Algebraic effects and effect handlers

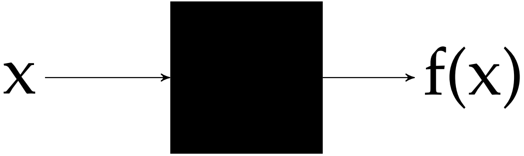
Sam Lindley

Laboratory for Foundations of Computer Science
The University of Edinburgh

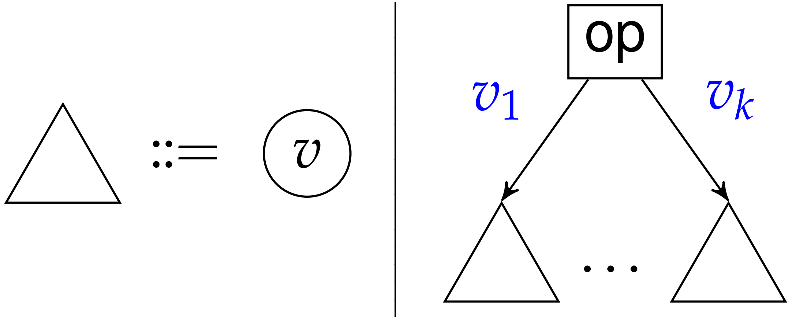
Sam.Lindley@ed.ac.uk

May 16th, 2017

What is a pure computation?



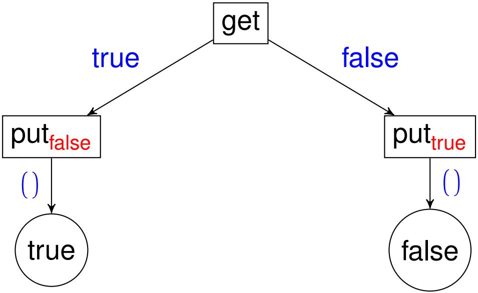
What is an effectful computation?



A *command-response* tree.

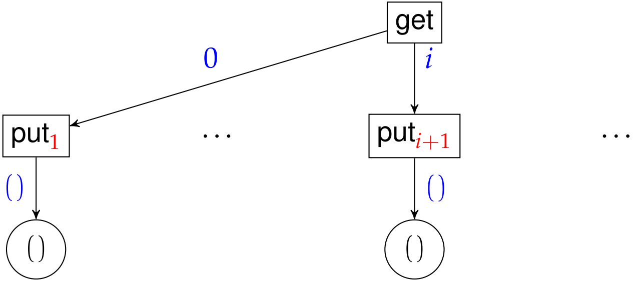
Example: bit toggling

$$\begin{aligned}\text{get} &: \text{bool}\\ \text{put}_{\text{true}} &: 1\\ \text{put}_{\text{false}} &: 1\end{aligned}$$



Example: increment

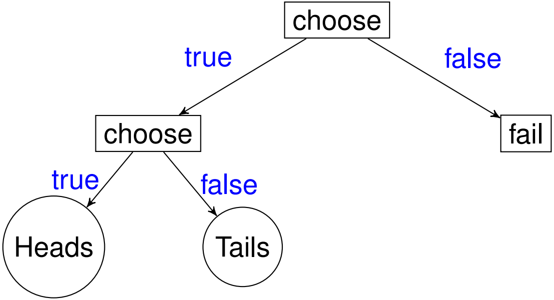
$$\begin{aligned} \mathsf{get} &: \mathbb{N} \\ \mathsf{put}_i &: 1, \quad i \in \mathbb{N} \end{aligned}$$



Example: drunk coin toss

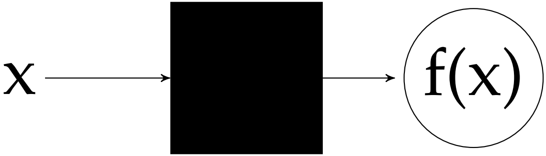
choose : bool

fail : 0

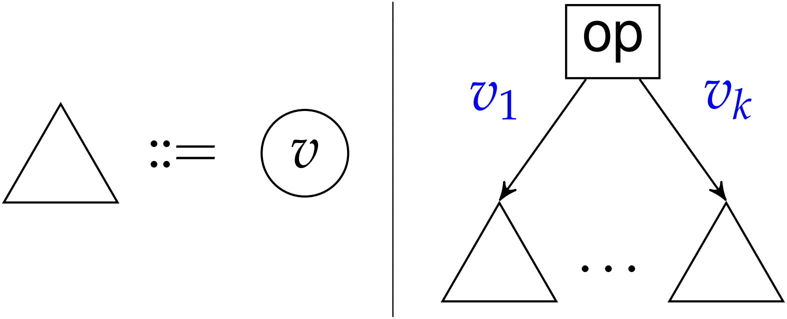


Pure computation of an effectful computation

Special case: pure function as an effectful computation



What is an effectful computation?



Equivalently (ignoring result values):

$$m ::= \textbf{return}\, v \mid \mathsf{op}\ \langle m_1, \ldots m_k \rangle$$

Equivalently (accounting for result values):

$$m ::= \textbf{return}\, v \mid \mathsf{op}\ (\lambda x.\textbf{case}\, x \{v_1 \mapsto m_1; \ldots; v_k \mapsto m_k\})$$

Examples

Boolean state

$$\mathsf{toggle} = \mathsf{get}\,\langle \mathsf{put}_{\mathsf{false}}\,\langle \mathbf{return}\,\mathsf{true}\rangle, \mathsf{put}_{\mathsf{true}}\,\langle \mathbf{return}\,\mathsf{false}\rangle\rangle$$

Natural number state

$$\mathsf{increment} = \mathsf{get}\,\langle \mathsf{put}_{1}\,\langle \mathbf{return}\,()\rangle, \ldots, \mathsf{put}_{i+1}\,\langle \mathbf{return}\,()\rangle, \ldots\rangle$$

Nondeterminism

$$\mathsf{drunkToss} = \mathsf{choose}\,\langle \mathsf{choose}\,\langle \mathbf{return}\,\mathsf{Heads}, \mathbf{return}\,\mathsf{Tails}\rangle, \mathsf{fail}\langle\rangle\rangle$$

Command response trees are free monads

- A computation of type A is a tree whose leaves have type A
- Return is **return**
- Bind perfoms substitution at the leaves

$$\begin{aligned} \mathbf{return}\ v \gg= f &= f\,v \\ \mathsf{op}\ \langle m_1, \ldots, m_n \rangle \gg= f &= \mathsf{op}\ \langle m_1 \gg= f, \ldots, m_n \gg= f \rangle \end{aligned}$$

Algebraic effects

An algebraic effect is given by a *signature* of operations and a collection of *equations*.

Example: boolean state

Signature

$$
\begin{aligned}
\mathsf{get} &: \mathsf{bool} \\
\mathsf{put}_{\mathsf{true}} &: 1 \\
\mathsf{put}_{\mathsf{false}} &: 1
\end{aligned}
$$

Equations

$$
\begin{aligned}
\mathsf{put}_s \langle \mathsf{put}_{s'} \langle m \rangle \rangle &\simeq \mathsf{put}_{s'} \langle m \rangle \\
\mathsf{put}_s \langle \mathsf{get} \langle m_{\mathsf{true}}, m_{\mathsf{false}} \rangle \rangle &\simeq \mathsf{put}_s \langle m_s \rangle \\
\mathsf{get} \langle \mathsf{put}_{\mathsf{true}} \langle m \rangle, n \rangle &\simeq \mathsf{get} \langle m, n \rangle \simeq \mathsf{get} \langle m, \mathsf{put}_{\mathsf{false}} \langle n \rangle \rangle \\
\mathsf{get} \langle \mathsf{get} \langle m, m' \rangle, n \rangle &\simeq \mathsf{get} \langle m, n \rangle \simeq \mathsf{get} \langle m, \mathsf{get} \langle n', n \rangle \rangle
\end{aligned}
$$

Interpreting algebraic effects

Example: boolean state

Standard interpretation ($[\![\mathsf{comp}\, A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{bool}$)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\, \langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\
[\![\mathsf{put}_{s'}\, \langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Discard interpretation ($[\![\mathsf{comp}\, A]\!] = \mathsf{bool} \to [\![A]\!]$)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.[\![v]\!] \\
[\![\mathsf{get}\, \langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\
[\![\mathsf{put}_{s'}\, \langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Logging interpretation ($[\![\mathsf{comp}\, A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{list}\, \mathsf{bool}$)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], [s]) \\
[\![\mathsf{get}\, \langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\
[\![\mathsf{put}_{s'}\, \langle m\rangle]\!] &= \lambda s.\mathbf{let}\ (x, ss) \leftarrow [\![m]\!] s'\ \mathbf{in}\ (x, s :: ss)
\end{aligned}
$$

Example: boolean state, standard interpretation

$$\begin{aligned} [\![\mathsf{comp}\, A]\!] &= \mathsf{bool} \to [\![A]\!] \times \mathsf{bool} \\ [\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\ [\![\mathsf{get}\, \langle m, n \rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\ [\![\mathsf{put}_{s'}\, \langle m \rangle]\!] &= \lambda s.[\![m]\!] s' \end{aligned}$$

Sound and complete with respect to the equations.

$$m \simeq n \iff [\![m]\!] = [\![n]\!]$$

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\, s\, \mathbf{then}\, (\mathsf{true}, \mathsf{false})\, \mathbf{else}\, (\mathsf{false}, \mathsf{true})$$

Example: boolean state, discard interpretation

$$[\![\mathsf{comp}\, A]\!] = \mathsf{bool} \to [\![A]\!]$$

$$\begin{aligned} [\![\mathbf{return}\, v]\!] &= \lambda s.[\![v]\!] \\ [\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\ [\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.[\![m]\!] s' \end{aligned}$$

Sound with respect to the equations.

$$m \simeq n \implies [\![m]\!] = [\![n]\!]$$

Not complete because:

$$[\![\mathsf{put}_{s}\,\langle \mathbf{return}\, v\rangle]\!] = [\![\mathbf{return}\, v]\!]$$

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\, s\, \mathbf{then}\, \mathsf{true}\, \mathbf{else}\, \mathsf{false} = \lambda s.s$$

Example: boolean state, logging interpretation

$$[\![\mathsf{comp}\, A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{list}\,\mathsf{bool}$$

$$\begin{aligned} [\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], [s]) \\ [\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\ [\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.\mathbf{let}\ (x, ss) \leftarrow [\![m]\!] s'\ \mathbf{in}\ (x, s :: ss) \end{aligned}$$

Complete with respect to the equations.

$$m \simeq n \iff [\![m]\!] = [\![n]\!]$$

Not sound because:

$$[\![\mathsf{put}_s\,\langle \mathsf{put}_{s'}\,\langle m\rangle\rangle]\!] \neq [\![\mathsf{put}_{s'}\,\langle m\rangle]\!]$$

$$[\![\mathsf{get}\,\langle \mathsf{put}_{\mathsf{true}}\,\langle m\rangle, n\rangle]\!] \neq [\![\mathsf{get}\,\langle m, n\rangle]\!] \neq [\![\mathsf{get}\,\langle m, \mathsf{put}_{\mathsf{false}}\,\langle n\rangle\rangle]\!]$$

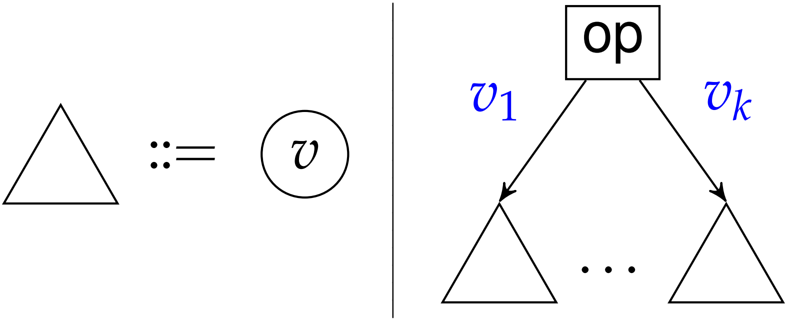
Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\, s\, \mathbf{then}\, (\mathsf{true}, [\mathsf{true}, \mathsf{false}])\ \mathbf{else}\ (\mathsf{false}, [\mathsf{false}, \mathsf{true}])$$

Algebraic effects without equations

- Different interpretations are useful in practice
- We adopt *free* algebraic effects: no equations
- *Algebraic computations* are command-response trees modulo equations
- *Abstract computations* are plain command-response trees
- Different interpretations give different meanings to the same abstract computation

What is an effectful computation?



Equivalently (ignoring result values):

$$m ::= \mathbf{return}\, v \mid \mathsf{op}\ \langle m_1, \ldots m_k \rangle$$

Equivalently (accounting for result values):

$$m ::= \mathbf{return}\, v \mid \mathsf{op}\ (\lambda x.\mathbf{case}\, x \{ v_1 \mapsto m_1; \ldots ; v_k \mapsto m_k \})$$

Interpretations as effect handlers

Example: boolean state

Meta level interpretation (enumerated continuations)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\, s\, \mathbf{then} [\![m]\!] s\, \mathbf{else}\, [\![n]\!] s \\
[\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Meta level interpretation (continuations as functions)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\, k]\!] &= \lambda s.[\![k\, s]\!]\, s \\
[\![\mathsf{put}_{s'}\, k]\!] &= \lambda s.[\![k\,()]\!]\, s'
\end{aligned}
$$

Object level effect handler

$$
\begin{aligned}
&\mathbf{return}\, v \mapsto \lambda s.(v, s) \\
&\mathsf{get}\,()\, k \quad \mapsto \lambda s.k\, s\, s \\
&\mathsf{put}\, s'\, k \quad \mapsto \lambda s.k\,()\, s'
\end{aligned}
$$

Interpretations as effect handlers

Example: nondeterminism

Meta level interpretation (enumerated continuations)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= [[\![v]\!]] \\
[\![\mathsf{choose}\, \langle m, n\rangle]\!] &= [\![m]\!] +\!\!+ [\![n]\!] \\
[\![\mathsf{fail}\, \langle\rangle]\!] &= []
\end{aligned}
$$

Meta level interpretation (continuations as functions)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= [[\![v]\!]] \\
[\![\mathsf{choose}\, k]\!] &= [\![k\, \mathsf{true}]\!] +\!\!+ [\![k\, \mathsf{false}]\!] \\
[\![\mathsf{fail}\, k]\!] &= []
\end{aligned}
$$

Object level effect handler

$$
\begin{aligned}
&\mathbf{return}\, v && \mapsto [v] \\
&\mathsf{choose}\, ()\, k && \mapsto k\, \mathsf{true} +\!\!+ k\, \mathsf{false} \\
&\mathsf{fail}\, ()\, k && \mapsto []
\end{aligned}
$$

Handlers in Links (demo)

Algebraic effects


Gordon Plotkin


John Power

Effect handlers


Gordon Plotkin


Matija Pretnar

Operational semantics

$$\begin{aligned}
\textbf{handle } V \textbf{ with } H &\rightsquigarrow N[V/x] \\
\textbf{handle } \mathcal{E}[\textbf{do } \mathsf{op}_i\ V] \textbf{ with } H &\rightsquigarrow N_i[V/p, \lambda x.\textbf{handle } \mathcal{E}[x] \textbf{ with } H/k]
\end{aligned}$$

where

$$\begin{aligned}
H = \textbf{return } x &\mapsto N \\
\mathsf{op}_1\ p\ k &\mapsto N_1 \\
&\dots \\
\mathsf{op}_n\ p\ k &\mapsto N_n
\end{aligned}$$

Typing rules

Operations

$$\frac{\Delta; \Gamma \vdash V : A}{\Delta; \Gamma \vdash \textbf{do}\ \mathsf{op}\ V : B!\{\mathsf{op} : A \to B; R\}}$$

Handlers

$$\frac{\Delta; \Gamma \vdash M : C \qquad \Delta; \Gamma \vdash H : C \Rightarrow D}{\Delta; \Gamma \vdash \textbf{handle}\ M\ \textbf{with}\ H : D}$$

$$\frac{\begin{array}{c} C = A!\{(\mathsf{op}_i : A_i \to B_i)_i; R\} \qquad D = B!\{(\mathsf{op}_i : P_i)_i; R\} \\ \Delta; \Gamma, x : A \vdash M : D \qquad [\Delta; \Gamma, p : A_i, k : B_i \to D \vdash N_i : D]_i \end{array}}{\Delta; \Gamma \vdash \begin{array}{l} \textbf{return}\ x \mapsto M \\ (\mathsf{op}_i\ p\ k \mapsto N_i)_i \end{array} : C \Rightarrow D}$$

Deep vs shallow handlers

Deep

$$\textbf{handle}\ \mathcal{E}[\textbf{do}\ \mathsf{op}_i\ V]\ \textbf{with}\ H \rightsquigarrow N_i[V/p, \lambda x.\textbf{handle}\ \mathcal{E}[x]\ \textbf{with}\ H/k]$$

$$\frac{\begin{array}{cc} C = A!\{(\mathsf{op}_i : A_i \to B_i)_i; R\} & D = B!\{(\mathsf{op}_i : P_i)_i; R\} \\ \Delta; \Gamma, x : A \vdash M : D & [\Delta; \Gamma, p : A_i, k : B_i \to D \vdash N_i : D]_i \end{array}}{\Delta; \Gamma \vdash \begin{array}{l} \textbf{return}\ x \mapsto M \\ (\mathsf{op}_i\ p\ k \mapsto N_i)_i \end{array} : C \Rightarrow D}$$

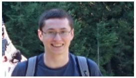
Shallow

$$\textbf{handle}\ \mathcal{E}[\textbf{do}\ \mathsf{op}_i\ V]\ \textbf{with}\ H \rightsquigarrow N_i[V/p, \lambda x.\mathcal{E}[x]/k]$$

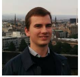
$$\frac{\begin{array}{cc} C = A!\{(\mathsf{op}_i : A_i \to B_i)_i; R\} & D = B!\{(\mathsf{op}_i : P_i)_i; R\} \\ \Delta; \Gamma, x : A \vdash M : D & [\Delta; \Gamma, p : A_i, k : B_i \to C \vdash N_i : D]_i \end{array}}{\Delta; \Gamma \vdash \begin{array}{l} \textbf{return}\ x \mapsto M \\ (\mathsf{op}_i\ p\ k \mapsto N_i)_i \end{array} : C \Rightarrow D}$$

Languages with explicit support for effect handlers

Eff  

Frank   

Koka 

Links  

Multicore OCaml     

Shonky 

Handlers in Frank (demo)

Implementation strategies for effect handlers

- Free monads (Eff; Koka; Haskell libraries)
- Delimited continuations (ML/Scheme/Racket libraries)
- Direct manipulation of the stack (Multicore OCaml)
- Continuation Passing Style (Koka; Links client backend)
- Abstract machine (Links server backend; Shonky; Frank)

Effects bibliography

http://github.com/yallop/effects-bibliography

References

Gordon Plotkin and John Power. Adequacy for algebraic effects. *FoSSaCS 2001*.

Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. *ESOP 2009*.

Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program. 2015*.

Stephen Dolan, Leo White, KC Sivaramakrishnan, Jeremy Yallop, and Anil Madhavapeddy. Effective concurrency through algebraic effects. *OCaml Workshop 2015*.

Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. *TyDe 2016*.

Daan Leijen. Type directed compilation of row-typed algebraic effects. *POPL 2017*.

Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. *POPL 2017*.

Frank type synthesis rules

$$\boxed{\Gamma\ [\Sigma]\!\vdash\ m \Rightarrow A}$$

$$\textsc{Var}\quad \frac{x : A \in \Gamma}{\Gamma\ [\Sigma]\!\vdash\ x \Rightarrow A}
\qquad
\textsc{PolyVar}\quad \frac{f : \forall \overline{Z}.A \in \Gamma}{\Gamma\ [\Sigma]\!\vdash\ f \Rightarrow \theta(A)}
\qquad
\textsc{Command}\quad \frac{c : \overline{A \to}\ B \in \Sigma}{\Gamma\ [\Sigma]\!\vdash\ c \Rightarrow \{\overline{\langle\iota\rangle A \to}\ [\Sigma]B\}}$$

$$\textsc{App}\quad \frac{\Gamma\ [\Sigma]\!\vdash\ m \Rightarrow \{\overline{\langle\Delta\rangle A \to}\ [\Sigma']B\} \qquad \Sigma' = \Sigma \qquad \overline{\Gamma\ [\Sigma \oplus \Delta]\!\vdash\ n : A}}{\Gamma\ [\Sigma]\!\vdash\ m\ \overline{n} \Rightarrow B}$$

Frank type checking rules

$$\boxed{\Gamma\ [\Sigma]\!\!- n : A}$$

Switch

$$\frac{\Gamma\ [\Sigma]\!\!- m \Rightarrow A \qquad A = B}{\Gamma\ [\Sigma]\!\!- m : B}$$

Data

$$\frac{k\ \overline{A} \in D\ \overline{R} \qquad \overline{\Gamma\ [\Sigma]\!\!- n : A}}{\Gamma\ [\Sigma]\!\!- k\ \overline{n} : D\ \overline{R}}$$

Thunk

$$\frac{\Gamma \vdash e : C}{\Gamma\ [\Sigma]\!\!- \{e\} : \{C\}}$$

$$\boxed{\Gamma \vdash e : C}$$

Comp

$$\frac{(r_{i,j} : T_j \dashv\![\Sigma]\ \Gamma'_{i,j})_{i,j} \qquad (\Gamma, (\Gamma'_{i,j})_j\ [\Sigma]\!\!- n_i : B)_i \qquad (r_{i,j})_{i,j} \text{ covers } (T_j)_j}{\Gamma \vdash ((r_{i,j})_j \mapsto n_i)_i : (T_j \to)_j\ [\Sigma]B}$$

Frank pattern matching rules

$$\boxed{p : A \dashv \Gamma}$$

P-Var

$$\frac{}{x : A \dashv x : A}$$

P-Data

$$\frac{k\,\overline{A} \in D\,\overline{R} \qquad \overline{p : A \dashv \Gamma}}{k\,\overline{p} : D\,\overline{R} \dashv \overline{\Gamma}}$$

$$\boxed{r : T \dashv[\Sigma]\ \Gamma}$$

P-Value

$$\frac{p : A \dashv \Gamma}{p : \langle\Delta\rangle A \dashv[\Sigma]\ \Gamma}$$

P-Request

$$\frac{c : \overline{A} \to B \in \emptyset \oplus \Delta \qquad (p_i : A_i \dashv \Gamma_i)_i}{\langle c\,\overline{p} \to z\rangle : \langle\Delta\rangle B' \dashv[\Sigma]\ \overline{\Gamma}, z : \langle\iota\rangle B \to [\Sigma \oplus \Delta]B'}$$

P-CatchAll

$$\frac{}{\langle x\rangle : \langle\Delta\rangle A \dashv[\Sigma]\ x : \{[\Sigma \oplus \Delta]A\}}$$