Equational Reasoning with Applicative Functors

Andreas Lochbihler Joshua Schneider

Institute of Information Security

ETH zürich

Equational Reasoning with Applicative Functors

Andreas Lochbihler   Joshua Schneider

Institute of Information Security

ETH zürich

model effects

- state
- probabilities
- error
- non-determinism
- 1 2 3 4 ··· streams

Equational Reasoning with Applicative Functors

Andreas Lochbihler Joshua Schneider

Institute of Information Security

ETH zürich

$f\ (g\ \bullet) = h\ (k\ \bullet)$

$$\begin{tikzcd} \bullet \arrow[r, "k"] \arrow[d, "g"'] & \blacksquare \arrow[d, "h"] \\ \blacktriangle \arrow[r, "f"] & \bigstar \end{tikzcd}$$

model effects

- state
- probabilities
- error
- non-determinism
- 1 2 3 4 · · · streams

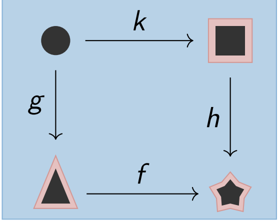
Equational Reasoning with Applicative Functors

Andreas Lochbihler Joshua Schneider

Institute of Information Security

ETH zürich

$\text{pure } f \diamond (g\ \bullet) = \text{pure } h \diamond (k\ \bullet)$



model effects

- state
- probabilities
- error
- non-determinism
- streams

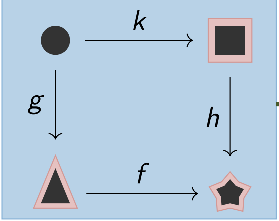
Equational Reasoning with Applicative Functors

Andreas Lochbihler   Joshua Schneider

Institute of Information Security

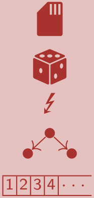
ETH zürich

$\mathsf{pure}\ f \diamond (g\ \bullet) = \mathsf{pure}\ h \diamond (k\ \bullet)$



don't care

model effects



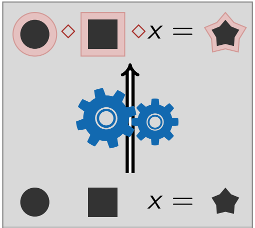
state
probabilities
error
non-determinism
streams

Contributions

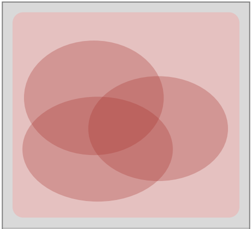
- Isabelle/HOL package for reasoning about applicative effects

```
applicative state
for
  pure: pure_state
  ap: ap_state
proof (prove)
goal (4 subgoals):
 1. ⋀f x. pure f ◇ pure x = pure (
 2. ⋀g f x. pure (λg f x. g (f x))
 3. ⋀x. pure (λx. x) ◇ x = x
 4. ⋀f x. f ◇ pure x = pure (λf. f
```

functor registration



proof tactic

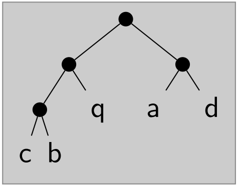


classify effects

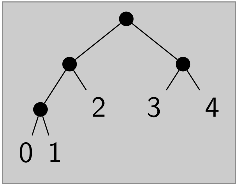
- Meta theory formalised and algorithms verified

- Used in several examples and case studies

Task: Label a binary tree with distinct numbers!



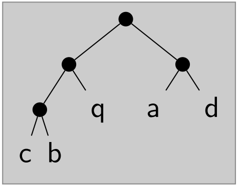
$\xrightarrow{\quad\text{lbl}\quad}$



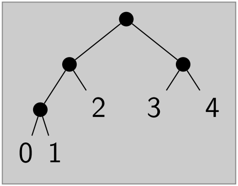
datatype α *tree* $=$
  L α | N (α *tree*) (α *tree*)

Example suggested by G. Hutton, D. Fulger: Reasoning about effects: seeing the wood through the trees. TFP 2008

Task: Label a binary tree with distinct numbers!



$\xrightarrow{\quad\text{lbl}\quad}$

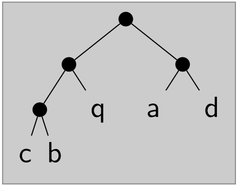


datatype α *tree* =
  L α | N (α *tree*) (α *tree*)

lbl :: α *tree* ⇒ *nat tree*

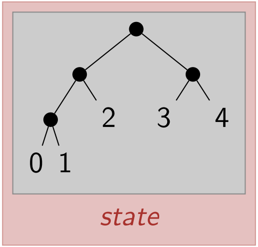
Example suggested by G. Hutton, D. Fulger: Reasoning about effects: seeing the wood through the trees. TFP 2008

Task: Label a binary tree with distinct numbers!



$$\xrightarrow{\quad \text{lbl} \quad}$$

datatype α *tree* $=$
L α | N (α *tree*) (α *tree*)



lbl :: α *tree* $\Rightarrow$ *nat tree state* where α *state* $=$ *nat* $\Rightarrow \alpha \times$ *nat*

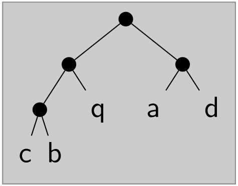
monadic α $M = \alpha$ *state*

return :: $\alpha \Rightarrow \alpha$ M
($\gg=$) :: α $M \Rightarrow (\alpha \Rightarrow \beta$ $M) \Rightarrow \beta$ M

lbl (L _) = fresh $\gg=$ $\lambda x'$. return (L x')
lbl (N l r) =
lbl l $\gg=$ $\lambda l'$. lbl r $\gg=$ $\lambda r'$. return (N l' r')

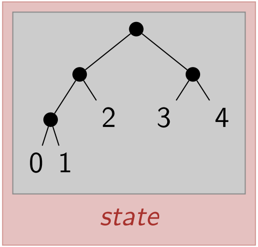
Example suggested by G. Hutton, D. Fulger: Reasoning about effects: seeing the wood through the trees. TFP 2008

Task: Label a binary tree with distinct numbers!



$\xrightarrow{\mathsf{lbl}}$

datatype α *tree* =
  L α | N (α *tree*) (α *tree*)



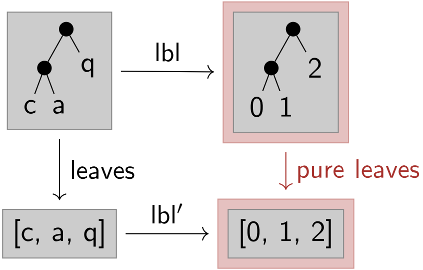
lbl :: α *tree* $\Rightarrow$ *nat tree state*   where α *state* = *nat* $\Rightarrow \alpha \times$ *nat*

monadic $\quad \alpha\ M = \alpha$ *state*
return :: $\alpha \Rightarrow \alpha\ M$
$(\ggg\!=)$ :: $\alpha\ M \Rightarrow (\alpha \Rightarrow \beta\ M) \Rightarrow \beta\ M$
lbl (L _) = fresh $\ggg\!= \lambda x'$. return (L x')
lbl (N l r) =
lbl l $\ggg\!= \lambda l'$. lbl r $\ggg\!= \lambda r'$. return (N l' r')

applicative $\quad \alpha\ F = \alpha$ *state*
pure :: $\alpha \Rightarrow \alpha\ F$
$(\diamond)$ :: $(\alpha \Rightarrow \beta)\ F \Rightarrow \alpha\ F \Rightarrow \beta\ F$
lbl (L _) = pure L $\diamond$ fresh
lbl (N l r) = pure N $\diamond$ lbl l $\diamond$ lbl r

Example suggested by G. Hutton, D. Fulger: Reasoning about effects: seeing the wood through the trees. TFP 2008

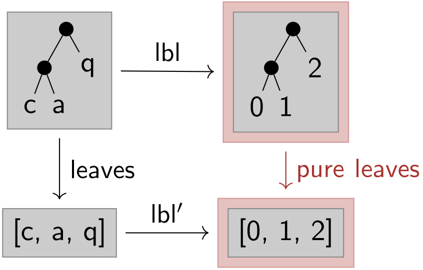
Labelling trees and lists



$$\begin{aligned}
&\mathsf{leaves} :: \alpha\ \mathit{tree} \Rightarrow \alpha\ \mathit{list}\\
&\mathsf{leaves}\ (\mathsf{L}\ x) \quad = x \cdot [\,]\\
&\mathsf{leaves}\ (\mathsf{N}\ l\ r) = \mathsf{leaves}\ l \mathbin{+\!\!+} \mathsf{leaves}\ r
\end{aligned}$$

$$\begin{aligned}
&\mathsf{lbl}' :: \alpha\ \mathit{list} \Rightarrow \mathit{nat}\ \mathit{list}\ \mathit{state}\\
&\mathsf{lbl}'\ [\,] \qquad\quad = \mathsf{pure}\ [\,]\\
&\mathsf{lbl}'\ (_ \cdot xs) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{lbl}'\ xs
\end{aligned}$$

Labelling trees and lists



$$
\begin{aligned}
&\text{leaves} :: \alpha\ \textit{tree} \Rightarrow \alpha\ \textit{list}\\
&\text{leaves}\ (\mathsf{L}\ x) \quad = x \cdot [\,]\\
&\text{leaves}\ (\mathsf{N}\ l\ r) = \text{leaves}\ l +\!\!+ \text{leaves}\ r
\end{aligned}
$$

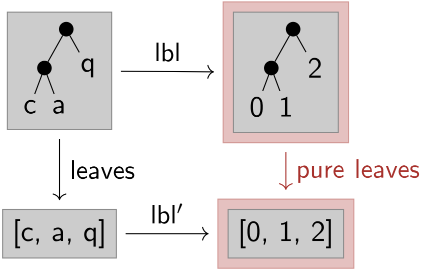
$$
\begin{aligned}
&\text{lbl}' :: \alpha\ \textit{list} \Rightarrow \textit{nat list state}\\
&\text{lbl}'\ [\,] \qquad\quad = \text{pure}\ [\,]\\
&\text{lbl}'\ (\_ \cdot xs) = \text{pure}\ (\cdot) \diamond \text{fresh} \diamond \text{lbl}'\ xs
\end{aligned}
$$

Lemma: $\text{pure leaves} \diamond \text{lbl}\ t = \text{lbl}'\ (\text{leaves}\ t)$

Proof by induction on t.

Case $\mathsf{L}\ x$: $\text{pure leaves} \diamond \text{lbl}\ (\mathsf{L}\ x) \qquad = \text{lbl}'\ (\text{leaves}\ (\mathsf{L}\ x))$

Labelling trees and lists



$\mathsf{leaves} :: \alpha\ \mathit{tree} \Rightarrow \alpha\ \mathit{list}$
$\mathsf{leaves}\ (\mathsf{L}\ x) \quad = x \cdot [\,]$
$\mathsf{leaves}\ (\mathsf{N}\ l\ r) = \mathsf{leaves}\ l +\!\!+\ \mathsf{leaves}\ r$

$\mathsf{lbl}' :: \alpha\ \mathit{list} \Rightarrow \mathit{nat}\ \mathit{list}\ \mathit{state}$
$\mathsf{lbl}'\ [\,] \qquad = \mathsf{pure}\ [\,]$
$\mathsf{lbl}'\ (\_ \cdot xs) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{lbl}'\ xs$

Lemma: $\mathsf{pure}\ \mathsf{leaves} \diamond \mathsf{lbl}\ t = \mathsf{lbl}'\ (\mathsf{leaves}\ t)$

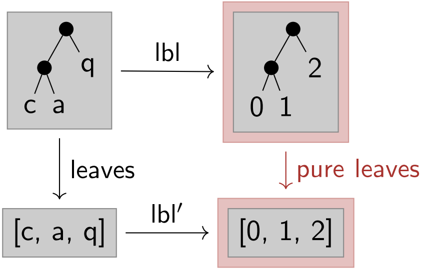
Proof by induction on t.

Case $\mathsf{L}\ x$: $\mathsf{pure}\ \mathsf{leaves} \diamond \mathsf{lbl}\ (\mathsf{L}\ x) = \mathsf{lbl}'\ (\mathsf{leaves}\ (\mathsf{L}\ x))$

$\mathsf{pure}\ \mathsf{leaves} \diamond (\mathsf{pure}\ \mathsf{L} \diamond \mathsf{fresh}) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$

$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) = (\cdot)\ x\ [\,]$

Labelling trees and lists



leaves :: α *tree* $\Rightarrow$ α *list*
leaves (L x) $= x \cdot [\,]$
leaves (N l r) $=$ leaves l $+\!+$ leaves r

lbl′ :: α *list* $\Rightarrow$ *nat list state*
lbl′ [ ] $=$ pure [ ]
lbl′ ($\_ \cdot xs$) $=$ pure ($\cdot$) $\diamond$ fresh $\diamond$ lbl′ xs

Lemma: pure leaves $\diamond$ lbl t $=$ lbl′ (leaves t)

Proof by induction on t.

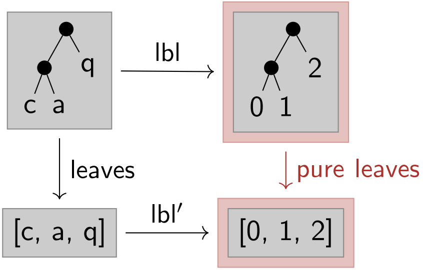
Case L x: pure leaves $\diamond$ lbl (L x) $=$ lbl′ (leaves (L x))

pure leaves $\diamond$ (pure L $\diamond$ fresh) $=$ pure ($\cdot$) $\diamond$ fresh $\diamond$ pure [ ]

holds by the applicative laws $\Uparrow$

$\forall x.$ leaves (L x) $=$ ($\cdot$) x [ ]

Labelling trees and lists



$\mathsf{leaves} :: \alpha\ \mathit{tree} \Rightarrow \alpha\ \mathit{list}$

$\mathsf{leaves}\ (\mathsf{L}\ x) \quad = x \cdot [\,]$

$\mathsf{leaves}\ (\mathsf{N}\ l\ r) = \mathsf{leaves}\ l +\!\!+ \mathsf{leaves}\ r$

$\mathsf{lbl}' :: \alpha\ \mathit{list} \Rightarrow \mathit{nat}\ \mathit{list}\ \mathit{state}$

$\mathsf{lbl}'\ [\,] \qquad = \mathsf{pure}\ [\,]$

$\mathsf{lbl}'\ (\_ \cdot xs) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{lbl}'\ xs$

Lemma: $\mathsf{pure}\ \mathsf{leaves} \diamond \mathsf{lbl}\ t = \mathsf{lbl}'\ (\mathsf{leaves}\ t)$

Proof by induction on t.

Case L x: $\mathsf{pure}\ \mathsf{leaves} \diamond \mathsf{lbl}\ (\mathsf{L}\ x) = \mathsf{lbl}'\ (\mathsf{leaves}\ (\mathsf{L}\ x))$

$\mathsf{pure}\ \mathsf{leaves} \diamond (\mathsf{pure}\ \mathsf{L} \diamond \mathsf{fresh}) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$

holds by the applicative laws $\Uparrow$ `apply applicative_lifting`

$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) = (\cdot)\ x\ [\,]$

Lifting equations over applicative functors

[Hinze 2010]

$$\text{pure leaves} \diamond (\text{pure L} \diamond \text{fresh}) \;=\; \text{pure} \; (\cdot) \diamond \text{fresh} \diamond \text{pure} \; [\,]$$

$$\Uparrow$$

$$\forall x.\ \text{leaves} \; (\text{L} \; x) \;=\; x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

$$\mathsf{pure}\ \mathsf{leaves} \diamond (\mathsf{pure}\ \mathsf{L} \diamond \mathsf{fresh}) \;=\; \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$$

$$\Uparrow$$

ML

syntactic formalisation

follows

$$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) \;=\; x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

$$\text{pure leaves} \diamond (\text{pure L} \diamond \text{fresh}) = \text{pure } (\cdot) \diamond \text{fresh} \diamond \text{pure } []$$

$\|$ 1. Convert to canonical form $\|$

$$\text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond \text{fresh} = \text{pure } (\lambda x.\ x \cdot []) \diamond \text{fresh}$$

$\Uparrow$

$$\forall x.\ \text{leaves } (\text{L } x) = x \cdot []$$

Lifting equations over applicative functors

[Hinze 2010]

Canonical form [McBride, Paterson]

applicative expression $\mapsto$ $\mathsf{pure}\ f \diamond x_1 \diamond x_2 \diamond \ldots \diamond x_n$

$$\mathsf{pure\ leaves} \diamond (\mathsf{pure\ L} \diamond \mathsf{fresh}) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ []$$

1. Convert to canonical form

$$\mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x)) \diamond \mathsf{fresh} = \mathsf{pure}\ (\lambda x.\ x \cdot []) \diamond \mathsf{fresh}$$

$$\Uparrow$$

$$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) = x \cdot []$$

Lifting equations over applicative functors

[Hinze 2010]

pure function   opaque arguments

Canonical form [McBride, Paterson]

applicative expression $\mapsto$ $\boxed{\mathsf{pure}\ f} \diamond \boxed{x_1} \diamond \boxed{x_2} \diamond \boxed{\ldots} \diamond \boxed{x_n}$

$$\mathsf{pure\ leaves} \diamond (\mathsf{pure\ L} \diamond \mathsf{fresh}) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$$

1. Convert to canonical form

$$\boxed{\mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x))} \diamond \boxed{\mathsf{fresh}} = \boxed{\mathsf{pure}\ (\lambda x.\ x \cdot [\,])} \diamond \boxed{\mathsf{fresh}}$$

$$\Uparrow$$

$$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) = x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

Canonical form [McBride, Paterson]

pure function — opaque arguments

applicative expression $\mapsto$ $\boxed{\mathsf{pure}\ f} \diamond \boxed{x_1} \diamond \boxed{x_2} \diamond \boxed{\ldots} \diamond \boxed{x_n}$

$$\mathsf{pure\ leaves} \diamond (\mathsf{pure\ L} \diamond \mathsf{fresh}) \;=\; \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$$

$\|$ 1. Convert to canonical form $\|$

$$\boxed{\mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x))} \diamond \boxed{\mathsf{fresh}} \;=\; \boxed{\mathsf{pure}\ (\lambda x.\ x \cdot [\,])} \diamond \boxed{\mathsf{fresh}}$$

2. Generalise opaque arguments $\Uparrow$

$$\boxed{\forall X.}\ \mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x)) \diamond \boxed{X} \;=\; \mathsf{pure}\ (\lambda x.\ x \cdot [\,]) \diamond \boxed{X}$$

$\Uparrow$

$$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) \;=\; x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

pure function

opaque arguments

Canonical form [McBride, Paterson]

applicative expression $\mapsto$ $\boxed{\mathsf{pure}\ f} \diamond \boxed{x_1} \diamond \boxed{x_2} \diamond \boxed{\ldots} \diamond \boxed{x_n}$

$$\mathsf{pure\ leaves} \diamond (\mathsf{pure\ L} \diamond \mathsf{fresh}) = \mathsf{pure}\ (\cdot) \diamond \mathsf{fresh} \diamond \mathsf{pure}\ [\,]$$

$\Vert$ 1. Convert to canonical form $\Vert$

$$\boxed{\mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x))} \diamond \boxed{\mathsf{fresh}} = \boxed{\mathsf{pure}\ (\lambda x.\ x \cdot [\,])} \diamond \boxed{\mathsf{fresh}}$$

2. Generalise opaque arguments $\Uparrow$

$$\boxed{\forall X.}\ \mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x)) \diamond \boxed{X} = \mathsf{pure}\ (\lambda x.\ x \cdot [\,]) \diamond \boxed{X}$$

3. Equality is a congruence $\Uparrow$

$$\forall X.\ \mathsf{pure}\ (\lambda x.\ \mathsf{leaves}\ (\mathsf{L}\ x)) \diamond X = \mathsf{pure}\ (\lambda x.\ x \cdot [\,]) \diamond X$$

$\Uparrow$

$$\forall x.\ \mathsf{leaves}\ (\mathsf{L}\ x) = x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

Canonical form [McBride, Paterson]

pure function / opaque arguments

applicative expression $\mapsto$ $\text{pure } f \diamond x_1 \diamond x_2 \diamond \ldots \diamond x_n$

$$\text{pure leaves} \diamond (\text{pure L} \diamond \text{fresh}) = \text{pure } (\cdot) \diamond \text{fresh} \diamond \text{pure } [\,]$$

1. Convert to canonical form

$$\text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond \text{fresh} = \text{pure } (\lambda x.\ x \cdot [\,]) \diamond \text{fresh}$$

2. Generalise opaque arguments

$$\forall X.\ \text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond X = \text{pure } (\lambda x.\ x \cdot [\,]) \diamond X$$

3. Equality is a congruence

$$\forall X.\ \text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond X = \text{pure } (\lambda x.\ x \cdot [\,]) \diamond X$$

4. Use extensionality

$$\forall x.\ \text{leaves } (\text{L } x) = x \cdot [\,]$$

Lifting equations over applicative functors

[Hinze 2010]

pure function

opaque arguments

Canonical form [McBride, Paterson]

applicative expression $\mapsto$ $\boxed{\text{pure } f} \diamond \boxed{x_1} \diamond \boxed{x_2} \diamond \boxed{\ldots} \diamond \boxed{x_n}$

$$\text{pure leaves} \diamond (\text{pure L} \diamond \text{fresh}) = \text{pure } (\cdot) \diamond \text{fresh} \diamond \text{pure } [\,]$$

1. Convert to canonical form

$$\boxed{\text{pure } (\lambda x.\ \text{leaves } (\text{L } x))} \diamond \boxed{\text{fresh}} = \boxed{\text{pure } (\lambda x.\ x \cdot [\,])} \diamond \boxed{\text{fresh}}$$

2. Generalise opaque arguments

$$\boxed{\forall X.}\ \text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond \boxed{X} = \text{pure } (\lambda x.\ x \cdot [\,]) \diamond \boxed{X}$$

3. Equality is a congruence

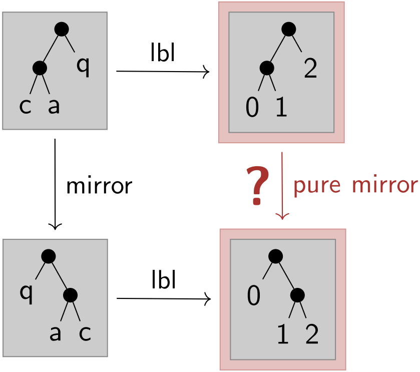
Same opaque args. on both sides!

$$\forall X.\ \text{pure } (\lambda x.\ \text{leaves } (\text{L } x)) \diamond X = \text{pure } (\lambda x.\ x \cdot [\,]) \diamond X$$

4. Use extensionality

$$\forall x.\ \text{leaves } (\text{L } x) = x \cdot [\,]$$

Tree mirroring



$$
\begin{aligned}
&\text{mirror} :: \alpha\ \textit{tree} \Rightarrow \alpha\ \textit{tree}\\
&\text{mirror}\ (\mathsf{L}\ x) \quad\ = \mathsf{L}\ x\\
&\text{mirror}\ (\mathsf{N}\ l\ r) = \mathsf{N}\ (\text{mirror}\ r)\ (\text{mirror}\ l)
\end{aligned}
$$

$$
\begin{aligned}
&\text{lbl} :: \alpha\ \textit{tree} \Rightarrow \textit{nat tree state}\\
&\text{lbl}\ (\mathsf{L}\ \_) \quad\ = \text{pure}\ \mathsf{L} \diamond \text{fresh}\\
&\text{lbl}\ (\mathsf{N}\ l\ r) = \text{pure}\ \mathsf{N} \diamond \text{lbl}\ l \diamond \text{lbl}\ r
\end{aligned}
$$

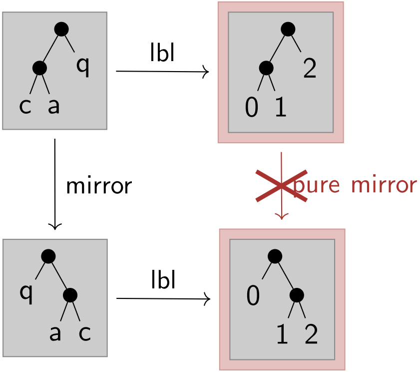
Lemma: $\text{lbl}\ (\text{mirror}\ t) = \text{pure mirror} \diamond \text{lbl}\ t$

Proof by induction on t.

Case $\mathsf{N}\ l\ r$:

$$
\begin{aligned}
&\boxed{\text{pure}\ (\lambda r'\ l'.\ \mathsf{N}\ (\text{mirror}\ r')\ (\text{mirror}\ l'))} \diamond \boxed{\text{lbl}\ r} \diamond \boxed{\text{lbl}\ l}\\
\overset{?}{=}\ &\boxed{\text{pure}\ (\lambda l'\ r'.\ \text{mirror}\ (\mathsf{N}\ l'\ r'))} \diamond \boxed{\text{lbl}\ l} \diamond \boxed{\text{lbl}\ r}
\end{aligned}
$$

Tree mirroring



$$
\begin{aligned}
&\text{mirror} :: \alpha\ \textit{tree} \Rightarrow \alpha\ \textit{tree}\\
&\text{mirror}\ (\text{L}\ x) \quad = \text{L}\ x\\
&\text{mirror}\ (\text{N}\ l\ r) = \text{N}\ (\text{mirror}\ r)\ (\text{mirror}\ l)
\end{aligned}
$$

$$
\begin{aligned}
&\text{lbl} :: \alpha\ \textit{tree} \Rightarrow \textit{nat tree state}\\
&\text{lbl}\ (\text{L}\ \_) \quad = \text{pure}\ \text{L} \diamond \text{fresh}\\
&\text{lbl}\ (\text{N}\ l\ r) = \text{pure}\ \text{N} \diamond \text{lbl}\ l \diamond \text{lbl}\ r
\end{aligned}
$$

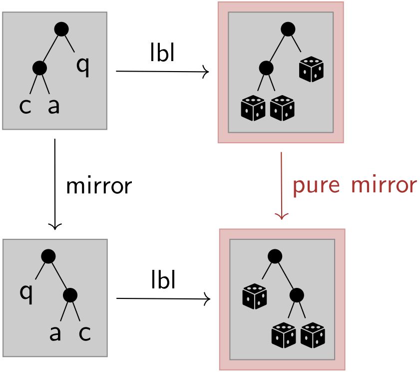
Lemma: $\text{lbl}\ (\text{mirror}\ t) = \text{pure mirror} \diamond \text{lbl}\ t$

Proof by induction on t.

Case N l r:

$$
\begin{aligned}
&\boxed{\text{pure}\ (\lambda r'\ l'.\ \text{N}\ (\text{mirror}\ r')\ (\text{mirror}\ l'))} \diamond \boxed{\text{lbl}\ r} \diamond \boxed{\text{lbl}\ l}\\
\stackrel{?}{=}\ &\boxed{\text{pure}\ (\lambda l'\ r'.\ \text{mirror}\ (\text{N}\ l'\ r'))} \diamond \boxed{\text{lbl}\ l} \diamond \boxed{\text{lbl}\ r}
\end{aligned}
$$

Tree mirroring and random labels



$\text{mirror} :: \alpha\ \mathit{tree} \Rightarrow \alpha\ \mathit{tree}$
$\text{mirror}\ (\text{L}\ x) \quad = \text{L}\ x$
$\text{mirror}\ (\text{N}\ l\ r) = \text{N}\ (\text{mirror}\ r)\ (\text{mirror}\ l)$

$\text{lbl} :: \alpha\ \mathit{tree} \Rightarrow \mathit{nat}\ \mathit{tree}\ \mathit{probability}$
$\text{lbl}\ (\text{L}\ \_) \quad = \text{pure}\ \text{L} \diamond \text{fresh}$
$\text{lbl}\ (\text{N}\ l\ r) = \text{pure}\ \text{N} \diamond \text{lbl}\ l \diamond \text{lbl}\ r$

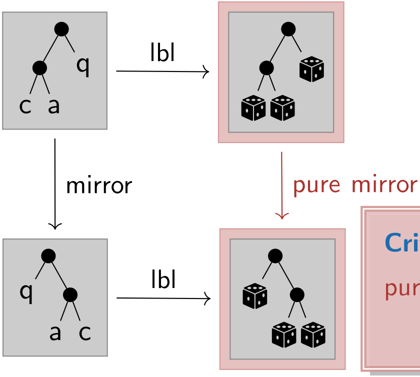
Lemma: $\text{lbl}\ (\text{mirror}\ t) = \text{pure}\ \text{mirror} \diamond \text{lbl}\ t$ **if effects commute**

Proof by induction on t.

Case $\text{N}\ l\ r$:

$$\boxed{\text{pure}\ (\lambda r'\ l'.\ \text{N}\ (\text{mirror}\ r')\ (\text{mirror}\ l'))} \diamond \boxed{\text{lbl}\ r} \diamond \boxed{\text{lbl}\ l}$$
$$= \boxed{\text{pure}\ (\lambda l'\ r'.\ \text{mirror}\ (\text{N}\ l'\ r'))} \diamond \boxed{\text{lbl}\ l} \diamond \boxed{\text{lbl}\ r}$$

Tree mirroring and random labels



$$
\begin{aligned}
&\mathsf{mirror} :: \alpha\ \mathit{tree} \Rightarrow \alpha\ \mathit{tree}\\
&\mathsf{mirror}\ (\mathsf{L}\ x) \quad = \mathsf{L}\ x\\
&\mathsf{mirror}\ (\mathsf{N}\ l\ r) = \mathsf{N}\ (\mathsf{mirror}\ r)\ (\mathsf{mirror}\ l)
\end{aligned}
$$

Criterion for commutative effects:

$$\mathsf{pure}\ (\lambda f\ x\ y.\ f\ y\ x) \diamond f \diamond x \diamond y \ = \ f \diamond y \diamond x$$

$$\mathbf{C} \qquad f\ \ x\ \ y \ = \ f\ \ y\ \ x$$

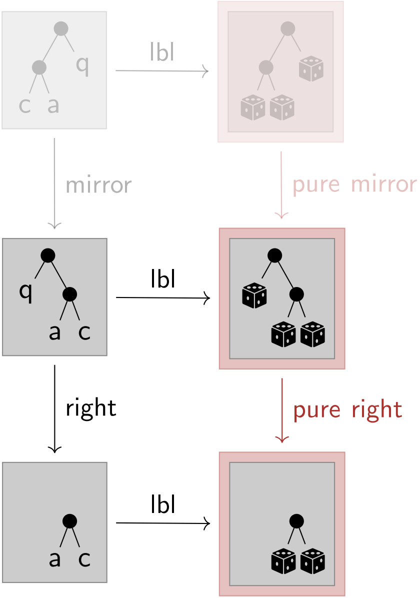
Lemma: $\mathsf{lbl}\ (\mathsf{mirror}\ t) = \mathsf{pure}\ \mathsf{mirror} \diamond \mathsf{lbl}\ t$ **if effects commute**

Proof by induction on t.

Case $\mathsf{N}\ l\ r$:

$$
\begin{aligned}
& \mathsf{pure}\ (\lambda r'\ l'.\ \mathsf{N}\ (\mathsf{mirror}\ r')\ (\mathsf{mirror}\ l')) \diamond \mathsf{lbl}\ r \diamond \mathsf{lbl}\ l\\
= \ & \mathsf{pure}\ (\lambda l'\ r'.\ \mathsf{mirror}\ (\mathsf{N}\ l'\ r')) \diamond \mathsf{lbl}\ l \diamond \mathsf{lbl}\ r
\end{aligned}
$$

Subtrees



Lemma:

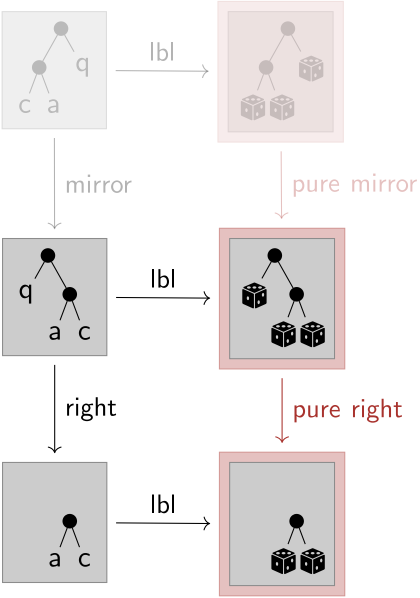
$\mathsf{lbl}\ (\mathsf{right}\ t) = \mathsf{pure}\ \mathsf{right} \diamond \mathsf{lbl}\ t$

Proof by case analysis on t.

Case N l r:

$$\mathsf{pure}\ (\lambda r'.\ \ r') \diamond \boxed{\mathsf{lbl}\ r}$$
$$\stackrel{?}{=}$$
$$\mathsf{pure}\ (\lambda\_\ r'.\ r') \diamond \boxed{\mathsf{lbl}\ l} \diamond \boxed{\mathsf{lbl}\ r}$$

Subtrees



Criterion for omissible effects:

$$\text{pure } (\lambda x\ y.\ x) \diamond x \diamond y = x$$
$$\mathbf{K}\quad x\quad y = x$$

Lemma: **if effects are omissible**

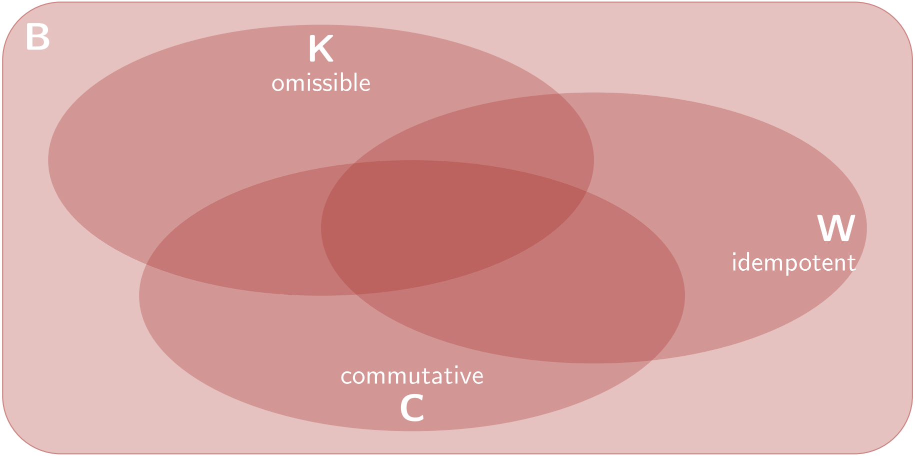
$$\text{lbl } (\text{right } t) = \text{pure right} \diamond \text{lbl } t$$

Proof by case analysis on t.

Case N l r:

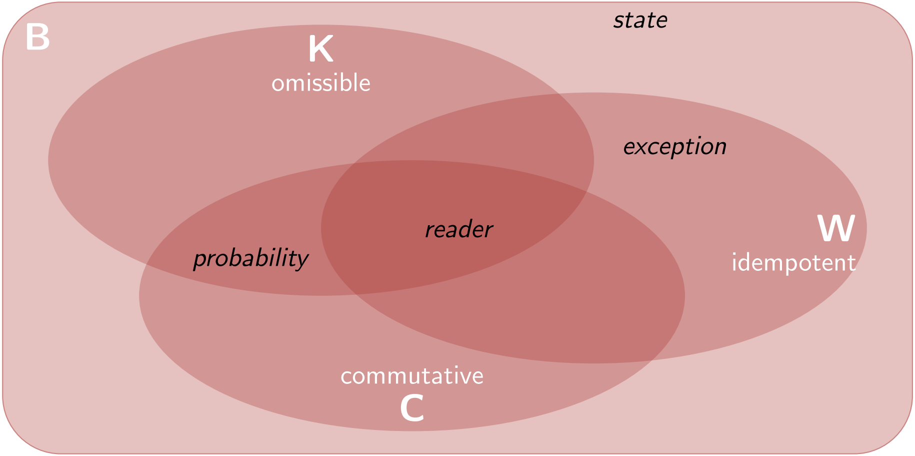
$$\text{pure } (\lambda r'.\ \ r') \qquad \diamond \boxed{\text{lbl } r}$$
$$=$$
$$\text{pure } (\lambda\_\ r'.\ r') \diamond \boxed{\text{lbl } l} \diamond \boxed{\text{lbl } r}$$

Combinatorial basis **BCKW**



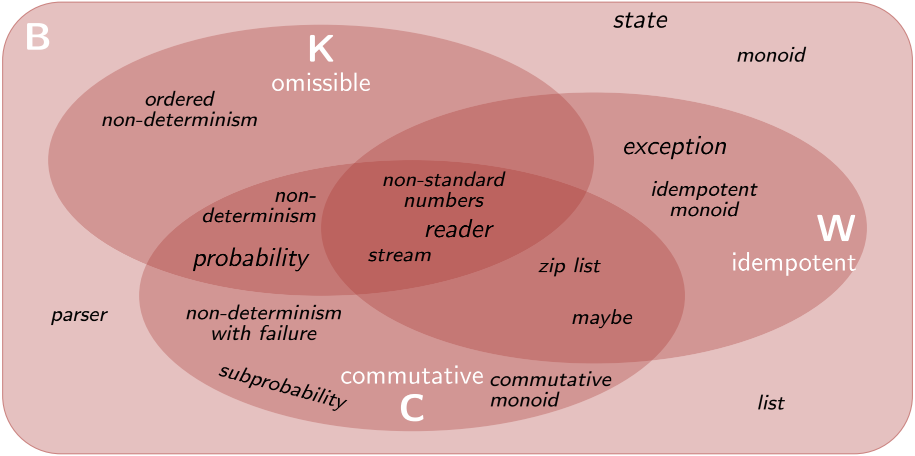
- Declarative characterisation of "liftable" equations
- Modular implementation via bracket abstraction

Combinatorial basis **BCKW**



- Declarative characterisation of “liftable” equations
- Modular implementation via bracket abstraction
- User declares and proves combinator properties at registration

Combinatorial basis **BCKW**



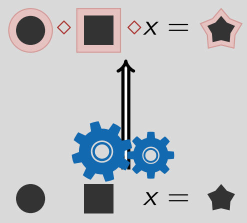
- Declarative characterisation of “liftable” equations
- Modular implementation via bracket abstraction
- User declares and proves combinator properties at registration

Summary

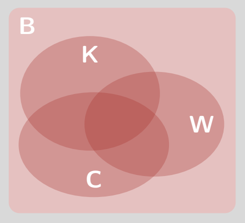
www.isa-afp.org/entries/Applicative_Lifting.shtml

```
applicative state
for
  pure: pure_state
  ap: ap_state
proof (prove)
goal (4 subgoals):
 1. ⋀f x. pure f ⋄ pure x = pure (
 2. ⋀g f x. pure (λg f x. g (f x))
 3. ⋀x. pure (λx. x) ⋄ x = x
 4. ⋀f x. f ⋄ pure x = pure (λf. f
```

functor registration



proof tactic



classify effects

guides

formalisation of the meta theory

Summary

www.isa-afp.org/entries/Applicative_Lifting.shtml