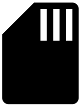
Effect polymorphism in higher-order logic
Proof pearl

Andreas Lochbihler

Institute of Information Security

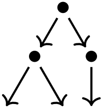
ETH zürich

Monadic effects in HOL

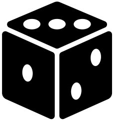


state

failure



non-determinism

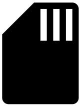


probabilities

$\text{return} :: \alpha \Rightarrow \alpha\ \mathsf{M}$
$\text{bind} :: \alpha\ \mathsf{M} \Rightarrow (\alpha \Rightarrow \beta\ \mathsf{M}) \Rightarrow \beta\ \mathsf{M}$

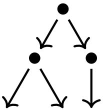
1. $(m \gg= f) \gg= g \;=\; m \gg= (\lambda x.\ f\ x \gg= g)$
2. $\text{return}\ x \gg= f \;=\; f\ x$
3. $m \gg= \text{return} \;=\; m$

Monadic effects in HOL

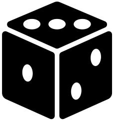


state

failure



non-determinism



probabilities

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

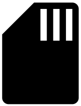
β M $\Rightarrow (\beta \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

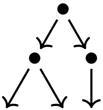
α M $\Rightarrow (\alpha \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

1. $(m \gg= f) \gg= g \;=\; m \gg= (\lambda x.\ f\ x \gg= g)$
2. return $x \gg= f \;=\; f\ x$
3. $m \gg=$ return $\;=\; m$

Monadic effects in HOL


state


failure


non-determinism


probabilities

return $:: \alpha \Rightarrow \alpha$ M
bind $:: \alpha$ M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

β M $\Rightarrow (\beta \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

value polymorphism

α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

α M $\Rightarrow (\alpha \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

1. $(m \gg\!= f) \gg\!= g \;=\; m \gg\!= (\lambda x.\ f\ x \gg\!= g)$
2. return $x \gg\!= f \;=\; f\ x$
3. $m \gg\!=$ return $\;=\; m$

Monadic effects in HOL

No effect polymorphism:

- HOL cannot express the notion of a monad
- HOL functions cannot abstract over the monad M

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

β M $\Rightarrow (\beta \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

value polymorphism

α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

α M $\Rightarrow (\alpha \Rightarrow \gamma$ M$) \Rightarrow \gamma$ M

1. $(m \ggg= f) \ggg= g \; = \; m \ggg= (\lambda x.\; f\; x \ggg= g)$
2. return $x \ggg= f \; = \; f\; x$
3. $m \ggg=$ return $\; = \; m$

Effect polymorphism with value monomorphism in

Monad τ

return :: $\forall \alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall \alpha\, \beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

Effect polymorphism with value monomorphism in

Monad τ

return :: $\forall\alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall\alpha\,\beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

effect monomorphism

value polymorphism

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

foo :: $\ldots \Rightarrow \mathbb{Z}$ state
bar :: $\ldots \Rightarrow$ unit option
goo :: $\ldots \Rightarrow \alpha$ state

Effect polymorphism with value monomorphism in

Monad τ

return :: $\forall\alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall\alpha\,\beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

effect monomorphism

value monomorphism

value polymorphism

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

effect polymorphism

return :: $\alpha \Rightarrow \alpha\ \tau$
bind :: $\alpha\ \tau \Rightarrow (\alpha \Rightarrow \alpha\ \tau) \Rightarrow \alpha\ \tau$

foo :: $\ldots \Rightarrow \mathbb{Z}$ state
bar :: $\ldots \Rightarrow$ unit option
goo :: $\ldots \Rightarrow \alpha$ state

Effect polymorphism with value monomorphism in Isabelle HOL

effect monomorphism

Monad τ

return :: $\forall\alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall\alpha\,\beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

value monomorphism

value polymorphism

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

effect polymorphism

return :: $\alpha \Rightarrow\ \ \mu$
bind :: $\ \ \mu\ \Rightarrow (\alpha \Rightarrow\ \ \mu\ ) \Rightarrow\ \ \mu$

foo :: $\ldots \Rightarrow \mathbb{Z}$ state
bar :: $\ldots \Rightarrow$ unit option
goo :: $\ldots \Rightarrow \alpha$ state

Effect polymorphism with value monomorphism in Isabelle/HOL

Monad τ

return :: $\forall \alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall \alpha\, \beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

effect monomorphism

value monomorphism

value polymorphism

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

foo :: $\ldots \Rightarrow \mathbb{Z}$ state
bar :: $\ldots \Rightarrow$ unit option
goo :: $\ldots \Rightarrow \alpha$ state

effect polymorphism

return :: $\alpha \Rightarrow \mu$
bind :: $\mu \Rightarrow (\alpha \Rightarrow \mu) \Rightarrow \mu$

```
locale monad =
  fixes return :: α ⇒ μ
    and bind :: μ ⇒ (α ⇒ μ) ⇒ μ
  assumes ...
```

Effect polymorphism with value monomorphism in Isabelle/HOL

- Abstract monads & effects
- Implementing monads and monad transformers
- Switching between monads

enables

Monad τ

return :: $\forall \alpha.\ \alpha \Rightarrow \alpha\ \tau$
bind :: $\forall \alpha\, \beta.\ \alpha\ \tau \Rightarrow (\alpha \Rightarrow \beta\ \tau) \Rightarrow \beta\ \tau$

effect monomorphism

value monomorphism

value polymorphism

return :: $\alpha \Rightarrow \alpha$ M
bind :: α M $\Rightarrow (\alpha \Rightarrow \beta$ M$) \Rightarrow \beta$ M

foo :: $\ldots \Rightarrow \mathbb{Z}$ state
bar :: $\ldots \Rightarrow$ unit option
goo :: $\ldots \Rightarrow \alpha$ state

effect polymorphism

return :: $\alpha \Rightarrow \mu$
bind :: $\mu \Rightarrow (\alpha \Rightarrow \mu) \Rightarrow \mu$

```
locale monad =
  fixes return :: α ⇒ μ
    and bind :: μ ⇒ (α ⇒ μ) ⇒ μ
  assumes ...
```

Example I: Monadic interpreter

$\mathsf{exp} ::= \mathsf{Const}\ \mathbb{Z} \mid \mathsf{Var}\ \mathcal{V} \mid \mathsf{exp} \oplus \mathsf{exp} \mid \mathsf{exp} \oslash \mathsf{exp}$

$$[\![\_]\!]_{\_} :: \mathsf{exp} \Rightarrow (\mathcal{V} \Rightarrow \mu) \Rightarrow \mu$$
$$[\![\mathsf{Const}\ i]\!]_E = \mathsf{return}\ \ i$$
$$[\![\mathsf{Var}\ x]\!]_E = E\ x$$
$$[\![e_1 \oplus e_2]\!]_E = [\![e_1]\!]_E \ggg= (\lambda i_1.\ [\![e_2]\!]_E \ggg= (\lambda i_2.\ \mathsf{return}\ \ (i_1 + i_2)))$$
$$[\![e_1 \oslash e_2]\!]_E =$$
$$[\![e_1]\!]_E \ggg= (\lambda i_1.\ [\![e_2]\!]_E \ggg= (\lambda i_2.\ \mathsf{if}\ i_2 \neq 0\ \mathsf{then}\ \ \mathsf{return}\ \ (i_1 / i_2)\ \mathsf{else}\ \ \mathsf{fail}\ ))$$

Example I: Monadic interpreter

$\mathsf{exp} ::= \mathsf{Const}\ \mathbb{Z} \mid \mathsf{Var}\ \mathcal{V} \mid \mathsf{exp} \oplus \mathsf{exp} \mid \mathsf{exp} \oslash \mathsf{exp}$

$\texttt{locale}\ \text{monad-fail} = \text{monad}\ \mathsf{return}\ \mathsf{bind}\ +$
$\quad \texttt{fixes}\ \mathsf{fail} :: \mu$
$\quad \texttt{assumes}\ \mathsf{fail} \gg\!= f = \mathsf{fail}$

failure

$[\![\_]\!]_{\_} :: \mathsf{exp} \Rightarrow (\mathcal{V} \Rightarrow \mu) \Rightarrow \mu$
$[\![\mathsf{Const}\ i]\!]_E = \mathsf{return}\ i$
$[\![\mathsf{Var}\ x]\!]_E = E\ x$
$[\![e_1 \oplus e_2]\!]_E = [\![e_1]\!]_E \gg\!= (\lambda i_1.\ [\![e_2]\!]_E \gg\!= (\lambda i_2.\ \mathsf{return}\ (i_1 + i_2)))$
$[\![e_1 \oslash e_2]\!]_E =$
$\quad [\![e_1]\!]_E \gg\!= (\lambda i_1.\ [\![e_2]\!]_E \gg\!= (\lambda i_2.\ \mathsf{if}\ i_2 \neq 0\ \mathsf{then}\ \mathsf{return}\ (i_1 / i_2)\ \mathsf{else}\ \mathsf{fail}))$

Example I: Monadic interpreter

$\text{exp} ::= \text{Const } \mathbb{Z} \mid \text{Var } \mathcal{V} \mid \text{exp} \oplus \text{exp} \mid \text{exp} \oslash \text{exp}$

$\texttt{locale}\ \text{monad-fail} = \text{monad return bind} +$
$\quad \texttt{fixes}\ \text{fail} :: \mu$
$\quad \texttt{assumes}\ \text{fail} \gg\!= f = \text{fail}$

failure

$[\![\_]\!]_{\_} :: \text{exp} \Rightarrow (\mathcal{V} \Rightarrow \mu) \Rightarrow \mu$

$[\![\text{Const } i]\!]_E = \text{return } i$

$[\![\text{Var } x]\!]_E = E\ x$

$[\![e_1 \oplus e_2]\!]_E = [\![e_1]\!]_E \gg\!= (\lambda i_1.\ [\![e_2]\!]_E \gg\!= (\lambda i_2.\ \text{return } (i_1 + i_2)))$

$[\![e_1 \oslash e_2]\!]_E =$
$\quad [\![e_1]\!]_E \gg\!= (\lambda i_1.\ [\![e_2]\!]_E \gg\!= (\lambda i_2.\ \text{if } i_2 \neq 0 \text{ then return } (i_1 / i_2) \text{ else fail}))$

Now prove your theorems **abstractly**!

Example I: Monadic interpreter

$\mathsf{exp} ::= \mathsf{Const}\ \mathbb{Z} \mid \mathsf{Var}\ \mathcal{V} \mid \mathsf{exp} \oplus \mathsf{exp} \mid \mathsf{exp} \oslash \mathsf{exp}$

$\texttt{locale}$ monad-fail $=$ monad return bind $+$
  $\texttt{fixes}$ fail $:: \mu$
  $\texttt{assumes}$ fail $\mathbin{>\!\!>\!\!=} f =$ fail

failure

$[\![\_]\!]_{\_} :: \mathsf{exp} \Rightarrow (\mathcal{V} \Rightarrow \mu) \Rightarrow \mu$

$[\![\mathsf{Const}\ i]\!]_E = \mathsf{return}\ i$

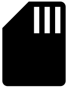
$[\![\mathsf{Var}\ x]\!]_E = E\ x$

$[\![e_1 \oplus e_2]\!]_E = [\![e_1]\!]_E \mathbin{>\!\!>\!\!=} (\lambda i_1.\ [\![e_2]\!]_E \mathbin{>\!\!>\!\!=} (\lambda i_2.\ \mathsf{return}\ (i_1 + i_2)))$

$[\![e_1 \oslash e_2]\!]_E =$
  $[\![e_1]\!]_E \mathbin{>\!\!>\!\!=} (\lambda i_1.\ [\![e_2]\!]_E \mathbin{>\!\!>\!\!=} (\lambda i_2.\ \text{if } i_2 \neq 0 \text{ then } \mathsf{return}\ (i_1 / i_2) \text{ else } \mathsf{fail}))$

Now prove your theorems **abstractly**! E.g.: compositionality

Example II: Memoisation



Conventional interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: $\ldots$
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$$\xmapsto[\text{passing}]{\text{continuation}}$$

Monomorphic interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: $\ldots$
get :: $(\sigma \Rightarrow \alpha$ M$) \Rightarrow \alpha$ M
put :: $\sigma \Rightarrow \alpha$ M $\Rightarrow \alpha$ M

Example II: Memoisation

Conventional interface

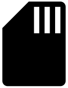
return :: $\alpha \Rightarrow \alpha$ M
bind :: ...
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$\xmapsto[\text{passing}]{\text{continuation}}$

Monomorphic interface

return :: $\alpha \Rightarrow \mu$
bind :: ...
get :: $(\sigma \Rightarrow \mu) \Rightarrow \mu$
put :: $\sigma \Rightarrow \mu \Rightarrow \mu$

Example II: Memoisation



Conventional interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: ...
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$\xmapsto[\text{passing}]{\text{continuation}}$

Monomorphic interface

return :: $\alpha \Rightarrow \mu$
bind :: ...
get :: $(\sigma \Rightarrow \mu) \Rightarrow \mu$
put :: $\sigma \Rightarrow \mu \Rightarrow \mu$

Algebraic specification

$$
\begin{aligned}
&\text{put } s \ (\text{get } f) &&= \text{put } s \ (f \ s) \\
&\text{put } s' \ (\text{put } s \ m) &&= \text{put } s \ m \\
&\text{get } (\lambda s.\ \text{get } (f \ s)) &&= \text{get } (\lambda s.\ f \ s \ s) \\
&\text{get } (\lambda s.\ \text{put } s \ m) &&= m \\
&\text{get } (\lambda \_.\ m) &&= m
\end{aligned}
$$

$$
\begin{aligned}
&\text{get } f \gg\!= g &&= \text{get } (\lambda s.\ f \ s \gg\!= g) \\
&\text{put } s \ m \gg\!= f &&= \text{put } s \ (m \gg\!= f)
\end{aligned}
$$

Example II: Memoisation

Conventional interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: ...
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$\xmapsto[\text{passing}]{\text{continuation}}$

Monomorphic interface

return :: $\alpha \Rightarrow \mu$
bind :: ...
get :: $(\sigma \Rightarrow \mu) \Rightarrow \mu$
put :: $\sigma \Rightarrow \mu \Rightarrow \mu$

Algebraic specification

$$
\begin{aligned}
\mathsf{put}\ s\ (\mathsf{get}\ f) &= \mathsf{put}\ s\ (f\ s) \\
\mathsf{put}\ s'\ (\mathsf{put}\ s\ m) &= \mathsf{put}\ s\ m \\
\mathsf{get}\ (\lambda s.\ \mathsf{get}\ (f\ s)) &= \mathsf{get}\ (\lambda s.\ f\ s\ s) \\
\mathsf{get}\ (\lambda s.\ \mathsf{put}\ s\ m) &= m \\
\mathsf{get}\ (\lambda\_.\ m) &= m
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{get}\ f \ggg g &= \mathsf{get}\ (\lambda s.\ f\ s \ggg g) \\
\mathsf{put}\ s\ m \ggg f &= \mathsf{put}\ s\ (m \ggg f)
\end{aligned}
$$

memo :: $(\beta \Rightarrow \mu) \Rightarrow \beta \Rightarrow \mu$

Example II: Memoisation

Conventional interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: . . .
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$\xmapsto[\text{passing}]{\text{continuation}}$

Monomorphic interface

return :: $\alpha \Rightarrow \mu$
bind :: . . .
get :: $(\sigma \Rightarrow \mu) \Rightarrow \mu$
put :: $\sigma \Rightarrow \mu \Rightarrow \mu$

Algebraic specification

$$
\begin{aligned}
\text{put } s \ (\text{get } f) &= \text{put } s \ (f \ s)\\
\text{put } s' \ (\text{put } s \ m) &= \text{put } s \ m\\
\text{get } (\lambda s.\ \text{get } (f \ s)) &= \text{get } (\lambda s.\ f \ s \ s)\\
\text{get } (\lambda s.\ \text{put } s \ m) &= m\\
\text{get } (\lambda\_.\ m) &= m
\end{aligned}
$$

$$
\begin{aligned}
\text{get } f \gg\!= g &= \text{get } (\lambda s.\ f \ s \gg\!= g)\\
\text{put } s \ m \gg\!= f &= \text{put } s \ (m \gg\!= f)
\end{aligned}
$$

$$
\begin{aligned}
&\text{memo} :: (\beta \Rightarrow \mu) \Rightarrow \beta \Rightarrow \mu\\
&\text{memo } f \ x =\\
&\quad \text{get } (\lambda T.\\
&\quad \text{case } T \ x \text{ of Some } y \Rightarrow \text{return } y\\
&\quad \mid \text{None} \Rightarrow f \ x \gg\!= (\lambda y.\ \text{get } (\lambda T.\ \text{put } (T(x \mapsto y)) \ (\text{return } y)))
\end{aligned}
$$

Example II: Memoisation

Conventional interface

return :: $\alpha \Rightarrow \alpha$ M
bind :: ...
get :: σ M
put :: $\sigma \Rightarrow$ unit M

$\xmapsto[\text{passing}]{\text{continuation}}$

Monomorphic interface

return :: $\alpha \Rightarrow \mu$
bind :: ...
get :: $(\sigma \Rightarrow \mu) \Rightarrow \mu$
put :: $\sigma \Rightarrow \mu \Rightarrow \mu$

Algebraic specification

$$
\begin{aligned}
\text{put } s\ (\text{get } f) &= \text{put } s\ (f\ s)\\
\text{put } s'\ (\text{put } s\ m) &= \text{put } s\ m\\
\text{get } (\lambda s.\ \text{get } (f\ s)) &= \text{get } (\lambda s.\ f\ s\ s)\\
\text{get } (\lambda s.\ \text{put } s\ m) &= m\\
\text{get } (\lambda\_.\ m) &= m
\end{aligned}
$$

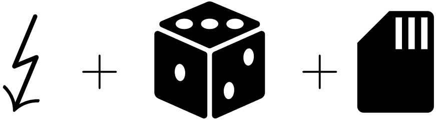
$$
\begin{aligned}
\text{get } f \gg\!= g &= \text{get } (\lambda s.\ f\ s \gg\!= g)\\
\text{put } s\ m \gg\!= f &= \text{put } s\ (m \gg\!= f)
\end{aligned}
$$

Verify memo abstractly:

- ⊕ correct
- ⊕ idempotent
- ⊕ parametric

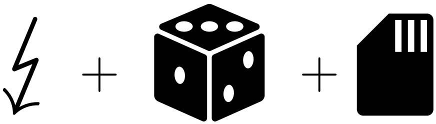
$$
\begin{aligned}
&\text{memo} :: (\beta \Rightarrow \mu) \Rightarrow \beta \Rightarrow \mu\\
&\text{memo } f\ x =\\
&\quad \text{get } (\lambda T.\\
&\quad \text{case } T\ x \text{ of Some } y \Rightarrow \text{return } y\\
&\quad \mid \text{None} \Rightarrow f\ x \gg\!= (\lambda y.\ \text{get } (\lambda T.\ \text{put } (T(x \mapsto y))\ (\text{return } y)))
\end{aligned}
$$

Combining monads



locale monad-fail-prob-state = monad-fail + monad-prob + monad-state
  assumes ...

Combining monads

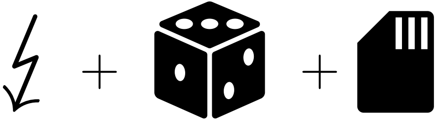


`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
  `assumes ...`

Interpreters for probabilistic expressions over random variables:

$$\mathsf{Var}\ X \oplus \mathsf{Var}\ X$$

Combining monads



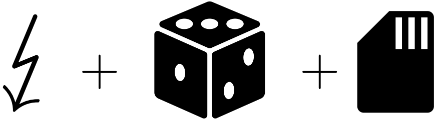
`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
  `assumes ...`

Interpreters for probabilistic expressions over random variables:

$$\text{Var } X \oplus \text{Var } X$$

lazy $\mathcal{X}$ $e = [\![e]\!]_{(\text{sample-var } \mathcal{X})}$

Combining monads



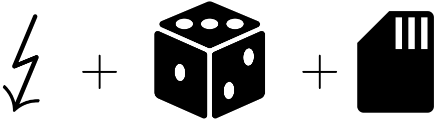
`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
  `assumes ...`

Interpreters for probabilistic expressions over random variables:

lazy  $(\text{Var } X \oplus \text{Var } X) \rightsquigarrow \begin{pmatrix} 0 \mapsto 1/4 \\ 1 \mapsto 1/2 \\ 2 \mapsto 1/4 \end{pmatrix}$

$\text{lazy } \mathcal{X} \; e = [\![e]\!]_{(\text{sample-var } \mathcal{X})}$

Combining monads



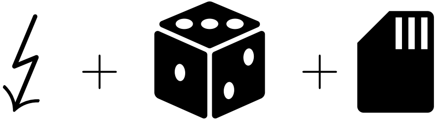
`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
`assumes ...`

Interpreters for probabilistic expressions over random variables:

$$\text{lazy}$$  $$(\text{Var } X \oplus \text{Var } X) \rightsquigarrow \cancel{\begin{pmatrix} 0 \mapsto 1/4 \\ 1 \mapsto 1/2 \\ 2 \mapsto 1/4 \end{pmatrix}} \quad \begin{pmatrix} 0 \mapsto 1/2 \\ 2 \mapsto 1/2 \end{pmatrix}$$

$\text{lazy } \mathcal{X} \ e = [\![e]\!]_{\boxed{\text{memo}}(\text{sample-var } \mathcal{X})}$

Combining monads



```
locale monad-fail-prob-state = monad-fail + monad-prob + monad-state
  assumes ...
```

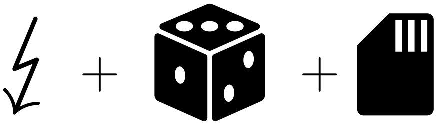
Interpreters for probabilistic expressions over random variables:

lazy  $(\mathsf{Var}\ X \oplus \mathsf{Var}\ X) \leadsto \xcancel{\begin{pmatrix} 0 \mapsto 1/4 \\ 1 \mapsto 1/2 \\ 2 \mapsto 1/4 \end{pmatrix}} \quad \begin{pmatrix} 0 \mapsto 1/2 \\ 2 \mapsto 1/2 \end{pmatrix}$

lazy $\mathcal{X}\ e = [\![e]\!]_{\boxed{\text{memo}}\,(\text{sample-var}\ \mathcal{X})}$

eager $\mathcal{X}\ e = \text{sample-vars}\ \mathcal{X}\ (\text{vars}\ e)\ [\![e]\!]_{\text{lookup}}$

Combining monads



`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
  `assumes ...`

Interpreters for probabilistic expressions over random variables:

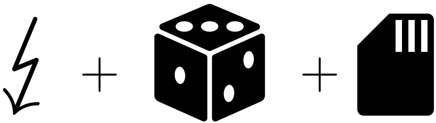
$$\text{lazy}$$  $$(\text{Var } X \oplus \text{Var } X) \rightsquigarrow \xcancel{\begin{pmatrix} 0 \mapsto 1/4 \\ 1 \mapsto 1/2 \\ 2 \mapsto 1/4 \end{pmatrix}} \quad \begin{pmatrix} 0 \mapsto 1/2 \\ 2 \mapsto 1/2 \end{pmatrix}$$

$\text{lazy } \mathcal{X} \; e = [\![e]\!]_{\boxed{\text{memo}}\,(\text{sample-var } \mathcal{X})}$

$\text{eager } \mathcal{X} \; e = \text{sample-vars } \mathcal{X} \; (\text{vars } e) \; [\![e]\!]_{\text{lookup}}$

$$\text{eager } \mathcal{X} \; e = \text{lazy } \mathcal{X} \; e$$

Combining monads



`locale` monad-fail-prob-state = monad-fail + monad-prob + monad-state
  `assumes ...`

Interpreters for probabilistic expressions over random variables:

lazy  $(\mathsf{Var}\ X \oplus \mathsf{Var}\ X) \leadsto \xcancel{\begin{pmatrix} 0 \mapsto 1/4 \\ 1 \mapsto 1/2 \\ 2 \mapsto 1/4 \end{pmatrix}} \quad \begin{pmatrix} 0 \mapsto 1/2 \\ 2 \mapsto 1/2 \end{pmatrix}$

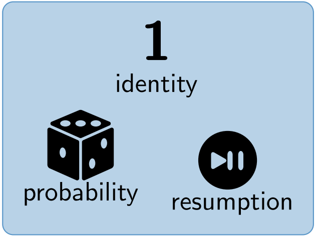
$\mathsf{lazy}\ \mathcal{X}\ e = [\![e]\!]_{\boxed{\mathsf{memo}}(\mathsf{sample\text{-}var}\ \mathcal{X})}$

$\mathsf{eager}\ \mathcal{X}\ e = \mathsf{sample\text{-}vars}\ \mathcal{X}\ (\mathsf{vars}\ e)\ [\![e]\!]_{\mathsf{lookup}}$

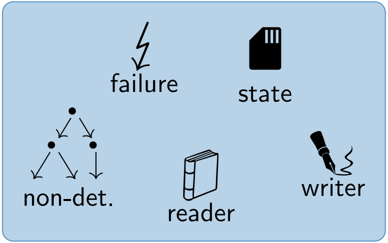
$$\mathsf{eager}\ \mathcal{X}\ e = \mathsf{lazy}\ \mathcal{X}\ e \quad \text{if} \quad \forall s.\ \mathsf{put}\ s\ \mathsf{fail} = \mathsf{fail}$$

Concrete monads and monad transformers

Monads

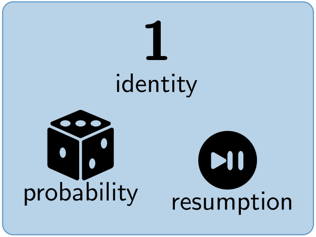


Transformers

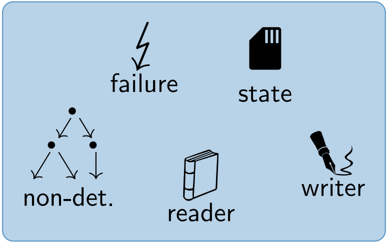


Concrete monads and monad transformers

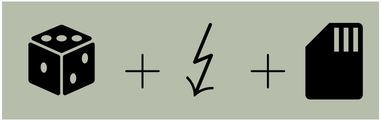
Monads



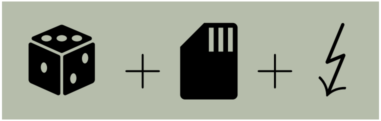
Transformers



Composing monads:

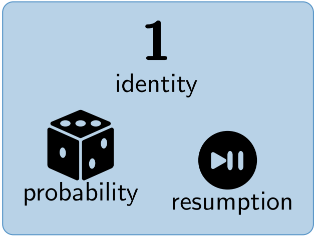


vs.

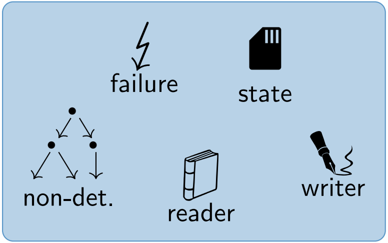


Concrete monads and monad transformers

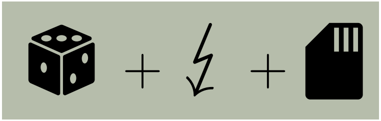
Monads



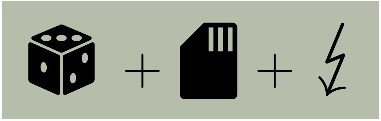
Transformers



Composing monads:



vs.

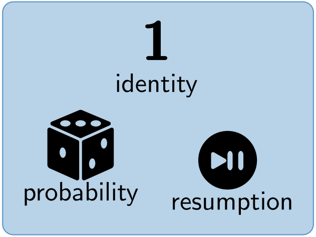


$\forall s.\ \mathsf{put}\ s\ \mathsf{fail} = \mathsf{fail}$

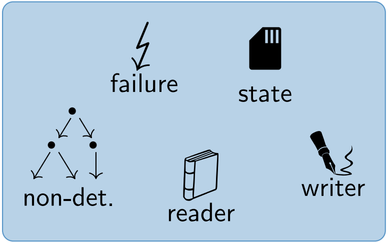
$\neg\forall s.\ \mathsf{put}\ s\ \mathsf{fail} = \mathsf{fail}$

Concrete monads and monad transformers

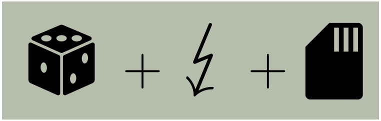
Monads



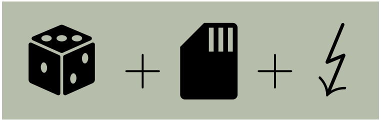
Transformers



Composing monads:



vs.



$\forall s.\ \mathsf{put}\ s\ \mathsf{fail} = \mathsf{fail}$

$\neg\forall s.\ \mathsf{put}\ s\ \mathsf{fail} = \mathsf{fail}$

$\mathsf{lazy}\ \mathcal{X}\ e = \mathsf{eager}\ \mathcal{X}\ e$

Example: Switching between monads within larger proofs

$\text{lazy}\ \mathcal{X}\ e = \text{eager}\ \mathcal{X}\ e = \text{sample-vars}\ \mathcal{X}\ (\text{vars}\ e)\ \ [\![e]\!]_{\text{lookup}}$

probabilities	🎲
failure	↯
state queries state updates	💾

Example: Switching between monads within larger proofs

$\text{lazy } \mathcal{X}\ e = \text{eager } \mathcal{X}\ e = \text{sample-vars } \mathcal{X}\ (\text{vars } e)\ [\![e]\!]_{\text{lookup}}$

probabilities

failure $\lightning$

state queries
state updates

failure

state queries

Example: Switching between monads within larger proofs

$$\text{lazy}\ \mathcal{X}\ e = \text{eager}\ \mathcal{X}\ e = \text{sample-vars}\ \mathcal{X}\ (\text{vars}\ e)\ [\![e]\!]_{\text{lookup}}$$

probabilities

failure

state queries

state updates

switch to simpler monad $\longrightarrow$

1 identity

failure

state queries

Example: Switching between monads within larger proofs

lazy $\mathcal{X}$ e = eager $\mathcal{X}$ e = sample-vars $\mathcal{X}$ (vars e) $[\![e]\!]_{\text{lookup}}$

probabilities	🎲
failure	↯
state queries	
state updates	

switch to simpler monad $\longrightarrow$

$\longleftarrow$ embed

1	identity
↯	failure
	state queries

$$[\![e]\!]^{🎲↯}_{\text{lookup}} = \text{embed } [\![e]\!]^{\mathbf{1}↯}_{\text{lookup}}$$

Example: Switching between monads within larger proofs

lazy $\mathcal{X}$ e = eager $\mathcal{X}$ e = sample-vars $\mathcal{X}$ (vars e) $[\![e]\!]_{\text{lookup}}$

probabilities	🎲
failure	$\lightning$
state queries state updates	💾

switch to simpler monad $\longrightarrow$

$\longleftarrow$ embed

$= \text{embed}^{💾}_{📖} \circ \text{lift}^{\lightning} \circ \text{embed}^{🎲}_{\mathbf{1}}$

$\mathbf{1}$	identity
$\lightning$	failure
📖	state queries

$$[\![e]\!]^{🎲\lightning💾}_{\text{lookup}} = \text{embed}\ [\![e]\!]^{\mathbf{1}\lightning📖}_{\text{lookup}}$$

Example: Switching between monads within larger proofs

$\text{lazy}\ \mathcal{X}\ e = \text{eager}\ \mathcal{X}\ e = \text{sample-vars}\ \mathcal{X}\ (\text{vars}\ e)\ [\![e]\!]_{\text{lookup}}$

probabilities	🎲
failure	↯
state queries, state updates	(SD card)

switch to simpler monad $\longrightarrow$

$\longleftarrow$ embed

$= \text{embed}^{\text{SD}}_{\text{book}} \circ \text{lift}^{↯} \circ \text{embed}^{🎲}_{\mathbf{1}}$

$\mathbf{1}$	identity
↯	failure
(book)	state queries

$[\![e]\!]^{🎲 ↯ \text{SD}}_{\text{lookup}} = \text{embed}\ [\![e]\!]^{\mathbf{1} ↯ \text{book}}_{\text{lookup}}$

Prove by induction???

Example: Switching between monads within larger proofs

lazy $\mathcal{X}$ e = eager $\mathcal{X}$ e = sample-vars $\mathcal{X}$ (vars e) $[\![e]\!]_{\text{lookup}}$

probabilities	🎲
failure	↯
state queries state updates	💾

switch to simpler monad ⟶

⟵ embed

$= \text{embed}^{💾}_{📖} \circ \text{lift}^{↯} \circ \text{embed}^{🎲}_{\mathbf{1}}$

$\mathbf{1}$	identity
↯	failure
📖	state queries

Free theorem! ⟶ $[\![e]\!]^{🎲↯💾}_{\text{lookup}} = \text{embed}\ [\![e]\!]^{\mathbf{1}↯📖}_{\text{lookup}}$

~~Prove by induction???~~

$[\![\_]\!]_{\_}$ is **relationally parametric** in the monad

See the paper for details!

value monomorphism $\rightsquigarrow$ effect polymorphism

abstract monads

free theorems

monad transformers

value monomorphism $\rightsquigarrow$ effect polymorphism

abstract monads

free theorems

monad transformers

Ideas developed while formalising cryptographic proofs with CryptHOL:

- switch to simpler monads in proofs
- lazy vs. eager sampling
- advanced kinds of memoisation

value monomorphism $\rightsquigarrow$ effect polymorphism

abstract monads

free theorems

monad transformers

Ideas developed while formalising cryptographic proofs with CryptHOL:

- switch to simpler monads in proofs
- lazy vs. eager sampling
- advanced kinds of memoisation

available in the AFP
`https://www.isa-afp.org/entries/Monomorphic_Monad.html`
`https://www.isa-afp.org/entries/CryptHOL.html`

value monomorphism $\rightsquigarrow$ effect polymorphism

- abstract monads
- monad transformers
- free theorems

Ideas developed while formalising cryptographic proofs with CryptHOL:

- switch to simpler monads in proofs
- lazy vs. eager sampling
- advanced kinds of memoisation

Next session:
David Butler using CryptHOL

Poster session:
Parametricity inference

available in the AFP
`https://www.isa-afp.org/entries/Monomorphic_Monad.html`
`https://www.isa-afp.org/entries/CryptHOL.html`

Appendix

Concrete monads and monad transformers

Monads

$\mathbf{1}$

identity

probability

resumption

Transformers

failure

state

non-det.

reader

writer

$$\alpha\ \mathsf{M} \xmapsto{\ \lightning\ } \alpha\ \mathsf{M\ failT}$$

Concrete monads and monad transformers

Monads

1

identity

probability

resumption

Transformers

failure

state

non-det.

reader

writer

$$\alpha \text{ option } \mathsf{M} \longmapsto \alpha \text{ option } \mathsf{M} \text{ failT}$$

Concrete monads and monad transformers

Monads

1

identity

probability

resumption

Transformers

failure

state

non-det.

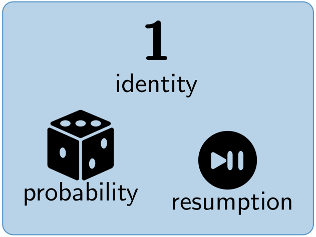
reader

writer

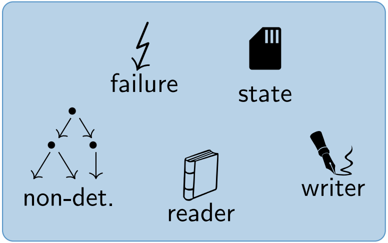
$$\overbrace{\alpha \text{ option } \mathsf{M}}^{\mu} \xmapsto{\ \lightning\ } \overbrace{\alpha \text{ option } \mathsf{M}}^{\mu} \text{ failT}$$

Concrete monads and monad transformers

Monads



Transformers



$$\overbrace{\alpha \ \mathsf{option}\ \mathsf{M}}^{\mu} \ \xmapsto{\ \lightning\ } \ \overbrace{\alpha \ \mathsf{option}\ \mathsf{M}}^{\mu}\ \mathsf{failT}$$

datatype μ failT $=$ FailT (run-fail: μ)

$\mathsf{return}_{\lightning} :: (\alpha\ \mathsf{option} \Rightarrow \mu) \Rightarrow \alpha \Rightarrow \mu\ \mathsf{failT}$

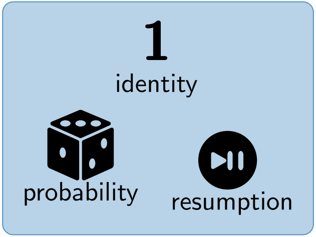
$\mathsf{return}_{\lightning}\ \mathsf{return}\ x = \mathsf{FailT}\ (\mathsf{return}\ (\mathsf{Some}\ x))$

$\mathsf{fail}_{\lightning} :: (\alpha\ \mathsf{option} \Rightarrow \mu) \Rightarrow \mu\ \mathsf{failT}$

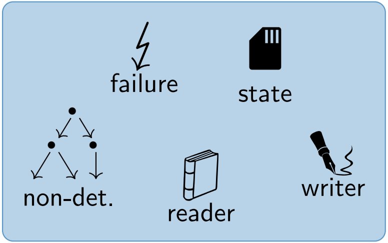
$\mathsf{fail}_{\lightning}\ \mathsf{return} = \mathsf{FailT}\ (\mathsf{return}\ \mathsf{None})$

Concrete monads and monad transformers

Monads



Transformers



$$\overbrace{\alpha\ \mathsf{option}\ \mathsf{M}}^{\mu} \xmapsto{\;\lightning\;} \overbrace{\alpha\ \mathsf{option}\ \mathsf{M}}^{\mu}\ \mathsf{failT}$$

`datatype` $\mu\ \mathsf{failT} = \mathsf{FailT}\ (\mathsf{run\text{-}fail}\colon \mu)$

$\mathsf{return}_{\lightning} :: (\alpha\ \mathsf{option} \Rightarrow \mu) \Rightarrow \alpha \Rightarrow \mu\ \mathsf{failT}$
$\mathsf{return}_{\lightning}\ \mathsf{return}\ x = \mathsf{FailT}\ (\mathsf{return}\ (\mathsf{Some}\ x))$

$\mathsf{fail}_{\lightning} :: (\alpha\ \mathsf{option} \Rightarrow \mu) \Rightarrow \mu\ \mathsf{failT}$
$\mathsf{fail}_{\lightning}\ \mathsf{return} = \mathsf{FailT}\ (\mathsf{return}\ \mathsf{None})$

lift operations of other effects

$\mathsf{get}_{\lightning} :: ((\sigma \Rightarrow \mu) \Rightarrow \mu) \Rightarrow (\sigma \Rightarrow \mu\ \mathsf{failT}) \Rightarrow \mu\ \mathsf{failT}$
$\mathsf{get}_{\lightning}\ \mathsf{get}\ f = \mathsf{FailT}\ (\mathsf{get}\ (\lambda s.\ \mathsf{run\text{-}fail}\ (f\ s)))$