Symmetric Programming and Reasoning

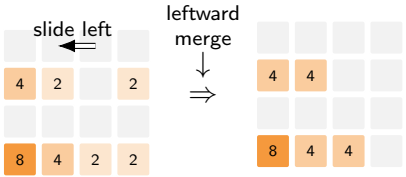
Ohad Kammar, LFCS, University of Edinburgh and
Matija Pretnar, FMF, University of Ljubljana

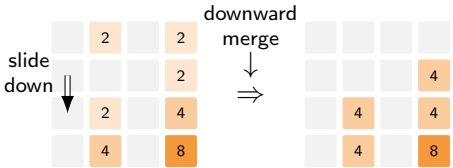
11th Workshop on
Mathematically Structured Functional Programming (MSFP 2026)
Lisbon, Portugal, 18 July 2026


Laboratory for Foundations of Computer Science


supported by:


Example: 2048 game[1] [Cirulli'14]





[1] Available to play: `https://play2048.co`

2048 in Idris 2 [Brady'21]

Merging left goes with the grain

```
Row, Board : Nat -> Type
Row    n = Vect n Int
Board n = Vect n (Row n)

(.pad0) : Row n -> Row (1 + n)
xs.pad0 = replace {p = Row}
                  (plusCommutative _ _)
          $ xs ++ [0]

mergeRow : Row n -> Row n
mergeRow []                = []
mergeRow (     0 :: xs)    = (mergeRow xs).pad0
mergeRow (     x :: [])    = [x]
mergeRow (x :: 0 :: xs)    = (mergeRow (x :: xs)).pad0
mergeRow (x :: y :: xs) with (x == y)
                        _ | True  = 2*x :: (mergeRow xs).pad0
                        _ | False =   x :: mergeRow (y :: xs)
mergeLeft : Board n -> Board n
mergeLeft = map mergeRow
```

Merging in 2048

Going against the grain

merging **up/down** incompatible with **inductive structure**

⇓

use a different representation

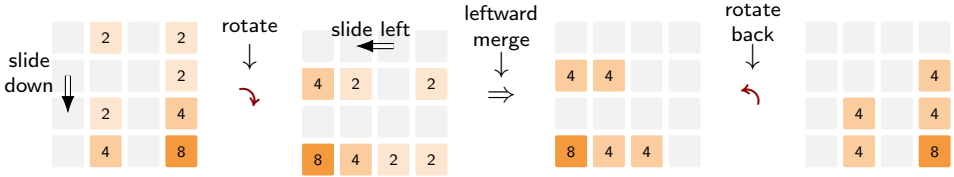
⇓

parameters galore![2]

```
  var vector     = this.getVector(direction);
  var traversals = this.buildTraversals(vector);
// ...
GameManager.prototype.getVector =
 function (direction) {// Vectors representing tile movement
  var map = { 0: { x:  0, y: -1 },  // Up
              1: { x:  1, y:  0 },  // Right
              2: { x:  0, y:  1 },  // Down
              3: { x: -1, y:  0 }}; // Left
  return map[direction];
};
```

[2] See the function `GameManager.prototype.move`:
https://github.com/gabrielecirulli/2048/blob/478b6ec346e3787f589e4af751378d06ded4cbbc/js/game_manager.js#L129.

Easy 2048 implementation using symmetry

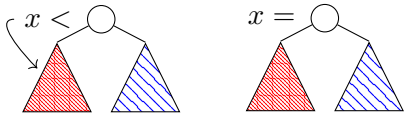


Anecdotal evidence with half-a-dozen UG students

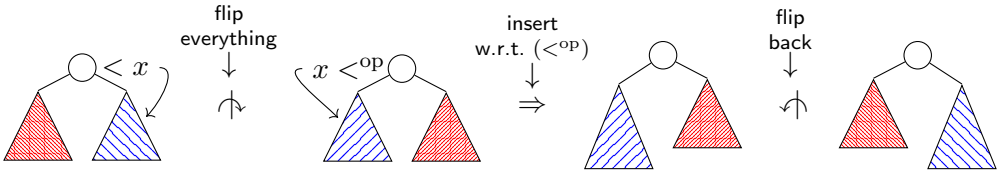
- ✗ buggy parametric solution
- ✓ mostly right symmetric solution

Example: binary search tree insertion using **symmetry**

Interesting cases



Symmetric case



NB: Yes, silly performance. We'll improve this later.

Symmetric reasoning without-loss-of-generality

Theorem (small pigeonhole principle[3])

If three balls are painted red or blue, at least two have the same color.

Proof

Without loss-of-generality, ball 1 is blue. If balls 2 and 3 are the same colour, pick them and conclude. If they have different colours, **without loss-of-generality**, ball 2 is blue. Pick balls 1 and 2 and conclude. ■

Valid proof, but dubious nesting of wlog.

Dijkstra [EWD1223, '95]: typical wlog proofs omit relevant details.
Presents a wlog principle for predicates over pairs.

Harrison ['09]: there ought to be a wlog principle.
Presents 6 wlog HOL light tactics in different domains (goal: formalisation fidelity)

[3] Adapted from the Wikipedia entry *wlog*.

Symmetric reasoning without-loss-of-generality

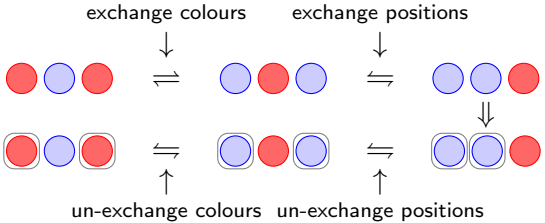
Theorem (small pigeonhole principle[3])

If three balls are painted red or blue, at least two have the same color.

Proof

Without loss-of-generality, ball 1 is blue. If balls 2 and 3 are the same colour, pick them and conclude. If they have different colours, **without loss-of-generality**, ball 2 is blue. Pick balls 1 and 2 and conclude. ■

today: symmetric wlog as group-action **conjugation** of **core constructions**



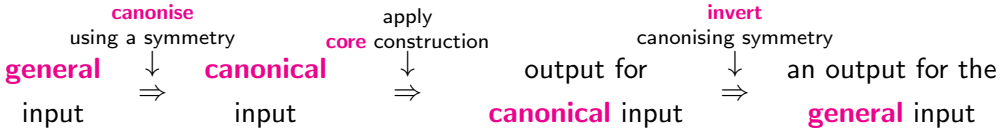
[3] Adapted from the Wikipedia entry *wlog*.

Symmetric programming

Goal: exploit **symmetry** for **computation** and **reasoning**

w.l.o.g. **anatomy**

w.l.o.g. = (input & output **symmetries**)+(**canonising** symmetries)+(**core** construction)



symmetry = **group** + **action**

Contribution

Symmetric programming

Exploit symmetries for computation and reasoning—**avoid symmetric cases**.
Mathematical foundations for symmetric programming semantics:

- **Abstractions** for symmetric programming
- **Simply-typed** reasoning about equivariance
- **Dependently-typed** and mathematical reasoning
- User-defined **data-types** through initial algebra semantics and induction principles
  - Simply-typed data-types
  - Dependently-typed/indexed data-types

Talk structure

- Introduction
- Simply-typed symmetric programming
- Further contributions
- Conclusion

Group theory basics

I'm **not** going to do:

Group G (textbook definition)

monoid $G = \langle \underline{G}, (*), e \rangle$ with inverses:

$$g * g^{-1} = e = g^{-1} * g$$

(Non)-intuition

Only helps you if you know about monoids, but anyway obscures intuition.

Basic group theory

Group action $\langle G, A \rangle$

Group $G = \langle \underline{G}, \ldots \rangle$

G-**action** $A = \langle \underline{A}, \ldots \rangle$

Intuition

- **symmetries**: $\underline{G}$ elements
- x **symmetric to** y **via** g

action and group axioms mean:

'**being symmetric**' is a
proof-relevant equivalence relation tracked by
symmetries

Rotations $\langle \overset{G:=}{\mathbf{C}_4}, \overset{=:A}{\mathsf{Dir}} \rangle$

$\underline{\mathbf{C}_4} := \{\curvearrowleft, \curvearrowright, \curvearrowright, \circlearrowright\}$

acting on

$\underline{\mathsf{Dir}} := \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$

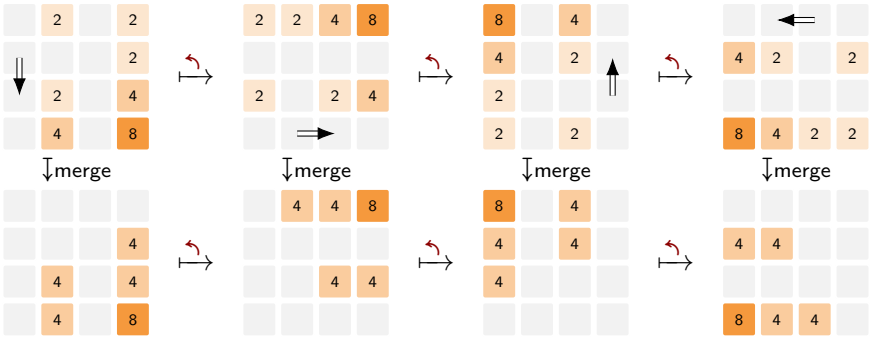
$\Uparrow$ symmetric to $\Leftarrow$ via $\curvearrowleft$
$\Downarrow$ symmetric to $\Leftarrow$ via $\curvearrowright$
$\Rightarrow$ symmetric to $\Leftarrow$ via $\curvearrowright$
$\Leftarrow$ symmetric to $\Leftarrow$ via $\circlearrowright$

Group theory basics

Equivariant map $\varphi : A \circledast\!\!\longmapsto B$ between G-actions A, B

$$\frac{x, \quad y \text{ symmetric via } g}{f\,x, f\,y \text{ symmetric via } g}$$

Example: merge : Dir $\times$ Board$_n$ $\circledast\!\!\longmapsto$ Board$_n$



Basic group theory

Group action $\langle G, A \rangle$

Group $G = \left\langle \underline{G}, (*), e, (-)^{-1} \right\rangle$:

$\underline{G}$(**carrier** set) $\quad \dfrac{g, h : \underline{G}}{g * h : \underline{G}}$(**multiplication**/**composition**)

$\dfrac{}{e : \underline{G}}$(**unit**/**identity**/**neutral**) $\quad \dfrac{g : \underline{G}}{g^{-1} : \underline{G}}$(**inverse**)

such that:

$$\mathsf{g}, \mathsf{h}, \mathsf{k} : \underline{G} \vdash (\mathsf{g} * \mathsf{h}) * \mathsf{k} = \mathsf{g} * (\mathsf{h} * \mathsf{k}) \qquad \text{(associativity)}$$

$$\mathsf{g} : \underline{G} \vdash \mathsf{g} * e = \mathsf{g} = e * \mathsf{g} \qquad \text{(neutrality)}$$

$$\mathsf{g} : \underline{G} \vdash \mathsf{g} * \mathsf{g}^{-1} = e = \mathsf{g}^{-1} * \mathsf{g} \qquad \text{(inverses)}$$

G-**action** $A = \langle \underline{A}, \ldots \rangle \ldots$

Rotations $\langle \overset{G:=}{\mathbf{C}_4}, \overset{=:A}{\mathsf{Dir}} \rangle$

$\underline{\mathbf{C}_4} := \{\circlearrowleft, \circlearrowright, \circlearrowright, \circlearrowright\}$

$\circlearrowleft * \circlearrowleft := \circlearrowright \quad \circlearrowleft * \circlearrowright := \circlearrowright$

$\circlearrowleft * \circlearrowright := \circlearrowright \quad \circlearrowleft * \circlearrowright := \circlearrowleft$

group laws imply rest and:

$$e = \circlearrowright$$

$$\circlearrowleft^{-1} = \circlearrowright \qquad \circlearrowright^{-1} = \circlearrowright$$

Basic group theory

Group action $\langle G, A \rangle$

Group $G = \left\langle \underline{G}, (*), e, (-)^{-1} \right\rangle$

G-**action** $A = \langle \underline{A}, \overset{A}{\circledast} \rangle$:

$\underline{A}$(**carrier** set) $\qquad \dfrac{g : \underline{G} \qquad x : \underline{A}}{g \circledast x : \underline{A}}$(**action** of G on $\underline{A}$)

such that:

$$\mathsf{g}, \mathsf{h} : \underline{G}, \mathsf{x} : \underline{A} \vdash \mathsf{g} \circledast (\mathsf{h} \circledast \mathsf{x}) = (\mathsf{g} * \mathsf{h}) \circledast \mathsf{x} \quad \text{(associativity)}$$

$$\mathsf{x} : \underline{A} \vdash e \circledast \mathsf{x} = \mathsf{x} \quad \text{(neutrality)}$$

Rotations $\overset{G:=}{\overset{=:A}{\langle}}\mathbf{C}_4, \mathsf{Dir}\rangle$

$\underline{\mathbf{C}_4} := \{\curvearrowleft, \curvearrowright, \curvearrowright, \circlearrowright\}$

$\curvearrowleft * \curvearrowleft := \curvearrowright \quad \curvearrowleft * \curvearrowright := \curvearrowright$

$\curvearrowleft * \curvearrowright := \circlearrowright \quad \curvearrowleft * \circlearrowright := \curvearrowleft$

acting on

$\underline{\mathsf{Dir}} := \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$

$\curvearrowleft \circledast \Leftarrow := \Downarrow \quad \curvearrowleft \circledast \Downarrow := \Rightarrow$

$\curvearrowleft \circledast \Rightarrow := \Uparrow \quad \curvearrowleft \circledast \Uparrow := \Leftarrow$

laws determine the rest

Basic group theory

Group action $\langle G, A \rangle$

Group $G = \left\langle \underline{G}, (*), e, (-)^{-1} \right\rangle$

G-**action** $A = \langle \underline{A}, \overset{A}{\circledast} \rangle$

Notation

- **symmetries**: $\underline{G}$ elements
- x **symmetric to** y **via** g when: $g \circledast x = y$

action and group axioms mean:

'**being symmetric**' is a
proof-relevant equivalence relation tracked by
symmetries

Rotations $\langle \overset{G:=}{\mathbf{C}_4}, \overset{=:A}{\mathrm{Dir}} \rangle$

$\underline{\mathbf{C}_4} := \{\curvearrowleft, \curvearrowright, \curvearrowright, \circlearrowleft\}$

$\curvearrowleft * \curvearrowleft := \curvearrowright \quad \curvearrowleft * \curvearrowright := \curvearrowright$

$\curvearrowleft * \curvearrowright := \circlearrowleft \quad \curvearrowleft * \circlearrowleft := \curvearrowleft$

acting on

$\underline{\mathrm{Dir}} := \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$

$\curvearrowleft \circledast \Leftarrow := \Downarrow \quad \curvearrowleft \circledast \Downarrow := \Rightarrow$

$\curvearrowleft \circledast \Rightarrow := \Uparrow \quad \curvearrowleft \circledast \Uparrow := \Leftarrow$

laws determine the rest

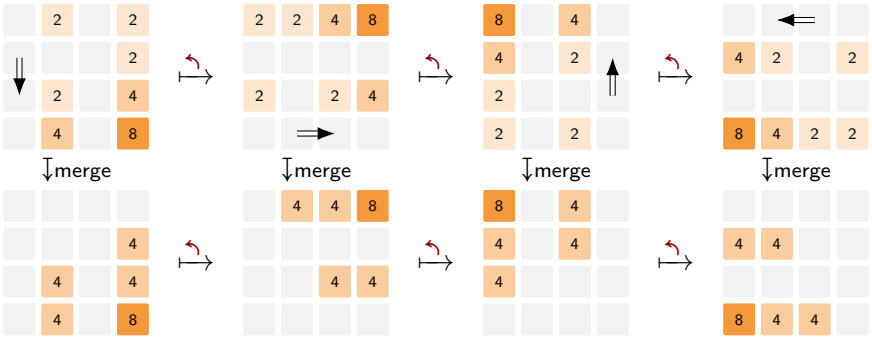
Group theory basics

Equivariant map $\varphi : A \circledast\!\!\longmapsto B$ between G-actions A, B

$$\underset{\text{function}}{\varphi : \underline{A} \to \underline{B}} \qquad \mathsf{g} : \underline{G}, \mathsf{x} : \underline{A} \vdash \varphi(\mathsf{g} \circledast \mathsf{x}) = \mathsf{g} \circledast (\varphi \mathsf{x}) : \underline{B}$$

$$\text{i.e.:} \quad \frac{x, \quad y \text{ symmetric via } g}{f\,x, f\,y \text{ symmetric via } g}$$

Example: merge : Dir $\times$ Board$_n$ $\circledast\!\!\longmapsto$ Board$_n$



w.l.o.g. anatomy: canonisation

w.l.o.g. = (input & output **symmetries**) +(**canonising** symmetries) +(**core** construction)

Canoniser for G-action A

function $c : \underline{A} \to \underline{G}$ assigning **canonising symmetries** to A-elements.

Canonical elements

$$\mathsf{Can}\, c := \{\, (c\, a) \circledast a \mid a \in \underline{A} \,\}$$

Example: mkLft : $\underline{\mathsf{Dir}} \to \underline{\mathbf{C}_4}$

Canonising symmetry makes direction 'left': mkLft $\Rightarrow$:= ↝ mkLft $\Uparrow$:= ↖ mkLft $\Leftarrow$:= ↺ mkLft $\Downarrow$:= ↘

Only canonical direction is 'left': $\mathsf{Can}\,\mathsf{mkLft} = \{\Leftarrow\}$

w.l.o.g. anatomy: core

w.l.o.g. = (input & output **symmetries**)+(**canonising** symmetries) +(**core** construction)

Core function $f : \mathrm{Can}\, c \to \underline{B}$

work on canonical elements, typically more concrete, after eliminating symmetric cases

Example: merging left

$$
\begin{aligned}
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow} && : \mathrm{Row}_n \to \mathrm{Row}_n \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \langle\rangle && := \langle\rangle \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \langle 0\rangle +\!\!+ r && := (\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ r) +\!\!+ \langle 0\rangle \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \langle x\rangle && := \langle x\rangle \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ (\langle x\ , 0\ \rangle +\!\!+ r) && := \mathrm{merge}_{\Leftarrow}\ (\langle x\rangle +\!\!+ r) +\!\!+ \langle 0\rangle \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \left(\left\langle 2^k, 2^k\right\rangle +\!\!+ r\right) && := \left\langle 2^{k+1}\right\rangle +\!\!+ (\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ r) +\!\!+ \langle 0\rangle \\
&\mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \left(\left\langle 2^k, 2^\ell\right\rangle +\!\!+ r\right) && := \left\langle 2^k\ \right\rangle +\!\!+ \mathrm{merge}^{\mathrm{row}}_{\Leftarrow}\ \left(\left\langle 2^\ell\right\rangle +\!\!+ r\right)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{merge}_{\Leftarrow} : \mathrm{Board}_n \to \mathrm{Board}_n \\
&\mathrm{merge}_{\Leftarrow} := \mathrm{map}\ \mathrm{merge}^{\mathrm{row}}_{\Leftarrow}
\end{aligned}
$$

Symmetric programming **abstraction**

w.l.o.g. = (input & output **symmetries**)+(**canonising** symmetries)+(**core** construction)

W.l.o.g. construction

Formally:

$$\frac{G\text{-actions } A, B \qquad c : \underline{A} \to \underline{G} \qquad f : \mathsf{Can}\, c \to \underline{B}}{\mathsf{wlog}(c, f) \coloneqq \left(a \mapsto (ca)^{-1} \circledast f((ca) \circledast a)\right) : \underline{A} \to \underline{B}}$$

Example

```
mergeUnchecked : {n : Nat} -> Dir -> Board n -> Board n
mergeUnchecked = Prelude.curry $ wlog
        (\(dir, _) => makeLeft dir)                    -- canoniser
        (\case (Lt, board) => mergeLeft board          -- core function
               _ => believe_me "oops")                 --   unreachable case
```

Symmetric programming **abstraction**

w.l.o.g. = (input & output **symmetries**)+(**canonising** symmetries)+(**core** construction)

W.l.o.g. construction

Formally:

$$\frac{G\text{-actions } A, B \qquad c : \underline{A} \to \underline{G} \qquad f : \mathsf{Can}\, c \to \underline{B}}{\mathsf{wlog}(c, f) \coloneqq \left(a \mapsto (ca)^{-1} \circledast f((ca) \circledast a)\right) : \underline{A} \to \underline{B}}$$

Example

```
core : {n : Nat} -> Canon (Canoniser {n}) -> Board n
core  (( d, b) ** ev) with 0 (canonicalView ev)  -- McBride-McKinna view
 core ((Lt, b) ** ev) | Refl = mergeLeft b

merge : {n : Nat} -> Dir -> Board n -> Board n
merge = Prelude.curry $ wlog' Canoniser core
```

Equivariant where defined core functions

Assume an equivariant $\varphi : A \circledast\!\!\rightarrow B$ extends a core function $f : \mathsf{Can}\, c \to \underline{B}$. Then:

$$\begin{array}{ccccc} & \varphi \text{ extends } f & & \varphi \text{ is equivariant} & & \varphi \text{ extends } f & \\ & \downarrow & & \downarrow & & \downarrow & \\ f(g \circledast x) & = & \varphi(g \circledast x) & = & g \circledast \varphi(x) & = & g \circledast fx \end{array}$$

Equivariant where defined core functions

Assume an equivariant $\varphi : A \circledast\!\!\rightarrow B$ extends a core function $f : \mathsf{Can}\, c \to \underline{B}$. Then:

$$\begin{array}{ccccc} & \varphi \text{ extends } f & & \varphi \text{ is equivariant} & & \varphi \text{ extends } f & \\ & \downarrow & & \downarrow & & \downarrow & \\ f(g \circledast x) & = & \varphi(g \circledast x) & = & g \circledast \varphi(x) & = & g \circledast fx \end{array}$$

Let A, B be G-actions and $X \subseteq \underline{A}$. Then, $f : X \to \underline{B}$ is **equivariant** when

$$\forall g \in \underline{G}. \forall x \in X. (g \overset{A}{\circledast} x \in X \implies f(g \overset{A}{\circledast} x) = g \overset{B}{\circledast} fx)$$

Equivariant where defined core functions

Assume an equivariant $\varphi : A \circledast\!\!\rightarrow B$ extends a core function $f : \mathsf{Can}\, c \to \underline{B}$. Then:

$$\begin{array}{ccccc} & \varphi \text{ extends } f & & \varphi \text{ is equivariant} & & \varphi \text{ extends } f & \\ & \downarrow & & \downarrow & & \downarrow & \\ f(g \circledast x) & = & \varphi(g \circledast x) & = & g \circledast \varphi(x) & = & g \circledast fx \end{array}$$

Let A, B be G-actions and $X \subseteq \underline{A}$. Then, $f : X \to \underline{B}$ is **equivariant** when

$$\forall g \in \underline{G}. \forall x \in X. (g \overset{A}{\circledast} x \in X \implies f(g \overset{A}{\circledast} x) = g \overset{B}{\circledast} fx)$$

Theorem

Given:
G-actions A, B; canoniser $c : \underline{A} \to \underline{G}$; equivariant core function $f : (\mathsf{Can}\, c \subseteq A) \circledast\!\!\rightarrow B$;
Then $\mathsf{wlog}(c, f) : A \circledast\!\!\rightarrow B$ *is the unique equivariant function extending f.*

Symmetric programming

Simply-typed w.l.o.g.

- Straightforward to implement
- Straightforward to use
- Plumbing group actions with type-classes/interfaces fiddly as usual
- Plenty of unreachable cases
- Doesn't cover mathematical reasoning

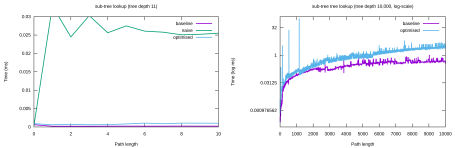
Straightforward extension to dependent types via indexed families.

Further contributions

- Dependent-types
- Data-types (simply-typed + dependently-typed)

simply-typed concept	dependently-typed concept	categorical concept
$\mathbf{Set}$	$\mathbf{Fam}_{\underline{\Gamma}}$	category $\mathcal{C}$
functor $F : \mathbf{Set} \to \mathbf{Set}$	functor $F : \mathbf{Fam}_{\underline{\Gamma}} \to \mathbf{Fam}_{\underline{\Gamma}}$	functor $F : \mathcal{C} \to \mathcal{C}$
free G-action monad	free Γ-indexed G-action monad	monad structure T over
$\underline{T}X \coloneqq \underline{G} \times X$	$\underline{T}(\underline{\Gamma} \vdash X) \coloneqq \underline{\Gamma} \vdash G \boxtimes X$	$\underline{T} : \mathcal{C} \to \mathcal{C}$
G-action	Γ-indexed G-action	T-algebra
G-strength st : $\underline{G} \times FX \to F(\underline{G} \times X)$	Γ-indexed G-strength $\mathrm{st} : G \boxtimes (FX) \to F(G \boxtimes X)$	monad-functor distributive law $\mathrm{st} : TF \to FT$
functor $S\underline{X} \coloneqq \underline{G} \to X$	functor $S(\underline{\Gamma} \vdash X) \coloneqq \underline{\Gamma} \vdash G \multimap X$	right-adjoint $\underline{T} \dashv S$
parameterised traversal $\underline{G} \times \boldsymbol{\mu} F \xrightarrow{\mathbf{pfold}\left(F X \xrightarrow{f} X\right)} X$	indexed parameterised traversal $G \boxtimes \boldsymbol{\mu} F \xrightarrow{\mathbf{pfold}\left(F X \xrightarrow{f} X\right)} X$	parameterised fold $T \boldsymbol{\mu} F \xrightarrow{\mathbf{pfold}\left(F X \xrightarrow{f} X\right)} X$
equivariant F-algebra $f : FA \looparrowright A$	indexed equivariant F-algebra $f : FA \looparrowright A$	F-algebra for lifted functor $F : \mathcal{C}^T \to \mathcal{C}^T$
canoniser $c : F \to \underline{G}$	indexed canoniser $c : F \to \underline{G}$	
canonical elements $\mathrm{Can}\, c : G\text{-}\mathbf{Act} \to \mathbf{Set}$	canonical elements $\mathrm{Can}\, c : G\text{-}\mathbf{Act}_{\Gamma} \to \mathbf{Fam}_{\underline{\Gamma}}$	canonical elements $\mathrm{Can}\, c : \mathcal{C}^T \to \mathcal{C}$

- Performance engineering and prototypes (Lean and Idris 2):



Road-map

Future work

- more examples:
  - Harrison's w.l.o.g. tactics
  - McBride's binary search trees
  - Steinitz exchange lemma
- connections: nominal techniques and symmetry breaking
- philosophy and history of mathematics:
  - comparison with vernacular and mechanised use
  - comparison with Dijkstra's wlog principle
- implementation:
  - ergonomic abstractions
  - systematise performance engineering

Contribution

Symmetric programming

Exploit symmetries for computation and reasoning—**avoid symmetric cases**.
Mathematical foundations for symmetric programming semantics:

- **Abstractions** for symmetric programming
- **Simply-typed** reasoning about equivariance
- **Dependently-typed** and mathematical reasoning
- User-defined **data-types** through initial algebra semantics and induction principles
  - Simply-typed data-types
  - Dependently-typed/indexed data-types