Mathematical Foundations for Symmetric Programming

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

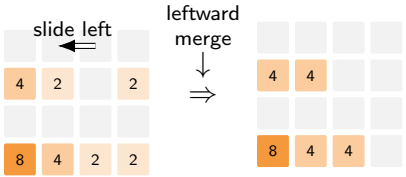
Shameless PLUG
(Programming Languages at University of Glasgow Seminar)
27 May 2026

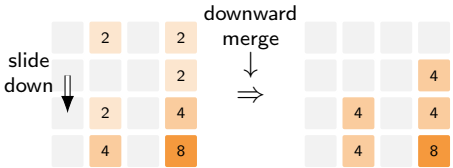
THE UNIVERSITY of EDINBURGH informatics lfcs | Laboratory for Foundations of Computer Science
BayesCentre supported by: THE ROYAL SOCIETY | Advanced Research + Invention Agency ARIA

Symmetric programming

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

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

```
(.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

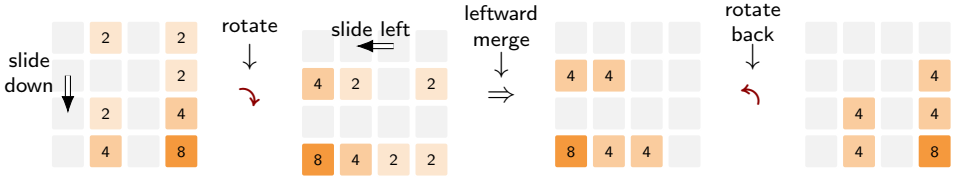
⇓

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

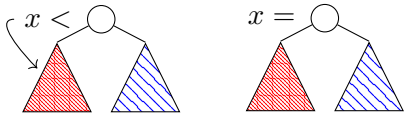


Anecdotal evidence with half-a-dozen UG students

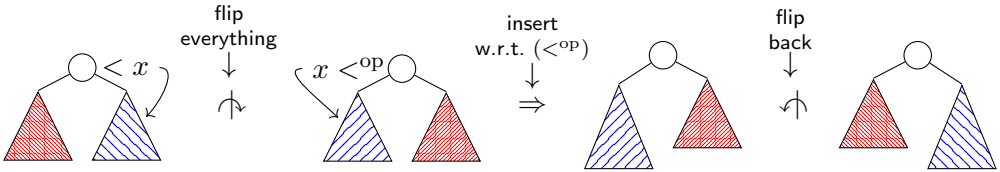
- ✗ buggy parametric solution
- ✓ mostly right symmetric solution

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

Interesting cases



Symmetric case



NB: Yes, silly performance. Won't improve performance today.

Example: **proofs** using **symmetry** [adapted from Harrison'09]

Theorem (Schur's inequality)

Then, for all $x_1, x_2, x_3 > 0$, we have, for all t:

$$x_1^t(x_1 - x_2)(x_1 - x_3) + x_2^t(x_2 - x_3)(x_2 - x_1) + x_3^t(x_3 - x_1)(x_3 - x_2) \geq 0$$

Proof

Without loss of generality, assume $x_1 \geq x_2 \geq x_3$. Then:

$$\begin{aligned} x_1^t(x_1 - x_2)(x_1 - x_3) + x_2^t(x_2 - x_3)(x_2 - x_1) + x_3^t(x_3 - x_1)(x_3 - x_2) = \\ (x_1 - x_2)\left(x_1^t(x_1 - x_3) - x_2^t(x_2 - x_3)\right) + x_3^t(x_1 - x_3)(x_2 - x_3) \geq \\ (x_1 - x_2)(x_1^t - x_2^t)(x_1 - x_3) + x_3^t(x_1 - x_3)(x_2 - x_3) \geq 0 \end{aligned}$$

■

Example: **proofs** using **symmetry** [adapted from Harrison'09]

Theorem (Schur's inequality)

Then, for all $x_1, x_2, x_3 > 0$, we have, for all t:

$$x_1^t(x_1 - x_2)(x_1 - x_3) + x_2^t(x_2 - x_3)(x_2 - x_1) + x_3^t(x_3 - x_1)(x_3 - x_2) \geq 0$$

Proof

Without loss of generality, assume $x_1 \geq x_2 \geq x_3$. Then:

$$\begin{aligned} x_1^t(x_1 - x_2)(x_1 - x_3) + x_2^t(x_2 - x_3)(x_2 - x_1) + x_3^t(x_3 - x_1)(x_3 - x_2) = \\ (x_1 - x_2)\left(x_1^t(x_1 - x_3) - x_2^t(x_2 - x_3)\right) + x_3^t(x_1 - x_3)(x_2 - x_3) \geq \\ (x_1 - x_2)(x_1^t - x_2^t)(x_1 - x_3) + x_3^t(x_1 - x_3)(x_2 - x_3) \geq 0 \end{aligned}$$

■

Rocq SSReflect[3]

The general syntax of without loss is:

```
wlog suff? clear_switch? i_item? : ident* / term
```

On a goal `G`, it creates two subgoals: a first one to prove the formula `(term -> G) -> G` and a second one to prove the formula `term -> G`. If the optional list of ident is present on the left side of `/`, these constants are generalized in the premise `(term -> G)` of the first subgoal.

[3] Rocq Reference Manual:
https://rocq-prover.org/doc/V9.2.0/refman/proof-engine/ssreflect-proof-language.html#rocq:tacn.wlog

w.l.o.g. mechanisations

Agda[4]

```
-- Without Loss of Generality
module _  {_R_ : Rel A ℓ1} {Q : Rel A ℓ2} where

wlog : Total _R_ → Symmetric Q →
       (∀ a b → a R b → Q a b) →
       ∀ a b → Q a b
```

[4] Agda Stdlib:
https://github.com/agda/agda-stdlib/blob/c2abce1b8e72ccae49f73791d0996eaf23674357/src/Relation/Binary/Consequences.agda#L197

w.l.o.g. mechanisations

Lean4[5]

The tactic `wlog h : P` will add an assumption `h : P` to the main goal, and add a new goal that requires showing that the case `¬h : P` can be reduced to the case where **P** holds (typically by symmetry).

[5] Mathlib4 documentation:
https://leanprover-community.github.io/mathlib4_docs/Mathlib/Tactic/WLOG.html

uniform w.l.o.g. using **symmetry**

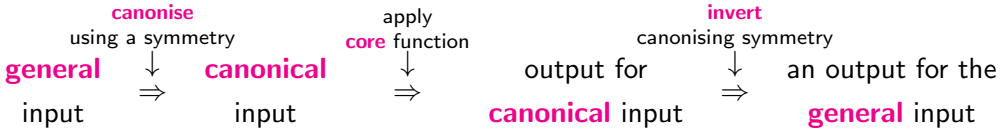
$$x_1, x_2, x_3 \in \mathbb{R}_+ \quad \overset{\text{sort using } \pi}{\Rightarrow} \quad x_{\pi 1} \geq x_{\pi 2} \geq x_{\pi 3} \quad \overset{\text{prove using additional assumption}}{\Rightarrow} \quad P(x_{\pi 1}, x_{\pi 2}, x_{\pi 3}) \quad \overset{\text{permute back since } P \text{ is invariant}}{\Rightarrow} \quad P(x_1, x_2, x_3)$$

NB: revisionism:

- ✓ Dijkstra'95 [EWD1223, not mechanised]
- ✓ HOL WLOG tactics [Harrison'09]
- ✗ Rocq SSReflect
- ✓ Agda Stdlib
- ✗ Lean

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

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



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
- Group theory basics
- Simply-typed symmetric programming
- Dependently-typed symmetric programming
  - Indexed actions and w.l.o.g.
  - Mathematical reasoning
- Conclusion

Not today / offline

- Dijkstra'95 & Harrison'09 w.l.o.g. principles
- Symmetric user-defined data-types
  - Simply-typed data-types
  - Indexed data-types
- Equivariant reasoning

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.

Group theory basics

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

$G{:=} \quad {=:}A$

Board rotations $\langle \mathbf{C}_4, \mathrm{Dir} \rangle$

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

acting on

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

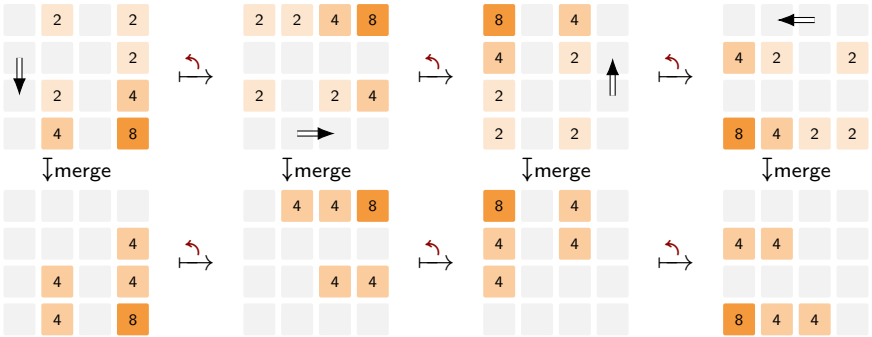
$\Uparrow$ symmetric to $\Leftarrow$ via $\curvearrowleft$
$\Downarrow$ symmetric to $\Leftarrow$ via $\curvearrowright$
$\Rightarrow$ symmetric to $\Leftarrow$ via $\curvearrowright$
$\Leftarrow$ symmetric to $\Leftarrow$ via $\circlearrowright$

Group theory basics

Equivariant map $\varphi : A \circledast\!\!\longrightarrow B$ 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$_n$ $\circledast\!\!\longrightarrow$ Board$_n$



Group theory basics

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:

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

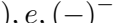
$$\mathsf{g} : \underline{G} \vdash \mathsf{g} * e = \mathsf{g} = e * \mathsf{g} \qquad \text{(neutrality)}$$

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

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

Board rotations $\langle \mathbf{C}_4, \mathrm{Dir} \rangle$ ($G :=$, $=: A$)

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



group laws imply rest and:

$$e = \text{}$$

Group theory basics

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:

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

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

Board rotations $\langle \underline{\mathbf{C}_4}, \mathsf{Dir} \rangle$ ($G := \underline{\mathbf{C}_4}$, $A := \mathsf{Dir}$)

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

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

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

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 acting similarly on: Board_n and $\mathsf{Dir} \times \mathsf{Board}_n$

Group theory basics

Group action $\langle G, A \rangle$

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

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

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

$G{:=} \quad {=:}A$

Board rotations $\langle \mathbf{C}_4, \mathrm{Dir} \rangle$

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

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

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

acting on

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

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

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

laws determine the rest
acting similarly on:
Board_n and $\mathrm{Dir} \times \mathrm{Board}_n$

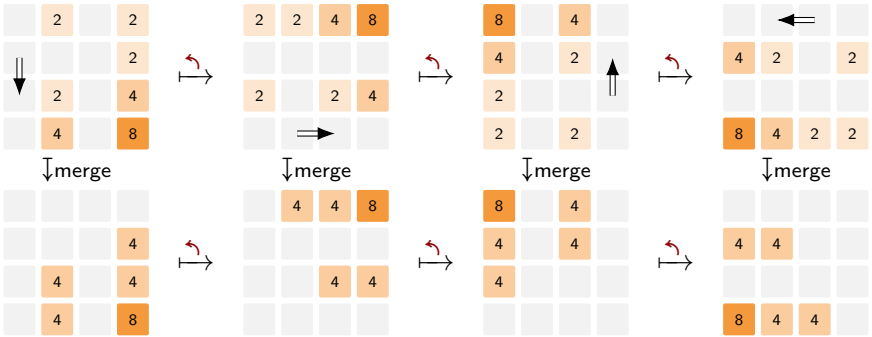
Group theory basics

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

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

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

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



Talk structure

- Introduction
- Group theory basics
- Simply-typed symmetric programming
- Dependently-typed symmetric programming
  - Indexed actions and w.l.o.g.
  - Mathematical reasoning
- Conclusion

Not today / offline

- Dijkstra'95 & Harrison'09 w.l.o.g. principles
- Symmetric user-defined data-types
  - Simply-typed data-types
  - Indexed data-types
- Equivariant reasoning

w.l.o.g. anatomy: canonisation

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

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':

$\mathsf{mkLft} \Rightarrow := \curvearrowright$ $\quad \mathsf{mkLft} \Uparrow := \curvearrowleft$

$\mathsf{mkLft} \Leftarrow := \circlearrowleft$ $\quad \mathsf{mkLft} \Downarrow := \curvearrowright$

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

w.l.o.g. anatomy: canonisation

Canonical elements

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

Lemma

$$\mathsf{Can}\left(\underline{A} \times \underline{B} \xrightarrow{\pi_1} \underline{A} \xrightarrow{c} \underline{G}\right) = \mathsf{Can}\left(\underline{A} \xrightarrow{c} \underline{G}\right) \times \underline{B}$$

Example

Canonise merge-inputs by canonising the direction:

$$c : \underline{\mathsf{Dir} \times \mathsf{Board}_n} \xrightarrow{\pi_1} \underline{\mathsf{Dir}} \xrightarrow{\mathsf{mkLft}} \underline{\mathbf{C}_4}$$

By this Lemma, in canonical merge-inputs, merge-direction left but board unrestricted:

w.l.o.g. anatomy: core

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

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

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

Example: merging left

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

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

Symmetric programming **abstraction**

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

W.l.o.g. construction

Formally:

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

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** function/proof)

W.l.o.g. construction

Formally:

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

Example

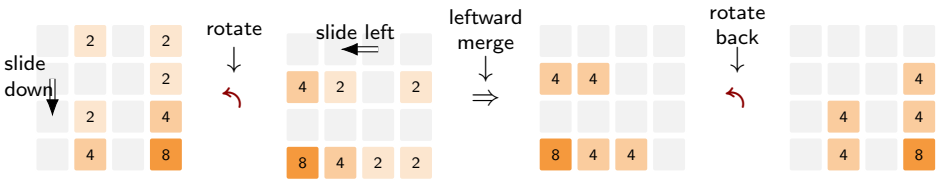
```
core : {n : Nat} -> Canon (Canoniser {n}) -> Board n
core  (( d, b) ** ev) with 0 (canonicalView ev)
 core ((Lt, b) ** ev) | Refl = mergeLeft b

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

Symmetric programming in Idris 2 [Brady'21]

Simply-typed w.l.o.g.

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



TwentyFortyEight.idr

Talk structure

- Introduction
- Group theory basics
- Simply-typed symmetric programming
- Dependently-typed symmetric programming
  - Indexed actions and w.l.o.g.
  - Mathematical reasoning
- Conclusion

Not today / offline

- Dijkstra'95 & Harrison'09 w.l.o.g. principles
- Symmetric user-defined data-types
  - Simply-typed data-types
  - Indexed data-types
- Equivariant reasoning

Back to mathematics

these concepts apply to **propositions** & **proofs**

concepts apply **uniformly** via dependent types:

types as propositions; programs as proofs

$\wr$

G-**actions** on propositions; **canonisers** and **core functions** in wlog proofs

easy with **simple types** (HOL/FOL), **duplicating** concepts in different **kinds/stages** (terms vs. propositions, object-level vs. meta-level)

analyse extant w.l.o.g. principles of Dijkstra & Harrison

Explicating type dependency

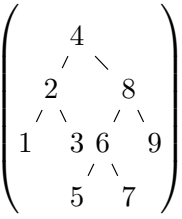
Family $\Gamma \vdash X$

- Γ set of **indices**
- X a family assigning a set X_γ to each index $\gamma \in \Gamma$—the **fibre** over γ

Example: internal tree nodes $\mathsf{t} : \mathsf{Tree}\, S \vdash \mathsf{Node}\, \mathsf{t}$

- index $\mathsf{Tree}\, S \ni t$ binary tree with labels in S;
- fibre $\mathsf{Node}\, t$: paths ending in internal node of $\mathsf{Tree}\, S \ni t$.

$\mathsf{R} :: \mathsf{L} :: \mathsf{Nil} \in \mathsf{Node}$



Fibres collectively form the smallest sets of lists over $\{\mathsf{L}, \mathsf{R}\}$ containing:

$$\frac{}{\mathsf{Nil} \in \mathsf{Node}} \qquad \frac{p \in \mathsf{Node}\, \ell}{(\mathsf{L} :: p) \in \mathsf{Node}} \qquad \frac{p \in \mathsf{Node}\, r}{(\mathsf{R} :: p) \in \mathsf{Node}}$$

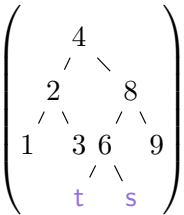
Explicating type dependency

Term/fibred element $\Gamma \vdash M : X$

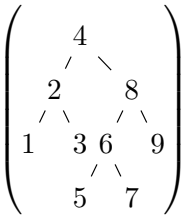
- family $\Gamma \vdash X$
- M a sequence assigning an element $M_\gamma \in X_\gamma$ at every fibre over $\gamma \in \Gamma$

Example

$\mathsf{t}, \mathsf{s} : \mathsf{Tree}\,\mathbb{N} \vdash \mathsf{R} :: \mathsf{L} :: \mathsf{Nil} : \mathsf{Node}$



element at fibre $\mathsf{t} \mapsto 5, \mathsf{s} \mapsto 7$: $\mathsf{R} :: \mathsf{L} :: \mathsf{Nil} \in \mathsf{Node}$



Formalising type dependency

Abstractions for type dependency:

$$\frac{\Gamma \vdash X}{\Gamma, \mathsf{x} : X \text{ set}} \qquad \frac{\theta : \Gamma \to \Delta \qquad \Delta \vdash Y}{\Gamma \vdash Y\,[\theta]} \qquad \frac{\Gamma \vdash X}{\mathsf{wkn} : (\Gamma, \mathsf{x} : X) \to \Gamma} \qquad \frac{\Gamma \vdash X}{\Gamma, \mathsf{x} : X \vdash \mathsf{x} : X\,[\mathsf{wkn}]}$$

$$\frac{\theta : \Delta \to \Gamma \qquad \Delta \vdash M : X\,[\theta]}{(\theta, \mathsf{x} \mapsto M) : \Delta \to (\Gamma, \mathsf{x} : X)} \qquad \frac{\Gamma \vdash X \qquad \Gamma, \mathsf{x} : X \vdash Y}{\Gamma \vdash (\mathsf{x} : X) \times Y} \qquad \frac{\Gamma \vdash M : X \quad \Gamma \vdash K : Y\,[\mathsf{x} \mapsto M]}{\Gamma \vdash M, K : (\mathsf{x} : X) \times Y}$$

$$\frac{\Gamma \vdash M : (\mathsf{x} : X) \times Y}{\Gamma \vdash M.\mathsf{fst} : X} \qquad \frac{\Gamma \vdash M : (\mathsf{x} : X) \times Y}{\Gamma \vdash M.\mathsf{snd} : Y\,[\mathsf{x} \mapsto M.\mathsf{fst}]} \qquad \frac{\Gamma \vdash X \;\; \Gamma, \mathsf{x} : X \vdash Y}{\Gamma \vdash (X : \mathsf{x}) \to Y}$$

$$\frac{\Gamma \vdash K : (X : \mathsf{x}) \to Y \qquad \Gamma \vdash M : X}{\Gamma \vdash K\,M : Y\,[\mathsf{x} \mapsto M]} \qquad \frac{\Gamma, \mathsf{x} : X \vdash M : Y}{\Gamma \vdash (\lambda \mathsf{x} : X.M) : (\mathsf{x} : X) \to Y}$$

Formalising type dependency

Families defer type-dependency difficulties to meta-level:

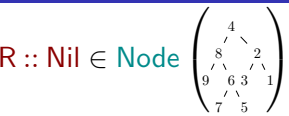
- Dependent-pairs/Grothendieck construction $\coprod_\Gamma X \coloneqq \{\gamma, x \mid \gamma \in \Gamma, x \in X_\gamma\}$
- Dependent-functions/products $\prod_\Gamma X$

Dependently-typed actions

Indexed G-action $\Gamma \vdash A$

- $\Gamma = \langle \underline{\Gamma}, (\circledast) \rangle$ **indexing** G-action
- $\underline{\Gamma} \vdash \underline{A}$ **carrier** family
- $$\frac{\vdash g : \underline{G} \qquad \gamma : \Gamma \vdash a : \underline{A}}{\gamma : \Gamma \vdash g \underset{\gamma}{\circledast} a : \underline{A}\left[\gamma \mapsto g \circledast \gamma\right]}$$ Γ-**indexed action** of G on $\underline{A}$

$$\curvearrowright\!\circledast\,(\mathsf{R} :: \mathsf{L} :: \mathsf{Nil}) = \mathsf{L} :: \mathsf{R} :: \mathsf{Nil} \in \mathsf{Node}$$



such that

$$\gamma : \Gamma, \mathsf{x} : \underline{A} \vdash e \underset{\gamma}{\circledast} \mathsf{x} = \mathsf{x} \qquad \mathsf{g}, \mathsf{k} : \underline{G}, \gamma : \underline{\Gamma}, \mathsf{x} : \underline{A} \vdash (\mathsf{g} * \mathsf{k}) \underset{\gamma}{\circledast} \mathsf{x} = \mathsf{g} \underset{\mathsf{k} \circledast \gamma}{\circledast} (\mathsf{k} \underset{\gamma}{\circledast} \mathsf{x})$$

equivalently: $\quad (\circledast) : (\mathsf{g} : \underline{G}) \to (\gamma : \Gamma) \to \underline{A}_{\gamma} \to \underline{A}_{\mathsf{g} \circledast \gamma}$

Proposition

Indexed actions amount to equivariant projections from the Grothendieck construction:

$$\underset{\text{indexed } G\text{-action}}{\Gamma \vdash A} \iff \underset{G\text{-action}}{\coprod_{\Gamma} A = (\coprod_{\underline{\Gamma}} \underline{A}, ((\circledast), (\circledast)))} \qquad \pi_1 : \coprod_{\Gamma} A \circledast\!\!\to \Gamma$$

Indexed w.l.o.g. anatomy: canonisers

Indexed canoniser for G-action $\Gamma \vdash A$

term $\underline{\Gamma} \vdash c : \underline{A} \to \underline{G}$

induced **family** $\underline{\Gamma} \vdash \mathsf{Can}\, c$ of canonical elements:

$$(\mathsf{Can}\, c)_\gamma := \left\{ \left. {}_{g^{-1} \circledast \gamma}(g \circledast a) \,\right|\, g \in \underline{G}, a \in \underline{A}_{g^{-1} \circledast \gamma}, c_{g^{-1} \circledast \gamma} a = g \right\}$$

Example

A canoniser for paths indexed by binary trees with action of $\mathbf{S}_2 := \{\mathtt{I}, \curvearrowright\}$:

$\mathsf{t} : \mathsf{Tree}\, \mathbb{N} \vdash c : \mathsf{Node}\, \mathsf{t} \to \mathbf{S}_2$

$$\begin{aligned} c_{\mathsf{Node}\langle \ell, n, r\rangle}\, \mathsf{Nil} &:= \mathtt{I} \\ c_{\mathsf{Node}\langle \ell, n, r\rangle}(\mathsf{L} :: p) &:= \mathtt{I} \\ c_{\mathsf{Node}\langle \ell, n, r\rangle}(\mathsf{R} :: p) &:= \curvearrowright \end{aligned}$$

Canonical paths may only turn R after first step:

$$(\mathsf{Can}\, c)_{\mathsf{Node}\langle \ell, n, r\rangle} = \{\mathsf{Nil}\} \cup \{\mathsf{L} :: p \mid p \in \mathsf{Node}\, \ell\}$$

Indexed w.l.o.g. anatomy: core functions

Indexed core function

term $\underline{\Gamma} \vdash f : (\mathsf{x} : \mathsf{Can}\, c) \to \underline{B}_{\mathsf{x}}$

Example

core function returning node pointed at by a path:

$$\begin{aligned} \mathsf{t} : \mathsf{Tree}\,\mathbb{N} \vdash \mathsf{lookup}^{\mathrm{Core}} &: \mathsf{Can}\, c \to \mathbb{N} \\ \mathsf{lookup}^{\mathrm{Core}}_{\mathsf{Node}\langle \ell, \mathsf{m}, r\rangle}\, \mathsf{Nil} &\coloneqq \mathsf{m} \\ \mathsf{lookup}^{\mathrm{Core}}_{\mathsf{Node}\langle \ell, \mathsf{m}, r\rangle}\, (\mathsf{L} :: p) &\coloneqq \mathsf{lookup}_{\ell} p \end{aligned}$$

✓ **eliminated** symmetric case

✗ **recursive** call to lookup, not $\mathsf{lookup}^{\mathrm{Core}}$
revisit through **data-types** (not today)

Symmetric programming **dependently-typed** abstraction

indexed w.l.o.g. = (**indexed** symmetries)+(**indexed** canoniser)+(**indexed** core function)

W.l.o.g. construction

Formally:

$$\frac{G\text{-actions } \Gamma \vdash A \qquad \Gamma, \mathsf{x} : A \vdash B \qquad \Gamma \vdash c : \underline{A} \to \underline{G} \qquad \Gamma \vdash f : \mathsf{Can}\, c \to \underline{B}}{\gamma : \underline{\Gamma} \vdash \mathsf{wlog}(c, f) \coloneqq \left(\lambda a.(ca)^{-1} \underset{(ca) \circledast \gamma}{\circledast} f((ca) \underset{\gamma}{\circledast} a) \right) : (\mathsf{x} : \underline{A}) \to \underline{B}}$$

Example (revisit later)

$$\mathsf{t} : \mathsf{Tree}\, \mathbb{N} \vdash \mathsf{lookup} \coloneqq \mathsf{wlog}(c, \mathsf{lookup}^{\mathrm{Core}}) : \mathsf{Node}\, \mathsf{t} \to \mathbb{N}$$

Talk structure

- Introduction
- Group theory basics
- Simply-typed symmetric programming
- Dependently-typed symmetric programming
  - Indexed actions and w.l.o.g.
  - Mathematical reasoning
- Conclusion

Not today / offline

- Dijkstra'95 & Harrison'09 w.l.o.g. principles
- Symmetric user-defined data-types
  - Simply-typed data-types
  - Indexed data-types
- Equivariant reasoning

Mathematical reasoning w.l.o.g.

Propositions as types, proofs as programs

For this part, take propositions $\varphi \in \mathsf{Prop} := \{\mathsf{True}, \mathsf{False}\}$. Use trivial action as needed.

Prop acts as a universe with:

$$\frac{\Gamma \vdash \varphi : \mathsf{Prop}}{\Gamma \vdash \varphi} \qquad \varphi_\gamma := \begin{cases} \varphi_\gamma = \mathsf{True} : & \mathbb{1} \\ \text{otherwise} : & \emptyset \end{cases}$$

E.g.: $(\mathsf{x}, \mathsf{y}, \mathsf{z}) : (0, \infty)^3 \vdash \mathsf{Schur} :=$

$$\forall t : \mathbb{R}. x_1^t(x_1 - x_2)(x_1 - x_3) + x_2^t(x_2 - x_3)(x_2 - x_1) + x_3^t(x_3 - x_1)(x_3 - x_2) \geq 0 : \mathsf{Prop}$$

At most one term $\Gamma \vdash M : \varphi$. It exists iff φ holds in every fibre $\gamma \in \Gamma$.

Equivariant terms $\Gamma \vdash M : A$ (where $\Gamma \vdash A$ indexed G-action)

terms $\underline{\Gamma} \vdash M : \underline{M}$ satisfying: $\mathsf{g} : \underline{G}, \gamma : \underline{\Gamma} \vdash g \underset{\gamma}{\circledast} M_\gamma = M_{\mathsf{g} \circledast \gamma} : A_{\mathsf{g} \circledast \gamma}$

Proposition

$\Gamma \vdash \varphi : \mathsf{Prop}$ **invariant** *i.e., equivariant w.r.t. trivial action,* $\iff$ $\Gamma \vdash \varphi$ *has an indexed action (this indexed action is unique)*

Mathematical reasoning w.l.o.g.

Revisionism

Few sources treat w.l.o.g. formally, we don't encompass all extant uses

Use our w.l.o.g. principle for **taxonomy**:

- ✓ Dijkstra'95 [EWD1223, not mechanised]
- ✓ HOL WLOG tactics [Harrison'09]
- ✗ Rocq SSReflect
- ✓ Agda Stdlib
- ✗ Lean

Road-map

More in the draft

- Analysis of Dijkstra's and Harrison's w.l.o.g. principles
- Extension to indexed data-types
- More examples:
  - lookup without Maybe;
  - insertion to McBride's binary search trees
- Meta-theoretic results for equivariant reasoning:
  - representation theorem for equivariant maps
  - completeness of w.l.o.g. reasoning

Future work

- more examples, connections nominal techniques and symmetry breaking
- mechanisation
- efficient implementation

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