{c})\\
(\mathsf{f} \ggg \mathsf{g})_n & = & \mathsf{Pair}^n\ \mathsf{b} \xrightarrow{\mathsf{f}_n} \mathsf{Pair}^n\ \mathsf{c} \xrightarrow{\mathsf{g}_n} \mathsf{Pair}^n\ \mathsf{d}
\end{array}
$$

$$
(\mathsf{first\ f})_n \quad = \quad
\begin{tikzcd}
 & \mathsf{Pair}^n\ \mathsf{b} \arrow[r, "\mathsf{f}_n"] & \mathsf{Pair}^n\ \mathsf{c} \arrow[dr] & \\
\mathsf{Pair}^n\ (\mathsf{b},\mathsf{d}) \arrow[ur] \arrow[dr] & & & \mathsf{Pair}^n\ (\mathsf{c},\mathsf{d})\\
 & \mathsf{Pair}^n\ \mathsf{d} \arrow[r, equal] & \mathsf{Pair}^n\ \mathsf{d} \arrow[ur] &
\end{tikzcd}
$$

The arrow instance

type $\mathsf{Pair\ a} = (\mathsf{a},\mathsf{a})$
data $\mathsf{Hom\ b\ c} = (\mathsf{b} \to \mathsf{c})\ {:}\&{:}\ \mathsf{Hom\ (Pair\ b)\ (Pair\ c)}$

instance Arrow Hom **where**

$$
\begin{array}{l}
\quad \mathsf{pure\ f} = \mathsf{f}\ {:}\&{:}\ \mathsf{pure}\ (\mathsf{f} \times \mathsf{f})\\
\quad (\mathsf{f}\ {:}\&{:}\ \mathsf{fs}) \ggg (\mathsf{g}\ {:}\&{:}\ \mathsf{gs}) = (\mathsf{g} \cdot \mathsf{f})\ {:}\&{:}(\mathsf{fs} \ggg \mathsf{gs})\\
\quad \mathsf{first}\ (\mathsf{f}\ {:}\&{:}\ \mathsf{fs}) = (\mathsf{f} \times \mathsf{id})\ {:}\&{:}\\
\qquad (\mathsf{pure\ transpose} \ggg \mathsf{first\ fs} \ggg \mathsf{pure\ transpose})
\end{array}
$$

$\mathsf{transpose} :: ((\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d})) \to ((\mathsf{a},\mathsf{c}),(\mathsf{b},\mathsf{d}))$
$\mathsf{transpose}\ ((\mathsf{a},\mathsf{b}),(\mathsf{c},\mathsf{d})) = ((\mathsf{a},\mathsf{c}),(\mathsf{b},\mathsf{d}))$

The scan operation

type $\mathsf{Pair\ a} = (\mathsf{a},\mathsf{a})$
data $\mathsf{Hom\ b\ c} = (\mathsf{b} \to \mathsf{c})\ {:}\&{:}\ \mathsf{Hom\ (Pair\ b)\ (Pair\ c)}$

For an associative operation $\oplus$ with identity u,

$$
\begin{array}{l}
\mathsf{scan}\ (\oplus)\ \mathsf{u} : ((x_1,x_2),(x_3,x_4)) \mapsto\\
\qquad\qquad ((x_1, x_1 \oplus x_2),(x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_2 \oplus x_3 \oplus x_4))
\end{array}
$$

Recursive pattern:

$$
\begin{array}{l}
\mathsf{scan} :: (\mathsf{b} \to \mathsf{b} \to \mathsf{b}) \to \mathsf{b} \to \mathsf{Hom\ b\ b}\\
\mathsf{scan}\ (\oplus)\ \mathsf{u} = \mathsf{id}\ {:}\&{:}\ \textbf{proc}\ (\mathsf{o},\mathsf{e}) \to \textbf{do}\\
\qquad\qquad\qquad ???\mathsf{scan}\ (\oplus)\ \mathsf{u}???
\end{array}
$$

Ladner-Fischer parallel scan

$$\begin{array}{l}
\textbf{type}\ \mathit{Pair}\ a = (a, a)\\
\textbf{data}\ \mathit{Hom}\ b\ c = (b \to c) :\&: \mathit{Hom}\ (\mathit{Pair}\ b)\ (\mathit{Pair}\ c)
\end{array}$$

$$\begin{array}{l}
\mathit{scan} :: (b \to b \to b) \to b \to \mathit{Hom}\ b\ b\\
\mathit{scan}\ (\oplus)\ u = \mathit{id} :\&: \textbf{proc}\ (o, e) \to \textbf{do}\\
\qquad\qquad\qquad\qquad e' \leftarrow \mathit{scan}\ (\oplus)\ u \prec o \oplus e\\
\qquad\qquad\qquad\qquad el \leftarrow \mathit{rsh}\ u \prec e'\\
\qquad\qquad\qquad\qquad \mathit{returnA} \prec (el \oplus o, e')
\end{array}$$

$$\begin{array}{l}
\mathit{rsh} :: b \to \mathit{Hom}\ b\ b\\
\mathit{rsh}\ v = \mathit{const}\ v :\&: \textbf{proc}\ (o, e) \to \textbf{do}\\
\qquad\qquad\qquad\qquad o' \leftarrow \mathit{rsh}\ v \prec e\\
\qquad\qquad\qquad\qquad \mathit{returnA} \prec (o', o)
\end{array}$$

Butterflies

$$\begin{array}{l}
\mathit{butterfly} :: (\mathit{Pair}\ b \to \mathit{Pair}\ b) \to \mathit{Hom}\ b\ b\\
\mathit{butterfly}\ f = \mathit{id} :\&: \textbf{proc}\ (o, e) \to \textbf{do}\\
\qquad\qquad\qquad\qquad o' \leftarrow \mathit{butterfly}\ f \prec o\\
\qquad\qquad\qquad\qquad e' \leftarrow \mathit{butterfly}\ f \prec e\\
\qquad\qquad\qquad\qquad \mathit{returnA} \prec f\ (o', e')
\end{array}$$

$$\begin{array}{l}
\mathit{rev} :: \mathit{Hom}\ b\ b\\
\mathit{rev} = \mathit{butterfly}\ \mathit{swap}
\end{array}$$

$$\begin{array}{l}
\mathit{bisort} :: \mathit{Ord}\ b \Rightarrow \mathit{Hom}\ b\ b\\
\mathit{bisort} = \mathit{butterfly}\ \mathit{cmp}\\
\qquad \textbf{where}\ \mathit{cmp}\ (x, y) = (\mathit{min}\ x\ y, \mathit{max}\ x\ y)
\end{array}$$

Exercises

Given the following definitions,

$$\textbf{newtype}\ \mathit{StateFunctor}\ s\ a\ b\ c = \mathsf{ST}\ (a\ (s, b)\ (s, c))$$

$$\textbf{newtype}\ \mathit{AutoFunctor}\ a\ b\ c = \mathsf{F}\ (a\ b\ (c, \mathit{AutoFunctor}\ a\ b\ c))$$

write the following instances, using arrow notation where it helps:

$$\textbf{instance}\ \mathit{Arrow}\ a \Rightarrow \mathit{Arrow}\ (\mathit{StateFunctor}\ s\ a)$$

$$\textbf{instance}\ \mathit{Arrow}\ a \Rightarrow \mathit{Arrow}\ (\mathit{AutoFunctor}\ a)$$

Arrow transformers

$$\textbf{newtype}\ \mathit{StateFunctor}\ s\ a\ b\ c = \mathsf{ST}\ (a\ (s, b)\ (s, c))$$

$$\begin{array}{l}
\textbf{instance}\ \mathit{Arrow}\ a \Rightarrow \mathit{Arrow}\ (\mathit{StateFunctor}\ s\ a)\ \textbf{where}\\
\qquad \mathit{pure}\ f = \mathsf{ST}\ (\textbf{proc}\ (s, b) \to \textbf{do}\\
\qquad\qquad\qquad\qquad \mathit{returnA} \prec (s, f\ b))\\
\qquad \mathsf{ST}\ f \ggg \mathsf{ST}\ g = \mathsf{ST}\ (f \ggg g)\\
\qquad \mathit{first}\ (\mathsf{ST}\ f) = \mathsf{ST}\ (\textbf{proc}\ (s, (b, d)) \to \textbf{do}\\
\qquad\qquad\qquad\qquad (s', c) \leftarrow f \prec (s, b)\\
\qquad\qquad\qquad\qquad \mathit{returnA} \prec (s', (c, d)))
\end{array}$$

$$\textbf{type}\ \mathit{StateTrans}\ s = \mathit{StateFunctor}\ s\ (\to)$$

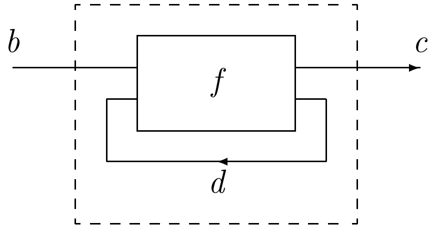
Arrow transformers

```haskell
newtype AutoFunctor a b c = A (a b (c, AutoFunctor a b c))

instance Arrow a => Arrow (AutoFunctor a) where
    pure f = A (proc b -> do
                    returnA -< (f b, pure f))
    A f >>> A g = A (proc b -> do
                    (c, f') <- f -< b
                    (d, g') <- g -< c
                    returnA -< (d, f' >>> g'))
    first (A f) = A (proc (b, d) -> do
                    (c, f') <- f -< b
                    returnA -< ((c, d), first f'))

type Auto = AutoFunctor (->)
```


Recursion: A feedback operator

Many (but not all) arrows have an operator

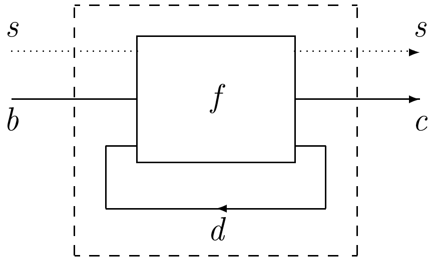
```haskell
class Arrow a => ArrowLoop a where
    loop :: a (b, d) (c, d) -> a b c
```



Generalizes traces (Joyal, Street and Verity, 1996) and recursive monads (Erkök and Launchbury, 2000).

State transformer instance

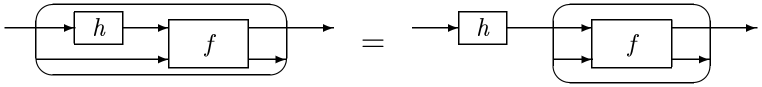
```haskell
newtype StateTrans s b c = ST ((s, b) -> (s, c))
```



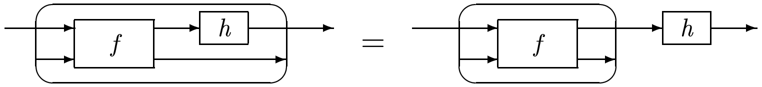

```haskell
instance ArrowLoop (StateTrans s) where
    loop (ST f) = ST (\~(s, b) -> let (s', (c, d)) = f (s, (b, d))
                                  in (s', c))
```


Loop axioms

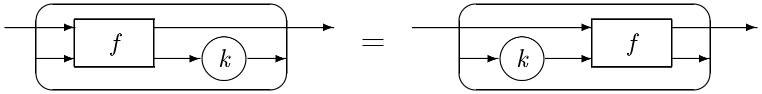
Left tightening: `loop (first h >>> f) = h >>> loop f`



Right tightening: `loop (f >>> first h) = loop f >>> h`



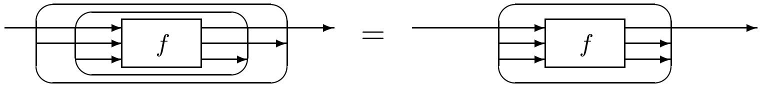
Sliding: `loop (f >>> pure (k × id)) = loop (pure (k × id) >>> f)`



Loop axioms

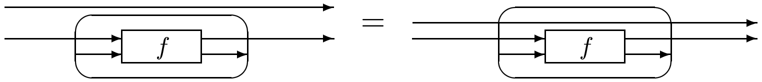
Vanishing:

$\mathsf{loop}\ (\mathsf{loop}\ \mathsf{f}) = \mathsf{loop}\ (\mathsf{pure}\ \mathsf{assoc}^{-1} \ggg \mathsf{f} \ggg \mathsf{pure}\ \mathsf{assoc})$



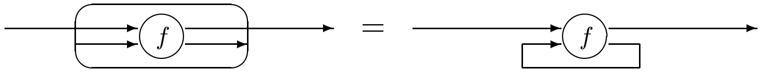
Superposing: $\mathsf{second}\ (\mathsf{loop}\ \mathsf{f}) =$

$\mathsf{loop}\ (\mathsf{pure}\ \mathsf{assoc} \ggg \mathsf{second}\ \mathsf{f} \ggg \mathsf{pure}\ \mathsf{assoc}^{-1})$



Extension:

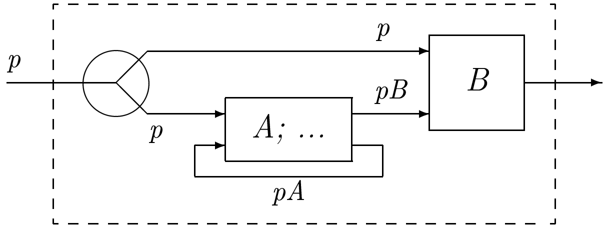
$\mathsf{loop}\ (\mathsf{pure}\ \mathsf{f}) = \mathsf{pure}\ (\lambda \mathsf{b} \to \textbf{let}\ (\mathsf{c}, \mathsf{d}) = \mathsf{f}\ (\mathsf{b}, \mathsf{d})\ \textbf{in}\ \mathsf{c})$



Recursive definitions

Translation rule

$$\boxed{\begin{array}{l} \textbf{proc}\ p \to \textbf{do}\ \{\ \textbf{rec}\ \{\ A\ \}; B\} \triangleq \\ \qquad \mathsf{returnA}\ \&\&\&\ \mathsf{loop}\ (\textbf{proc}\ (p, p_A) \to \textbf{do} \\ \qquad\qquad \{\ A; \mathsf{returnA} \prec (p_B, p_A)\ \})\ \ggg \\ \qquad \textbf{proc}(p, p_B) \to \textbf{do}\ \{\ B\ \} \end{array}}$$



More instances

$\textbf{newtype}\ \mathsf{StreamProc}\ \mathsf{b}\ \mathsf{c} = \mathsf{SP}\ ([\mathsf{b}] \to [\mathsf{c}])$

$\textbf{instance}\ \mathsf{ArrowLoop}\ \mathsf{StreamProc}\ \textbf{where}$
$\quad \mathsf{loop}\ (\mathsf{SP}\ \mathsf{f}) = \mathsf{SP}\ (\lambda \mathsf{bs} \to$
$\qquad\qquad \textbf{let}\ \mathsf{cds} = \mathsf{f}\ (\mathsf{zip}\ \mathsf{bs}\ (\mathsf{map}\ \mathsf{snd}\ \mathsf{cds}))$
$\qquad\qquad \textbf{in}\ \mathsf{map}\ \mathsf{fst}\ \mathsf{cds})$

$\textbf{newtype}\ \mathsf{Auto}\ \mathsf{b}\ \mathsf{c} = \mathsf{A}\ (\mathsf{b} \to (\mathsf{c}, \mathsf{Auto}\ \mathsf{b}\ \mathsf{c}))$

$\textbf{instance}\ \mathsf{ArrowLoop}\ \mathsf{Auto}\ \textbf{where}$
$\quad \mathsf{loop}\ (\mathsf{A}\ \mathsf{f}) = \mathsf{A}\ (\lambda \mathsf{b} \to \textbf{let}\ ((\mathsf{c}, \mathsf{d}), \mathsf{f}') = \mathsf{f}\ (\mathsf{b}, \mathsf{d})$
$\qquad\qquad\qquad \textbf{in}\ (\mathsf{c}, \mathsf{loop}\ \mathsf{f}'))$

A class for synchronous circuits

$\textbf{class}\ \mathsf{ArrowLoop}\ a \Rightarrow \mathsf{ArrowCircuit}\ a\ \textbf{where}$
$\quad \mathsf{delay} :: \mathsf{b} \to a\ \mathsf{b}\ \mathsf{b}$

Stream processor instance:

$\textbf{newtype}\ \mathsf{StreamProc}\ \mathsf{b}\ \mathsf{c} = \mathsf{SP}\ ([\mathsf{b}] \to [\mathsf{c}])$

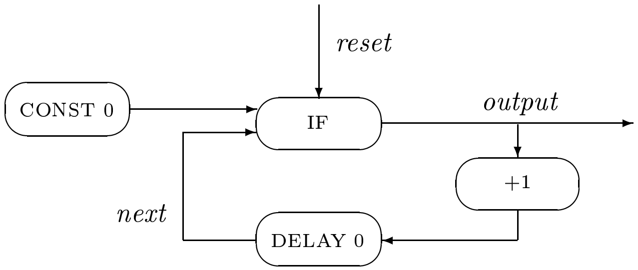
$\textbf{instance}\ \mathsf{ArrowCircuit}\ \mathsf{StreamProc}\ \textbf{where}$
$\quad \mathsf{delay}\ \mathsf{b} = \mathsf{SP}\ (\mathsf{b}{:})$

Simple automaton instance:

$\textbf{newtype}\ \mathsf{Auto}\ \mathsf{b}\ \mathsf{c} = \mathsf{A}\ (\mathsf{b} \to (\mathsf{c}, \mathsf{Auto}\ \mathsf{b}\ \mathsf{c}))$

$\textbf{instance}\ \mathsf{ArrowCircuit}\ \mathsf{Auto}\ \textbf{where}$
$\quad \mathsf{delay}\ \mathsf{b} = \mathsf{A}\ (\lambda \mathsf{b}' \to (\mathsf{b}, \mathsf{delay}\ \mathsf{b}'))$

Example: a resettable counter



$\mathsf{counter} :: \mathsf{ArrowCircuit}\ a \Rightarrow a\ \mathsf{Bool}\ \mathsf{Int}$

$\mathsf{counter} = \textbf{proc}\ \mathsf{reset} \rightarrow \textbf{do}$

$\quad \textbf{rec}\ \mathsf{output} \leftarrow \mathsf{returnA} \prec \textbf{if}\ \mathsf{reset}\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{next}$

$\qquad \mathsf{next} \leftarrow \mathsf{delay}\ 0 \prec \mathsf{output} + 1$

$\quad \mathsf{returnA} \prec \mathsf{output}$

Embedded languages

- A library providing an *Arrow* instance plus special constants defines an embedded mini-language. e.g. StateTrans (fetch and store), *Writer* (*write*).
- e.g. Arrow FRP (Yale, MSR) is notionally $(\mathsf{Time} \rightarrow b) \rightarrow (\mathsf{Time} \rightarrow c)$ and includes

$$\mathsf{time} :: a\ b\ \mathsf{Time}$$
$$\mathsf{integral} :: a\ \mathsf{Double}\ \mathsf{Double}$$

- Sometimes we need operations to build new arrows from existing ones (e.g. or).
- These may pass additional data to the component arrows (e.g. handle).
- Next week: a more primitive arrow syntax.