Arrows and Computations
3. Control operators

Ross Paterson
City University, London

23rd November 2001

Deterministic parsing

(adapted from Swierstra and Duponcheel)

$$\textbf{newtype}\ \mathit{DynamicParser}\ b\ c = \\ \quad \mathsf{DP}\ (([\mathit{Sym}], b) \to \mathit{Maybe}\ ([\mathit{Sym}], c))$$

$$\textbf{data}\ \mathit{StaticParser}\ b\ c = \\ \quad \mathsf{SP}\ [(\mathit{Sym}, \mathit{DynamicParser}\ b\ c)]\ (\mathit{Maybe}\ (b \to c))$$

$$\mathit{mkParser} :: \mathit{StaticParser}\ b\ c \to \mathit{DynamicParser}\ b\ c$$

Parsers are arrows. Also assume primitives

$$\mathit{symbol} :: \mathit{Sym} \to \mathit{StaticParser}\ ()\ ()$$
$$\mathit{ident} :: \mathit{StaticParser}\ ()\ \mathit{String}$$
$$\mathit{number} :: \mathit{StaticParser}\ ()\ \mathit{Int}$$

Part of a simple parser

$$\textbf{type}\ \mathit{Bindings} = [(\mathit{String}, \mathit{Int})]$$

$$\begin{aligned} &\mathit{letExpr} :: \mathit{StaticParser}\ \mathit{Bindings}\ \mathit{Int} \\ &\mathit{letExpr} = \textbf{proc}\ \mathit{binds} \to \textbf{do} \\ &\qquad \mathit{symbol}\ \mathsf{LET} \prec ()\\ &\qquad n \leftarrow \mathit{ident} \prec ()\\ &\qquad \mathit{symbol}\ \mathsf{EQUALS} \prec ()\\ &\qquad v \leftarrow \mathit{expr} \prec \mathit{binds}\\ &\qquad \mathit{symbol}\ \mathsf{IN} \prec ()\\ &\qquad \mathit{expr} \prec (n, v) : \mathit{binds} \end{aligned}$$

$$\mathit{expr} :: \mathit{StaticParser}\ \mathit{Bindings}\ \mathit{Int}$$

Alternatives

Failure and alternatives are handed with the classes

$$\textbf{class}\ \mathit{Arrow}\ a \Rightarrow \mathit{ArrowZero}\ a\ \textbf{where} \\ \quad \mathit{zeroArrow} :: a\ b\ c$$

$$\textbf{class}\ \mathit{ArrowZero}\ a \Rightarrow \mathit{ArrowPlus}\ a\ \textbf{where} \\ \quad (\mathbin{<\!+\!>}) :: a\ b\ c \to a\ b\ c \to a\ b\ c$$

$\mathit{DynamicParser}$ and $\mathit{StaticParser}$ are instances.

$$\textbf{newtype}\ \mathit{DynamicParser}\ b\ c = \\ \quad \mathsf{DP}\ (([\mathit{Sym}], b) \to \mathit{Maybe}\ ([\mathit{Sym}], c))$$

$$\textbf{data}\ \mathit{StaticParser}\ b\ c = \\ \quad \mathsf{SP}\ [(\mathit{Sym}, \mathit{DynamicParser}\ b\ c)]\ (\mathit{Maybe}\ (b \to c))$$

Refactored arrow expression syntax

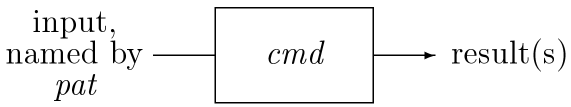
$$
\begin{array}{lcll}
\mathit{exp} & = & \ldots & \\
& | & \textbf{proc}\ \mathit{pat} \rightarrow \mathit{cmd} & \text{arrow expression} \\
\mathit{cmd} & = & \mathit{exp} \prec \mathit{exp} & \text{arrow application} \\
& | & \cdots & \\
& | & \textbf{do}\ \{\ \mathit{stmt}; \ldots; \mathit{stmt}; \mathit{cmd}\ \} & \\
\mathit{stmt} & = & \mathit{pat} \leftarrow \mathit{cmd} & \\
& | & \mathit{cmd} & \text{same as } \_ \leftarrow \mathit{cmd} \\
& | & \textbf{rec}\ \{\ \mathit{stmt}; \ldots; \mathit{stmt}\ \} &
\end{array}
$$

Commands

New syntactic category:

$$
\begin{array}{lcll}
\mathit{cmd} & = & \mathit{exp} \prec \mathit{exp} & \text{arrow application} \\
& | & \cdots & \\
& | & \textbf{do}\ \{\ \mathit{stmt}; \ldots; \mathit{stmt}; \mathit{cmd}\ \} &
\end{array}
$$

Just as an expression denotes a function from environments, a command denotes an arrow



Refactored translation rules

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

$$
\begin{array}{rcl}
\textbf{proc}\ p \rightarrow \textbf{do}\ \{\ p' \leftarrow c; B\ \} & \triangleq & \text{pure id} \,\&\&\&\, (\textbf{proc}\ p \rightarrow c) \ggg \\
& & (\textbf{proc}\ (p, p') \rightarrow \textbf{do}\ \{\ B\ \}) \\
\textbf{proc}\ p \rightarrow \textbf{do}\ \{\ c; B\ \} & \triangleq & \text{pure id} \,\&\&\&\, (\textbf{proc}\ p \rightarrow c) \ggg \\
& & \text{pure fst} \ggg (\textbf{proc}\ p \rightarrow \textbf{do}\ \{\ B\ \})
\end{array}
$$

Combining commands with control operators

$$
\begin{array}{lcl}
\mathit{cmd} & = & \cdots \\
& | & \mathit{cmd}_1\ \mathit{op}\ \mathit{cmd}_2
\end{array}
$$

Translation rule:

$$
\boxed{\textbf{proc}\ p \rightarrow c_1\ \mathit{op}\ c_2 \ \triangleq\ (\textbf{proc}\ p \rightarrow c_1)\ \mathit{op}\ (\textbf{proc}\ p \rightarrow c_2)}
$$

An example:

$$
\begin{array}{l}
\textbf{proc}\ (x, y) \rightarrow (f \prec x) \lt\!\!+\!\!\gt (g \prec y + 1) \\
= (\textbf{proc}\ (x, y) \rightarrow f \prec x) \lt\!\!+\!\!\gt (\textbf{proc}\ (x, y) \rightarrow g \prec y + 1) \\
= (\text{pure}\ (\lambda(x, y) \rightarrow x) \ggg f) \lt\!\!+\!\!\gt (\text{pure}\ (\lambda(x, y) \rightarrow y + 1) \ggg g)
\end{array}
$$

Part of an expression parser

$$
\begin{array}{l}
\mathsf{expr} :: \mathsf{StaticParser\ Bindings\ Int} \\
\mathsf{expr} = \textbf{proc}\ \mathsf{binds} \rightarrow \textbf{do} \\
\qquad\qquad\qquad \mathsf{l} \leftarrow \mathsf{term} \prec \mathsf{binds} \\
\qquad\qquad\qquad \mathsf{returnA} \prec \mathsf{l} \\
\qquad\qquad\qquad\quad \lt\!\!+\!\!\gt\ \textbf{do} \\
\qquad\qquad\qquad\qquad\qquad \mathsf{symbol\ PLUS} \prec () \\
\qquad\qquad\qquad\qquad\qquad \mathsf{r} \leftarrow \mathsf{expr} \prec \mathsf{binds} \\
\qquad\qquad\qquad\qquad\qquad \mathsf{returnA} \prec \mathsf{l} + \mathsf{r} \\
\qquad\qquad\qquad\quad \lt\!\!+\!\!\gt\ \textbf{do} \\
\qquad\qquad\qquad\qquad\qquad \mathsf{symbol\ MINUS} \prec () \\
\qquad\qquad\qquad\qquad\qquad \mathsf{r} \leftarrow \mathsf{expr} \prec \mathsf{binds} \\
\qquad\qquad\qquad\qquad\qquad \mathsf{returnA} \prec \mathsf{l} - \mathsf{r} \\
\\
\mathsf{term} :: \mathsf{StaticParser\ Bindings\ Int} \\
\mathsf{term} = \ldots
\end{array}
$$

Naturality of control operators

We require

$$op :: \ldots \Rightarrow \forall\, \mathsf{e}.\ \mathsf{a}_1\ \mathsf{e}\ t_1 \rightarrow \mathsf{a}_2\ \mathsf{e}\ t_2 \rightarrow \mathsf{a}\ \mathsf{e}\ t$$

satisfying a corresponding naturality property

$$(\mathsf{pure\ k} \ggg \mathsf{f})\ op\ (\mathsf{pure\ k} \ggg \mathsf{g}) \quad = \quad \mathsf{pure\ k} \ggg (\mathsf{f}\ op\ \mathsf{g})$$

Note that if $\mathsf{a\ b\ c} \cong \mathsf{b} \rightarrow \mathsf{m\ c}$ we have

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall\, \mathsf{e}.\ \mathsf{a\ e}\ t_1 \rightarrow \mathsf{a\ e}\ t_2 \rightarrow \mathsf{a\ e}\ t} \\
\cong & \forall\, \mathsf{e}.\ (\mathsf{e} \rightarrow \mathsf{m}\ t_1) \rightarrow (\mathsf{e} \rightarrow \mathsf{m}\ t_2) \rightarrow (\mathsf{e} \rightarrow \mathsf{m}\ t) \\
\cong & \mathsf{m}\ t_1 \rightarrow \mathsf{m}\ t_2 \rightarrow \mathsf{m}\ t
\end{array}
$$

Control operators of any arity

$$
\begin{array}{rcll}
cmd & = & \cdots & \\
 & | & \textbf{form}\ exp\ cmd\ \cdots\ cmd & \text{composite command} \\
 & | & cmd_1\ op\ cmd_2 & \text{same as } \textbf{form}\ (op)\ cmd_1\ cmd_2
\end{array}
$$

with similar naturality conditions.

Some other control operators:

$$(\&\&\&) :: \mathsf{Arrow\ a} \Rightarrow \mathsf{a\ b\ c} \rightarrow \mathsf{a\ b\ c'} \rightarrow \mathsf{a\ b\ (c, c')}$$

$$\mathsf{zeroArrow} :: \mathsf{ArrowZero\ a} \Rightarrow \mathsf{a\ b\ c}$$

$$\mathsf{lift} :: \mathsf{Arrow\ a} \Rightarrow \mathsf{a\ b\ c} \rightarrow \mathsf{StateTrans\ s\ a\ b\ c}$$

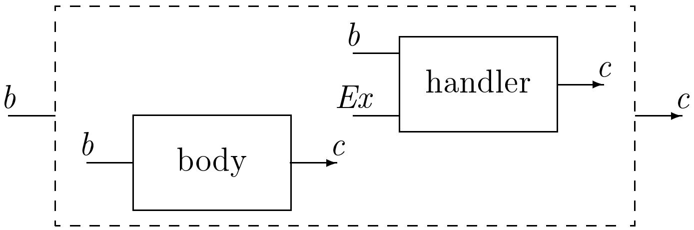
$$\mathsf{mkParser} :: \mathsf{StaticParser\ b\ c} \rightarrow \mathsf{DynamicParser\ b\ c}$$

Parameter passing

A control operator may pass additional inputs to a constituent arrow.

Consider an exception-handling operator:

$$\mathsf{handle} :: \mathsf{ArrowExcept\ a} \Rightarrow \mathsf{a\ b\ c} \rightarrow \mathsf{a\ (b, Ex)\ c} \rightarrow \mathsf{a\ b\ c}$$



Naturality

The control operator

$$\mathsf{handle} :: \mathsf{ArrowExcept}\ \mathsf{a} \Rightarrow \mathsf{a}\ \mathsf{b}\ \mathsf{c} \to \mathsf{a}\ (\mathsf{b}, \mathsf{Ex})\ \mathsf{c} \to \mathsf{a}\ \mathsf{b}\ \mathsf{c}$$

will be required to satisfy

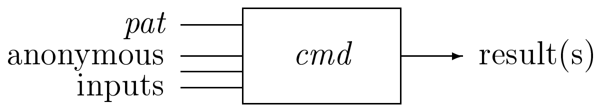
$$\begin{aligned}(\mathsf{pure}\ \mathsf{k} \ggg \mathsf{f})\ \text{`}\mathsf{handle}\text{`}\ (\mathsf{pure}\ (\mathsf{k} \times \mathsf{id}) \ggg \mathsf{g})\ &= \\ \mathsf{pure}\ \mathsf{k} \ggg (\mathsf{f}\ \text{`}\mathsf{handle}\text{`}\ \mathsf{g})&\end{aligned}$$

Note that if $\mathsf{a}\ \mathsf{b}\ \mathsf{c} \cong \mathsf{b} \to \mathsf{m}\ \mathsf{c}$ we have

$$\begin{aligned}&\forall\, \mathsf{b}.\ \mathsf{a}\ \mathsf{b}\ \mathsf{c} \to \mathsf{a}\ (\mathsf{b}, \mathsf{Ex})\ \mathsf{c} \to \mathsf{a}\ \mathsf{b}\ \mathsf{c} \\ \cong\ &\forall\, \mathsf{b}.\ (\mathsf{b} \to \mathsf{m}\ \mathsf{c}) \to ((\mathsf{b}, \mathsf{Ex}) \to \mathsf{m}\ \mathsf{c}) \to (\mathsf{b} \to \mathsf{m}\ \mathsf{c}) \\ \cong\ &\forall\, \mathsf{b}.\ (\mathsf{b} \to \mathsf{m}\ \mathsf{c}) \to (\mathsf{b} \to \mathsf{Ex} \to \mathsf{m}\ \mathsf{c}) \to (\mathsf{b} \to \mathsf{m}\ \mathsf{c}) \\ \cong\ &\mathsf{m}\ \mathsf{c} \to (\mathsf{Ex} \to \mathsf{m}\ \mathsf{c}) \to \mathsf{m}\ \mathsf{c}\end{aligned}$$

Generalizing arrow expressions

To deal with additional inputs, **proc** *pat* $\to$ *cmd* is in general translated to an arrow



New quantifier κ names the first anonymous input.

$$\begin{aligned}\mathit{cmd}\ &=\ \cdots \\ &\mid\ \ \kappa\ \mathit{pat} \to \mathit{cmd} \quad \text{parameter binding}\end{aligned}$$

The κ quantifier

$$\begin{aligned}\mathit{cmd}\ &=\ \cdots \\ &\mid\ \ \kappa\ \mathit{pat} \to \mathit{cmd} \quad \text{parameter binding}\end{aligned}$$

The κ quantifier names the first anonymous input.

$$\boxed{\textbf{proc}\ p \to \kappa\ p' \to c\ \triangleq\ \textbf{proc}\ (p, p') \to c}$$

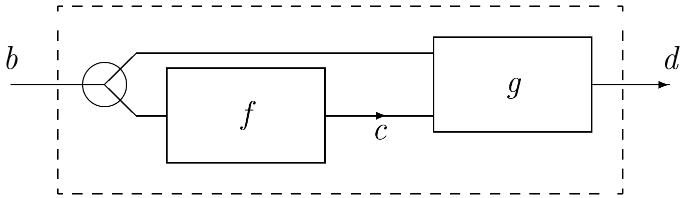
For example,

$$\begin{aligned}&\textbf{proc}\ \mathsf{y} \to (c_1\ \text{`}\mathsf{handle}\text{`}\ \kappa\ \mathsf{ex} \to c_2) \\ &= (\textbf{proc}\ \mathsf{y} \to c_1)\ \text{`}\mathsf{handle}\text{`}\ (\textbf{proc}\ \mathsf{y} \to \kappa\ \mathsf{ex} \to c_2) \\ &= (\textbf{proc}\ \mathsf{y} \to c_1)\ \text{`}\mathsf{handle}\text{`}\ (\textbf{proc}\ (\mathsf{y}, \mathsf{ex}) \to c_2)\end{aligned}$$

and so on.

A sequencing operator

$$\begin{aligned}&\mathsf{bind} :: \mathsf{Arrow}\ \mathsf{a} \Rightarrow \mathsf{a}\ \mathsf{b}\ \mathsf{c} \to \mathsf{a}\ (\mathsf{b}, \mathsf{c})\ \mathsf{d} \to \mathsf{a}\ \mathsf{b}\ \mathsf{d} \\ &\mathsf{u}\ \text{`}\mathsf{bind}\text{`}\ \mathsf{f} = \mathsf{pure}\ \mathsf{id}\ \&\&\&\ \mathsf{u} \ggg \mathsf{f}\end{aligned}$$



A related operator:

$$\begin{aligned}&\mathsf{bind\_} :: \mathsf{Arrow}\ \mathsf{a} \Rightarrow \mathsf{a}\ \mathsf{b}\ \mathsf{c} \to \mathsf{a}\ \mathsf{b}\ \mathsf{d} \to \mathsf{a}\ \mathsf{b}\ \mathsf{d} \\ &\mathsf{u}\ \text{`}\mathsf{bind\_}\text{`}\ \mathsf{v} = \mathsf{u}\ \text{`}\mathsf{bind}\text{`}\ (\mathsf{pure}\ \mathsf{fst} \ggg \mathsf{v})\end{aligned}$$

Command equivalences

We write

$$c \equiv_p c' \quad \text{if} \quad \textbf{proc}\ p \to c = \textbf{proc}\ p \to c'$$
$$c \equiv c' \quad \text{if} \quad c \equiv_p c' \text{ for all legal } p$$

Some laws:

$$(\mathtt{returnA} \prec e)\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x} \to c \quad \equiv \quad [e/x]c$$
$$c\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x} \to \mathtt{returnA} \prec \mathtt{x} \quad \equiv \quad c$$
$$c_1\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x}_1 \to (c_2\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x}_2 \to c_3) \quad \equiv \quad (c_1\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x}_1 \to c_2)\ \text{`}\mathtt{bind}\text{`}\ \kappa\, \mathtt{x}_2 \to c_3$$

Moreover

$$\textbf{do}\ \{\ p \leftarrow c; A\ \} \quad \equiv \quad c\ \text{`}\mathtt{bind}\text{`}\ \kappa\, p \to \textbf{do}\ \{\ A\ \}$$
$$\textbf{do}\ \{\ c; A\ \} \quad \equiv \quad c\ \text{`}\mathtt{bind\_}\text{`}\ \textbf{do}\ \{\ A\ \}$$
$$\textbf{do}\ \{\ c\ \} \quad \equiv \quad c$$

Parameter passing II

Some control operators (e.g. encapsulation operators) produce an arrow that expects additional arguments:

$$\mathtt{runState} :: \mathtt{StateTrans\ s\ a\ b\ c} \to \mathtt{a\ (b, s)\ c}$$

with a similar naturality condition.

A typical use:

$$\mathtt{mkState}\ \text{`}\mathtt{bind}\text{`}\ (\textbf{form}\ \mathtt{runState}\ (\textbf{do}\ \{\ldots\}))$$

using

$$\mathtt{bind} :: \mathtt{Arrow\ a} \Rightarrow \mathtt{a\ b\ c} \to \mathtt{a\ (b, c)\ d} \to \mathtt{a\ b\ d}$$

General form of control operator

A Haskell expression e is a *control operator* if it has a type

$$e :: \ldots \Rightarrow \forall\, \mathtt{b}.\ \mathtt{a}_1\ (\mathtt{b}, \overline{t_1})\ t'_1 \to \cdots\ \mathtt{a}_n\ (\mathtt{b}, \overline{t_n})\ t'_n \to \mathtt{a}\ (\mathtt{b}, \overline{t})\ t'$$

with a corresponding naturality property

$$\begin{aligned} & e\ (\mathtt{first}^{|\overline{t_1}|}\ (\mathtt{pure\ k}) \ggg \mathtt{f\_1})\ \cdots\ (\mathtt{first}^{|\overline{t_n}|}\ (\mathtt{pure\ k}) \ggg \mathtt{f\_n}) \\ = \quad & (\mathtt{first}^{|\overline{t}|}\ (\mathtt{pure\ k}) \ggg e\ \mathtt{f\_1}\ \cdots\ \mathtt{f\_n} \end{aligned}$$

When $\mathtt{a\ b\ c} \cong \mathtt{b} \to \mathtt{m\ c}$ we have

$$\begin{aligned} & \forall\, \mathtt{b}.\ \mathtt{a}\ (\mathtt{b}, \overline{t_1})\ t'_1 \to \cdots\ \mathtt{a}\ (\mathtt{b}, \overline{t_n})\ t'_n \to \mathtt{a}\ (\mathtt{b}, \overline{t})\ t' \\ \cong \quad & \forall\, \mathtt{b}.\ ((\mathtt{b}, \overline{t_1}) \to \mathtt{m}\ t'_1) \to \cdots ((\mathtt{b}, \overline{t_n}) \to \mathtt{m}\ t'_n) \to ((\mathtt{b}, \overline{t}) \to \mathtt{m}\ t') \\ \cong \quad & \forall\, \mathtt{b}.\ (\mathtt{b} \to \overline{t_1} \to \mathtt{m}\ t'_1) \to \cdots (\mathtt{b} \to \overline{t_n} \to \mathtt{m}\ t'_n) \to (\mathtt{b} \to \overline{t} \to \mathtt{m}\ t') \\ \cong \quad & (\overline{t_1} \to \mathtt{m}\ t'_1) \to \cdots (\overline{t_n} \to \mathtt{m}\ t'_n) \to (\overline{t} \to \mathtt{m}\ t') \end{aligned}$$

New syntax: arrow expressions

exp	=	. . .	
	|	**proc** *pat* → *cmd*	arrow expression
cmd	=	*exp* ≺ *exp*	arrow application
	|	**form** *exp cmd* · · · *cmd*	composite command
	|	cmd_1 *op* cmd_2	same as **form** (*op*) cmd_1 cmd_2
	|	κ *pat* → *cmd*	parameter binding
	|	**do** { *stmt*; . . . ; *stmt*; *cmd* }	
stmt	=	*pat* ← *cmd*	
	|	*cmd*	same as _ ← *cmd*
	|	**rec** { *stmt*; . . . ; *stmt* }	

Exercise

Given the definitions

$$\textbf{newtype}\ \mathsf{Fix}\ f = \mathsf{In}\ (f\ (\mathsf{Fix}\ f))$$

$$\begin{aligned}&\mathsf{out} :: \mathsf{Fix}\ f \to f\ (\mathsf{Fix}\ f)\\&\mathsf{out}\ (\mathsf{In}\ x) = x\end{aligned}$$

generalize the fold operator

$$\begin{aligned}&\mathsf{fold} :: \mathsf{Functor}\ f \Rightarrow (f\ c \to c) \to \mathsf{Fix}\ f \to c\\&\mathsf{fold}\ f = f \cdot \mathsf{fmap}\ (\mathsf{fold}\ f) \cdot \mathsf{out}\end{aligned}$$

to a control operator on arrows, assuming

$$\begin{aligned}&\textbf{class}\ (\mathsf{Arrow}\ a, \mathsf{Functor}\ f) \Rightarrow \mathsf{GenFunctor}\ f\ a\ \textbf{where}\\&\qquad \mathsf{mapA} :: a\ b\ c \to a\ (f\ b)\ (f\ c)\end{aligned}$$

Fold for arrows

$$\mathsf{foldA} :: \mathsf{GenFunctor}\ f\ a \Rightarrow a\ (b, f\ c)\ c \to a\ (b, \mathsf{Fix}\ f)\ c$$

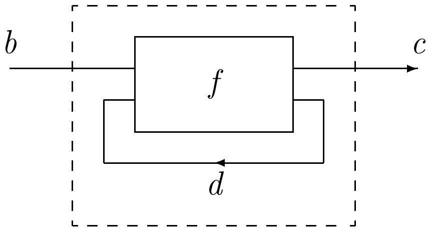
$$\begin{array}{l}(b, \mathsf{Fix}\ f)\\ \quad\downarrow\ \ \mathsf{pure}\ (\mathsf{id} \times \mathsf{out})\\ (b, f\ (\mathsf{Fix}\ f))\\ \quad\downarrow\ \ \mathsf{pure}\ (\lambda(b,t) \to (b, \mathsf{fmap}\ (\mathsf{mkPair}\ b)\ t))\\ (b, f\ (b, \mathsf{Fix}\ f))\\ \quad\downarrow\ \ \mathsf{second}\ (\mathsf{mapA}\ (\mathsf{foldA}\ f))\\ (b, f\ c)\\ \quad\downarrow\ \ f\\ c\end{array}$$

$$\begin{aligned}\mathsf{foldA}\ f = {}& \mathsf{pure}\ (\lambda(b,t) \to (b, \mathsf{fmap}\ (\mathsf{mkPair}\ b)\ (\mathsf{out}\ t))) \ggg\\ & \mathsf{second}\ (\mathsf{mapA}\ (\mathsf{foldA}\ f)) \ggg f\\ & \textbf{where}\ \mathsf{mkPair}\ x\ y = (x, y)\end{aligned}$$

Recursion: A feedback operator

Many (but not all) arrows have an control operator

$$\begin{aligned}&\textbf{class}\ \mathsf{Arrow}\ a \Rightarrow \mathsf{ArrowLoop}\ a\ \textbf{where}\\&\qquad \mathsf{loop} :: a\ (b,d)\ (c,d) \to a\ b\ c\end{aligned}$$

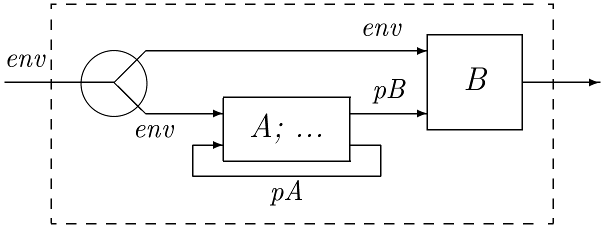


Naturality is implied by the axioms.

Recursive definitions

Translation rule

$$\begin{array}{|l|}\hline \textbf{do}\ \{\ \textbf{rec}\ \{\ A\ \}; B\} \triangleq \textbf{do}\\ \qquad\qquad p_B \leftarrow \mathsf{loop}\ (\kappa\ p_A \to \textbf{do}\ \{\ A; \mathsf{returnA} \prec (p_B, p_A)\ \})\\ \qquad\qquad B\\ \hline\end{array}$$



Defining an embedded language

- an arrow type constructor

  data FRP b c = ...

- an instance of $Arrow$ and possibly related classes

  instance $Arrow$ FRP

  instance $ArrowLoop$ FRP

- primitive arrows

  $time$ :: FRP () $Time$

  $integral$ :: FRP $Double$ $Double$

  $when$ :: FRP $Bool$ ($Event$ ())

- control operators

  $$switch :: \mathsf{FRP}\ b\ c \to \mathsf{FRP}\ b\ (Event\ e) \to \mathsf{FRP}\ (b, e)\ c \to \mathsf{FRP}\ b\ c$$

- axioms (command equivalences)