

# 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



streams

# Equational Reasoning with Applicative Functors

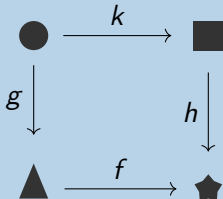
Andreas Lochbihler

Joshua Schneider

Institute of Information Security

**ETH** zürich

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



model effects



state



probabilities



error



non-determinism



streams

# Equational Reasoning with Applicative Functors

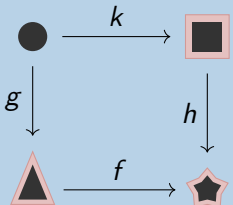
Andreas Lochbihler

Joshua Schneider

Institute of Information Security

**ETH** zürich

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



model effects



state



probabilities



error



non-determinism



streams

# Equational Reasoning with Applicative Functors

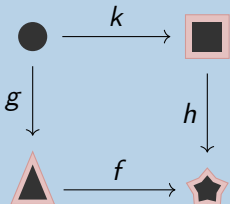
Andreas Lochbihler

Joshua Schneider

Institute of Information Security

**ETH** zürich

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



don't  
care

model effects



state



probabilities



error



non-determinism



streams

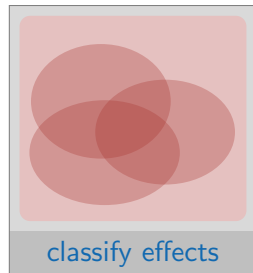
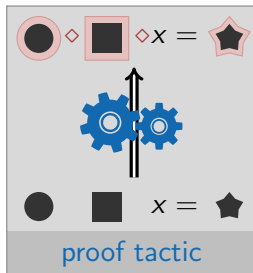
# Contributions

- ▶ Isabelle/HOL package for reasoning about applicative effects

```
applicative state
for
  pure: pure_state
  ap: ap_state

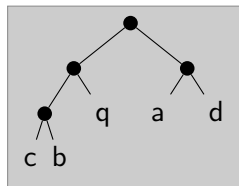
proof (prove)
goal (4 subgoals):
1.  $\bigwedge f x. \text{pure } f \diamond \text{pure } x = \text{pure } (f x)$ 
2.  $\bigwedge g f x. \text{pure } (\lambda g f x. g (f x))$ 
3.  $\bigwedge x. \text{pure } (\lambda x. x) \diamond x = x$ 
4.  $\bigwedge f x. f \diamond \text{pure } x = \text{pure } (\lambda f. f x)$ 
```

functor registration

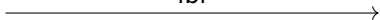


- ▶ Meta theory formalised and algorithms verified
- ▶ Used in several examples and case studies

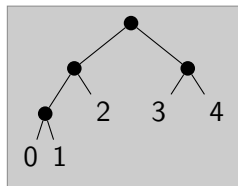
## Task: Label a binary tree with distinct numbers!



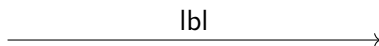
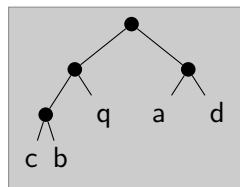
lbl



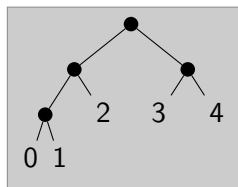
**datatype**  $\alpha$  tree =  
L  $\alpha$  | N ( $\alpha$  tree) ( $\alpha$  tree)



## Task: Label a binary tree with distinct numbers!



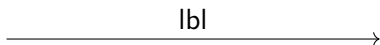
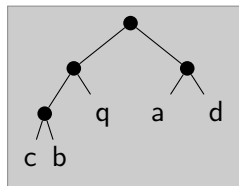
**datatype**  $\alpha$  tree =  
L  $\alpha$  | N ( $\alpha$  tree) ( $\alpha$  tree)



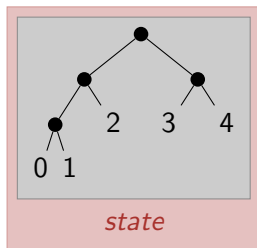
$lbl :: \alpha$  tree  $\Rightarrow$  nat tree



# Task: Label a binary tree with distinct numbers!



**datatype**  $\alpha$  tree =  
L  $\alpha$  | N ( $\alpha$  tree) ( $\alpha$  tree)



$\text{lbl} :: \alpha \text{ tree} \Rightarrow \text{nat tree state}$

where

$\alpha \text{ state} = \text{nat} \Rightarrow \alpha \times \text{nat}$

**monadic**

$\alpha M = \alpha \text{ state}$

$\text{return} :: \alpha \Rightarrow \alpha M$

$(\gg) :: \alpha M \Rightarrow (\alpha \Rightarrow \beta M) \Rightarrow \beta M$

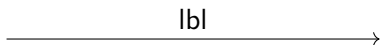
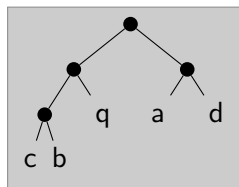
$\text{lbl } (L \_) = \text{fresh } \gg \lambda x'. \text{return } (L x')$

$\text{lbl } (N \ / \ r) =$

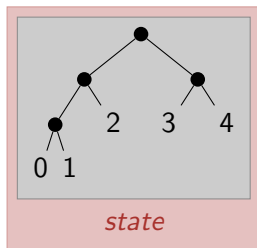
$\text{lbl } \ / \ \gg \lambda l'. \text{lbl } r \gg \lambda r'. \text{return } (N \ / \ r')$

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!



**datatype**  $\alpha$  tree =  
 $L \ \alpha \mid N \ (\alpha \ \text{tree}) \ (\alpha \ \text{tree})$



$\text{lbl} :: \alpha \ \text{tree} \Rightarrow \text{nat tree state}$

where

$\alpha \ \text{state} = \text{nat} \Rightarrow \alpha \times \text{nat}$

**monadic**

$\alpha \ M = \alpha \ \text{state}$

$\text{return} :: \alpha \Rightarrow \alpha \ M$

$(\gg) :: \alpha \ M \Rightarrow (\alpha \Rightarrow \beta \ M) \Rightarrow \beta \ M$

$\text{lbl} \ (L \ \_) = \text{fresh} \gg \lambda x'. \text{return} \ (L \ x')$

$\text{lbl} \ (N \ l \ r) =$

$\text{lbl} \ l \gg \lambda l'. \text{lbl} \ r \gg \lambda r'. \text{return} \ (N \ l' \ r')$

**applicative**

$\alpha \ F = \alpha \ \text{state}$

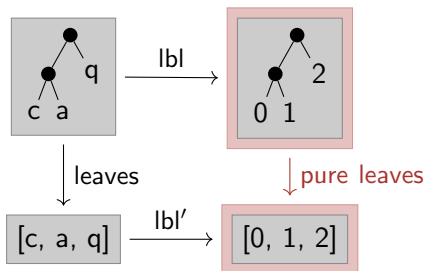
$\text{pure} :: \alpha \Rightarrow \alpha \ F$

$(\diamond) :: (\alpha \Rightarrow \beta) \ F \Rightarrow \alpha \ F \Rightarrow \beta \ F$

$\text{lbl} \ (L \ \_) = \text{pure} \ L \ \diamond \ \text{fresh}$

$\text{lbl} \ (N \ l \ r) = \text{pure} \ N \ \diamond \ \text{lbl} \ l \ \diamond \ \text{lbl} \ r$

## Labelling trees and lists



$\text{leaves} :: \alpha \text{ tree} \Rightarrow \alpha \text{ list}$

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

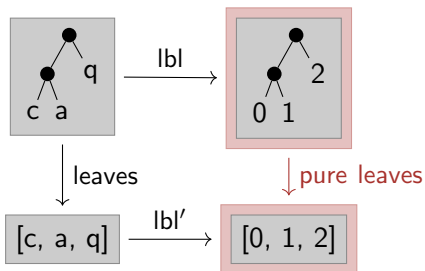
$\text{leaves (N } l \ r) = \text{leaves } l \ ++ \ \text{leaves } r$

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

$\text{lbl}' [] = \text{pure } []$

$\text{lbl}' (- \cdot xs) = \text{pure } (\cdot) \diamond \text{fresh} \diamond \text{lbl}' xs$

## Labelling trees and lists



$leaves :: \alpha \text{ tree} \Rightarrow \alpha \text{ list}$

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

$leaves (N l r) = leaves l ++ leaves r$

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

$lbl' [] = \text{pure} []$

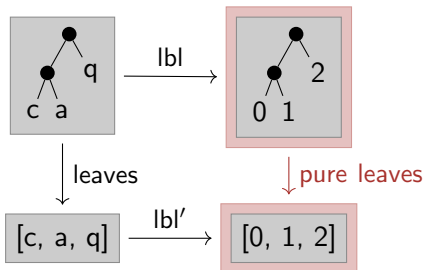
$lbl' (- \cdot xs) = \text{pure} (\cdot) \diamond \text{fresh} \diamond lbl' xs$

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

Proof by induction on  $t$ .

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

## Labelling trees and lists



$\text{leaves} :: \alpha \text{ tree} \Rightarrow \alpha \text{ list}$

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

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

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

$\text{lbl}' [] = \text{pure } []$

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

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

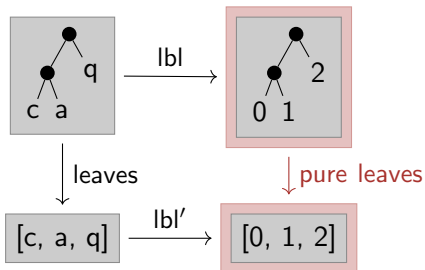
Proof by induction on  $t$ .

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

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

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

# Labelling trees and lists



$leaves :: \alpha \text{ tree} \Rightarrow \alpha \text{ list}$

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

$leaves (N \ l \ r) = leaves \ l \ ++ \ leaves \ r$

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

$lbl' [] = \text{pure } []$

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

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

Proof by induction on  $t$ .

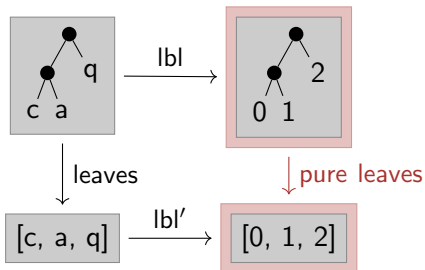
Case L  $x$ :  $\text{pure leaves} \diamond lbl (L \ x) = lbl' (\text{leaves } (L \ x))$

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

holds by the applicative laws  $\uparrow$

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

# Labelling trees and lists



$leaves :: \alpha \text{ tree} \Rightarrow \alpha \text{ list}$

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

$leaves (N \ l \ r) = leaves \ l \ ++ \ leaves \ r$

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

$lbl' [] = \text{pure} []$

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

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

Proof by induction on  $t$ .

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

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

holds by the applicative laws  $\uparrow$  apply applicative\_lifting

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

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

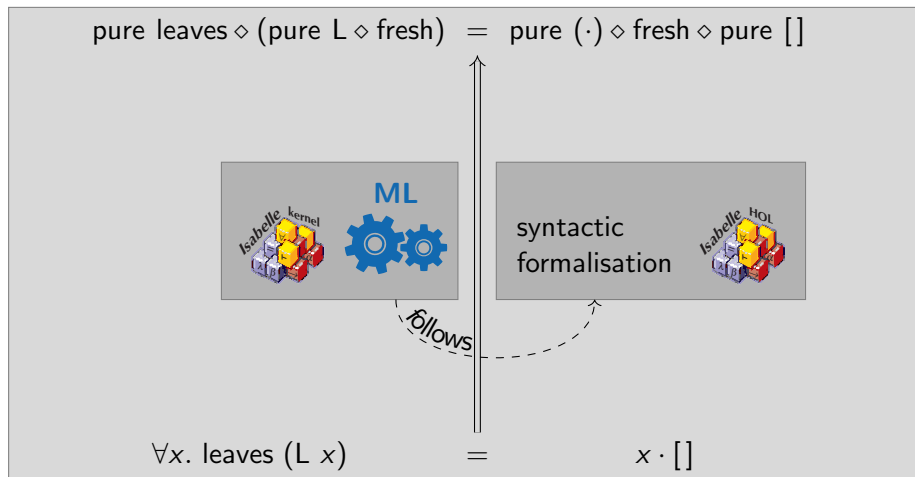


$$\forall x. \text{leaves } (L \ x)$$

=

$$x \cdot []$$





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

|| 1. Convert to canonical form ||

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

$$\forall x. \text{leaves } (L x)$$



=

$$x \cdot []$$

**Canonical form**

[McBride, Paterson]

applicative expression  $\mapsto$  pure  $f \diamond x_1 \diamond x_2 \diamond \dots \diamond x_n$ 

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

$$\parallel \boxed{\text{1. Convert to canonical form}} \parallel$$

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

$$\forall x. \text{leaves } (L x)$$



=

$$x \cdot []$$

# Lifting equations over applicative functors

[Hinze 2010]

## Canonical form

pure function

opaque arguments

[McBride, Paterson]

applicative expression  $\mapsto$  `pure f`  $\diamond$  `x1`  $\diamond$  `x2`  $\diamond$  `...`  $\diamond$  `xn`

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

1. Convert to canonical form

`pure (λx. leaves (L x))`  $\diamond$  `fresh` = `pure (λx. x · [])`  $\diamond$  `fresh`

$\forall x. \text{leaves } (L\ x)$

=

$x \cdot []$

# Lifting equations over applicative functors

[Hinze 2010]

pure function

opaque arguments

## Canonical form

[McBride, Paterson]

applicative expression  $\mapsto$  pure  $f$   $\diamond$   $x_1$   $\diamond$   $x_2$   $\diamond$  ...  $\diamond$   $x_n$

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

1. Convert to canonical form

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

2. Generalise opaque arguments

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

$\forall x$ . leaves (L x)

=

x  $\cdot$  []

pure function

opaque arguments

## Canonical form

[McBride, Paterson]

applicative expression  $\mapsto$  pure  $f$   $\diamond$   $x_1$   $\diamond$   $x_2$   $\diamond$  ...  $\diamond$   $x_n$

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

1. Convert to canonical form

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

2. Generalise opaque arguments

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

3. Equality is a congruence

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

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

pure function

opaque arguments

## Canonical form

[McBride, Paterson]

applicative expression  $\mapsto$  pure  $f$   $\diamond$   $x_1$   $\diamond$   $x_2$   $\diamond$  ...  $\diamond$   $x_n$

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

1. Convert to canonical form

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

2. Generalise opaque arguments

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

3. Equality is a congruence

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

4. Use extensionality

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

pure function

opaque arguments

## Canonical form

[McBride, Paterson]

applicative expression  $\mapsto$  **pure**  $f$   $\diamond$   $x_1$   $\diamond$   $x_2$   $\diamond$   $\dots$   $\diamond$   $x_n$

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

1. Convert to canonical form

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

2. Generalise opaque arguments

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

3. Equality is a congruence

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

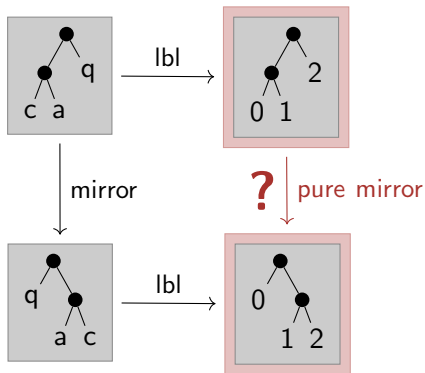
*Same opaque args. on both sides!*

4. Use extensionality

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



# Tree mirroring



$\text{mirror} :: \alpha \text{ tree} \Rightarrow \alpha \text{ tree}$

$\text{mirror} (L \ x) = L \ x$

$\text{mirror} (N \ l \ r) = N \ (\text{mirror } r) \ (\text{mirror } l)$

$\text{lbl} :: \alpha \text{ tree} \Rightarrow \text{nat tree state}$

$\text{lbl} (L \ _) = \text{pure } L \diamond \text{fresh}$

$\text{lbl} (N \ l \ r) = \text{pure } N \diamond \text{lbl } l \diamond \text{lbl } r$

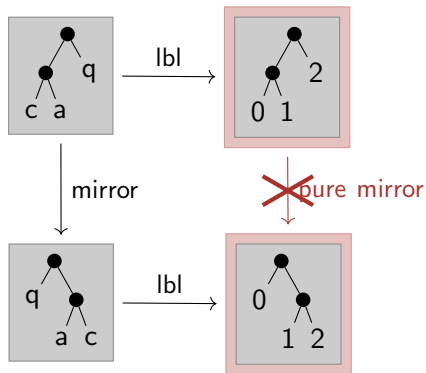
**Lemma:**  $\text{lbl} (\text{mirror } t) = \text{pure mirror} \diamond \text{lbl } t$

Proof by induction on  $t$ .

Case  $N \ l \ r$ :

$$\begin{array}{c}
 \text{pure } (\lambda r' l'. N \ (\text{mirror } r') \ (\text{mirror } l')) \diamond \text{lbl } r \diamond \text{lbl } l \\
 \stackrel{?}{=} \begin{array}{c} \text{X} \\ \text{X} \end{array} \quad \begin{array}{c} \updownarrow \\ \updownarrow \end{array} \quad \begin{array}{c} \text{X} \\ \text{X} \end{array} \\
 \text{pure } (\lambda l' r'. \text{mirror} (N \ l' \ r')) \diamond \text{lbl } l \diamond \text{lbl } r
 \end{array}$$

# Tree mirroring



$\text{mirror} :: \alpha \text{ tree} \Rightarrow \alpha \text{ tree}$

$\text{mirror} (L \ x) = L \ x$

$\text{mirror} (N \ l \ r) = N \ (\text{mirror } r) \ (\text{mirror } l)$

$\text{lbl} :: \alpha \text{ tree} \Rightarrow \text{nat tree state}$

$\text{lbl} (L \ _) = \text{pure } L \diamond \text{fresh}$

$\text{lbl} (N \ l \ r) = \text{pure } N \diamond \text{lbl } l \diamond \text{lbl } r$

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

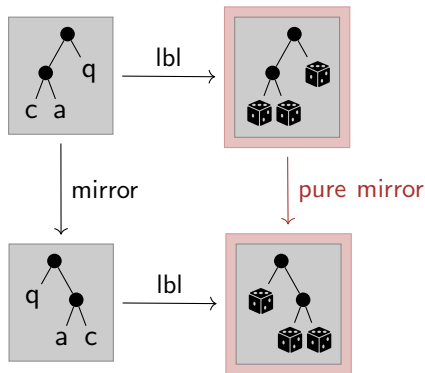
Proof by induction on  $t$ .

Case  $N \ l \ r$ :

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

$\begin{array}{ccc} \text{X} & & \text{X} \\ \text{X} & & \text{X} \\ \text{X} & & \text{X} \end{array}$

# Tree mirroring and random labels



$\text{mirror} :: \alpha \text{ tree} \Rightarrow \alpha \text{ tree}$

$\text{mirror} (L \ x) = L \ x$

$\text{mirror} (N \ l \ r) = N \ (\text{mirror} \ r) \ (\text{mirror} \ l)$

$\text{lbl} :: \alpha \text{ tree} \Rightarrow \text{nat tree probability}$

$\text{lbl} (L \ _) = \text{pure } L \diamond \text{fresh}$

$\text{lbl} (N \ l \ r) = \text{pure } N \diamond \text{lbl } l \diamond \text{lbl } r$

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

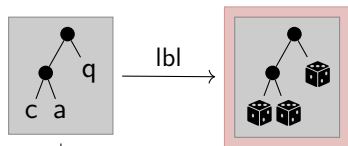
Proof by induction on  $t$ .

Case  $N \ l \ r$ :

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

The equation shows a commutative diagram with two boxes connected by a double-headed arrow. The top box is  $\text{pure } (\lambda r' \ l'. N \ (\text{mirror } r') \ (\text{mirror } l')) \diamond \text{lbl } r \diamond \text{lbl } l$  and the bottom box is  $\text{pure } (\lambda l' \ r'. \text{mirror} (N \ l' \ r')) \diamond \text{lbl } l \diamond \text{lbl } r$ . There are also arrows between the  $\text{lbl } r$  and  $\text{lbl } l$  terms in both boxes, indicating a commutative property of the effects.

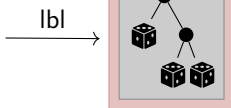
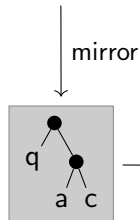
# Tree mirroring and random labels



$\text{mirror} :: \alpha \text{ tree} \Rightarrow \alpha \text{ tree}$

$\text{mirror} (\text{L } x) = \text{L } x$

$\text{mirror} (\text{N } l \ r) = \text{N} (\text{mirror } r) (\text{mirror } l)$



pure mirror

**Criterion for commutative effects:**

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

$\text{C} \quad f \ x \ y = f \ y \ x$

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

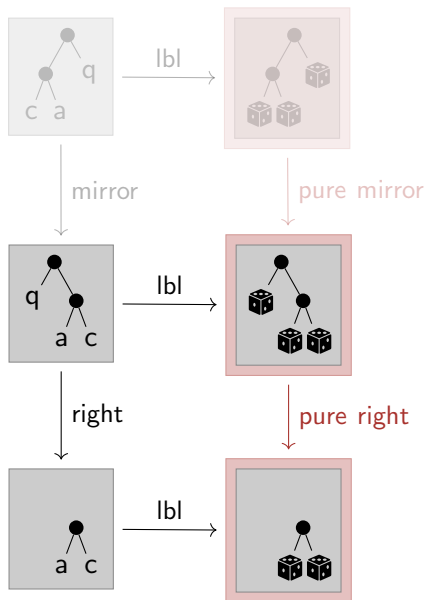
**if effects commute**

Proof by induction on  $t$ .

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

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

# Subtrees



## Lemma:

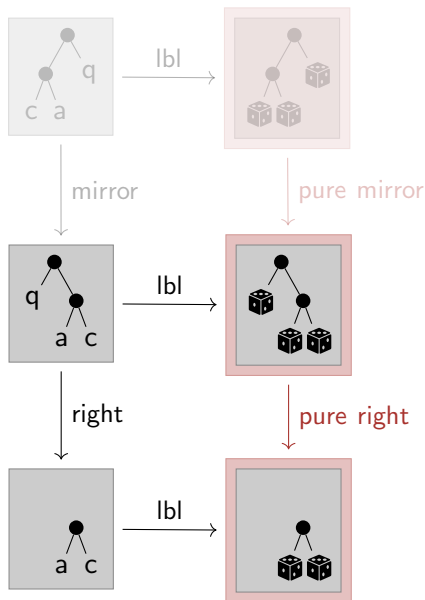
$$\text{lbl}(\text{right } t) = \text{pure right} \diamond \text{lbl } t$$

Proof by case analysis on  $t$ .

Case  $N / r$ :

$$\begin{aligned} & \text{pure}(\lambda r'. r') \quad \diamond \text{lbl } r \\ \stackrel{?}{=} & \text{pure}(\lambda_-. r'. r') \diamond \text{lbl } l \diamond \text{lbl } r \end{aligned}$$

# 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

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

Proof by case analysis on  $t$ .

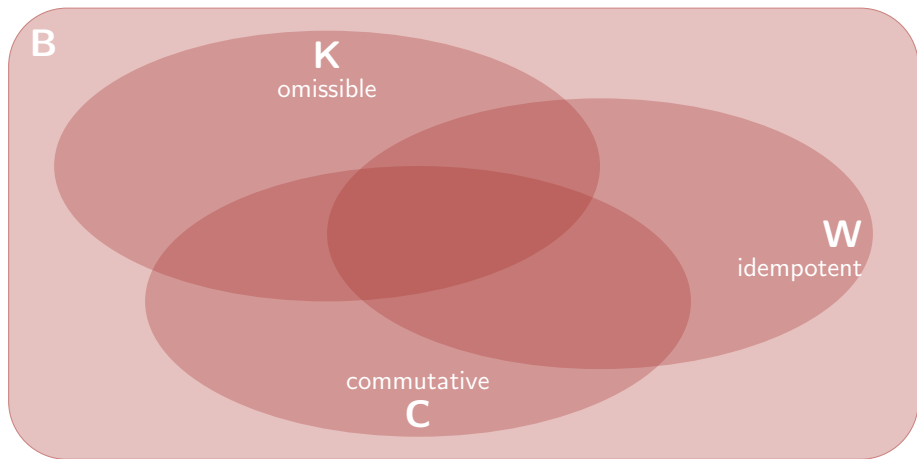
Case  $N / r$ :

$$\text{pure } (\lambda r'. r') \quad \diamond \quad lbl r$$

=

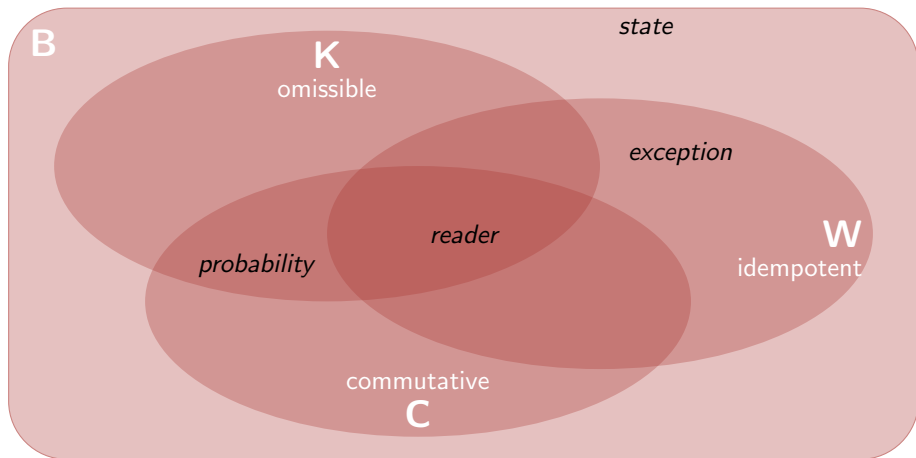
$$\text{pure } (\lambda_-. r'. r') \diamond lbl l \diamond lbl r$$

# Combinatorial basis **BCKW**



- ▶ Declarative characterisation of “liftable” equations
- ▶ Modular implementation via bracket abstraction

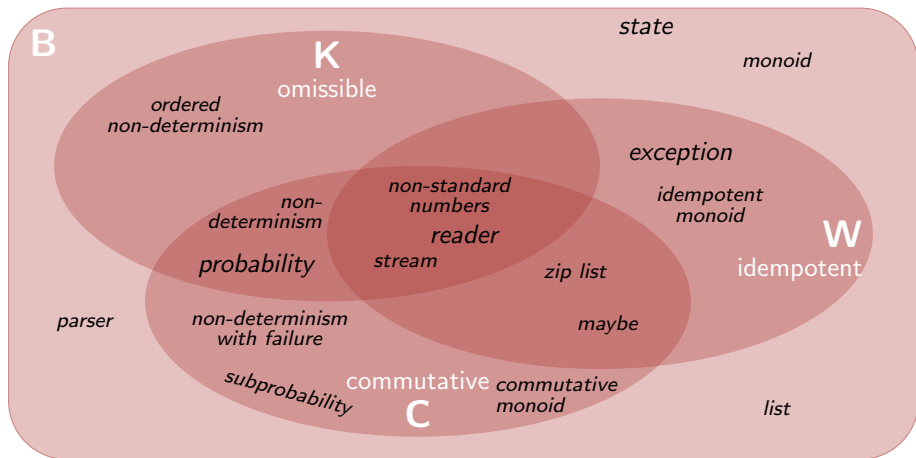
# Combinatorial basis **BCKW**



- ▶ 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](http://www.isa-afp.org/entries/Applicative_Lifting.shtml)

```
applicative state
for
  pure: pure_state
  ap: ap_state

proof (prove)
goal (4 subgoals):
1.  $\forall f x. \text{pure } f \diamond \text{pure } x = \text{pure } (f x)$ 
2.  $\forall g f x. \text{pure } (\lambda g f x. g (f x))$ 
3.  $\forall x. \text{pure } (\lambda x. x) \diamond x = x$ 
4.  $\forall f x. f \diamond \text{pure } x = \text{pure } (\lambda f. f x)$ 
```

functor registration

proof tactic

The diagram illustrates a proof tactic. At the bottom, there is a concrete state represented by a black circle, a black square, and a black star, with the text  $x =$  to the right. Above this, two blue gears are shown. A vertical double-headed arrow points from the gears to an abstract state at the top, which consists of a pink circle, a pink square, and a pink star, also with the text  $x =$  to the right. The abstract state is enclosed in a pink box.

classify effects

A Venn diagram with a large light red background labeled 'B'. Inside, there are four overlapping circles: a top circle labeled 'K', a bottom circle labeled 'C', a right circle labeled 'W', and a central circle that overlaps with all three others.

formalisation  
of the meta theory

guides

An upward-pointing arrow labeled 'guides' connects the 'formalisation of the meta theory' box to the 'proof tactic' box.

# Summary

[www.isa-afp.org/entries/Applicative\\_Lifting.shtml](http://www.isa-afp.org/entries/Applicative_Lifting.shtml)

```
applicative state
for
  pure: pure_state
  ap: ap_state

proof (prove)
goal (4 subgoals):
1.  $\forall f x. \text{pure } f \circ \text{pure } x = \text{pure } (f x)$ 
2.  $\forall g f x. \text{pure } (\lambda g f x. g (f x))$ 
3.  $\forall x. \text{pure } (\lambda x. x) \circ x = x$ 
4.  $\forall f x. f \circ \text{pure } x = \text{pure } (\lambda f. f x)$ 
```

functor registration

proof tactic

classify effects

monads?

generate?

beyond equality?

formalisation  
of the meta theory

guides