Mathematical Foundations for Symmetric Programming

OHAD KAMMAR, University of Edinburgh, United Kingdom
MATIJA PRETNAR, University of Ljubljana & Institute of Mathematics, Physics and Mechanics, Slovenia

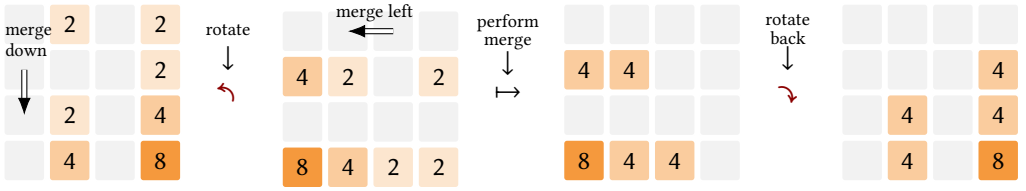
ACM Reference Format:
Ohad Kammar and Matija Pretnar. 2026. Mathematical Foundations for Symmetric Programming. 11, MSFP, Article 8 (July 2026), 59 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

When faced with repetition, programmers naturally turn to boilerplate or parametrisation. In this paper, we focus on repetition that arises from *symmetry*. We develop mathematical foundations that explicitly and uniformly capture existing implicit patterns in programming and reasoning, and use them to define a construction that removes redundant symmetric cases from function definitions and proofs.

Consider Gabriele Cirulli's famous game of 2048[1], in which the player slides tiles labelled by powers of 2, merging tiles with identical labels. The player can slide tiles in all four directions, and the game source-code[2] accommodates that ability through a general merging code that takes the direction as a parameter, builds a sequence of matrix indices to consider, and traverses over them[3].

Our anecdotal experience with half a dozen student projects suggests this parametrisation can be difficult to implement correctly. Here is an approach that is simpler to understand and to implement. Define the merging logic only for a single direction, say left, and exploit the inherent symmetry of the game for the other three directions: rotate the board so that the task is to slide left, perform the merge, and rotate back, restoring the original orientation. Here is an example downwards merge:



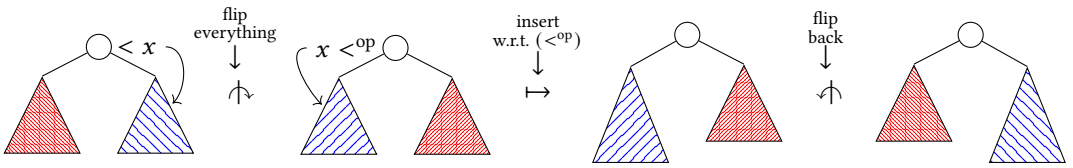
Another common example of symmetry in programming is that of binary search trees, in which elements smaller than the root go to the left subtree, while larger elements go to the right. Operations such as search, insertion, deletion, or rotation behave completely symmetrically. One way of avoiding repetition by parametrisation is to store both subtrees in an array of length two, and then use the offset $1 - i$ to refer to the tree opposite to the one at offset i. In practice, though, the additional complexity is often not worth losing the high-level abstraction, and programmers prefer to duplicate code.

[1]Available to play: https://play2048.co
[2]Available from https://github.com/gabrielecirulli/2048
[3]See the function `GameManager.prototype.move`:
https://github.com/gabrielecirulli/2048/blob/478b6ec346e3787f589e4af751378d06ded4cbbc/js/game_manager.js#L129.

Authors' Contact Information: Ohad Kammar, ohad.kammar@ed.ac.uk, University of Edinburgh, Laboratory for the Foundations of Computer Science, School of Informatics, Edinburgh, Scotland, United Kingdom; Matija Pretnar, matija.pretnar@fmf.uni-lj.si, University of Ljubljana & Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia.

2026. ACM XXXX-XXXX/2026/7-ART8
https://doi.org/XXXXXXX.XXXXXXX

But we can again avoid this repetition with the same approach and implement the code only for one case. For example, when inserting an element into the right subtree, we can flip the tree (and its ordering relation!), insert the element into the left subtree, and then flip the resulting tree back:



A reader may justifiably worry about the cost of flipping the whole tree each time we descend right, but a more efficient implementation fits the same pattern.

Our final example following the same pattern comes from mathematics, where one often invokes a *without loss of generality* (or w.l.o.g.) argument when we treat a canonical case, and derive the remaining cases by symmetry. Consider the pigeonhole principle given as an example in the Wikipedia entry for w.l.o.g. [31]:

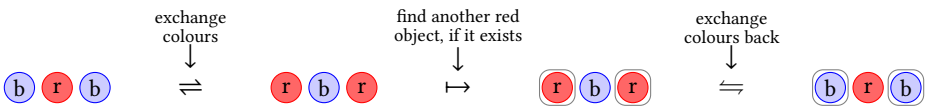
> If three objects are each painted either red or blue, then there must be at least two objects of the same color.
>
> **Proof.** Assume, without loss of generality, that the first object is red. If either of the other two objects is red, then we are finished; if not, then the other two objects must both be blue and we are still finished.

On paper, w.l.o.g. appeals to an intuitive understanding that nothing has been lost by considering a special case. But if the symmetry is complex or if we want to convince a proof assistant, we need to explicate the reduction of the general case to the special one.

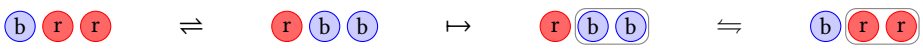
We can generalise the special case as a parametric proof that covers all cases. Above, we could take the colour of the first object as a parameter and then instantiate it to both red and blue. But as Harrison argues in his seminal study of w.l.o.g. in the context of the HOL proof assistant—and we agree—this formulation is not faithful to the informal proof [16].

Modern proof assistants offer special tactics to treat w.l.o.g. arguments, but approach them from various directions. Some tactics or theorems named `wlog` work in a very restricted setting, such as symmetric relations or negation, and some cut in arbitrary additional statements that do not involve symmetries. These divergent approaches point towards a missing mathematical foundation. Furthermore, none treat the underlying symmetry as a first-class citizen, and as Dijkstra argues—and we agree—"omissions [of the underlying symmetry] make most 'w.l.o.g. proofs' rather unsatisfactory" [8].

We can explicate this symmetry by following the same pattern we used for programming. In particular, if the first object is blue, we can exchange the colours to make it red, use the specialised argument, and then revert the colours back to the original core. For example, one of the three cases where there is another blue object in addition to the first one proceeds as:



while if both remaining objects are red, we similarly proceed as:



In this paper, we develop this pattern into a uniform construction, and use it to eliminate symmetric cases in definitions of functions and in reasoning. On the surface, this pattern is conceptually

simple, and so one may, just like us, be quickly tempted to implement it, either in a programming language or in a proof assistant. We made multiple prototypes in simply- and dependently-typed languages, and unsurprisingly, each obstacle pushed us back to pen-and-paper and forced us to enrich the underlying mathematical structure.

For this reason, while still keeping computational applications in mind, this paper focuses on presenting these mathematical foundations. We postpone a bona fide implementation to future work. Such future implementations will allow us to explore symmetric programming on more involved examples, in a machine-checked environment, design the best interface for exposing it to the programmer, to build a comprehensive library for symmetric programming and mechanised reasoning, and revisit existing w.l.o.g. formalisations systematically. With these downstream goals in mind, we pursue none of those future goals in the present work, and remain purely mathematical.

Our contributions:

- In §2, we identify three ingredients: *group actions* capturing the symmetries of the input and output, a *canoniser* map that assigns a canonising group element to each input, and a *core function* that defines the construction on canonical inputs. We then characterise our symmetric w.l.o.g. construction as the unique symmetry preserving (i.e., equivariant) extension of the core function to all the inputs.
- In §3, we extend this theory to the setting of *dependent types*. We show how via the Curry-Howard correspondence, symmetry-invariant propositions correspond to types equipped with a group action. We show how proofs via the symmetric w.l.o.g. argument amount to dependent functions obtained by the above construction.
- In §4, we further extend the theory to (simply- and dependently-typed) *algebraic data types* and their induction principles. The main construct we introduce is *symmetry strength*, which defines a group action in terms of data constructors, paving the way for symmetric reasoning on data types, and efficient partial evaluation.

We conclude by discussing related and future work in §5.

The Appendix contains details of a proof-of-concept Idris 2 [6] implementation removing the asymptotic performance bottlenecks symmetric programming incurs (Appendix A), and expanded technical development, including further details and proofs of most mathematical claims in the paper (Appendices B–G).

References

[1] Robert Atkey. 2014. From parametricity to conservation laws, via Noether's theorem. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 491–502. https://doi.org/10.1145/2535838.2535867

[2] Robert Atkey. 2018. The Syntax and Semantics of Quantitative Type Theory. In *LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom*. Association for Computing Machinery, 56–65. https://doi.org/10.1145/3209108.3209189

[3] Ramzan Basheer and Deepak Mishra. 2025. Symmetry group equivariant convolutions for representation learning: a survey. *International Journal of Data Science and Analytics* 22 (12 2025). https://doi.org/10.1007/s41060-025-00994-7

[4] Jon Beck. 1969. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, B. Eckmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 119–140.

[5] Richard Bird and Ross Paterson. 1999. Generalised folds for nested datatypes. *Form. Asp. Comput.* 11, 2 (Sept. 1999), 200–222. https://doi.org/10.1007/s001650050047

[6] Edwin C. Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021, Aarhus, Denmark (Virtual Conference), July 11-17, 2021 (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:26. https://doi.org/10.4230/LIPICS.ECOOP.2021.9

[7] Rob Cornish. 2025. Stochastic Neural Network Symmetrisation in Markov Categories. arXiv:2406.11814 [stat.ML] https://arxiv.org/abs/2406.11814

[8] Edsger W. Dijkstra. 1997. WLOG, or the misery of the unordered pair (EWD1223). In *Mathematical Methods in Program Development*, Manfred Broy and Birgit Schieder (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–34.

[9] Marcelo Fiore and Matías Menni. 2005. Reflective Kleisli subcategories of the category of Eilenberg-Moore algebras for factorization monads. *Theory and Applications of Categories [electronic only]* 15 (2005), 40–65. http://eudml.org/doc/125160

[10] Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.* 6, POPL, Article 53 (Jan. 2022), 29 pages. https://doi.org/10.1145/3498715

[11] Marcelo Fiore and Dmitrij Szamozvancev. 2025. Familial Model of Second-Order Abstract Syntax. (2025). unpublished.

[12] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 193–202. https://doi.org/10.1109/LICS.1999.782615

[13] Marcelo P. Fiore and Sanjiv Ranchod. 2025. Substructural Abstract Syntax with Variable Binding and Single-Variable Substitution. In *2025 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 196–208. https://doi.org/10.1109/LICS65433.2025.00022

[14] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. 2006. Symmetry in Constraint Programming. In *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). Foundations of Artificial Intelligence, Vol. 2. Elsevier, 329–376. https://doi.org/10.1016/S1574-6526(06)80014-3

[15] Georges Gonthier and Assia Mahboubi. 2010. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning* 3, 2 (Jan. 2010), 95–152. https://doi.org/10.6092/issn.1972-5787/1979

[16] John Harrison. 2009. Without Loss of Generality. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 43–59. https://doi.org/10.1007/978-3-642-03359-9_3

[17] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. 2005. Handbook of Computational Group Theory. https://api.semanticscholar.org/CorpusID:116937731

[18] Mikoláš Janota, Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. 2025. Breaking Symmetries in Quantified Graph Search: A Comparative Study. arXiv:2502.15078 [cs.LO] https://arxiv.org/abs/2502.15078

[19] Anton Lorenzen, Daan Leijen, Wouter Swierstra, and Sam Lindley. 2025. First-Order Laziness. *Proc. ACM Program. Lang.* 9, ICFP (2025), 734–762. https://doi.org/10.1145/3747530

[20] Saunders Mac Lane. 1978. *Categories for the Working Mathematician* (2nd ed.). Springer New York, NY. https://doi.org/10.1007/978-1-4757-4721-8

[21] Conor McBride and James McKinna. 2004. The view from the left. *Journal of Functional Programming* 14, 1 (2004), 69–111. https://doi.org/10.1017/S0956796803004829

[22] Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94–112. https://doi.org/10.1016/j.jsc.2013.09.003

[23] Vaibhav Mehta and Justin Hsu. 2025. A Hoare Logic for Symmetry Properties. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 302 (Oct. 2025), 25 pages. https://doi.org/10.1145/3763080

[24] Andrew M. Pitts. 2003. Nominal logic, a first order theory of names and binding. *Inf. Comput.* 186, 2 (2003), 165–193. https://doi.org/10.1016/S0890-5401(03)00138-X

[25] Andrew M. Pitts. 2016. Nominal techniques. *ACM SIGLOG News* 3, 1 (2016), 57–72. https://doi.org/10.1145/2893582.2893594

[26] Ákos Seress. 2003. *Permutation Group Algorithms.* Cambridge University Press.

[27] Wouter Swierstra. 2008. Data types à la carte. *J. Funct. Program.* 18, 4 (2008), 423–436. https://doi.org/10.1017/S0956796808006758

[28] Dmitrij Szamozvancev. 2026. *Categorical models of second-order abstract syntax.* Ph. D. Dissertation. University of Cambridge.

[29] The mathlib Community. 2020. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020).* ACM, New Orleans, LA, USA. https://doi.org/10.1145/3372885.3373824

[30] Toby Walsh. 2012. Symmetry Breaking Constraints: Recent Results. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, Jörg Hoffmann and Bart Selman (Eds.). AAAI Press, 2192–2198. https://doi.org/10.1609/AAAI.V26I1.8437

[31] Wikipedia contributors. 2026. Without loss of generality — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Without_loss_of_generality&oldid=1336885984 [Online; accessed 8-March-2026].

[32] Bo Zhao, Robin Walters, and Rose Yu. 2025. Symmetry in Neural Network Parameter Spaces. arXiv:2506.13018 [cs.LG] https://arxiv.org/abs/2506.13018

Abstract

We develop a mathematical abstraction for uniformly exploiting symmetries in programming and reasoning based on the mathematical proof principle 'without-loss-of-generality' (w.l.o.g.). Given: group actions that describe symmetries of the input and the output; and a canoniser, which captures how arbitrary inputs get sent to canonical ones, our construction extends any function from the canonical inputs to the whole domain.

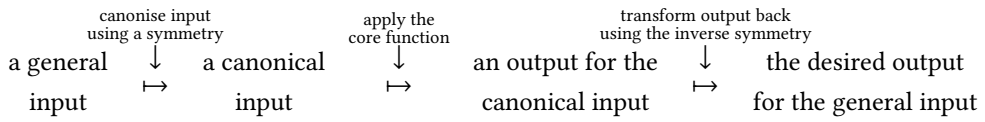
We characterise the conditions on the canonisers which guarantee that the construction gives the unique equivariant, i.e. symmetry preserving extension. Furthermore, we adapt the construction to a dependently-typed setting, where it captures traditional symmetric w.l.o.g. arguments via the Curry-Howard correspondence. Finally, we extend the construction to algebraic data types, where the symmetries of the constructors both uniquely determine the symmetries of the data type and pave the way for efficient execution.

2 Without loss of generality

Having observed the common pattern in the preceding examples, let us detail the ingredients:

(1) Capture the inherent symmetries of our problem using **group actions** (§2.1).
(2) Obtain a **canonising** symmetry for each input (§2.2).
(3) Give the **core function**, defined only on canonical elements (§2.3).

The **symmetric w.l.o.g. construction** combines these ingredients, covering all inputs (§2.4):



Our previous examples fit this pattern as:

	2048 merge	**tree insertion**	**pigeonhole principle**
general input	board & direction	element & tree	arbitrary colouring
desired output	merged board	extended tree	general proof
symmetry	rotation	flipping	colour exchange
canonical input	direction is left	inserted key ≤ root key	first object is red
core function	merge left	insert left as needed	special-case proof

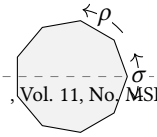
2.1 Groups and actions

A *concrete symmetry* on a set is a bijection from the set to itself. When we compare different sets, we need to relate their symmetries. A *group* gives these symmetries a name, and its *actions* allow us to use the same abstract symmetry to refer to concrete symmetries on different set.

Groups. Recall that a *group* $G = \langle \underline{G}, e, (*), (-)^{-1} \rangle$ is a tuple consisting of: a *carrier* set $\underline{G}$, whose elements we call *symmetries*; a *neutral* element $e \in \underline{G}$; a *composition* $(*) : \underline{G} \to \underline{G} \to \underline{G}$; and an *inverse* map $(-)^{-1} : \underline{G} \to \underline{G}$; such that:

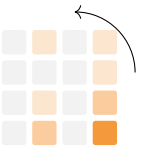
- composition is associative: $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$ for all $g_1, g_2, g_3 \in \underline{G}$;
- the neutral element is neutral w.r.t. composition: $e * g = g * e = g$ for all $g \in \underline{G}$; and
- the inverse map picks inverses: $g * g^{-1} = g^{-1} * g = e$ for all $g \in \underline{G}$.

Two famous examples of groups are the *symmetric group* S_n of all permutations of the set $\mathsf{Fin}\, n := \{0, \ldots, n-1\}$, and the *dihedral group* D_n of all symmetries of a regular



n-gon, i.e. n rotations and n reflections. The dihedral group is *generated* by a $2\pi/n$ rotation ρ and a single reflection σ, meaning every other symmetry can be obtained by a suitable composition of ρ and σ.

In our examples, both the mirroring symmetry of trees and exchange of colours are described by the symmetric group $\mathrm{S}_2 = \{\mathtt{I}, \mathtt{F}\}$, where $\mathtt{I}$ is the identity permutation and $\mathtt{F}$ flips the two elements. The 2048 board is symmetric with respect to the dihedral group D_4, though for our use, it is sufficient to consider the *cyclic group* C_4, generated by the quarter-turn rotation marked here as ↰.



Actions. Groups capture symmetries abstractly. Symmetries between elements of a particular set are captured by *group actions*. Given a group G, recall that a G-*action* $A = \langle \underline{A}, (\circledast) \rangle$ consists of a *carrier* set $\underline{A}$; and a map $(\circledast) : \underline{G} \to \underline{A} \to \underline{A}$, writing $(\circledast^A)$ when ambiguous, such that:

- every element is symmetric to itself via the neutral symmetry: $e \circledast x = x$ for $x \in \underline{A}$; and
- symmetry is transitive via composition: $(g_1 * g_2) \circledast x = g_1 \circledast (g_2 \circledast x)$ for $g_1, g_2 \in \underline{G}$, $x \in \underline{A}$.

Example 2.1. The symmetric group S_n acts on the finite sets $\mathsf{Fin}\, n = \{0, 1, \ldots, n-1\}$ by $\pi \circledast i := \pi i$, and on vectors $\mathsf{Vect}_n X := X^n$ by $\pi \circledast \langle x_0, \ldots, x_{n-1} \rangle := \left\langle x_{\pi^{-1}0}, \ldots, x_{\pi^{-1}(n-1)} \right\rangle$ sending the i-th element to the position πi. □

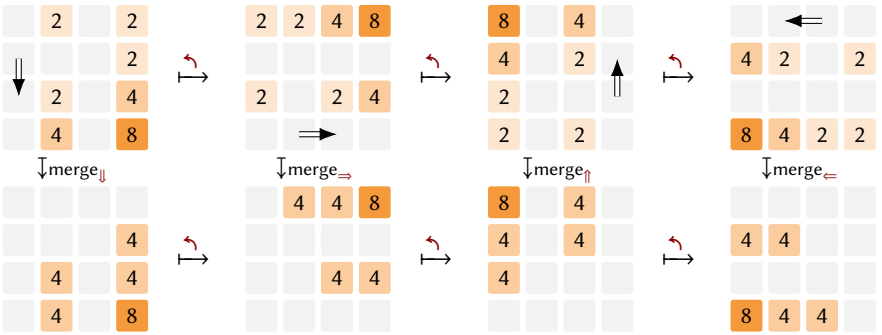
Example 2.2. We define the sets of valid $n \times n$ boards of 2048 by:

$$\mathsf{Tile} := \{0\} \cup \left\{ 2^i \,\middle|\, i \in \mathbb{N}_+ \right\} \qquad \mathsf{Row}_n = \mathsf{Vect}_n \mathsf{Tile} \qquad \mathsf{Board}_n = \mathsf{Vect}_n(\mathsf{Row}_n)$$

And the action $(\circledast) : \mathrm{C}_4 \to \mathsf{Board}_n \to \mathsf{Board}_n$ is given by:

$$\text{↰} \circledast \begin{pmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,n} \end{pmatrix} = \begin{pmatrix} x_{1,n} & \cdots & x_{n,n} \\ \vdots & \ddots & \vdots \\ x_{1,1} & \cdots & x_{n,1} \end{pmatrix} \quad \text{i.e.} \quad (\text{↰} \circledast -) := \mathsf{reverse} \circ \mathsf{transpose}$$ □

Equivariance. The game 2048 exhibits symmetry not only by permitting rotations of the board, but also by the fact that merging respects it: if we rotate the board and then merge in the rotated direction, we get the same result as first merging and then rotating the resulting board:



Equivariance captures symmetry preservation. Given G-actions A and B, recall that an *equivariant* map $\varphi : A \circledast\!\!\longrightarrow B$ is a map $\varphi : \underline{A} \to \underline{B}$ satisfying $\varphi(g \circledast^A a) = g \circledast^B \varphi a$ for all $g \in \underline{G}$ and $a \in \underline{A}$.

Example 2.3. Take any $f : X \to Y$. Its pointwise application $(\mathsf{Vect}_n) f : \mathsf{Vect}_n X \to \mathsf{Vect}_n Y$ is equivariant with respect to the permuting action of S_n on vectors since:

$$\begin{aligned} F f(\pi \circledast \langle x_0, \ldots, x_{n-1} \rangle) &= F f(\left\langle x_{\pi^{-1}0}, \ldots, x_{\pi^{-1}(n-1)} \right\rangle) = \left\langle f x_{\pi^{-1}0}, \ldots, f x_{\pi^{-1}(n-1)} \right\rangle \\ &= \pi \circledast \langle f x_0, \ldots, f x_{n-1} \rangle = \pi \circledast (F f \langle x_0, \ldots, x_{n-1} \rangle) \end{aligned}$$ □

Before the next example, recall that a *product* $A \times B$ of G-actions A and B is given on $\underline{A \times B} \coloneqq \underline{A} \times \underline{B}$ component-wise by: $g \circledast^{A \times B} \langle a, b \rangle \coloneqq \langle g \circledast^{A} a, g \circledast^{B} b \rangle$. Furthermore, the *exponential action* $A \to B$ is given on the set of all functions, not only the equivariant ones, $\underline{A \to B} \coloneqq \underline{A} \to \underline{B}$ by *conjugation*: $(g \circledast^{A \to B} f) \coloneqq a \mapsto g \circledast^{B} f(g^{-1} \circledast^{A} a)$. Finally, the *trivial* action $\mathrm{Triv} X$ on a set X is defined by $\underline{\mathrm{Triv} X} \coloneqq X$ and $g \circledast x \coloneqq x$. An *invariant* map is an equivariant map $f : A \circledast\!\!\to \mathrm{Triv} X$.

Example 2.4. We can equip the set Tree of binary trees with values in $\mathbb{N}$ (fixed for simplicity) with the recursive S_2-action $\mathsf{F} \circledast^{\mathsf{Tree}} \mathsf{Leaf} = \mathsf{Leaf}$ and $\mathsf{F} \circledast^{\mathsf{Tree}} \mathsf{Node}(\ell, n, r) = \mathsf{Node}(\mathsf{F} \circledast r, n, \mathsf{F} \circledast \ell)$. Furthermore, let Path be the set of lists over step directions $\mathsf{Step} = \{\mathsf{L}, \mathsf{R}\}$. We can equip Path with an S_2-action by mapping the S_2-action $\mathsf{F} \circledast \mathsf{L} = \mathsf{R}$ and $\mathsf{F} \circledast \mathsf{R} = \mathsf{L}$.

Take $\mathsf{lookup} : \mathsf{Tree} \to \mathsf{Path} \to \mathsf{Maybe}\,\mathbb{N}$, which takes a path through a tree (a list of steps left or right) and returns the resulting value, if it exists. This function is invariant under symmetry: if we flip both the tree and the direction of all the steps, we should get back the same value. More precisely, lookup is equivariant in its curried form $\mathsf{Tree} \circledast\!\!\to (\mathsf{Path} \to \mathrm{Triv}(\mathsf{Maybe}\,\mathbb{N}))$ with respect to the exponential action, while the uncurried form $\mathsf{Tree} \times \mathsf{Path} \circledast\!\!\to \mathrm{Triv}(\mathsf{Maybe}\,\mathbb{N})$ is invariant. □

This example concludes our group theory tutorial. We return to symmetric w.l.o.g. abstractions.

2.2 Canonisers

The next step is to map each input to a *canonical* representative. It is crucial that we do this through an explicit symmetry in the group, rather than through an arbitrary function, as we need to apply the inverse symmetry to reconstruct the output for the original input.

For example, we canonise search trees by flipping the tree when the inserted key is strictly larger than the root key. Thus, the remaining canonical inputs are those that: insert into an empty tree, insert the redundant key equal to the one in the root; and insert to the left subtree. For the pigeonhole principle example, the canonising symmetry is the identity if the first object is red, and F that exchanges the colours if the first object is blue. Thus, the canonical elements are triples in which the first object is red.

Definition 2.5. Let A be a G-action. A *canoniser* for A is a function $c : \underline{A} \to \underline{G}$. For each $a \in \underline{A}$, we call the symmetry $ca \in \underline{G}$ the *c-canonising symmetry* of a, and the element $ca \circledast a \in \underline{A}$ the *canonical form* of a. We define the set of *canonical* elements to be $\mathrm{Can}_A\, c \coloneqq \{\, ca \circledast a \mid a \in \underline{A} \,\}$.

Canonisers need not respect the structure of the action on A, as many of the interesting examples indeed do not. However, there are still some examples that respect it, and we study them and their impact on the equivariance of the w.l.o.g. construction in §2.4.

Example 2.6. Let P be a *strict partial order*, i.e., a set equipped with an irreflexive and transitive relation $<_P$. We say that an n-tuple $\vec{p} \in \mathrm{Vect}_n P$ is *sorted* when $p_i < p_j$ implies $i < j$ for all indices $i, j \in \mathrm{Fin}\, n$. Then, a *sorting function* is a function $\mathsf{sort} : \mathrm{Vect}_n P \to \underline{\mathrm{S}_n}$ that maps a tuple to a permutation that sorts it. Every such function is a canoniser for the permuting action of S_n on $\mathrm{Vect}_n P$ from Example 2.1.

Given an n-tuple $\vec{p}$, the sorted tuple $(\mathsf{sort}\, \vec{p}) \circledast \vec{p}$ is the canonical form of $\vec{p}$, and every $p \in \mathrm{Can}\ \mathsf{sort}$ is sorted. If the sorting function returns the identity permutation for sorted tuples—for example when the sorting is stable—the set $\mathrm{Can}\ \mathsf{sort}$ is exactly the set of sorted tuples. □

Example 2.7. Continuing Example 2.2, the input for the 2048's merge is the board together with the merge direction. Define $\mathsf{Dir} \coloneqq \{\Rightarrow, \Uparrow, \Leftarrow, \Downarrow\}$ and its action $(\circledast) : \mathrm{C}_4 \to \mathsf{Dir} \to \mathsf{Dir}$ by:

$$\curvearrowleft \circledast \Rightarrow \coloneqq \Uparrow \quad \curvearrowleft \circledast \Uparrow \coloneqq \Leftarrow \quad \curvearrowleft \circledast \Leftarrow \coloneqq \Downarrow \quad \curvearrowleft \circledast \Downarrow \coloneqq \Rightarrow$$

Define $\mathsf{mkLft} : \mathsf{Dir} \to \underline{\mathrm{C}_4}$ to pick the unique rotation turning its input left, i.e. $(\mathsf{mkLft}\, d) \circledast d = \Leftarrow$:

$$\mathsf{mkLft} \Rightarrow := \curvearrowleft^2 = \curvearrowright \qquad \mathsf{mkLft} \Uparrow := \curvearrowleft \qquad \mathsf{mkLft} \Leftarrow := e = \circlearrowleft \qquad \mathsf{mkLft} \Downarrow := \curvearrowleft^{-1} = \curvearrowright$$

Thus the only canonical direction is left $\mathsf{Can}\, \mathsf{mkLft} = \{\Leftarrow\}$. We can extend the canoniser to input pairs $\mathsf{Dir} \times \mathsf{Board}_n$ by $\langle d, \_ \rangle \mapsto \mathsf{mkLft}\, d$. □

We frequently encounter similar canonisers that depend only on a part of the input.

Lemma 2.8. *Let A and B be G-actions. Every A-canoniser $c : \underline{A} \to \underline{G}$ induces an $(A \times B)$-canoniser $c \circ \pi_1 : \underline{A \times B} \to \underline{G}$. We then have* $\mathsf{Can}\, (c \circ \pi_1) = \mathsf{Can}\, c \times \underline{B}$.

2.3 Core functions

Take G-actions A and B between which we want to define an equivariant function $\varphi : A \circledast\!\!\longrightarrow B$. Once we have an A-canoniser c and its set of canonical elements $\mathsf{Can}\, c$ in place, the only thing missing is the *core function* $f : \mathsf{Can}\, c \to \underline{B}$: merging of tiles in 2048, insertion into a subtree, or the special case in the pigeonhole principle. Typical symmetric w.l.o.g. arguments only explicate canonical elements and core functions, while groups, actions, and gloss over canonisers.

Example 2.9. Continuing Example 2.7, let us define the core function merging the tiles in 2048. We choose left merges as the canonical direction since vertical merges go against the structure of vectors of rows, while rightward merges have edge cases coalescing empty tiles. First, we define $\mathsf{merge}^{\mathsf{Row}}_{\Leftarrow} : \mathsf{Row}_n \to \mathsf{Row}_n$ using pattern matching as all the relevant tiles are on the left:

$$\mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, \langle\rangle := \langle\rangle \qquad \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (x :: \langle\rangle) := \langle x \rangle \qquad \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (0 :: r) := \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (r) +\!\!+ \langle 0 \rangle$$

$$\mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (x :: y :: r) := \begin{cases} 2x :: \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (r) +\!\!+ \langle 0 \rangle & (x = y) \\ \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (x :: r) +\!\!+ \langle 0 \rangle & (y = 0) \\ x :: \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}\, (y :: r) & (x \neq y \neq 0) \end{cases}$$

We merge the whole board simply by merging each row:

$$\mathsf{merge}_{\Leftarrow} : \mathsf{Board}_n \to \mathsf{Board}_n \qquad \mathsf{merge}_{\Leftarrow} := F\, \mathsf{merge}^{\mathsf{Row}}_{\Leftarrow}$$ □

In addition to extending the core function $f : \mathsf{Can}\, c \to \underline{B}$ to the whole $\varphi : \underline{A} \to \underline{B}$, we often want the extension to be equivariant. In that case, if we happen to have a canonical element $x \in \mathsf{Can}\, c$ that also has a symmetric canonical element $g \circledast x \in \mathsf{Can}\, c$, then:

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

So equivariant w.l.o.g. extensions have core functions that are equivariant on canonical elements. We generalise the notion of equivariance to domains that may not be closed under the action:

Definition 2.10. Take G-actions A, B, a subset $X \subseteq \underline{A}$, and a function $f : X \to \underline{B}$. We call f *equivariant*, writing $f : (X \subseteq A) \circledast\!\!\longrightarrow B$, when $g \circledast x \in X \implies f(g \circledast x) = g \circledast fx$ for $g \in \underline{G}$, $x \in X$.

Example 2.11. Every core function for the 2048 canoniser from Example 2.7 is vacuously equivariant: only $\Leftarrow$ is canonical, and any $g \in \mathrm{C}_4 \setminus \{e\}$ changes the direction to a non-canonical input. Therefore, if $x, g \circledast x$ are canonical, then $g = e$ and $g \circledast f\, x = f\, x = f(g \circledast x)$.

Insertion into binary search trees features a non-trivial corner case of core function equivariance. Flipping the empty tree yields back the empty tree, which is again a canonical element. Thus inserting a key into a "flipped" empty tree needs to be the same as first inserting the key into the empty tree and then flipping the result. However, the resulting tree is a single root with two empty subtrees, which is invariant under flipping, thus the core function is equivariant. □

2.4 The w.l.o.g. construction

We now formally define the w.l.o.g. construction we sketched a number of times, extending a core function $f : \mathsf{Can}\, c \to \underline{B}$ to the whole carrier $\underline{A}$. Take any $a \in \underline{A}$ and compute its canonical form $ca \circledast a$. By definition, the canonical form lies in $\mathsf{Can}\, c$, so we can apply the core function to obtain $f(ca \circledast a)$. Even though this result has the expected type B, we still need to apply the inverse symmetry $(ca)^{-1}$. Indeed, making a downward merge in 2048 would result in a rotation, followed by a leftward merge, followed by the inverse rotation.

Definition 2.12. Let A and B be G-actions, $c : \underline{A} \to \underline{G}$ a canoniser, and $f : \mathsf{Can}\, c \to \underline{B}$ a core function. Then, the *w.l.o.g. construction* $\mathsf{wlog}(c, f) : \underline{A} \to \underline{B}$ is defined as

$$\mathsf{wlog}(c, f) \coloneqq a \mapsto (ca)^{-1} \circledast^{B} f(ca \circledast^{A} a)$$

The symmetric w.l.o.g. induces a bijection between equivariant functions and core functions:

Theorem 2.13. *Let A and B be G-actions and $c : \underline{A} \to \underline{G}$ a canoniser. If $f : (\mathsf{Can}\, c \subseteq A) \circledast\!\!\to B$ is an equivariant core function, then $\mathsf{wlog}(c, f) : A \circledast\!\!\to B$ is the unique equivariant function extending f to the whole $\underline{A}$, along the inclusion $\mathsf{Can}\, c \subseteq \underline{A}$.*

Example 2.14. Completing Example 2.9, we can define $\mathsf{merge} : \mathsf{Dir} \times \mathsf{Board}_n \to \mathsf{Board}_n$ as $\mathsf{wlog}(\mathsf{mkLft}, \mathsf{merge}_{\Leftarrow})$. It indeed behaves as expected: for leftward merges, it extends $\mathsf{merge}_{\Leftarrow}$, while for other directions, it is uniquely determined by the equivariant behaviour. □

Equivariance of core functions is the only additional assumption in Theorem 2.13. In concrete cases, it is often easily discharged, but we can further relax it when the canoniser c has additional properties. A natural property is the equivariance of the canoniser. Since $\left(ca * g^{-1}\right) \circledast (g \circledast a) = ca \circledast a$ is canonical, it is natural to ask whether $ca * g^{-1}$ is the c-canonising symmetry of $g \circledast a$, i.e. whether $c(g \circledast a) = ca * g^{-1}$. More succinctly, this properties means c is an equivariant map $c : A \circledast\!\!\to G^{-1}$ where G^{-1} is the action of G on itself, defined by $g \circledast^{G^{-1}} h \coloneqq h * g^{-1}$. It validates the axioms since:

$$e \circledast^{G^{-1}} s = s * e^{-1} = s \qquad g \circledast^{G^{-1}} (h \circledast^{G^{-1}} s) = (s * h^{-1}) * g^{-1} = s * (g * h)^{-1} = (g * h) \circledast^{G^{-1}} s$$

Definition 2.15. A canoniser $c : \underline{A} \to \underline{G}$ is *strictly equivariant* when $c\,(g \circledast a) = ca * g^{-1}$, i.e. when it is an equivariant map $c : A \circledast\!\!\to G^{-1}$.

As we shall soon see, strict equivariance is a powerful assumption, but also a very strong restriction, thus we reserve the name *equivariant canonisers* for a weaker and more natural notion that we introduce in §3.4, after we discuss dependent types.

Example 2.16. The canoniser mkLft from Example 2.7 is strictly equivariant since $\mathsf{mkLft}\, d$ is the unique rotation that makes d canonical. On the other hand, if in Example 2.6, we take a non-total ordering P and a stable sorting algorithm, the canoniser $\mathsf{sort} : \mathsf{Vect}_n P \to \underline{\mathsf{S}_n}$ is not strictly equivariant since for every two incomparable elements p, q, we have:

$$\mathsf{sort}\left((0 \leftrightarrow 1) \circledast \langle p, q\rangle\right) = \mathsf{sort}\,\langle q, p\rangle = \mathrm{id} \neq (0 \leftrightarrow 1) = \mathrm{id} * (0 \leftrightarrow 1)^{-1} = (0 \leftrightarrow 1) \circledast \mathsf{sort}\,\langle p, q\rangle \qquad \square$$

Strict equivariance guarantees equivariance of arbitrary core functions, and thus also of the w.l.o.g. construction. The converse also holds, though we do not need it for our applications.

Theorem 2.17. *Let $c : \underline{A} \to \underline{G}$ be a canoniser. Then, c is strictly equivariant iff for every action B, every core function $f : \mathsf{Can}\, c \to \underline{B}$ is equivariant.*

Strict equivariance is too strong, as it restricts the action to be *semi-regular*, meaning that only the identity can fix an element: $g \circledast a = a \implies g = e$.

Lemma 2.18 (semi-regularity). *If A has a strictly equivariant canoniser, then A is semi-regular.*

For example, the C_4 action on Board_n is not semi-regular, since all rotations preserve the empty 2048 board. It is only the addition of the direction, which all non-trivial symmetries affect, that makes the action on $\mathsf{Dir} \times \mathsf{Board}_n$ semi-regular.

A weaker natural property of canonisers, and one that all examples in this paper enjoy, is that canonisation is a one-step process and that further canonisation symmetries are identities:

Definition 2.19. A canoniser c is *immediate* when for all $a \in \underline{A}$, we have $c(ca \circledast a) = e$.

To apply the w.l.o.g. construction, we need to provide both the canoniser c and the core function f. If we have $ca = e$ for some $a \in \underline{A}$, then $a = ca \circledast a \in \mathsf{Can}\, c$, and so we need to define the value fa. In the ideal case, we want to consider each element of $\underline{A}$ only once and define the core function only on elements with trivial canonising symmetries. More precisely, we want the sets $\{a \in \underline{A} \mid ca \neq e\}$ and $\mathsf{Can}\, c$ to be disjoint. But in general, the two sets may overlap. In the extreme case, if $c : \mathsf{Dir} \to \underline{C_4}$ is the constant function $\_ \mapsto \curvearrowleft$, they are both equal to the whole domain Dir. However, immediate canonisers are exactly those for which the domain $\underline{A}$ cleanly splits into two:

Proposition 2.20. *A canoniser $c : \underline{A} \to \underline{G}$ is immediate iff* $\mathsf{Can}\, c = c^{-1}[e] \coloneqq \{a \in \underline{A} \mid ca = e\}$.

Immediateness is strictly weaker than strict equivariance. As we saw in Example 2.16, a stable sorting function is not a strictly equivariant canoniser, but it is immediate since the canonical elements are exactly the sorted tuples, and the sorting permutation for sorted tuples is the identity.

Proposition 2.21. *A strictly equivariant canoniser $c : A \circledast\!\!\longrightarrow G^{-1}$ is immediate.*

We will often derive a canoniser for A from a canoniser for $A \times B$ through projection. For example, we derive the canoniser for $\mathsf{Dir} \times \mathsf{Board}_n$ from that of Dir.

Lemma 2.22. *For G-actions A, B, if $\underline{A} \xrightarrow{c} \underline{G}$ is strictly equivariant/immediate, then so is $\underline{A} \times \underline{B} \xrightarrow{c \circ \pi_1} \underline{G}$.*

3 Symmetric dependent types

To precisely show how our construction captures traditional w.l.o.g. arguments, we need to move to a more expressive setting of dependent types. Similarly, we want dependent types to impose richer invariants on programming examples such as binary search trees.

We start by recalling the framework of *families* (§3.1), which gives a mathematical interpretation of familiar dependent-type notions. Afterwards, we generalise our constructs and results to a dependent setting (§3.2) and show how via the Curry-Howard correspondence, they capture proofs that use w.l.o.g. arguments (§3.3). Finally, we use the additional power of dependent types to define a more refined notion of canoniser equivariance (§3.4).

3.1 Families

A *family* $\Gamma \vdash X$ is a pair consisting of a set Γ and an assignment of a set $X\gamma$ to each element $\gamma \in \Gamma$. We call Γ the *indexing* set, its elements $\gamma \in \Gamma$ *indices*, and each set $X\gamma$ the *fibre* over the index γ. A *term* $\Gamma \vdash t : X$ in a family $\Gamma \vdash X$ is an assignment of an element $t\gamma \in X\gamma$ to each $\gamma \in \Gamma$. A family $\Gamma \vdash X$ is a *subfamily* of $\Gamma \vdash Y$, which we write as $\Gamma \vdash X \subseteq Y$ if $X\gamma \subseteq Y\gamma$ for all $\gamma \in \Gamma$.

Example 3.1. To eliminate the necessity for the Maybe type in the tree lookup from Example 2.4, let us define the family $\mathsf{Tree} \vdash \mathsf{InPath}$ of *node paths* that contains all paths ending at an internal node of the index tree. More precisely, the fibre $\mathsf{InPath}\, t$ is empty if t is empty, and is otherwise the set of lists over $\mathsf{Step} = \{\mathsf{L}, \mathsf{R}\}$ which are either Nil, representing the root of the index tree, or

$d :: p$ that start with a step direction d followed by a path p in that subtree. This set can be given inductively as:

$$\frac{}{\mathsf{Nil} \in \mathsf{InPath}\left(\begin{smallmatrix} & & x & & \\ & \ell & & r & \end{smallmatrix}\right)} \qquad \frac{p \in \mathsf{InPath}\,\ell}{(\mathsf{L} :: p) \in \mathsf{InPath}\left(\begin{smallmatrix} & & x & & \\ & \ell & & r & \end{smallmatrix}\right)} \qquad \frac{p \in \mathsf{InPath}\,r}{(\mathsf{R} :: p) \in \mathsf{InPath}\left(\begin{smallmatrix} & & x & & \\ & \ell & & r & \end{smallmatrix}\right)}$$ □

When specifying fibres or terms as assignments of the form $\gamma \mapsto X$ or $\gamma \mapsto t$, which explicitly use the index $\gamma \in \Gamma$, we write them as $\gamma : \Gamma \vdash X$ and $\gamma : \Gamma \vdash t : X$. Note that despite using a convenient type-theoretic notation, we continue to keep our development purely mathematical, constructing objects without any recourse to a syntactic type system.

Continuing this approach, we allow fibres and terms to depend on a context of variables. For the empty context, we take the indexing set $\mathbb{1} \coloneqq \{\star\}$ and identify families $\mathbb{1} \vdash X$ with their only fibre $X\star$. Then, we write fibres over $\mathbb{1}$ simply as $\vdash X$, and similarly for terms.

For context extension, we employ the *Grothendieck construction*. Recall that for a family $\Gamma \vdash X$, it is defined as the set $\coprod_\Gamma X \coloneqq \coprod_{\gamma \in \Gamma} X\gamma$, consisting of pairs $\langle \gamma, x \rangle$ where $\gamma \in \Gamma$ and $x \in X\gamma$, together with a projection function $\pi_1 \coloneqq (\langle \gamma, x \rangle \mapsto \gamma) : \coprod_\Gamma X \to \Gamma$. We write $\Gamma, x : X \vdash Y$ and $\Gamma, x : X \vdash t : Y$ for fibres and terms indexed by $\coprod_\Gamma X$ and use x to refer to the second component of the index pair in $\coprod_\Gamma X$ in Y and t.

Given two families $\Gamma \vdash X$ and $\Gamma, x : X \vdash Y$, we have the family of *dependent pairs* $\Gamma \vdash (x : X) \times Y$ whose fibre at $\gamma \in \Gamma$ is the Grothendieck construction $\coprod_{x \in X\gamma} Y\,\langle \gamma, x \rangle$, and the family of *dependent functions* $\Gamma \vdash (x : X) \to Y$ whose fibre at γ is the product $\prod_{x \in X\gamma} Y\,\langle \gamma, x \rangle$. As usual, we write $X \to Y$ when the codomain family Y does not depend on the argument of the function. To distinguish between indices and arguments of dependent functions $\Gamma \vdash f : (x : X) \to Y$, we apply them as $f_\gamma x$ for some $\gamma \in \Gamma$ and $x \in X\gamma$, or even as $f\,x$ when γ can be inferred.

Example 3.2. With dependent types, we can refine the lookup function $\mathsf{Tree} \to \mathsf{Path} \to \mathsf{Maybe}\,\mathbb{N}$ from Example 2.4 to a dependent function $\vdash \mathsf{lookup} : (t : \mathsf{Tree}) \to \mathsf{InPath}\,t \to \mathbb{N}$, as the path now must point to a node in the tree. We can define it as:

$$\mathsf{lookup}\left(\begin{smallmatrix} & & n & & \\ & \ell & & r & \end{smallmatrix}\right)\mathsf{Nil} \coloneqq n \qquad \mathsf{lookup}\left(\begin{smallmatrix} & & n & & \\ & t_\mathsf{L} & & t_\mathsf{R} & \end{smallmatrix}\right)(d :: p) \coloneqq \mathsf{lookup}\,t_d\,p$$

Note that since we have two trees $t_\mathsf{L}, t_\mathsf{R}$ rather than a function $\mathsf{Step} \to \mathsf{Tree}$, the parametrisation t_d that we use in the second case is just repetition in disguise. Later, we will show how to properly avoid this repetition by using the w.l.o.g. construction. □

3.2 Indexed actions

Just as in Example 2.4, the function lookup from Example 3.2 is invariant with respect to flipping: flipping both the tree and the path returns the same value. But note that once we flip the tree t, the flipped path no longer lies in $\mathsf{InPath}\,t$, but in $\mathsf{InPath}\,(\mathsf{F} \circledast t)$. For example:

$$\mathsf{F} \circledast \left(\mathsf{L} :: \mathsf{R} :: \mathsf{Nil} \in \mathsf{InPath}\left(\begin{smallmatrix} & & & 3 & \\ & & 2 & & \\ & 1 & & 4 & \end{smallmatrix}\right)\right) = \mathsf{R} :: \mathsf{L} :: \mathsf{Nil} \in \mathsf{InPath}\left(\begin{smallmatrix} & & 3 & & \\ & 4 & & 2 & \\ & & & & 1 \end{smallmatrix}\right)$$

In general, an action over a family $\Gamma \vdash A$ should thus consist of an action over the index set Γ, and an action over each fibre $A\gamma$, which tracks the indexing action: if $a \in A\gamma$, then $g \circledast a \in A(g \circledast \gamma)$.

Definition 3.3. Given a G-action Γ, an *indexed G-action* $\Gamma \vdash A$ consists of a *carrier* family $\underline{\Gamma} \vdash \underline{A}$ and a dependent function $(\circledast\_) : (g : \underline{G}) \to (\gamma : \underline{\Gamma}) \to \underline{A}\gamma \to \underline{A}(g \circledast^\Gamma \gamma)$, which we apply as $g \circledast_\gamma a$, such that $e \circledast_\gamma a = a$ and $(g * h) \circledast_\gamma a = g \circledast_{h \circledast \gamma} (h \circledast_\gamma a)$.

When A is ambiguous, we add a superscript and write $\circledast^A_\gamma$, and when we can infer the index γ, we elide it and write $g \circledast a$ instead of $g \circledast_\gamma a$.

Example 3.4. The action of the cyclic group C_4 on 2048 boards can be generalised to an indexed action on the family of arbitrary matrices $\mathbb{N} \times \mathbb{N} \vdash \mathsf{Mat}$. First, take Γ to be the action of C_4 on dimensions $\underline{\Gamma} \coloneqq \mathbb{N} \times \mathbb{N}$, defined as: $\circlearrowleft \circledast \langle m, n\rangle \coloneqq \langle n, m\rangle$. We then have a Γ-indexed action defined as in Example 2.2:

$$\circlearrowleft \circledast_{\langle m,n\rangle} \left(\begin{pmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{pmatrix} \in \mathsf{Mat}_{m\times n} \right) = \begin{pmatrix} x_{1,n} & \cdots & x_{m,n} \\ \vdots & \ddots & \vdots \\ x_{1,1} & \cdots & x_{m,1} \end{pmatrix} \in \mathsf{Mat}_{n\times m} \qquad \square$$

Example 3.5. Take indexed G-actions $\Gamma \vdash A$ and $\Gamma, x : A \vdash B$. Then, we can define the *dependent pair* action $\Gamma \vdash (x : A) \times B$ as:

$$\underline{\Gamma \vdash (x : A) \times B} \coloneqq \underline{\Gamma} \vdash (x : \underline{A}) \times \underline{B} \qquad g \circledast^{(x:A)\times B}_\gamma \langle a, b\rangle \coloneqq \left\langle g \circledast^A_\gamma a, g \circledast^B_{\langle \gamma, a\rangle} b \right\rangle$$

Furthermore, we can define the *dependent function* action $\Gamma \vdash (x : A) \to B$ as:

$$\underline{\Gamma \vdash (x : A) \to B} \coloneqq \underline{\Gamma} \vdash (x : \underline{A}) \to \underline{B}$$

$$g \circledast^{(x:A)\to B}_\gamma f \coloneqq (a : \underline{A}(g \circledast \gamma)) \mapsto g \circledast^B_{\langle \gamma, g^{-1}\circledast a\rangle} f_\gamma(g^{-1} \circledast^A_{g\circledast\gamma} a)$$

It is straightforward to check that both definitions satisfy the indexed action laws. $\square$

Definition 3.6. Given a G-action Γ and an indexed G-action $\Gamma \vdash A$, a term $\underline{\Gamma} \vdash t : \underline{A}$ is *equivariant* when its components are: $t(g \circledast^\Gamma \gamma) = g \circledast_\gamma t\gamma$ for all $\gamma \in \underline{\Gamma}, g \in \underline{G}$.

Equivariance of terms is preserved by reindexing and substitution. Furthermore, an equivariant term $\Gamma, x : A \vdash t : B$ induces an equivariant dependent function $\Gamma \vdash (x : A \mapsto t) : (x : A) \to B$, and two equivariant terms $\Gamma \vdash t : A$ and $\gamma : \Gamma \vdash s : B\langle \gamma, t\gamma\rangle$ induce an equivariant term $\Gamma \vdash \langle t, s\rangle : (x : A) \times B$. For more details and proofs, see Appendix D.

The ease of compositionality should come as no surprise given that the definition of an indexed action is in fact just a reformulation of the notion of a family internally to the topos of G-actions:

Proposition 3.7. *Let Γ be a G-action. Then, a Γ-indexed action A amounts to an action $\coprod_\Gamma A$ over the Grothendieck construction for which the projection is equivariant: $\underline{\coprod_\Gamma A} = \coprod_{\underline{\Gamma}} \underline{A}$ and $\pi_1 : \coprod_\Gamma A \circledast\longrightarrow \Gamma$.*

We have used this connection above to reformulate standard notions from dependent types to the symmetric setting, and continue to do so with canonisers, core functions, and the w.l.o.g. construction. Ignoring the additional indices, we recover the simply-typed notions from §2.

Definition 3.8. Let $\Gamma \vdash A$ be an indexed G-action. An *indexed canoniser* for A is a term $\underline{\Gamma} \vdash c : \underline{A} \to \underline{G}$. For each $a \in \underline{A}\gamma$, we call the symmetry $c_\gamma a \in \underline{G}$ the *c-canonising symmetry* of a, and the element $ca \circledast_\gamma a$ the *canonical form* of a. We define the family of *canonical* elements $\underline{\Gamma} \vdash \mathsf{Can}\, c$ with the fibres $(\mathsf{Can}\, c)\gamma \coloneqq \left\{ ca \circledast_{\gamma'} a \,\middle|\, \gamma' \in \Gamma, a \in \underline{A}\gamma', ca \circledast \gamma' = \gamma \right\}$ for each $\gamma \in \underline{\Gamma}$.

Example 3.9. To capture the symmetry of the lookup function from Example 3.2, we define a canoniser for the indexed S_2-action on $t : \mathsf{Tree}\, X \vdash \mathsf{InPath}\, t$:

$$c_{\mathsf{Node}\langle \ell,n,r\rangle}\mathsf{Nil} \coloneqq \mathtt{I} \qquad c_{\mathsf{Node}\langle \ell,n,r\rangle}(\mathsf{L} :: p) \coloneqq \mathtt{I} \qquad c_{\mathsf{Node}\langle \ell,n,r\rangle}(\mathsf{R} :: p) \coloneqq \mathtt{F}$$

The canonical elements in $\mathsf{InPath}\, t$ are Nil and $(\mathsf{L} :: p) \in \mathsf{InPath}(\mathsf{Node}\, \langle \ell, n, r\rangle)$ where $p \in \mathsf{InPath}\, \ell$ can be an arbitrary path, even one that moves right on any of its steps. $\square$

Definition 3.10. Take indexed G-actions $\Gamma \vdash A$ and $\Gamma, x : A \vdash B$, a subfamily $\underline{\Gamma} \vdash X \subseteq \underline{A}$, and a dependent function $\underline{\Gamma} \vdash f : (x : X) \to \underline{B}$. We write $\Gamma \vdash f : (x : X \subseteq A) \circledast\!\!\to B$ and say that f is *equivariant* when we have $g \circledast^A_\gamma x \in X(g \circledast \gamma) \implies f_{g \circledast \gamma}(g \circledast^A_\gamma x) = g \circledast^B_{\langle \gamma, x \rangle} f_\gamma x$ for all $g \in \underline{G}$, $\gamma \in \underline{\Gamma}$, and $x \in X\gamma$.

Definition 3.11. Let $\Gamma \vdash A$ and $\Gamma, x : A \vdash B$ be indexed G-actions, $\Gamma \vdash c : \underline{A} \to \underline{G}$ an indexed canoniser, and $\Gamma \vdash f : (x : \mathsf{Can}\, c) \to \underline{B}$ a dependent core function. Then, the *indexed w.l.o.g. construction* $\mathsf{wlog}(c, f) : \left(x : \underline{A}\right) \to \underline{B}$ is defined as:

$$\mathsf{wlog}(c, f)_\gamma \coloneqq (a : \underline{A}\gamma) \mapsto \left(c_\gamma a\right)^{-1} \circledast^B_{\langle c_\gamma a \circledast^\Gamma \gamma, c_\gamma a \circledast^A_\gamma a \rangle} f_{c_\gamma a \circledast^\Gamma \gamma}(c_\gamma a \circledast^A_\gamma a)$$

With all the indices precisely spelt out, the definition can look a bit intimidating, but since all of them are uniquely determined, there is no harm in writing $\mathsf{wlog}(c, f) = a \mapsto (ca)^{-1} \circledast^B f(ca \circledast^A a)$ as in the simply-typed case. And just as before in Theorem 2.13, the w.l.o.g. construction gives the unique equivariant extension of a core function.

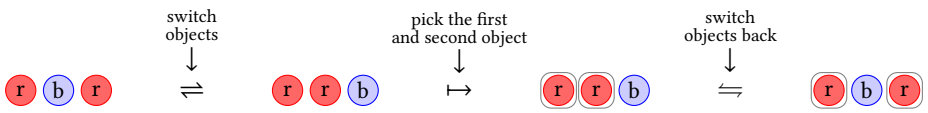
Theorem 3.12 (indexed representation)**.** *Let $\Gamma \vdash A$ and $\Gamma, x : A \vdash B$ be indexed G-actions and $\Gamma \vdash c : \underline{A} \to \underline{G}$ an indexed canoniser for A. If $\Gamma \vdash f : (x : \mathsf{Can}\, c \subseteq A) \circledast\!\!\to B$ is an equivariant dependent core function, then $\mathsf{wlog}(c, f) : (x : A) \circledast\!\!\to B$ is the unique equivariant function extending f to the whole $\underline{\Gamma} \vdash \underline{A}$.*

Example 3.13. Let us revisit the pigeonhole principle [31] from the introduction and extend it in two ways. First, instead of a proof, let us construct a function $\mathsf{findSame} : \left(xyz : \{\mathsf{r}, \mathsf{b}\}^3\right) \to \mathsf{Same}\, xyz$ where $\mathsf{Same}\, xyz \coloneqq \left\{(i, j) : (\mathsf{Fin}\, 3)^2 \,\middle|\, i \neq j \wedge \pi_i xyz = \pi_j xyz\right\}$ is the family of distinct positions of the same colour, indexed by the colouring xyz.

And second, we shall see that in the special case of the proof, there is another opportunity to use the w.l.o.g. argument. Informally, our proof goes as follows:

> W.l.o.g., the first object is red. The other two objects either have the same colour or different colours. If they have the same colour, choose $(1, 2)$. If they have different colours, w.l.o.g. the second object is red, so choose $(0, 1)$.

Such nested uses of w.l.o.g. may be suspicious and confusing, which is exactly why it is important to make all aspects of w.l.o.g. arguments explicit. In this case, this use is valid. For example, in when the second object is not red and the third is red, the inner argument exchanges their positions:



In the outer level, S_2 acts on $\{\mathsf{r}, \mathsf{b}\}$ by flipping the two colours, and on the domain $\{\mathsf{r}, \mathsf{b}\}^3$ with the component-wise product action we recalled in §2.2. On the codomain $\{\mathsf{r}, \mathsf{b}\}^3 \vdash \mathsf{Same}$, we lift the trivial action to preserve the elements of the fibre, but change the index. This is a sub-action (i.e. an action that preserves a subset) since if objects at two positions are of the same colour, they remain so after we exchange the colours. For example, $\mathsf{F} \circledast_{(\mathsf{r},\mathsf{r},\mathsf{b})} (0, 1) = (0, 1) \in \mathsf{Same}\, (\mathsf{b}, \mathsf{b}, \mathsf{r})$

By swapping colours, we make the first object red, thus the outer canoniser is:

$$\mathsf{makeFirstRed}(\mathsf{r}, b, c) \coloneqq \mathsf{I} \qquad \mathsf{makeFirstRed}(\mathsf{b}, b, c) \coloneqq \mathsf{F}$$

and using it, we can define

$$\mathsf{findSame} \coloneqq \mathsf{wlog}\left(\mathsf{makeFirstRed}, (\mathsf{r}, b, c) \mapsto \begin{cases} (1, 2) & (b = c) \\ \mathsf{findOtherRed}(b, c) & (b \neq c) \end{cases}\right)$$

where we still need to define

$$\mathsf{findOtherRed} : \left((y,z) : \left\{(b,c) \in \{\mathsf{r},\mathsf{b}\}^2 \,\middle|\, b \neq c\right\}\right) \to \mathsf{Same}\,(\mathsf{r},y,z)$$

Again, we employ symmetry, but now, the S_2-action on the domain swaps the two components of the pair, but not their colour, i.e., $\mathsf{F} \circledast (b,c) \coloneqq (c,b)$. For the codomain, define an action on $\mathsf{Fin}\,3$ that swaps 1 and 2, and extend it component-wise to the product $(\mathsf{Fin}\,3)^2$. Since b and c are of different colours, it is easy to check that this action again preserves the defining property of Same and thus lifts to a sub-action on the codomain. For example, $\mathsf{F} \circledast_{(\mathsf{r},\mathsf{b})} (0,1) = (0,2) \in \mathsf{Same}\,(\mathsf{r},\mathsf{b},\mathsf{r})$.

The inner canoniser now makes the second object red:

$$\mathsf{makeSecondRed}(\mathsf{r},\mathsf{b}) \coloneqq \mathsf{I} \qquad \mathsf{makeSecondRed}(\mathsf{b},\mathsf{r}) \coloneqq \mathsf{F}$$

and so $\mathsf{findOtherRed} \coloneqq \mathsf{wlog}(\mathsf{makeSecondRed}, (\mathsf{r},\mathsf{b}) \mapsto (0,1))$. Equivariance is immediate since both core functions are vacuously equivariant. □

3.3 Propositional reasoning

So far we have used the w.l.o.g. construction to define functions. To formalise mathematical proofs, we apply the Curry-Howard correspondence and identify propositions with types, and their proofs with terms. We could have also approached this differently and e.g., transferred the w.l.o.g. construction to higher-order logic or first-order logic, but this would require us to duplicate the definitions for concepts such as group actions or equivariance for proofs.

For concreteness, let us first borrow a running example of Schur's inequality from Harrison [16], which states that for all $x,y,z,t \geq 0$, we have

$$x^t(x-y)(x-z) + y^t(y-z)(y-x) + z^t(z-x)(z-y) \geq 0$$

Since the inequality is symmetric in the three variables, the standard proof assumes that w.l.o.g., $x \geq y \geq z$ and rearranges the expression to an evidently non-negative form.

Let $\mathsf{Prop} \coloneqq \{\mathsf{True},\mathsf{False}\}$ be the set of logical values. Each *proposition* $\Gamma \vdash \varphi : \mathsf{Prop}$ determines a family $\Gamma \vdash \mathsf{Proof}\,\varphi$ such that for each $\gamma \in \Gamma$, the fibre $(\mathsf{Proof}\,\varphi)\gamma = \mathbb{1} = \{\star\}$ when $\varphi\gamma = \mathsf{True}$ and $(\mathsf{Proof}\,\varphi)\gamma = \emptyset$ when $\varphi\gamma = \mathsf{False}$. As is conventional, we overload the notation and identify propositions with their proofs, writing $\Gamma \vdash \mathsf{Proof}\,\varphi$ simply as $\Gamma \vdash \varphi$. Then, any proof term $\Gamma \vdash p : \varphi$ is an explicit witness that φ holds at every $\gamma \in \Gamma$.

Example 3.14. We can write Schur's inequality as the proposition $(x,y,z) : \mathbb{R}^3_+ \vdash \mathsf{Schur}(x,y,z) : \mathsf{Prop}$ where

$$\mathsf{Schur}(x,y,z) \coloneqq \forall t > 0.\; x^t(x-y)(x-z) + y^t(y-z)(y-x) + z^t(z-x)(z-y) \geq 0$$

Recall from Example 2.1 that S_3 acts on $\mathbb{R}^3_+$ by permutation. The proposition Schur is invariant under those permutations. □

Definition 3.15. A proposition $\Gamma \vdash \varphi : \mathsf{Prop}$ is *invariant*, if it is equivariant as a term with respect to the trivial G-action on Prop, i.e. $\varphi(g \circledast \gamma) = \varphi\gamma$ for all $\gamma \in \Gamma$.

In addition to equivariant substitutions, like all equivariant terms, invariant propositions are also preserved by logical connectives and quantifiers. Each invariant proposition $\Gamma \vdash \varphi : \mathsf{Prop}$ induces a Γ-indexed action on its proof family $\Gamma \vdash \varphi$. Moreover, given a Γ-indexed action A, each invariant proposition $\Gamma, x : A \vdash \varphi : \mathsf{Prop}$ induces a Γ-indexed action on the subfamily $\Gamma \vdash \{x : A \mid \varphi\} \subseteq A$. For more details and proofs, again see Appendix D.

Example 3.16. Coming back to Schur's inequality, we can see that the w.l.o.g. argument again fits the now familiar pattern:

$$x_0, x_1, x_2 \in \mathbb{R}_+ \quad \overset{\text{sort using a permutation } \pi}{\mapsto} \quad x_{\pi 0} \geq x_{\pi 1} \geq x_{\pi 2} \quad \overset{\text{prove using additional assumption}}{\mapsto} \quad \mathsf{Schur}(x_{\pi 0}, x_{\pi 1}, x_{\pi 2}) \quad \overset{P \text{ is invariant under permutations}}{\mapsto} \quad \mathsf{Schur}(x_0, x_1, x_2)$$

Let us thus use the w.l.o.g. construction to obtain a proof, i.e. a function $\big((x, y, z) : \mathbb{R}^3_+\big) \to \mathsf{Schur}(x, y, z)$. From Example 3.14 we already have an S_3-action on $\mathbb{R}^3_+$, and since Schur is an invariant proposition, we also get an indexed S_3-action on $\mathbb{R}^3_+ \vdash \mathsf{Schur}$. For the canoniser, we take a stable sorting function $\mathsf{sort} : \mathbb{R}^3_+ \to \mathrm{S}_3$ from Example 2.6 so that Can sort is exactly the set of sorted triples. The last thing we need is the core function $f : \left\{(x, y, z) : \mathbb{R}^3_+ \,\middle|\, x \leq y \leq z\right\} \to \mathsf{Schur}(x, y, z)$, which we can define using the rearrangement in the standard proof. □

The resulting function $\mathsf{wlog}(\mathsf{sort}, f)$ is equivariant since in the case of propositions, core functions are always equivariant due to proof irrelevance:

Lemma 3.17. *Let* $\Gamma, x : A \vdash \varphi : \mathsf{Prop}$ *be an invariant proposition, and* $\Gamma \vdash c : \underline{A} \to \underline{G}$ *a canoniser. Every core function* $\Gamma \vdash f : \mathsf{Can}\ c \to \varphi$ *is equivariant.*

Harrison suggests proving the Schur's inequality using the following tactic[4], which states that it is sufficient to show permutation invariant propositions only for sorted triples [16, page 44]:

$$\frac{\forall x, y, z.\ P(x, y, z) \Rightarrow P(y, x, z) \wedge P(x, z, y) \qquad \forall x, y, z.\ (x \leq y) \wedge (y \leq z) \Rightarrow P(x, y, z)}{\forall x, y, z.\ P(x, y, z)}$$

Let us see how this bespoke tactic relates to our general w.l.o.g. construction. In contrast to our approach, the symmetry group S_3 is left implicit, as is the canoniser. Instead, only its set of canonical elements $\left\{(x, y, z) : \mathbb{R}^3_+ \,\middle|\, x \leq y \leq z\right\}$ is spelt out. Furthermore, the first assumption corresponds to an indexed S_3-action on the proposition P, and the second assumption is exactly the core function.

The latter is evident, while for the former, recall that the symmetry group S_3 is generated by the two transpositions $(0 \leftrightarrow 1)$ and $(1 \leftrightarrow 2)$. Since the generators determine the behaviour of the action on the whole group, one defines a S_3 action with one map for each of the generators:

$$(0 \leftrightarrow 1) \circledast_{(x,y,z)} - : P(x, y, z) \to P((0\,1) \circledast (x, y, z)) = P(y, x, z)$$
$$(1 \leftrightarrow 2) \circledast_{(x,y,z)} - : P(x, y, z) \to P((1\,2) \circledast (x, y, z)) = P(x, z, y)$$

These are exactly the contents of the first assumption.

What is even more important is that exposing the symmetric structure in proofs allows us to use existing results in group theory. For example, Harrison's proof for the tactic still explicitly lists all six permutations [16]. For S_3, this number may still be manageable, but in general, the group S_n has $n!$ elements, so a proof of a potential wlog_5 rule would already need to list prohibitively many cases, while the proof of wlog_n would not even be expressible. In contrast, we can lean on the general result that S_n is generated by a single cycle $(0\,1\,2 \cdots n-1)$ and a single adjacent transposition, for example $(0 \leftrightarrow 1)$, so it is always enough to spell out only two cases for the action.

A careful reader may observe that such definitions by generator cases need to respect all the equalities between elements of a particular group. For example in any S_n-action, $(0 \leftrightarrow 1) \circledast -$ must be its own inverse (amongst other relations) since $(0 \leftrightarrow 1) * (0 \leftrightarrow 1) = e$. However, in case of propositions, once more due to proof irrelevance, the condition is trivially satisfied.

[4] Available here: https://github.com/jrh13/hol-light/blob/master/int.ml#L612.

3.4 Equivariant canonisers

In §2.4 we saw that the issue with strict equivariance is that both $c(g \circledast a)$ and $g \circledast^{G^{-1}} ca$ canonise $g \circledast a$, but that they are not necessarily identical. This leads us to a weaker notion in which we collapse all symmetries that act equivalently on a particular element.

Definition 3.18. Take a G-action A. Then, for each $a \in \underline{A}$, we equip $\underline{G}$ with an *indistinguishability* relation $g \overset{a}{\sim} h \iff g \circledast a = h \circledast a$. If $[g]_{\overset{a}{\sim}}$ is the equivalence class of g in $\underline{G}/\overset{a}{\sim}$, we define an indexed G-action $A \vdash G/A$ with the fibre $(G/A)\ a \coloneqq \underline{G}/\overset{a}{\sim}$ and with $\circledast^{G/A}$ defined as: $g \circledast_a^{G/A} [h]_{\overset{a}{\sim}} \coloneqq [h * g^{-1}]_{\overset{g \circledast a}{\sim}}$.

The action is well-defined since $h_1 \overset{a}{\sim} h_2$ implies $h_1 * g^{-1} \overset{g \circledast a}{\sim} h_2 * g^{-1}$. The relation $\overset{a}{\sim}$ is closely related to the *stabiliser* $G_a \coloneqq \left\{ g \in \underline{G} \,\middle|\, g \circledast a = a \right\}$, i.e. a subgroup of all symmetries that preserve a. In particular, $g \overset{a}{\sim} h$ is equivalent to $(h^{-1} * g) \circledast a = a$, i.e. $h^{-1} * g \in G_a$.

Remark 3.19. Since the equivalence classes of the indistinguishability relation are precisely the cosets of the stabiliser group, we can recast the Orbit-Stabiliser theorem as an *equivariant* isomorphism between the actions on orbits and stabiliser cosets.

Given a G-action A, the fibres of a dependent family of *orbits* contain all the symmetric elements of an index, i.e. $a : \underline{A} \vdash G \circledast [a] \coloneqq \left\{ h \circledast a \,\middle|\, h \in \underline{G} \right\}$. On orbits, we can define an indexed action as:

$$g \circledast_a (h \circledast a \in G \circledast [a]) \coloneqq h \circledast a = (h * g^{-1}) \circledast (g \circledast a) \in G \circledast [g \circledast a]$$

Even though the action does not change the fibres, it is not trivial, since it changes the indices.

Theorem 3.20 (Orbit-Stabiliser). *Let A be a G-action. We have an equivariant isomorphism:*

$$a : A \vdash \varphi_a : (G/A)a \overset{\cong}{\circledast\longrightarrow} G \circledast [a] \qquad \varphi_a [h]_{\overset{a}{\sim}} \coloneqq h \circledast a \qquad \varphi_a^{-1}(h \circledast a) \coloneqq [h]_{\overset{a}{\sim}}$$

Coming back to equivariance of canonisers, the indexed action G/A generalises G^{-1} and allows us to consider a weaker property:

Definition 3.21. A canoniser $c : \underline{A} \to \underline{G}$ is *equivariant* when post-composing it with the quotient map yields an equivariant map $(a \mapsto [c\ a]_a) : (a : A) \circledast\longrightarrow (\underline{G}/A)a$, i.e. when $c(g \circledast^A a) \overset{g \circledast a}{\sim} g \circledast^{G^{-1}} ca$.

Example 3.22. For a totally ordered set P, we can define the S_2-action on $P \times P$ by $\mathsf{F} \circledast \langle p, q \rangle \coloneqq \langle q, p \rangle$ and a canoniser $c : \underline{A} \to \underline{S_2}$ such that $c\ \langle p, q \rangle = \mathsf{I}$ if $p \leq q$, and $c\ \langle p, q \rangle = \mathsf{F}$ if $p > q$. Then c is not strictly equivariant since for some $p \in P$, we have:

$$c\ (\mathsf{F} \circledast \langle p, p \rangle) = c\ \langle p, p \rangle = \mathsf{I} \neq \mathsf{F} = \mathsf{I} * \mathsf{F}^{-1} = \mathsf{F} \circledast_{\langle p,p \rangle} c\ \langle p, p \rangle$$

It is, however, equivariant, since $c\ (\mathsf{F} \circledast \langle p, p \rangle) = \mathsf{I} \overset{\langle p,p \rangle}{\sim} \mathsf{F} = \mathsf{F} \circledast_{\langle p,p \rangle} c\ \langle p, p \rangle$. □

Note that for every equivariant function f and every stabilising element $g \in G_a$, we have $g \circledast fa = f(g \circledast a) = fa$, thus $G_a \subseteq G_{fa}$, leading to a weaker condition on core functions.

Definition 3.23. Take indexed G-actions $\Gamma \vdash A$ and $\Gamma, x : A \vdash B$, a subfamily $\underline{\Gamma} \vdash X \subseteq \underline{A}$. A function $\underline{\Gamma} \vdash f : (x : X) \to \underline{B}$ is *stable* when it preserves the stabiliser: $G_x \subseteq G_{fx}$ for all $x \in X\gamma$.

Example 3.24. Just as in Example 2.16, consider a non-total ordering P and a stable sorting function, and the permutation sub-action on all tuples with distinct elements $\left\{ \vec{p} \in P^n \,\middle|\, \forall i, j \in \mathsf{Fin}\, n. p_i \neq p_j \right\}$. Then, we can define $\mathsf{indexMin} : \underline{P^n} \to \mathsf{Fin}\, n$ that finds the index of a minimal element by $\mathsf{indexMin} \coloneqq \mathsf{wlog}(\mathsf{sort}, (\_ \mapsto 0))$.

Now, for any two incomparable p, q, both $\langle p, q \rangle$ and $\langle q, p \rangle$ are sorted, thus canonical. However:

$$(0 \leftrightarrow 1) \circledast ((\_ \mapsto 0)\ \langle p, q \rangle) = (0 \leftrightarrow 1) \circledast 0 = 1 \neq 0 = (\_ \mapsto 0)\ \langle q, p \rangle = (\_ \mapsto 0)\ \big((0 \leftrightarrow 1) \circledast \langle p, q \rangle\big)$$

and so $(\_ \mapsto 0)$ is not equivariant on Can sort. It is, however, vacuously stable. Indeed, take a canonical $\vec{p}$ and a $\vec{p}$-stabilising permutation $\pi \in G_{\vec{p}}$. Since all the elements in $\vec{p}$ are distinct, yet $\pi \circledast \vec{p} = \vec{p}$, we have that $\pi = \mathrm{id}$. Therefore $\pi \in G_{f\vec{p}} = G_0$. □

So, in general, stability is strictly weaker than equivariance. Nonetheless, in the presence of an equivariant canoniser, stability and equivariance coincide.

Theorem 3.25. *Let $\Gamma \vdash A$ be an indexed G-action and let $\Gamma \vdash c : \underline{A} \to \underline{G}$ be an indexed canoniser. Then, c is equivariant iff for every indexed G-action $\Gamma, x : A \vdash B$, every stable core function $\Gamma \vdash f : (x : \mathsf{Can}\, c) \to \underline{B}$ is equivariant.*

Unlike before, where only semi-regular actions admitted strictly equivariant canonisers, here every action admits an equivariant canoniser, although we may need to invoke the axiom of choice.

Theorem 3.26. *Using the axiom of choice, every indexed G-action has an equivariant canoniser.*

To us, this use of the axiom of choice indicates that identifying symmetry is a creative activity, rather than a mechanical one.

4 Symmetric algebraic data types

Let us return to the tree lookup function $\mathsf{lookup} : (t : \mathsf{Tree}) \to \mathsf{InPath}\, t \to \mathbb{N}$ from Example 3.2. We can notice the symmetry in its two recursive cases

$$\mathsf{lookup}\left(\begin{array}{ccc} & \nearrow^{n} & \\ \ell & & \searrow_{r} \end{array}\right)(\mathsf{L} :: p) \coloneqq \mathsf{lookup}\, \ell\, p \qquad \mathsf{lookup}\left(\begin{array}{ccc} & \nearrow^{n} & \\ \ell & & \searrow_{r} \end{array}\right)(\mathsf{R} :: p) \coloneqq \mathsf{lookup}\, r\, p$$

and define the canoniser on paths by $\mathsf{goLeft}\, \mathsf{Nil} = \mathsf{goLeft}(\mathsf{L} :: p) = \mathtt{I}$ and $\mathsf{goLeft}(\mathsf{R} :: p) = \mathtt{F}$. Then, lookup can be defined as follows (with dependent types ruling out inaccessible cases):

$$\mathsf{lookup} \coloneqq \left(\begin{array}{ccc} & \nearrow^{n} & \\ \ell & & \searrow_{r} \end{array}\right) \mapsto \mathsf{wlog}\left(\mathsf{goLeft}, \begin{cases} \mathsf{Nil} & \mapsto n \\ \mathsf{L} :: p' & \mapsto \mathsf{lookup}\, \ell\, p' \end{cases}\right)$$

However, note that lookup and the core function are mutually recursive: lookup is defined in terms of the core function, which then recursively calls lookup on descent. Even though this recursion is structural, the interdependence through a higher-order function wlog makes it hard to convince the termination checker. Furthermore, as defined, the function is prohibitively inefficient, as it flips the whole tree on each step to the right.

We resolve both issues by framing our development in the framework of *initial algebras*. After a brief recall of functors and their initial algebras (§4.1), we generalise them to the symmetric setting (§4.2), show how we can reuse canonisers for traversals over the same constructors (§4.3), and show how to recover the asymptotic complexity through partial evaluation (§4.4). Finally, we combine these results with the previous section and adapt data types to the indexed setting (§4.5).

4.1 Initial algebras

In order to treat data types mathematically, we use the *initial algebra semantics*, in which each data type signature is specified by a *structure functor* $F : \mathrm{Set} \to \mathrm{Set}$. For example, trees over natural numbers and paths in them from Example 2.4 are given by the following two functors, which closely correspond to their usual data type definitions:

$$\begin{array}{ll} \mathsf{data}\ \mathsf{Tree} = \mathsf{Leaf} \mid \mathsf{Node}\ \mathsf{Tree}\ \mathbb{N}\ \mathsf{Tree} & \mathsf{data}\ \mathsf{Path} = \mathsf{Nil} \mid \mathsf{Step} :: \mathsf{Path} \\ F_{\mathsf{Tree}} X \coloneqq \mathbb{1} + X \times \mathbb{N} \times X & F_{\mathsf{Path}} X \coloneqq \mathbb{1} + \mathsf{Step} \times X \end{array}$$

Each functor F has an evident component-wise map $F : (X \to Y) \to (FX \to FY)$, for example:

$$F_{\mathsf{Tree}}\ f\ (\iota_1 \star) \coloneqq \iota_1 \star \qquad F_{\mathsf{Path}}\ f\ (\iota_1 \star) \coloneqq \iota_1 \star$$
$$F_{\mathsf{Tree}}\ f\ (\iota_2\ \langle \ell, n, r\rangle) \coloneqq \iota_2\ \langle f\ \ell, n, f\ r\rangle \qquad F_{\mathsf{Path}}\ f\ (\iota_2\ \langle d, p\rangle) \coloneqq \iota_2\ \langle d, f\ p\rangle$$

An *algebra* $\langle X, f\rangle$ for a functor F is a set X together with a function $f : FX \to X$, which describes the interaction with the constructors. An example F_{Tree}-algebra is the node-counting algebra $\langle \mathbb{N}, c : (\mathbb{1} + \mathbb{N} \times \mathbb{N} \times \mathbb{N}) \to \mathbb{N}\rangle$, given by $c(\iota_1 \star) \coloneqq 0$ and $c(\iota_2\ \langle m_1, \_, m_2\rangle) \coloneqq 1 + m_1 + m_2$. A *homomorphism* of F-algebras $h : \langle X, f\rangle \to \langle Y, g\rangle$ is a function $h : X \to Y$ such that $h \circ f = g \circ F\,h$.

We interpret data types by *initial* algebras, denoted by $\langle \boldsymbol{\mu} F, (\mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F)\rangle$, whose defining universal property is that for every other F-algebra $\langle X, f\rangle$, there is a unique homomorphism $\mathrm{fold}\ f : \langle \boldsymbol{\mu} F, \mathrm{roll}\rangle \to \langle X, f\rangle$. This homomorphism interprets structurally recursive functions. For example, the initial F_{Tree}-algebra, which we label by $\langle \mathsf{Tree}, \mathrm{roll}_{\mathsf{Tree}}\rangle$, satisfies:

$$\mathrm{roll}_{\mathsf{Tree}}\ \iota_1 \star = \mathsf{Leaf} \qquad \mathrm{roll}_{\mathsf{Tree}}\ \iota_2\ \langle \ell, n, r\rangle = \mathsf{Node}\ \ell\ n\ r$$

When readability is more important than precision, we are going to elide roll, and identify e.g. $\iota_1 \star$ with Leaf, and $\iota_2\ \langle \ell, n, r\rangle$ with $\mathsf{Node}\ \ell\ n\ r$.

For a given F-algebra $\langle X, f\rangle$, the universal property of the initial algebra becomes the recurrence:

$$\mathrm{fold}\ f\ \mathsf{Leaf} = f(\iota_1 \star) \qquad \mathrm{fold}\ f\ (\mathsf{Node}\ \ell\ n\ r) = f(\iota_2\ \langle \mathrm{fold}\ f\ \ell, n, \mathrm{fold}\ f\ r\rangle)$$

Through the algebra $\langle \mathbb{N}, c\rangle$, we can compute the number of nodes in a tree t by $\mathrm{fold}\ c\ t$ since

$$\mathrm{fold}\ c\ \mathsf{Leaf} = 0 \qquad \mathrm{fold}\ c\ (\mathsf{Node}\ \ell\ n\ r) = 1 + \mathrm{fold}\ c\ \ell + \mathrm{fold}\ c\ r$$

4.2 Initial equivariant algebras

We could transfer our framework to the context of data types by equipping the initial F-algebra with a G-action $(\circledast) : G \to \boldsymbol{\mu}F \to \boldsymbol{\mu}F$. However, that is not ideal, as it does not tell us what to do for other F-algebras. Instead, we take a more refined construct that describes the interaction of the group with the data type constructors:

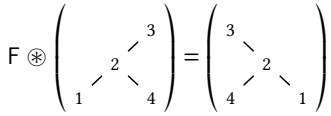
Definition 4.1. Let G be a group. A *G-symmetry-strong functor* $F = \langle \underline{F}, \mathrm{st}^{G;F}\rangle$, or a *$G$-strong functor* for short, is a functor $\underline{F} : \mathrm{Set} \to \mathrm{Set}$ together with a *G-symmetry strength* $\mathrm{st}^{G;F}$, or *G-strength* for short, which is a natural transformation $\mathrm{st}^{G;F}_X : (\underline{G} \times \underline{F}X) \to \underline{F}(\underline{G} \times X)$ satisfying

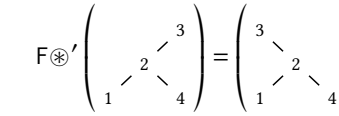
$$\mathrm{st}_X(e, u) = \underline{F}(x \mapsto \langle e, x\rangle)\ u$$
$$\mathrm{st}_X(g * h, u) = \underline{F}(\langle g', h', x\rangle \mapsto \langle g' * h', x\rangle)(\mathrm{st}_{G\times X}(g, \mathrm{st}_X(h, u)))$$

for all $u \in \underline{F}X$ and for all $g, h \in \underline{G}$.

Note that unlike group actions, symmetry strengths also feature the group in the codomain, which allows us to specify how to act on particular sub-terms.

Example 4.2. We can imagine two different S_2-actions on trees, one that recursively flips the whole tree, and one that flips only the top node, with corresponding strengths:

$\mathsf{F} \circledast$  $=$

$\mathsf{F} \circledast'$  $=$

$$\mathrm{st}\ \langle \mathsf{F}, \mathsf{Node}\ \ell\ n\ r\rangle \coloneqq \mathsf{Node}\ \langle \mathsf{F}, r\rangle\ n\ \langle \mathsf{F}, \ell\rangle \qquad \mathrm{st}'\ \langle \mathsf{F}, \mathsf{Node}\ \ell\ n\ r\rangle \coloneqq \mathsf{Node}\ \langle \mathtt{I}, r\rangle\ n\ \langle \mathtt{I}, \ell\rangle$$

The first strength propagates F to the subtrees, while the second strength does not by pairing with $\mathtt{I}$.

We only need to define the action for F since the first axiom forces the strength at $\langle \mathtt{I}, u\rangle$ to be $F(x \mapsto \langle \mathtt{I}, x\rangle)u$, i.e., $\mathrm{st}\,\langle \mathtt{I}, \mathsf{Node}\,\ell n r\rangle \coloneqq \mathrm{st}'\,\langle \mathtt{I}, \mathsf{Node}\,\ell n r\rangle \coloneqq \mathsf{Node}\,\langle \mathtt{I}, \ell\rangle n \langle \mathtt{I}, r\rangle$. For leaves, we define both strengths to act trivially $\mathrm{st}\,\langle g, \mathsf{Leaf}\rangle \coloneqq \mathrm{st}'\,\langle g, \mathsf{Leaf}\rangle \coloneqq \mathsf{Leaf}$. □

Remark 4.3. The notion of a symmetry strength is related, but different from the usual Cartesian or monoidal strength. Recall that a *Cartesian strength* for a functor $F : \mathrm{Set} \to \mathrm{Set}$ is a natural transformation $\mathrm{st}_X^{\Gamma;F} : \Gamma \times FX \to F(\Gamma \times X)$ satisfying $\mathrm{st}_{\mathbb{1},X}(\star, u) = F(x \mapsto \langle \star, x\rangle)u$ for all $u \in FX$, and $\mathrm{st}_{\Gamma\times\Delta,X}((\gamma,\delta),u) = F(\langle\gamma',\delta',x\rangle \mapsto \langle\langle\gamma',\delta'\rangle,x\rangle)\left(\mathrm{st}_{\Gamma,\Delta\times X}(\gamma, \mathrm{st}_{\Delta,X}(\delta,u))\right)$ for all $\gamma \in \Gamma$, and $\delta \in \Delta$. Every Cartesian strength is also a G-strength, which leaves constructors as they are and propagates the same group element into the sub-terms. For example, for trees, it acts as:

$$\mathrm{st}_X\,\langle \gamma, \mathsf{Leaf}\rangle \coloneqq \mathsf{Leaf} \qquad \mathrm{st}_X\,\langle \gamma, \mathsf{Node}\ \ell\ n\ r\rangle \coloneqq \mathsf{Node}\ \langle \gamma, \ell\rangle\ n\ \langle \gamma, r\rangle$$

We use G-symmetry strengths to lift any action on X to an action on FX:

Lemma 4.4. *Let G be a group and F a G-strong functor. Then, for every G-action A, we have a G-action FA given by:*

$$\underline{FA} \coloneqq \underline{F}\,\underline{A} \qquad (\circledast^{FA}) : \underline{G} \times \underline{FA} \xrightarrow{\mathrm{st}_{\underline{A}}} \underline{F}(\underline{G} \times \underline{A}) \xrightarrow{F(\circledast^{A})} \underline{FA}$$

Furthermore, if $h : A \circledcirc\!\!\to B$ is equivariant, so is $F\,h : FA \circledcirc\!\!\to FB$.

Example 4.5. The two tree symmetry strengths from Example 4.2 induce the two expected actions: action induced by the first strength flips the node and propagates symmetry, while for the second strength, the action only flips the top node and does not propagate the symmetry further. Cartesian strengths induce the pointwise lifting of actions. □

Symmetry-strong functors not only lift actions, but as we will see shortly, also uniquely determine actions on their initial algebras. Even more, the resulting algebra structure and the action are compatible in the following sense:

Definition 4.6. Let F be a G-strong functor. An *equivariant F-algebra* $\langle A, f\rangle$ consists of a G-action A and an equivariant map $f : FA \circledcirc\!\!\to A$, i.e., an $\underline{F}$-algebra $\langle \underline{A}, f\rangle$ such that:

$$\begin{array}{ccccc} \underline{G} \times \underline{FA} & & \xrightarrow{\mathbf{st}} & & \underline{F}(\underline{G} \times \underline{A}) \xrightarrow{F(\circledast)} \underline{FA} \\ {\scriptstyle \mathrm{id} \times f}\downarrow & & = & & \downarrow{\scriptstyle f} \\ \underline{G} \times \underline{A} & & \xrightarrow[(\circledast)]{} & & \underline{A} \end{array}$$

An *equivariant algebra homomorphism* $h : \langle A, f\rangle \circledcirc\!\!\to \langle B, g\rangle$ is simultaneously an $\underline{F}$-homomorphism $h : \langle \underline{A}, f\rangle \to \langle \underline{B}, g\rangle$ that is also equivariant $h : A \circledcirc\!\!\to B$.

Recall that initial algebra semantics supports a modular account of data types [27]: coproducts of functors are given pointwise, and an algebra for the coproduct functor is given by an algebra structures for each functor over the same carrier set. This modularity extends to symmetry-strong functors and their equivariant algebras:

Proposition 4.7 (modularity). *Let F_1, F_2 be G-strong functors with symmetry strengths $\mathrm{st}_1, \mathrm{st}_2$. The cotupled map $[\mathrm{st}_1, \mathrm{st}_2]\,\langle g, \iota_i x\rangle \coloneqq \iota_i\,\mathrm{st}_i\,\langle g, x\rangle$ is a G-symmetry strength for the coproduct functor $\underline{F}_1 + \underline{F}_2$. Moreover, given an action A, an equivariant algebra $f : (F_1 + F_2)A \circledcirc\!\!\to A$ for this strength amounts to a pair of equivariant algebra structures $f_i : F_i A \circledcirc\!\!\to A$.*

Using (cf. Appendix E) an appropriate notion of parameterised initiality originally proposed by Bird and Paterson [5], the symmetry strength lets us equip the initial algebra with an action that makes it the *initial equivariant algebra*. Then, to define an equivariant homomorphism $h : \boldsymbol{\mu} F \circledast\!\!\longrightarrow A$, we need to define an equivariant F-algebra $g : FA \circledast\!\!\longrightarrow A$, which we can do using the established w.l.o.g. construction.

Theorem 4.8 (equivariant initiality)**.** *Let F be a G-strong functor and* roll $: \underline{F}\boldsymbol{\mu}\underline{F} \to \boldsymbol{\mu}\underline{F}$ *the initial algebra of $\underline{F}$. Then, there is a unique action $\boldsymbol{\mu} F = \langle \boldsymbol{\mu}\underline{F}, (\circledast^{\mu F}) \rangle$ satisfying*

$$\begin{array}{ccccc} \underline{G} \times \underline{F}\boldsymbol{\mu}\underline{F} & \xrightarrow{\mathbf{st}} & \underline{F}(\underline{G} \times \boldsymbol{\mu}\underline{F}) & \xrightarrow{F(\circledast^{\mu F})} & \underline{F}\boldsymbol{\mu}\underline{F} \\ {\scriptstyle \mathrm{id} \times \mathbf{roll}}\downarrow & & = & & \downarrow{\scriptstyle \mathbf{roll}} \\ \underline{G} \times \boldsymbol{\mu}\underline{F} & & \xrightarrow[(\circledast^{\mu F})]{} & & \boldsymbol{\mu}\underline{F} \end{array}$$

making $\langle \boldsymbol{\mu} F, \mathrm{roll} \rangle$ into an initial equivariant F-algebra: for any equivariant F-algebra $f : FA \circledast\!\!\longrightarrow A$, the unique $\underline{F}$-homomorphism is equivariant: fold $f : \boldsymbol{\mu} F \circledast\!\!\longrightarrow A$.

Example 4.9. To see all the modular building blocks in action, define the path lookup function from Example 2.4. The mirroring symmetry of the type Path is captured by the strength:

$$\mathrm{st}^{\mathsf{Path}}\ \langle g, \mathsf{Nil} \rangle \coloneqq \mathsf{Nil} \qquad \mathrm{st}^{\mathsf{Path}}\ \langle g, d :: x \rangle \coloneqq (g \circledast d) :: \langle g, x \rangle$$

and we can compute that the action given by Theorem 4.8 satisfies:

$$\begin{aligned} g \circledast (d :: p) &= \big((\circledast) \circ (\mathrm{id} \times \mathrm{roll})\big)\ \langle g, \iota_2\ \langle d, p \rangle \rangle \underset{\text{equivariant algebra}}{=} \big(\mathrm{roll} \circ F_{\mathsf{Path}}(\circledast) \circ \mathrm{st}\big)\ \langle g, \iota_2\ \langle d, p \rangle \rangle \\ &\underset{\text{strength def}}{=} \big(\mathrm{roll} \circ F_{\mathsf{Path}}(\circledast)\big)(\iota_2\ \langle g \circledast d, \langle g, p \rangle \rangle) = \mathrm{roll}(\iota_2\ \langle g \circledast d, g \circledast p \rangle) = (g \circledast d) :: g \circledast p \end{aligned}$$

A similar calculation for Nil shows we get the expected action for Path, and a similar set of calculations shows we get the expected recursive action for Tree from the propagating strength from Example 4.2. We now construct lookup and simultaneously show it is equivariant as a map $\mathsf{Path} \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N})$. By Theorem 4.8, we can instead define an equivariant algebra:

$$\mathrm{lookup}^{\mathrm{Elim}} : F_{\mathsf{Path}}(\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N}) \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N})$$

Thanks to the Modularity Proposition 4.7, it suffices to define two equivariant algebra structures for each case of summand of $F_{\mathsf{Path}} = \mathbb{1} + \mathsf{Step} \times -$.

The Nil summand's algebra structure $\mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}} : \mathbb{1} \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N})$:

$$\mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}} \star\ \mathsf{Leaf} \coloneqq \mathsf{Nothing} \qquad \mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}} \star\ (\mathsf{Node}\ \ell\ n\ r) \coloneqq \mathsf{Just}\ n$$

To show that $\mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}}$ is equivariant, we refer to the fact that currying preserves equivariance, and calculate the behaviour of the uncurried version on the only interesting case:

$$\mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}}(\mathsf{F} \circledast \langle \star, \mathsf{Node}\ \ell\ n\ r \rangle) = \mathrm{lookup}^{\mathrm{Elim}}_{\mathsf{Nil}} \langle \star, \mathsf{Node}\ (\mathsf{F} \circledast r)\ n\ (\mathsf{F} \circledast \ell) \rangle = \mathsf{Just}\ n = \mathsf{F} \circledast \mathsf{Just}\ n$$

For $(::)$, define the algebra $\mathrm{lookup}^{\mathrm{Elim}}_{(::)} : \mathsf{Step} \times (\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N}) \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Maybe}\,\mathbb{N})$. Here, we use w.l.o.g. reasoning, as stepping right is the same as stepping left on the flipped tree with the rest of the path flipped. Define the Step-canoniser $c' : \mathsf{Step} \circledast\!\!\longrightarrow {\mathrm{S}_2}^{-1}$ that ensures canonical steps are left steps, i.e. $c'\mathsf{L} = \mathsf{I}$ and $c'\mathsf{R} = \mathsf{F}$. Then, extend it to a canoniser c for the whole domain through projection via Lemma 2.8. Its canonical elements are the tuples $\langle \mathsf{L}, f \rangle$, where f is the recursive

call we make when traversing (necessarily left). Thus, we eliminated the symmetric cases in the following core function $\mathsf{lookLeft} : \mathsf{Can}\ c \longrightarrow \mathsf{Maybe}\ \mathbb{N}$:

$$\mathsf{lookLeft}\ \langle \mathsf{L}, f\rangle \coloneqq \begin{cases} \mathsf{Leaf} & \mapsto \mathsf{Nothing} \\ \mathsf{Node}\ \ell\ n\ r & \mapsto f\ell \end{cases}$$

Now, we only need to assemble the pieces. As c' is strictly equivariant, so is c by Lemma 2.22, and by Theorem 2.17, all core functions are equivariant, including lookLeft. Thus $\mathsf{lookup}^{\mathrm{Elim}}_{(::)}$ is equivariant by Theorem 2.13. By Proposition 4.7, so is $\mathsf{lookup}^{\mathrm{Elim}}$, and by Theorem 4.8 so is lookup. □

4.3 Data type traversals w.l.o.g.

The steps in Example 4.9 are common, and we often traverse a data type in different ways but with the same canoniser. In this section, we present reusable abstractions that streamline these steps.

Definition 4.10. Let Γ be a G-action, and F a G-strong functor. A *canoniser* for F is a natural transformation $c_X : \underline{FX} \longrightarrow \underline{G}$. Each such canoniser induces the *canonical elements functor*:

$$\mathsf{Can}\ c : G\text{-}\mathsf{Act} \longrightarrow \mathsf{Set} \quad (\mathsf{Can}\ c)A \coloneqq \mathsf{Can}_{FA}\ c_{\underline{A}} \coloneqq \left\{(c_{\underline{A}}u) \circledast^{FA} u \,\middle|\, u \in \underline{FA}\right\} \quad (\mathsf{Can}\ c)h : u \mapsto (\underline{Fh})u$$

A canoniser c thus induces a natural assignment of canonisers $c_{\underline{A}} : \underline{FA} \longrightarrow \underline{G}$ for each lifted G-action FA. The naturality condition means canonisers are determined by their component at the singleton $\mathbb{1}$, i.e., by a canoniser for $\underline{F\mathbb{1}}$. The unique extension of such a canoniser $c_{\mathbb{1}} : \underline{F\mathbb{1}} \longrightarrow \underline{G}$ to a canoniser for F is given by the following composition $c_X : \underline{FX} \xrightarrow{\underline{F\langle\rangle}} \underline{F\mathbb{1}} \xrightarrow{c_{\mathbb{1}}} \underline{G}$.

Example 4.11. The S_2-strong functor F_{Path} has the canoniser (cf. p. 18) and canonical elements:

$$\mathsf{goLeft}\ \mathsf{Nil} \coloneqq \mathtt{I} =: \mathsf{goLeft}(\mathsf{L} :: p) \quad \mathsf{goLeft}(\mathsf{R} :: p) \coloneqq \mathtt{F} \qquad \mathsf{Can}_A\ \mathsf{goLeft} \coloneqq \left\{\mathsf{Nil}, \mathsf{L} :: a \,\middle|\, a \in \underline{A}\right\}$$ □

We can now streamline the data we need to specify to traverse a symmetric data type using an algebra, or to reason about such traversals using symmetry:

Definition 4.12. Let F be a G-strong functor and $c : \underline{F} \longrightarrow \underline{G}$ a canoniser for it. A *c-core F-algebra* $\langle A, f\rangle$ consists of a G-action A equipped with a function $f : (\mathsf{Can}\ c)A \longrightarrow \underline{A}$. It is *equivariant* when f is equivariant $f : ((\mathsf{Can}\ c)A \subseteq FA) \circledast\!\!\longrightarrow A$. A *homomorphism* of equivariant c-core F-algebras $h : \langle A, f\rangle \longrightarrow \langle B, g\rangle$ is an equivariant map $h : A \circledast\!\!\longrightarrow B$ satisfying: $h(f\,u) = g\,((\mathsf{Can}\ c)h\,u)$.

We can rearrange the constructs in Example 4.9 using these concepts:

Example 4.13. We have the following equivariant goLeft-core F_{Path}-algebra:

$$\mathsf{lookup}^{\mathrm{Core}} : (\mathsf{Can}\ \mathsf{goLeft})(\mathsf{Tree} \longrightarrow \mathsf{Triv}\mathsf{Maybe}\ \mathbb{N}) \longrightarrow \underline{\mathsf{Tree} \longrightarrow \mathsf{Triv}\mathsf{Maybe}\ \mathbb{N}} = (\mathsf{Tree} \longrightarrow \mathsf{Maybe}\ \mathbb{N})$$

$$\mathsf{Nil} \mapsto \begin{cases} \mathsf{Leaf} & \mapsto \mathsf{Nothing} \\ \mathsf{Node}\ \ell\ n\ r & \mapsto n \end{cases} \qquad \left(\mathsf{L} :: (\mathsf{Tree} \xrightarrow{\mathrm{rec}} \mathsf{Maybe}\ \mathbb{N})\right) \mapsto \begin{cases} \mathsf{Leaf} & \mapsto \mathsf{Nothing} \\ \mathsf{Node}\ \ell\ n\ r & \mapsto \mathrm{rec}\ \ell \end{cases}$$ □

These ingredients streamline the steps of Example 4.9, using the portmanteau fold wlog:

$$\left(\mathsf{Can}_{FA}\ c_{\underline{A}} = (\mathsf{Can}\ c)A \xrightarrow{f} \underline{A}\right) \mapsto \left(F\underline{A} \xrightarrow{\mathrm{wlog}(c_{\underline{A}}, f)} \underline{A}\right) \mapsto \left(\mathrm{flog}_A(c, f) : \boldsymbol{\mu}\underline{F} \xrightarrow{\text{fold wlog}(c_{\underline{A}}, f)} \underline{A}\right)$$

We chain Theorem 2.13 and the Equivariant Initiality Theorem 4.8:

Theorem 4.14 (core initiality). *Let F be a G-strong functor,* $\mathrm{roll} : \underline{F\boldsymbol{\mu}\underline{F}} \longrightarrow \boldsymbol{\mu}\underline{F}$ *an initial algebra of $\underline{F}$, and c a canoniser for F. Then restricting* roll *to canonical elements makes $\boldsymbol{\mu}F$ into the initial c-core F-algebra* $\mathrm{roll} : (\mathsf{Can}\ c)\boldsymbol{\mu}F \circledast\!\!\longrightarrow \boldsymbol{\mu}F$*: for any equivariant c-core F-algebra $f : (\mathsf{Can}\ c)A \circledast\!\!\longrightarrow A$, the unique equivariant c-core F-algebra homomorphism into f is:* $\mathrm{flog}(c, f) : \boldsymbol{\mu}F \circledast\!\!\longrightarrow A$.

This theorem implies, for example, that the lookup function from Example 4.13 is the unique equivariant goLeft-core F_{Path}-homomorphism given by flog.

Example 4.15. We reuse the canoniser to define the *subtree projection* function

$$\mathsf{subtree} \coloneqq \mathsf{flog}(\mathsf{goLeft}, \mathsf{subtree}^{\mathrm{Core}}) : \mathsf{Path} \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Tree})$$

$$\mathsf{subtree}^{\mathrm{Core}} : (\mathsf{Can}\,\mathsf{goLeft})(\mathsf{Tree} \to \mathsf{Tree}) \circledast\!\!\longrightarrow (\mathsf{Tree} \to \mathsf{Tree})$$

$$\mathsf{Nil} \mapsto \begin{cases} \mathsf{Leaf} & \mapsto \mathsf{Nothing} \\ \mathsf{Node}\ \ell\, n\, r & \mapsto \mathsf{Just}(\mathsf{Node}\ \ell\, n\, r) \end{cases} \qquad (\mathsf{L} :: \mathsf{Tree} \xrightarrow{\mathrm{rec}} \mathsf{Tree}) \mapsto \begin{cases} \mathsf{Leaf} & \mapsto \mathsf{Nothing} \\ \mathsf{Node}\ \ell\, n\, r & \mapsto \mathrm{rec}\,\ell \end{cases} \qquad \square$$

4.4 Mathematical foundations for performance engineering

By unfolding the definition of lookup, we can compute its behaviour on the symmetric case, recalling that the exponential action conjugates, $(g \circledast f)(a) = g \circledast f(g^{-1} \circledast a)$:

$$\begin{aligned} &\mathsf{lookup}\,(\mathsf{R} :: p)\,(\mathsf{Node}\ \ell\, n\, r) = \big(\mathsf{fold}(\mathsf{lookup}^{\mathrm{Elim}})\,(\mathsf{R} :: p)\big)(\mathsf{Node}\ \ell\, n\, r) \\ &= \big(\mathsf{lookup}^{\mathrm{Elim}}\,(\iota_2\,\langle \mathsf{R}, \mathsf{lookup}\,p\rangle)\big)(\mathsf{Node}\ \ell\, n\, r) = \big(\mathsf{F} \circledast \mathsf{lookLeft}\,\langle \mathsf{L}, \mathsf{F} \circledast \mathsf{lookup}\,p\rangle\big)(\mathsf{Node}\ \ell\, n\, r) \\ &= \mathsf{F} \circledast \Big(\mathsf{lookLeft}\,\langle \mathsf{L}, \mathsf{F} \circledast \mathsf{lookup}\,p\rangle\,\big(\mathsf{Node}\,(\mathsf{F} \circledast r)\,n\,(\mathsf{F} \circledast \ell)\big)\Big) \\ &= \mathsf{F} \circledast \big((\mathsf{F} \circledast \mathsf{lookup}\,p)\,(\mathsf{F} \circledast r)\big) = \mathsf{F} \circledast \big(\mathsf{F} \circledast (\mathsf{lookup}\,p\,(\mathsf{F} \circledast \mathsf{F} \circledast r))\big) = \mathsf{lookup}\,p\,r \end{aligned}$$

This is the result we want, but in the process, it first flips the whole tree $\mathsf{Node}\ \ell\, n\, r$ and then flips the remaining subtree $\mathsf{F} \circledast r$ back to r. This effect is compounded for subtree, where we flip the input tree on every symmetric call, and then flip the result on the way back. Naively implementing the functions lookup and subtree might retrace these steps computationally, leading high time and space complexity. In this section, we demonstrate three techniques that reduces this overhead. Faithful to our motivation, here we only develop the mathematical foundations: demonstrating which mathematical structures underlie the efficient implementation. We want to explore tools for systematic performance engineering in future work based on these foundations.

We employ three different techniques. First, instead of working with S_2-action over Tree, we use the free S_2-action $\mathsf{S}_2 \times \mathsf{Tree}$. As we will see, this design accumulates the symmetry and leaves the tree unchanged. Second, as we traverse the input Path, we also pattern-match on the tree. We will use the S_2-strength to do so, giving us the appropriately symmetric perspective on the data structure $\mathsf{S}_2 \times \mathsf{Tree}$. An implementation that employs this technique exhibits performance that is similar to parameterised implementations, but uses symmetric programming. Employing these techniques, we define an auxiliary function $\mathsf{subtree}^{\mathrm{aux}} : \mathsf{Path} \to (\mathsf{S}_2 \times \mathsf{Tree} \circledast\!\!\longrightarrow \mathsf{S}_2 \times \mathsf{Maybe}\,\mathsf{Tree})$. While we could implement $\mathsf{subtree}\,p\,t \coloneqq \mathsf{let}\ \langle g, s\rangle = \mathsf{subtree}^{\mathrm{aux}}\,p\,\langle \mathtt{I}, t\rangle\ \mathsf{in}\ g \circledast s$, the final multiplication is not necessary. Using the Core Initiality Theorem 4.14, we will show that if $(g, s) = \mathsf{subtree}^{\mathrm{aux}}\,p\ \langle \mathtt{I}, t\rangle$, then necessarily $g = \mathtt{I}$, and so we may implement $\mathsf{subtree}\,p\,t \coloneqq \pi_2 \circ \mathsf{subtree}^{\mathrm{aux}}\,p\ \langle \mathtt{I}, t\rangle$.

Delayed evaluation with free actions. We use a general principle from modern partial evaluation folklore. Instead of performing an expensive operation repeatedly, axiomatise its properties and design an efficient implementation for its free algebra. Then evaluate the free algebra back into the desired structure after partial evaluation. In our case, the free action $\mathsf{S}_2 \times \mathsf{Tree}$ is efficient enough for our purposes. Accordingly, we define a goLeft-core F_{Path}-algebra over $\mathsf{S}_2 \times \mathsf{Tree} \to \mathsf{S}_2 \times \mathsf{Maybe}\,\mathsf{Tree}$.

(De-)construction of auxiliary data types with their G-strength. The subtree function deconstructs the tree alongside the input Path. If we used the delayed evaluation technique, we no longer have a tree that's compatible with the given path. Instead, we have a pair $\langle g, t\rangle$, and it is the tree $g \circledast t$ that is compatible with the given path. The crucial observation is that we only need the top

constructor from the tree. We can therefore use the S_2-strength for F_{Tree} to obtain $\mathsf{st}(g, \mathsf{unroll}\ t)$ from $F_{\mathsf{Tree}}(\mathsf{S}_2 \times \mathsf{Tree})$. Whether the tree in question is flipped or not, applying the strength ensures the correct sub-structure resides in the left sub-tree.

Combining the two techniques, we define the following goLeft-core F_{Path}-algebra:

$$\mathsf{subtree}^{\mathsf{auxCore}} : (\mathsf{Can}\ \mathsf{goLeft})\ (\mathsf{S}_2 \times \mathsf{Tree} \to \mathsf{S}_2 \times \mathsf{Maybe}\ \mathsf{Tree}) \to (\mathsf{S}_2 \times \mathsf{Tree} \to \mathsf{S}_2 \times \mathsf{Maybe}\ \mathsf{Tree})$$

$$\mathsf{Nil} \mapsto \left(\langle g, t \rangle \mapsto \begin{cases} \mathsf{Nothing} & (\mathsf{st}\ \langle g, t \rangle = \mathsf{Leaf}) \\ \langle g, \mathsf{Just}\ t \rangle & \text{otherwise} \end{cases} \right)$$

$$(\mathsf{L} :: \mathsf{S}_2 \times \mathsf{Tree} \xrightarrow{\mathsf{rec}} \mathsf{S}_2 \times \mathsf{Tree}) \mapsto \left(\langle g, t \rangle \mapsto \begin{cases} \mathsf{Nothing} & (\mathsf{st}\ \langle g, t \rangle = \mathsf{Leaf}) \\ \mathsf{rec}\ \langle g_\ell, \ell \rangle & (\mathsf{st}\ \langle g, t \rangle = \mathsf{Node}\ \langle g_\ell, \ell \rangle n\ \langle g_r, r \rangle) \end{cases} \right)$$

This goLeft-core F_{Path}-algebra is equivariant. The only case that needs checking has the canonical paths Nil, and $\mathsf{F} \circledast \mathsf{Nil}$. We show that $\mathsf{F} \circledast \mathsf{subtree}^{\mathsf{auxCore}}\ \mathsf{Nil}\ \langle g, t \rangle = \mathsf{subtree}^{\mathsf{auxCore}}\ \mathsf{Nil}\ \langle \mathsf{F} * g, t \rangle$. If $\mathsf{st}\ \langle g, t \rangle = \mathsf{Leaf}$, then by the strength axiom, $\mathsf{st}\ \langle \mathsf{F} * gt, =\rangle\ F_{\mathsf{Path}}(\circledast)\ \mathsf{st}(\mathsf{F}, \mathsf{Leaf}) = \mathsf{Leaf}$. The proof obligation follows: $LHS = \mathsf{F} \circledast \mathsf{Nothing} = \mathsf{Nothing} = RHS$. If $\mathsf{st}\ \langle g, t \rangle = \mathsf{Node}\ \langle g_\ell, \ell \rangle n \langle g_r, r \rangle$, then by the strength axiom, $\mathsf{st}\ \langle \mathsf{F} * g, t \rangle = \mathsf{Node}\ \langle \mathsf{F} * g_r, r \rangle n \langle \mathsf{F} * g_\ell, \ell \rangle$, i.e., both sides take the second branch. The proof obligation follows too: $LHS = \mathsf{F} \circledast \langle g, \mathsf{Just}\ t \rangle = \langle \mathsf{F} \circledast g, \mathsf{Just}\ t \rangle = RHS$.

We can now define $\mathsf{subtree}^{\mathsf{aux}} \coloneqq \mathsf{flog}(\mathsf{goLeft}, \mathsf{subtree}^{\mathsf{auxCore}})$. We have equivariant core algebras, and by the Core Initiality Theorem 4.14, we can relate the auxiliary definition with our target definition $(\circledast^{\mathsf{Maybe}\ \mathsf{Tree}}) \circ \mathsf{subtree}^{\mathsf{aux}}\ p\ \langle g, t \rangle = \mathsf{subtree}\ p (g \circledast^{\mathsf{Tree}} t)$. Similarly, we have $\mathsf{subtree}^{\mathsf{aux}}\ p(g, t) = g$. These results show that $\mathsf{subtree}\ p\ t = \pi_2\ (\mathsf{subtree}^{\mathsf{aux}}\ p(\mathsf{I}, t))$. An implementation following this design never flips the tree. For example, when $\ell \neq \mathsf{Leaf}$, we have:

$$\begin{aligned} \mathsf{subtree}(\mathsf{R} :: \mathsf{Nil})(\mathsf{Node}\ \ell\ nr) &= \pi_2\ (\mathsf{subtree}^{\mathsf{aux}}(\mathsf{R} :: \mathsf{Nil})\ \langle \mathsf{I}, \mathsf{Node}\ \ell\ nr \rangle) \\ &= \pi_2 \left(\mathsf{F} \circledast \mathsf{subtree}^{\mathsf{auxCore}}(\mathsf{L} :: \mathsf{subtree}^{\mathsf{aux}} \mathsf{Nil})\ \langle \mathsf{F}, \mathsf{Node}\ \ell\ nr \rangle \right) \\ &= \pi_2\ (\mathsf{F} \circledast \mathsf{subtree}^{\mathsf{aux}} \mathsf{Nil}\ \langle \mathsf{F}, \ell \rangle) = \pi_2\ (\mathsf{F} \circledast \langle \mathsf{F}, \mathsf{Just}\ \ell \rangle) = \langle \mathsf{I}, \mathsf{Just}\ \ell \rangle = \mathsf{Just}\ \ell \end{aligned}$$

There are further opportunities for optimisation in this specific example. For example, the result symmetry is never used. It is merely a reasoning device for the correctness of the implementation. In other examples, the result symmetry may be relevant. Our goal is to demonstrate how the lion's share of the overhead can be removed by general principles. We leave a systematic account of performance engineering to future work, implementing these ideas and evaluating them empirically.

4.5 Indexed data types

Thanks to the generality of the initial algebra semantics approach, the dependently-typed development follows the same lines, but also uncovers nuances that degenerate in the simply-typed case. For example, the notion of a symmetry strength is phrased, not w.r.t. the Cartesian product, but w.r.t. the free action over a family. In the simply-typed case these notions coincide, but in the indexed case they differ. Once we explicate this structure, much of the development amounts to repeating the simply-typed development with the adjective 'indexed' peppered throughout. Appendix B repeats the development, and we describe the changes here. Fig. 1 summarises the difference between the simply typed theory and its dependently typed counterpart as follows. Where appropriate, we also describe the categorical concept.

Although the categorical theory does not capture all concepts, e.g. canonisers, it suffices to assign to almost concept its type. The theory includes indexed versions of the Modularity Proposition 4.7, the Equivariant Initiality Theorem 4.8 and Core Initiality Theorem 4.14.

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

Fig. 1. Summary of symmetric data type theory

We generalise from the category Set to $\mathrm{Fam}_{\underline{\Gamma}}$, the category of $\underline{\Gamma}$-indexed sets and vertical maps. The main technical device we use to generalise from Set to $\mathrm{Fam}_{\underline{\Gamma}}$ is the free Γ-indexed G-action adjunction between $\mathrm{Fam}_{\underline{\Gamma}}$ and $G\text{-Act}_{\Gamma}$—the category of Γ-indexed G-sets and vertical equivariant maps. Its monadicity means we can capture its structure categorically, and we use the existence of a right adjoint S to the resulting monad T over $\mathrm{Fam}_{\underline{\Gamma}}$.

For G-action Γ, we define, for each $\underline{\Gamma}$-family X, the *free* Γ*-indexed* G*-action over* X:

$$\underline{(g : G) \times g \circledast X_{\gamma}} \coloneqq (g : \underline{G}) \times X_{g^{-1} \circledast \gamma} \qquad g \circledast_{\gamma} \langle h, x : X_{h^{-1} \circledast \gamma} \rangle \coloneqq \langle g * h, x : X_{h^{-1} \circledast g^{-1} \circledast g \circledast \gamma} \rangle$$

We also define $(g : G) \to g^{-1} \circledast X$, the *cofree* Γ*-indexed* G*-action over* X. Its fibres are the dependent-function fibres $\underline{(g : G) \to g^{-1} \circledast X_{\gamma}} \coloneqq (g : \underline{G}) \to X_{g \circledast \gamma}$, defined by:

$$g \circledast_{\gamma} (\varphi : (k : \underline{G}) \to X_{k \circledast \gamma}) \coloneqq h \mapsto f(h * g) \in (h : \underline{G}) \to X_{h \circledast g \circledast \gamma}$$

They are the two adjoints to the forgetful functor from Γ-indexed actions to Γ-indexed families:

Theorem 4.16. *For a* G*-action* Γ *and family* $\underline{\Gamma} \vdash X$*, the map* $\Gamma \vdash x \mapsto \langle e, x \rangle : X \to \underline{(g : G) \times (g \circledast X)}$ *is the universal* Γ*-indexed* G*-action over* X*. The map* $\underline{\Gamma} \vdash \mathrm{eval}\, \langle -, e \rangle : \underline{(g : G) \to g^{-1} \circledast X} \to X$ *is the universal* Γ*-indexed* G*-action under* X*. The resulting two adjunction are (co)monadic.*

This theorem generalises the non-indexed case, where $\underline{G} \times X$ has the structure of the free G-action over X. The existence of the adjoints and their (co)monadicity follow from abstract reasons. Let $\oint F$ be the Grothendieck construction for a presheaf $F : \mathbb{C}^{\mathrm{op}} \to \mathrm{Set}$; $|C|$ be the discrete category underlying a category C, and $\hat{\mathbb{C}}$ the presheaf category of $\mathbb{C}$. Considering the G-action Γ as a presheaf over the category with one object and morphisms the G-elements. We then have the following two equivalences. First, Γ-indexed actions are equivalent to presheaves over $\oint \Gamma$ $G\text{-Act}_{\Gamma} \simeq \widehat{\oint \Gamma}$. Forgetting the functorial action then recovers the category of $\underline{\Gamma}$-indexed families, a presheaf category itself: $\mathrm{Fam}_{\underline{\Gamma}} \simeq \widehat{\oint \underline{\Gamma}} = \widehat{\left|\oint \Gamma\right|}$. The forgetful functor $G\text{-Act}_{\Gamma} \to \mathrm{Fam}_{\underline{\Gamma}}$ arises as precomposition by $J : \oint \Gamma \leftarrow \left|\oint \Gamma\right|$. Consequently, the two adjoints are given by (co)end formulae,

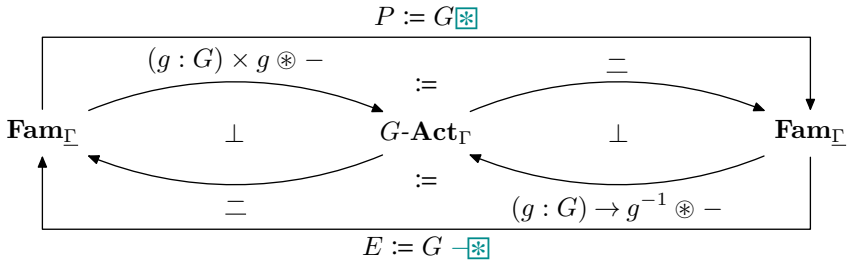


Fig. 2. Categorical situation for indexed G-action

and their monadicity follows by surjectivity of this functor [9, Prop. 1.1]. This situation is exploited by Fiore and Szamozvancev to account for mathematical and computational representations of abstract syntax with binding and substitution [10, 11, 28]. One can use the coend formula to calculate the explicit descriptions we gave the two adjoints. We show directly (cf. Appendix F) that these descriptions are the adjoints, and for completeness, we also show monadicity and comonadicity using Beck's theorem. We then have the situation in Fig. 2. Therefore, P is a monad, E is a comonad, and $P \dashv E$. Moreover, $P^2 := P \circ P \dashv E^2$.

5 Conclusion

5.1 Related work

W.l.o.g. in theorem provers. Given that w.l.o.g. is a common phrase in mathematical proofs, it comes as no surprise that several major theorem provers feature a similarly named concept. What is more compelling is that the treatment is far from uniform, pointing toward a missing uniform foundation, which this paper aims to provide.

For example, both the Rocq library SSReflect [15], and the Lean 4 Mathlib [29], define tactics named `wlog`, which cut in an additional statement ψ when proving φ. The former tactic[5] creates an additional subgoal $(\psi \Rightarrow \varphi) \Rightarrow \varphi$, while the latter[6] additionally assumes $\neg\psi$, adding a subgoal $(\psi \Rightarrow \varphi) \Rightarrow \neg\psi \Rightarrow \varphi$. In both tactics, the additional assumption ψ can be arbitrary and not tied to any particular symmetry, so it could be worth investigating how many of its uses correspond to the assumption of canonicity in our framework.

At the other end of the spectrum, Agda Stdlib defines[7] `wlog` : $(\forall(a\,b : A).a\,R\,b \Rightarrow Q\,a\,b) \Rightarrow (\forall a\,b.Q\,a\,b)$ for any total relation R and a symmetric predicate Q. Here, we have a swapping S_2-action on the domain $A \times A$, while Q provides an appropriate indexed action on the codomain. The canoniser is implicitly encoded by the totality of R, and the assumption exactly corresponds to our core function. A similar setting is used as an example in Dijkstra's [1997] critique of the w.l.o.g. arguments without an explicitly given underlying symmetry.

A richer collection of symmetries can be found in Harrison's [2009] seminal paper on w.l.o.g. tactics in HOL. In addition to the tactic discussed in §3.3, Harrison develops four additional tactics for w.l.o.g. reasoning in geometry, using a bank of results about invariance and equivariance of various operations under translations and orthonormal transformations. Three tactics are immediate special cases of our principle, while the fourth one is a reduction from a predicate on points of the form $(x, y, 0) \in \mathbb{R}^3$ to a predicate on $\mathbb{R}^2$.

[5]Rocq Reference Manual: https://rocq-prover.org/doc/V9.2.0/refman/proof-engine/ssreflect-proof-language.html#rocq:tacn.wlog
[6]Mathlib4 documentation: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Tactic/WLOG.html
[7]Agda Stdlib: https://github.com/agda/agda-stdlib/blob/c2abce1b8e72ccae49f73791d0996eaf23674357/src/Relation/Binary/Consequences.agda#L197

In that sense, our work is revisionist: we do not account for all previous principles named `wlog`. Instead, we offer a crisp criterion by which one can distinguish between w.l.o.g. arguments based on symmetry and other, more general reductions.

Symmetry in programming languages and type theory. A close topic in programming languages is nominal logic, an established approach to treating bound variables, in which α-equivalence is represented with the group of (finitely supported) permutations of atoms acting on terms built from such atoms [24, 25]. A possible application of our work would be to treat w.l.o.g. assumptions of atom freshness in proofs of correctness. Another could be symmetry strengths, as the changing of the group element when descending into a data type might capture the way the renaming permutation changes when one introduces fresh names while moving under a binder.

Another possible connection is the work of Atkey [1] on using parametricity and Noether's theorem to derive invariants of physical systems. A shared theme is identifying the group actions that describe a system's symmetries. However, applying Noether's theorem requires a notion of differentiation, which is not present in our general setting. One direction for future work is to investigate which of our action combinators can be translated to smooth-manifold settings, and whether they can serve as compositional operators for building physical systems.

A more recent development on symmetries in programming is a Hoare logic for symmetry properties by Mehta and Hsu [23]. It features triples that relate the symmetries of preconditions to those of postconditions. In particular, a triple of the form $(G, A)C(H, B)_\varphi$ uses a G-action A to capture symmetries of input states and an H-action B to capture symmetries of output states after executing the program C. The program C is required to be equivariant with respect to a group homomorphism $\varphi : G \to H$. More precisely: $\forall g \in G, a \in \underline{A}.C(g \circledast^A a) = \varphi(g) \circledast^B C(a)$. Beyond general action constructions such as (non-dependent) products and pre-/post-composition, the paper develops a forward-reasoning system that simultaneously derives an action on the output and proves the equivariance of C. As we move from mathematical foundations toward more practical symmetric programming, it will be interesting to adapt these rules to our setting to ease the construction of the group actions required by the w.l.o.g. construction.

Symmetry elsewhere in computer science. Computer science is vast, and group theory is pervasive, so there are many points of contact between the two. Rather than attempting a comprehensive review, we highlight a few representative areas and point to recent surveys for further reading.

In computer algebra and computational group theory, groups are both the object of study and a computational resource, shaping algorithm design and complexity analysis in areas such as permutation-group algorithms and constructive recognition [17, 26]. The concept of *canonical labelling* from nauty/Traces [22], which finds the vertex permutation mapping a graph into its representative element w.r.t. graph isomorphism, exactly matches our canonisers. It is worth investigating how many algorithms building on it fit into our w.l.o.g. pattern.

In constraint programming and satisfiability, symmetries have long been a central concern. A large body of work studies symmetry-breaking techniques, which add constraints that rule out symmetric solutions while preserving satisfiability [14, 18, 30]. More recently, in machine learning, symmetry appears both as an architectural prior (equivariance and invariance) and as an analysis tool. In computer vision in particular, it supports tasks such as recognition, reconstruction, and compression of geometric data [3, 7, 32].

5.2 Future work

As the title suggests, the aim of this paper was to lay common foundations for two future directions. The first is refining the developed modular mathematical abstractions into practical programming constructs. We plan on implementing more involved examples (e.g. a full set of operations on

self-balancing binary search trees) to guide us both in terms of efficiency, as well as on how to most ergonomically present groups, actions, canonisers, core functions, and equivariance to the programmer. For example, immediate canonisers (see Proposition 2.20) could admit an interface similar to *lazy constructors* [19], where we extend a data type definition with non-canonical constructors together with their non-trivial canonising symmetries, but list only canonical cases in pattern matching, since the non-canonical ones are covered by the w.l.o.g. construction.

The second direction is to mechanise our development in a proof assistant. We already explored the landscape by building two prototypes in Idris and Lean. Having a clean mathematical resolution of issues that surfaced, such as termination and efficiency, we can now return to a proof assistant: not to increase confidence in the correctness of the results, but to develop a library of reusable results and lay a foundation for future mechanisation work.

Acknowledgments

Copyright addenadum. For the purpose of open access, the authors have applied a Creative Commons attribution (CC BY) licence to any Author Accepted Manuscript version arising. The second author used Anthropic's Claude models to obtain feedback on the prose and to proofread drafts for errors. Supported by Royal Society University Research Fellowship, Advanced Research and Invention Agency (ARIA) Safeguarded AI grant Qbs4Safety, and an Alan Turing Institute Seed Funding grant. We are grateful to the following individuals for useful comments, suggestions, and pointers: Guillaume X. Allais, Rob Cornish, Pierre-Evariste Dagand, Marcelo P. Fiore, Nicola Gambino, Jeremy Gibbons, Justin Hsu, Martin E. Hyland, András Kovács, Neel Krishnaswami, Sam Lindley, Justus Matthiesen, James McKinna, Georg Moser, Sean K. Moss, Bruno Oliveira, Andrew M. Pitts, Tom Schrijvers, Sam Staton, Armin Walch, Nick Wu, and Jeremy D. Yallop.

Supplementary material

We proceed as follows:

- Appendix A evaluates a proof-of-concept implementation of dependently-typed symmetric programming. The main goal is to determine whether the performance engineering techniques of §4.4 can eliminate the asymptotically significant overhead that symmetric programming incurs.
- Appendix B presents the theory of symmetric indexed data types, as it follows almost identical structure to its simply-typed counterpart.
- Appendix C contains further details about the simply-typed w.l.o.g. construction (§2).
- Appendix D contains further details about the dependently-typed w.l.o.g. construction (§3).
- Appendix E contains further details and proofs about symmetric simply typed data types (§4.1–§4.4). This material is a special case of its dependently-typed counterparts. Much of it can also be done more abstractly and categorically. Here we prove much of it more concretely and with fewer abstractions to help make the material more accessible.
- Appendix F contains further details and proofs about symmetric dependently typed data types (§4.5 and Appendix B). Here we give concrete descriptions of additional auxiliary results, and omitted details from concrete examples.
- Appendix G contains a more abstract category theoretic development that unifies parts of both theories of data types (cf. summary figure in §4.5).

A Proof-of-Concept: removing performance overheads for symmetric data types

In §4.4 we described techniques for removing some of the performance overhead incurred by symmetric programming when projecting a subtree. There, we used the free action construction and the symmetric strength. A skeptical reader may doubt whether these techniques indeed remove the performance bottleneck. While this is the topic of future work, we include a preliminary experiment demonstrating that these techniques indeed remove the bottlenecks, and are feasible even for the dependently-typed setting.

We use Idris 2 [6], a dependently-typed programming language. We implemented a dependently-typed version of the subtree projection function from Example 4.15. Idris 2 implements a version of McBride and Atkey's *quantitative type theory* [2]. It allows us to ensure some values must be erased at runtime. We erase indices in indexed actions. Some functions, such as subtree projection, in fact require this index, and so we use a singleton type. We implemented three functions of the type:

$$t : \mathsf{Tree} \vdash \mathsf{subtree} : \mathsf{Path}\, t \circledast\!\!\longrightarrow (\{t\} \to \mathsf{Tree})$$

The function requires little computation, and so most of the computational complexity concerns the overhead of deconstructing the path and the tree, and projecting out the result. The baseline implementation includes both L :: p and R :: p cases explicitly. The naive implementation uses symmetric programming to remove the symmetric case R :: p, but in turn flips the input tree given via the singleton, and the result. The optimised implementation employs dependently-typed counterparts to the performance engineering techniques from §4.4. The benchmarks construct a tree of depth d for $d = 11$ (including the naive implementation) and $d = 10,000$ (excluding the naive implementation), and then run each implementation for all paths of length $0 \leq |p| < d$. We use alternating paths of the form R :: L :: $\cdots$ :: Nil to make the symmetric programming tests flip both inputs and outputs in every recursive call. For simplicity, we only used the generic action for the Path indexed symmetric data type, and did not use a generic action for the simply-typed Tree. The optimised implementation did use the S_2-strength for the functor $F_{\mathsf{Tree}} : \mathsf{Set} \to \mathsf{Set}$ when deconstructing the tree alongside the path. The test harness does not isolate the measurements from the interference of garbage collection.

Fig. 3 presents sample runtimes from an HP EliteBook 640 14 inch G11 Notebook PC with Intel Core Ultra 5 135U×14, 32.0GiB memory, running Ubuntu 24.04.3 LTS. Fig. 3(a) plots the runtimes of the three benchmarks on a tree of depth $d = 11$. While the naive implementation can cope with trees of depth larger than $d = 11$, the difference in performance to the baseline and optimised implementation is so large that they are indiscernible in the resulting diagram, and having the paths run up to length 10 is a nice round number. As we can see, the naive test indeed incurs several orders of magnitude of overhead, and the optimised version removes much of this overhead. The shape of the naive test runtime curve may seem surprising, and we suggest partial explanations below.

Fig. 3(b) plots the runtimes of baseline and optimised tests for a larger tree of depth $d = 10,000$. The test that uses symmetric programming still incurs overhead. Fig. 3(c) plots the same data against a log scale to gauge whether this overhead changes the asymptotic runtime. The two graphs seem to have similar shapes, suggesting the techniques of §4.4 remove the asymptotic overhead.

We explain the shape of the naive test's runtime as follows. The path of length 0, Nil incurs no tree flipping, returning the input tree immediately. Accordingly, the test has similar runtime to the other tests. Every recursive call with tree t, flips the tree t and the result s. The longer the path, the smaller the result s. For the path of length 1, we flip a tree of depth 11 on the input and a tree of depth 10 on the output. For the path of length 2, we flip a tree of depth 11 and a tree of depth 10 on the input, but flip a tree of depth 9 twice on the output. And so on. This account does not incorporate the overhead of memory allocation. The initial tree representation shares the same subtree for both children at every node. Once the naive implementation starts flipping the tree, this sharing is lost, incurring a large memory footprint, and potentially spends some time on garbage collection. This may have averaging effects on the curve. Further experimentation and analysis are needed to fully explain the curve.

We conclude with some qualitative analysis of the experiment. While the symmetric code never considers redundant symmetric cases, it requires substantial infrastructure. Much of the symmetric programming code is, however, shared between the two benchmarks: the structure functors, their strength, their canonisers, and relevant McBride and McKinna views [21] for convenient pattern-matching on the canonical elements. This suggests that additional examples could amortise the cost of the development. Further experimentation should also identify the right programming abstractions to alleviate layers of nested pattern matching and Idris 2 specific plumbing.

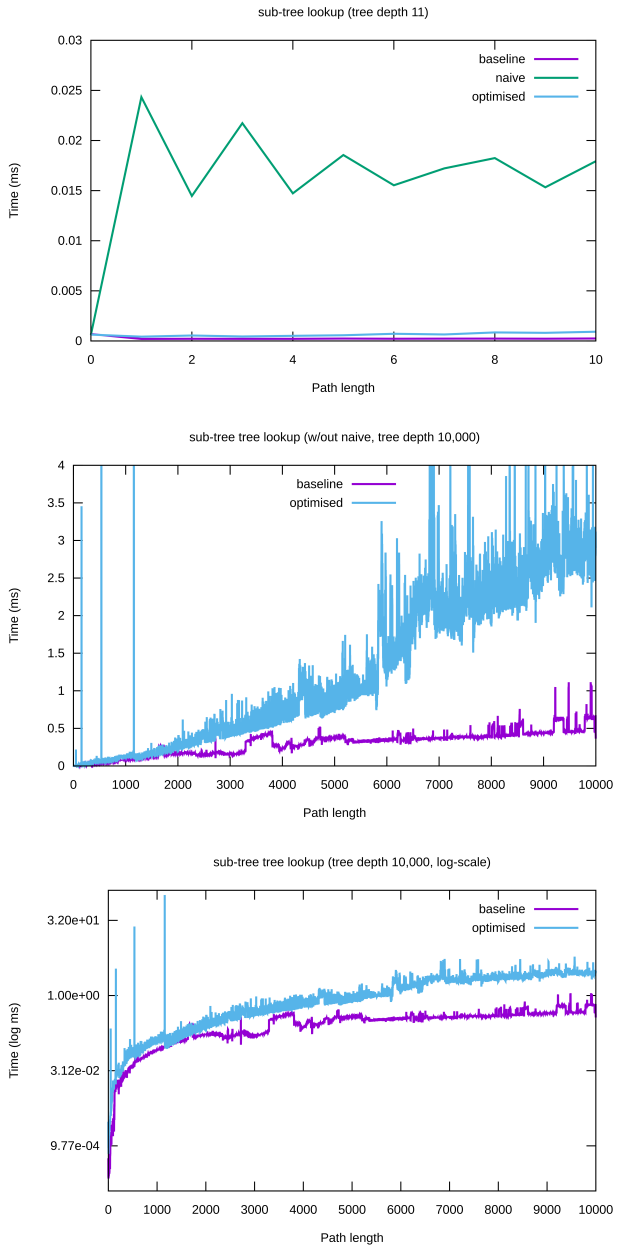


Fig. 3. Proof-of-concept: Idris 2 dependently-typed subtree projection implementations. The baseline implementation includes redundant symmetric cases, the naive implementation flips the tree repeatedly, and the optimised implementation uses the free action and strength to remove the performance bottlenecks. (a) runtimes on a tree of depth 11, projecting subtrees of depth 0–10; (b) runtimes of baseline and optimised benchmarks on a tree of depth 10000; and (c) log-runtime of the benchmark from (b).

B Indexed algebraic data types

In §4.5 we summarised the changes to the simply-typed development of symmetric data types. In this appendix, we spell the theory out in full detail, for completeness.

To extend the theory to indexed data types $\underline{\Gamma} \vdash \mathsf{D}$, we use endofunctors $F_{\mathsf{D}} : \mathrm{Fam}_{\underline{\Gamma}} \to \mathrm{Fam}_{\Gamma}$ over the category $\mathrm{Fam}_{\underline{\Gamma}}$ of families indexed by a set $\underline{\Gamma}$ and vertical maps between them. The data type D denotes an initial F_{D}-algebra $\underline{\Gamma} \vdash \mathrm{roll} : F_{\mathsf{D}}\mathsf{D} \to \mathsf{D}$. As in the non-indexed case, we equip those endofunctors with a symmetry strength relative to an ambient G-action $\Gamma = \langle \underline{\Gamma}, (\circledast) \rangle$ on the set of indices. This strength will be a certain natural transformation $\mathrm{st} : G \boxed{\circledast} F_{\mathsf{D}}X \to F_{\mathsf{D}}(G \boxed{\circledast} X)$, where $(G \boxed{\circledast} -)$ is the functor part of the *free* Γ*-indexed* G*-action* monad. Initial algebras for such endofunctors enjoy an indexed version of the parameterised initiality and representation theorems, and stating and establishing those is our main goal.

We'll use the indexed family $t : \mathsf{Tree} \vdash \mathsf{InPath}\, t$ as our running example, but unlike Example 3.2, we avoid the redundant cases through symmetric programming. As our endofunctor, we take:

$$(F_{\mathsf{InPath}}X)_{\mathsf{Leaf}} \coloneqq \emptyset \qquad (F_{\mathsf{InPath}}X)_{\mathsf{Node}\ \ell\, n\, r} \coloneqq \mathsf{Nil} \mid \mathsf{L} :: X_\ell \mid \mathsf{R} :: X_r$$
$$(F\,f)_u\mathsf{Nil} \coloneqq \mathsf{Nil} \qquad (F\,f)_u(\mathsf{L} :: x) \coloneqq \mathsf{L} :: f_\ell x \qquad (F\,f)_u(\mathsf{R} :: y) \coloneqq \mathsf{R} :: f_r y$$
$$(\text{where: } t : \mathsf{Tree} \vdash f : X \to Y,\ u = \mathsf{Node}\ \ell\, n\, r)$$

The initial algebra has as carrier F_{InPath} is $t : \mathsf{Tree} \vdash \mathsf{InPath}\, t$ from Example 3.1.

We denote the multiplication of the free Γ-indexed G-action monad P by:

$$(\boxtimes) : G \boxed{\circledast} (G \boxed{\circledast} X) \to G \boxed{\circledast} X \qquad \left\langle g, \left\langle h, x : X_{h^{-1} \circledast g^{-1} \circledast \gamma} \right\rangle \right\rangle \coloneqq \left\langle g * h, x : X_{(g*h)^{-1} \circledast \gamma} \right\rangle$$

It is the functor $(G \boxed{\circledast})$ that generalises the notion of strength to the indexed setting:

Definition B.1. Let Γ be a G-action and $F : \mathrm{Fam}_{\underline{\Gamma}} \to \mathrm{Fam}_{\underline{\Gamma}}$ be an endofunctor. An Γ*-indexed* G*-symmetry strength* is a natural transformation $\mathrm{st} : G \boxed{\circledast} FX \to F(G \boxed{\circledast} X)$, satisfying:

$$\mathrm{st}_\gamma(e, u) = F(x \mapsto \langle e, x \rangle)u \qquad F(\boxtimes) \circ \mathrm{st}(g, \mathrm{st}(h, x)) = \mathrm{st}(g * h, x)$$

Example B.2. We define an S_2-symmetry strength for $t : \mathsf{Tree} \vdash \mathsf{InPath}\, t$. Only the F case is interesting, as the first strength axioms forces the definition for $\mathtt{I}$:

$$\mathrm{st}^{\mathsf{InPath}}\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)\left\langle \mathsf{F}, \mathsf{Nil} : \mathsf{InPath}\ \begin{smallmatrix} & & n & & \\ & \mathsf{F} \circledast r & & & \mathsf{F} \circledast \ell \end{smallmatrix} \right\rangle \coloneqq \mathsf{Nil} : \mathsf{InPath}\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)$$

$$\mathrm{st}^{\mathsf{InPath}}\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)\langle \mathsf{F}, \mathsf{L} :: (x : X_{\mathsf{F} \circledast r}) \rangle \coloneqq \mathsf{R} :: \langle \mathsf{F}, x : X_{\mathsf{F} \circledast r} \rangle \in (F_{\mathsf{InPath}}(G \boxed{\circledast} X))\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)$$

$$\mathrm{st}^{\mathsf{InPath}}\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)\langle \mathsf{F}, \mathsf{R} :: (x : X_{\mathsf{F} \circledast \ell}) \rangle \coloneqq \mathsf{L} :: \langle \mathsf{F}, x : X_{\mathsf{F} \circledast \ell} \rangle \in (F_{\mathsf{InPath}}(G \boxed{\circledast} X))\left(\begin{smallmatrix} & n & \\ \ell & & r \end{smallmatrix}\right)$$

To prove the second strength axiom, only the case $g \coloneqq h \coloneqq \mathsf{F}$ is interesting though straightforward:

$$\begin{aligned} F(\boxtimes) \circ \mathrm{st}(\mathsf{F}, \mathrm{st}(\mathsf{F}, \mathsf{Nil})) &= F(\boxtimes) \circ \mathrm{st}(\mathsf{F}, \mathsf{Nil}) = F(\boxtimes)\mathsf{Nil} = \mathsf{Nil} \\ &= \mathrm{st}(\mathsf{F} * \mathsf{F}, \mathsf{Nil}) \\ F(\boxtimes) \circ \mathrm{st}(\mathsf{F}, \mathrm{st}(\mathsf{F}, \mathsf{L} :: x)) &= F(\boxtimes) \circ \mathrm{st}(\mathsf{F}, \mathsf{R} :: \langle \mathsf{F}, x \rangle) \\ &= F(\boxtimes)(\mathsf{L} :: \langle \mathsf{F} * \mathsf{F}, x \rangle) = \mathsf{L} :: \langle \mathtt{I}, x \rangle = \mathrm{st}(\mathsf{F} * \mathsf{F}, \mathsf{L} :: x) \end{aligned}$$

and symmetrically for $\mathsf{R} :: x$. We cannot avoid this symmetric argument using wlog as we are setting up the symmetric action by defining the strength. Later uses of wlog will exploit this work. □

The appropriate category-theoretic concept that abstracts this notion of strength is a *monad-functor distributive law*, and we will use this perspective in our category theoretic development in Appendix G. As may be clear, when $\mathbb{1} \vdash X$ is a single-fibred family, $\mathbb{1} \vdash G \circledast X$ is the single-fibred family $\underline{G} \times X$, subsuming the simply-typed case.

The following lemma justifies checking only the interesting cases, as we did in Example B.2:

Lemma B.3. *Let* st *be the structure of an indexed G-symmetric strength, satisfying the first axiom. Then the second axiom holds when either $g = e$ or $h = e$.*

Our main initiality theorem uses an indexed notion of equivariant algebras (cf. Definition 4.6):

Definition B.4. Let Γ be a G-action, $F : \mathrm{Fam}_{\underline{\Gamma}} \to \mathrm{Fam}_{\underline{\Gamma}}$ an endofunctor equipped with a symmetry strength. For every indexed action A, the *lifted action* is given by:

$$\underline{F_{\mathrm{st}} A} \coloneqq F\underline{A} \qquad g \circledast_\gamma u \coloneqq F(\circledast)\, \mathrm{st}_{g \circledast \gamma} \langle g, t \rangle$$

An *equivariant F-algebra* $\langle A, f \rangle$ for an endofunctor equipped with a symmetry strength, consists of an indexed action A equipped with an equivariant algebra structure $\Gamma \vdash f : F_{\mathrm{st}} A \circledast\!\!\longrightarrow A$. An equivariant F-algebra homomorphism is then a family map between the carrier families of two such algebras that is simultaneously equivariant and F-algebra homomorphic.

We generalise the Equivariant Initiality Theorem 4.8 to indexed families:

Theorem B.5 (indexed equivariant initiality)**.** *Let* roll $: F\boldsymbol{\mu}F \longrightarrow \boldsymbol{\mu}F$ *be an initial algebra for a functor F over Γ-indexed families with a G-symmetry strength* st $: G \circledast FX \longrightarrow F(G \circledast X)$*. The map* $(\circledast) : G \circledast \boldsymbol{\mu}F \longrightarrow \boldsymbol{\mu}F$ *is the unique such map satisfying* $g \circledast (\mathrm{roll}\, u) = \mathrm{roll}(F(\circledast)(\mathrm{st}(g, u)))$*. It makes* $\langle \boldsymbol{\mu} f, \mathrm{roll} \rangle$ *into an initial equivariant F-algebra: for all equivariant F-algebra $\langle A, f : F_{\mathrm{st}} A \circledast\!\!\longrightarrow A \rangle$, the F-homomorphism into $\underline{A}$ is equivariant:* fold $f : \boldsymbol{\mu}F \circledast\!\!\longrightarrow A$*.*

Like its simply-typed variant, the proof of this theorem uses an appropriate notion of parameterised initiality originally proposed by Bird and Paterson [5].

Example B.6. The action induced by the Indexed Equivariant Initiality Theorem B.5 satisfies:

$$\begin{aligned} \mathsf{F} \circledast (\mathsf{L} :: \mathsf{R} :: \mathsf{Nil}) &= \mathrm{roll} \circ F(\circledast)\, \mathrm{st}(\mathsf{F}, \mathsf{L} :: (\mathsf{R} :: \mathsf{Nil})) = \mathrm{roll} \circ F(\circledast)(\mathsf{R} :: \langle \mathsf{F}, \mathsf{R} :: \mathsf{Nil} \rangle) \\ &= \mathsf{R} :: (\mathsf{F} \circledast (\mathsf{R} :: \mathsf{Nil})) = \mathsf{R} :: \mathsf{L} :: (\mathsf{F} \circledast \mathsf{Nil}) = \mathsf{R} :: \mathsf{L} :: \mathsf{Nil} \end{aligned}$$

as we may expect from the example at the beginning of §3.2. □

Example B.7. Let's implement the dependently-typed node lookup function and simultaneously show its equivariance lookup $: (t : \mathsf{Tree}) \times \mathsf{InPath}\, t \circledast\!\!\longrightarrow \mathbb{N}$. We'll construct an indexed equivariant map $t : \mathsf{Tree} \vdash \mathrm{lookup}_t : \mathsf{InPath}\, t \circledast\!\!\longrightarrow \mathbb{N}$. By the Indexed Equivariant Initiality Theorem B.5, it suffices to equip $\mathbb{N}$ with an equivariant F_{InPath}-algebra structure $\mathrm{lookup}^{\mathrm{Elim}} : F_{\mathsf{InPath}} \mathbb{N} \circledast\!\!\longrightarrow \mathbb{N}$. Informally, w.l.o.g., the path to the node is either empty or starts with the Left subtree, eliminating one symmetric case. Formally, define a canoniser by $c\mathsf{Nil} \coloneqq c(\mathsf{L} :: n) \coloneqq \mathsf{I}$ and $c(\mathsf{R} :: n) \coloneqq \mathsf{F}$. Its canonical elements are Nil and $\mathsf{L} :: n$. This canoniser is equivariant, but not strictly:

$$c(\mathsf{F} \circledast \mathsf{Nil}) = c\mathsf{Nil} = \mathsf{I} \overset{\mathsf{Nil}}{\sim} \mathsf{F} = \mathsf{I} * \mathsf{F}^{-1} = \mathsf{F} \circledast c\,\mathsf{Nil}$$

$$c(\mathsf{F} \circledast (\mathsf{L} :: n)) = c(\mathsf{R} :: n) = \mathsf{F} = \mathsf{I} * \mathsf{F}^{-1} = \mathsf{F} \circledast c(\mathsf{L} :: n)$$

and symmetrically for $\mathsf{R} :: n$. Again, we cannot avoid this symmetry with wlog since we are setting up the symmetric argument.

Let's define the core function for the eliminator $t : \mathsf{Tree} \vdash \mathrm{lookup}^{\mathrm{Core}} : \mathrm{Can}\, c \to \mathbb{N}$:

$$\mathrm{lookup}^{\mathrm{Core}}_{(\mathsf{Node}\ \ell\, n\, r)} \mathsf{Nil} \coloneqq n \qquad \mathrm{lookup}^{\mathrm{Core}}_{(\mathsf{Node}\ \ell\, n\, r)} (\mathsf{L} :: p) \coloneqq p$$

and we do not need to consider the symmetric case. The stabiliser subgroup for $\mathbb{N}$ is the whole of S_2 since its action is trivial. Consequently, every core function is stable, and we don't need to work out the stabilisers for canonical elements. For completeness, here they are. The stabiliser subgroup for Nil is the whole of S_2, and the stabiliser subgroup for $\mathsf{L} :: p$ is the trivial subgroup $\{\mathtt{I}\}$.

We use the Indexed Representation Theorem 3.12 to show the extended function is equivariant: $\mathsf{lookup}^{\mathsf{Elim}} := \mathsf{wlog}(c, f) : F_{\mathsf{InPath}}\mathbb{N} \circledast\!\!\longrightarrow \mathbb{N}$. □

As in the simply-typed case (cf. §4.3), we reuse the same canoniser for multiple traversals of the same symmetric indexed data type, and streamline the steps in Example B.7.

Definition B.8. Let Γ be a G-action, and F be a Γ-indexed G-strong. An *indexed canoniser* for F is a natural transformation $\underline{\Gamma} \vdash c : F \longrightarrow \underline{G}$. Each such canoniser induces the *canonical elements* functor $\mathsf{Can}\, c : G\text{-}\mathrm{Act}_\Gamma \longrightarrow \mathrm{Fam}_{\underline{\Gamma}}$ given by:

$$((\mathsf{Can}\, c)A)_\gamma := (\mathsf{Can}_{FA}\, c_{\underline{A}})_\gamma := \left\{ g \circledast^{FA}_{\delta} u \;\middle|\; \begin{array}{l} g = c_{\underline{A};\delta} u, g \circledast \delta = \gamma, \\ u \in (\underline{FA})_\delta \end{array} \right\} \qquad (\mathsf{Can}\, c)h : u \mapsto (\underline{Fh})u$$

An indexed canoniser is uniquely determined by its component at the terminal family $\underline{\Gamma} \vdash \mathbb{1}$.

Example B.9. The indexed S_2-strong functor F_{InPath} has the canoniser (cf.Example 4.11)

$$\mathsf{goLeft}_{\mathsf{Node}\, n\, \ell\, r}\, \mathsf{Nil} := \mathtt{I} =: \mathsf{goLeft}_{\mathsf{Node}\, \ell\, n\, r}(\mathsf{L} :: p) \quad \mathsf{goLeft}_{\mathsf{Node}\, \ell\, n\, r}(\mathsf{R} :: p) := \mathtt{F}$$

The canonical elements are:

$$(\mathsf{Can}_A\, \mathsf{goLeft})_{\mathsf{Leaf}} := \emptyset \qquad (\mathsf{Can}_A\, \mathsf{goLeft})_{\mathsf{Node}\, \ell\, n\, r} := \left\{ \mathsf{Nil}, \mathsf{L} :: a \;\middle|\; a \in \underline{A}_{\mathsf{Node}\, \ell\, n\, r} \right\}$$ □

Continuing in tandem with the development of §4.5, we define the data need to traverse an indexed symmetric data type:

Definition B.10. Let F be a Γ-indexed G-strong functor and $c : \underline{F} \longrightarrow \underline{G}$ a canoniser for it. An *indexed c-core F-algebra* $\langle A, f\rangle$ consists of a Γ-indexed G-action A equipped with a vertical map $\underline{\Gamma} \vdash f : (\mathsf{Can}\, c)A \longrightarrow \underline{A}$. It is *equivariant* when f is equivariant $\Gamma \vdash f : ((\mathsf{Can}\, c)A \subseteq FA) \circledast\!\!\longrightarrow A$. A *homomorphism* of equivariant indexed c-core F-algebras $h : \langle A, f\rangle \longrightarrow \langle B, g\rangle$ is an equivariant map $\Gamma \vdash h : A \circledast\!\!\longrightarrow B$ satisfying: $h(f\, u) = g\,((\mathsf{Can}\, c)h\, u)$.

Example B.11. We have the following equivariant goLeft-core F_{InPath}-algebra:

$$\mathsf{lookup}^{\mathsf{Core}} : (\mathsf{Can}\, \mathsf{goLeft})(\mathsf{Triv}\mathbb{N}) \longrightarrow \underline{\mathsf{Triv}\mathbb{N}} = \mathbb{N}$$

$$\mathsf{lookup}^{\mathsf{Core}}_{\mathsf{Node}\, \ell\, n\, r}\, \mathsf{Nil} := n \qquad \mathsf{lookup}^{\mathsf{Core}}_{\mathsf{Node}\, \ell\, n\, r}\, (\mathsf{L} :: (k \in \mathbb{N})) := k$$ □

Next, chain together the traversal and w.l.o.g. constructions:

$$\underline{\Gamma} \vdash \left(\mathsf{Can}_{FA}\, c_{\underline{A}} = (\mathsf{Can}\, c)A \xrightarrow{f} \underline{A} \right) \mapsto \left(F\underline{A} \xrightarrow{\mathsf{wlog}(c_{\underline{A}}, f)} \underline{A} \right) \mapsto \left(\mathsf{flog}_A(c, f) : \boldsymbol{\mu}\underline{F} \xrightarrow{\mathsf{fold}\, \mathsf{wlog}(c_{\underline{A}}, f)} \underline{A} \right)$$

And similarly chain Theorem 3.12 and the Indexed Equivariant Initiality Theorem B.5:

Theorem B.12 (indexed core initiality)**.** *Let F be a Γ-indexed G-strong functor, $\underline{\Gamma} \vdash \mathsf{roll} : \underline{F}\boldsymbol{\mu}\underline{F} \longrightarrow \boldsymbol{\mu}\underline{F}$ an initial algebra of $\underline{F}$, and c a canoniser for F. Then restricting* roll *to canonical elements makes $\boldsymbol{\mu}F$ into the initial indexed c-core F-algebra* $\mathsf{roll} : (\mathsf{Can}\, c)\boldsymbol{\mu}F \circledast\!\!\longrightarrow \boldsymbol{\mu}F$*: for any equivariant indexed c-core F-algebra $f : (\mathsf{Can}\, c)A \circledast\!\!\longrightarrow A$, the unique equivariant indexed c-core F-algebra homomorphism into f is:* $\mathsf{flog}(c, f) : \boldsymbol{\mu}F \circledast\!\!\longrightarrow A$.

This theorem implies, by Example B.11, that we have an equivariant function:

$$t : \mathsf{Tree} \vdash \mathsf{lookup} := \mathsf{flog}(\mathsf{goLeft}, \mathsf{lookup}^{\mathsf{Core}}) : \mathsf{InPath}\, t \circledast\!\!\longrightarrow \mathsf{Triv}\mathbb{N}$$

C Simply-typed w.l.o.g.: further details

In this Appendix, we provide further details to §2. Our first proof obligation is a comment we immediately before Example 2.4:

Lemma C.1 (well-known). *Let A and B be G-actions. The exponential $A \to B$ in G-Act is given by the set of functions between their carriers:*

$$\underline{A \to B} \coloneqq \underline{A} \to \underline{B} \qquad (g \circledast^{A\to B} f) \coloneqq a \mapsto g \circledast^{B} f(g^{-1} \circledast^{A} a)$$
$$\mathrm{eval} : (A \to B) \times A \circledast\!\!\to B \qquad \mathrm{eval}\,\langle f, a\rangle \coloneqq f\,a$$

Proof
We indeed have an action. Unit preservation:

$$(\overset{G}{e} \circledast^{A\to B} f)a = \overset{G}{e} \circledast^{B} f(\overset{G}{e}^{-1} \circledast a) = f\,a \implies \overset{G}{e} \circledast^{A\to B} f = f$$

Action preservation:

$$\begin{aligned}(g \circledast^{A\to B} h \circledast^{A\to B} f)a &= g \circledast^{B} (h \circledast^{A\to B} f)(g^{-1} \circledast^{A} a) = g \circledast^{B} (h \circledast^{B} f(h^{-1} \circledast^{A} (g^{-1} \circledast^{A} a)))\\ &= (g \overset{G}{*} h) \circledast^{B} f(\left(g \overset{G}{*} h\right)^{-1} \circledast^{A} a) = \left((g \overset{G}{*} h) \circledast^{A\to B} f\right) a\\ &\implies (g \circledast^{A\to B} h \circledast^{A\to B} f) = (g \overset{G}{*} h) \circledast^{A\to B} f\end{aligned}$$

Evaluation is equivariant. Calculate:

$$\begin{aligned}\mathrm{eval}(g \circledast^{(A\to B)\times A} \langle f, a\rangle) &= \mathrm{eval}(\left\langle g \circledast^{A\to B} f, g \circledast^{A} a\right\rangle) = (g \circledast^{A\to B} f)\,(g \circledast^{A} a)\\ &= g \circledast^{B} (f(g^{-1} \circledast^{A} (g \circledast^{A} a))) = g \circledast^{B} (f\,a) = g \circledast^{B} \mathrm{eval}(f, a)\end{aligned}$$

Given any equivariant map $h : C \times A \circledast\!\!\to B$, its curried map is also equivariant:

$$\begin{aligned}(g \circledast^{A\Rightarrow B} \mathrm{curry}\, hc)a &= g \circledast^{B} \mathrm{curry}\, h(c; g^{-1} \circledast^{A} a) = g \circledast^{B} h(c, g^{-1} \circledast^{A} a) \overset{\text{equivariance}}{\downarrow} = h(g \circledast^{C} c, g \circledast g^{-1} \circledast^{A} a)\\ &= h(ng \circledast^{C} c, a) = (\mathrm{curry}\, h(g \circledast^{C} c)a\\ &\implies \quad g \circledast^{A\to B} \mathrm{curry}\, hc = \mathrm{curry}\, h(g \circledast^{C} c)\end{aligned}$$

and so $\mathrm{curry}\, h : C \circledast\!\!\to (A \to B)$. The curried function lifts h along evaluation. Now if we have an equivariant map $g : C \circledast\!\!\to (A \to B)$ that lifts h along evaluation, then by the set-theoretic argument, it coincides with $\mathrm{curry}\, h$. ∎

Next, we characterise the canonical elements of canonisers of the form $\underline{A} \times \underline{B} \xrightarrow{\pi_1} A \xrightarrow{c} \underline{G}$:

Proof of Lemma 2.8
Take an arbitrary canonical element $(c \circ \pi_1)\,\langle a, b\rangle \circledast \langle a, b\rangle \in \mathrm{Can}\,(c \circ \pi_1)$. We then have

$$(c \circ \pi_1)\,\langle a, b\rangle \circledast \langle a, b\rangle = ca \circledast \langle a, b\rangle = \langle ca \circledast a, ca \circledast b\rangle \in \mathrm{Can}\, c \times \underline{B}$$

Conversely, given an arbitrary pair $\langle ca \circledast a, b\rangle \in \mathrm{Can}\, c \times \underline{B}$, let $b' = (ca)^{-1} \circledast b$. We then have:

$$\langle ca \circledast a, b\rangle = \left\langle ca \circledast a, (ca * (ca)^{-1}) \circledast b\right\rangle = ca \circledast \langle a, b'\rangle = ((c \circ \pi_1)\,\langle a, b'\rangle) \circledast \langle a, b'\rangle \in \mathrm{Can}\,(c \circ \pi_1)$$

Therefore $\mathrm{Can}\,(c \circ \pi_1) = \mathrm{Can}\, c \times \underline{B}$. ∎

Next, we proof the main equivariance theorem for the w.l.o.g. construction:

Proof of Theorem 2.13

Let us first show that $\mathsf{wlog}(c, f)$ is equivariant. Take arbitrary $g \in \underline{G}$ and $a \in \underline{A}$. By definition and by the group and action axioms, we can rewrite:

$$\begin{aligned}\mathsf{wlog}(c, f)(g \circledast a) &= c(g \circledast a)^{-1} \circledast f(c(g \circledast a) \circledast (g \circledast a)) \\ &= c(g \circledast a)^{-1} \circledast f(c(g \circledast a) \circledast (g \circledast (ca)^{-1} \circledast ca \circledast a)) \\ &= c(g \circledast a)^{-1} \circledast f\big((c(g \circledast a) * g * (ca)^{-1}) \circledast (ca \circledast a)\big)\end{aligned}$$

Since both $ca \circledast a$ and $c(g \circledast a) \circledast (g \circledast a)$ belong to $\mathsf{Can}\, c$, equivariance of f allows us to rewrite:

$$\begin{aligned}&= c(g \circledast a)^{-1} \circledast \big((c(g \circledast a) * g * (ca)^{-1}\big) \circledast f(ca \circledast a)) \\ &= g \circledast ((ca)^{-1} \circledast f(ca \circledast a)) \\ &= g \circledast \mathsf{wlog}(c, f)\, a\end{aligned}$$

as desired.

Next, let us show that $\mathsf{wlog}(c, f)$ extends f. Take an arbitrary canonical element $a \in \mathsf{Can}\, c$. Since $ca \circledast a$ also lies in $\in \mathsf{Can}\, c$, action axioms imply:

$$\mathsf{wlog}(c, f)\, a = (ca)^{-1} \circledast f(ca \circledast a) \overset{f \text{ is equivariant}}{=} (ca)^{-1} \circledast ca \circledast fa = fa$$

Finally, take an arbitrary $\varphi : A \circledast\!\!\longrightarrow B$ that extends f along $\mathsf{Can}\, c \subseteq \underline{A}$. Then, for an arbitrary $a \in \underline{A}$, we have

$$\varphi a = \varphi((ca)^{-1} \circledast ca \circledast a) \overset{\varphi \text{ equivariant}}{=} (ca)^{-1} \circledast \varphi(ca \circledast a) \overset{\varphi \text{ extends } f}{=} (ca)^{-1} \circledast f(ca \circledast a) \overset{\text{by definition}}{=} \mathsf{wlog}(c, f)\, a$$

as desired. ∎

The order of the next few results concerning characterisation of strictly equivariant and immediate canonisers from §2.4 differs from the order in which they appear there. Strict equivariance seem to be a more natural concept to the audience when we present this work, and so the main body leads with it. Here we first characterise immediate canonisers, and use this characterisation as part of the proof. This order make it easier to see that we avoid circular reasoning.

We start with immediate canonisers as every strictly equivariant canoniser is also immediate:

Proof of Proposition 2.21

Calculate:

$$c(ca * a) \overset{\text{equivariance}}{=} ca * (ca)^{-1} = e$$

as desired. ∎

We characterise immediate canonisers via the condition $\mathsf{Can}\, c = c^{-1}[e]$:

Proof of Proposition 2.20

Take an arbitrary $ca \circledast a \in \mathsf{Can}\, c$. Since c is immediate, we have $cca \circledast a = e$, and so $ca \circledast a \in c^{-1}[e]$, thus $\mathsf{Can}\, c \subseteq c^{-1}[e]$. The converse inclusion always holds. If $a \in c^{-1}[e]$, then $ca = e$ and so $a = ca \circledast a \in \mathsf{Can}\, c$.

Conversely, let us show that c is immediate and take an arbitrary $a \in \underline{A}$. We have $ca \circledast a \in \mathsf{Can}\, c$ by definition, thus $ca \circledast a \in c^{-1}[e]$ by assumption, and so $c(ca \circledast a) = e$ as required. ∎

The next result characterises strict equivariance of canoniser via the equivariance of every core function. The proof uses the concept of stabilisers that we introduce in §3.4. The proof is dependently-typed in nature, and it is best understood in the context of §3 and §3.4 in particular. However, we make the proof self-contained and re-introduce the concepts that are relevant for it.

Proof of Theorem 2.17

First, take a strictly equivariant canoniser $c : \underline{A} \to \underline{G}$, and let B be an arbitrary action. We need to show that every core function $f : \mathsf{Can}\, c \to \underline{B}$ is equivariant. Take $a, g \circledast a \in \mathsf{Can}\, c$. Since c is strictly equivariant, by Proposition 2.21, it is an immediate canoniser. So by Proposition 2.20, $ca = e = c(g \circledast a)$, therefore:

$$g = g * ca \underset{\substack{\uparrow \\ c \text{ strictly equivariant}}}{=} c\,(g \circledast a) = e$$

and equivariance holds vacuously.

Conversely, assume that every core function is equivariant, and let us show that c is strictly equivariant. Recall that for every $a \in \underline{A}$, we define the stabiliser sub-group of a as the set of symmetries that fix a:

$$G_a \coloneqq \left\{ s \in \underline{G} \,\middle|\, s \circledast a = a \right\}$$

Take the following action over the the set of dependent pairs consisting of an element in $\underline{A}$ and one of its stabilising symmetries:

$$\underline{B} \coloneqq (a : \underline{A}) \times G_a \qquad g \circledast \langle a, s \rangle \coloneqq \left\langle g \circledast a, s * g^{-1} \right\rangle$$

Indeed, this definition is well-typed because:

$$(s * g^{-1}) * (g \circledast a) = s \circledast (g^{-1} * g) \circledast a = s \circledast a = a$$

It is also an action, since:

$$e \circledast \langle a, s \rangle = \left\langle e \circledast a, s \circledast e^{-1} \right\rangle = \langle a, s \rangle$$

$$g \circledast h \circledast \langle a, s \rangle = \left\langle g \circledast h \circledast a, s * h^{-1} * g^{-1} \right\rangle = \left\langle (g * h) \circledast a, s * (g * h)^{-1} \right\rangle = (g * h) \circledast \langle a, s \rangle$$

Define:

$$f : \mathsf{Can}\, c \to \underline{B} \qquad f\, a \coloneqq \langle a, e \rangle$$

By our assumption, its w.l.o.g. extension is equivariant: $\mathsf{wlog}(c, f) : A \circledast\!\!\longrightarrow B$. Note that this map is given by:

$$\mathsf{wlog}(f, c)a = (ca)^{-1} \circledast f\,(ca * a) = (ca)^{-1} \circledast \langle ca * a, e \rangle = \left\langle a, \left((ca)^{-1}\right)^{-1} \right\rangle = \langle a, ca \rangle$$

Now take any $a \in \underline{A}$ and $g \in \underline{G}$, and calculate:

$$\langle g \circledast a, c\,(g \circledast a) \rangle = \mathsf{wlog}(c, f)(g \circledast a) = g \circledast \mathsf{wlog}(c, f)a = g \circledast \langle a, ca \rangle = \left\langle g \circledast a, (ca) * g^{-1} \right\rangle$$

Comparing the two second components, we have $c\,(g \circledast a) = (ca) * g^{-1}$ and the canoniser is strictly equivariant. ■

The next result shows that actions that admit strictly equivariant canonisers must be semi-regular, which makes this condition too strong for some actions:

Proof of Lemma 2.18

Assume $g \circledast a = a$. By equivariance of the canoniser, we thus have $ca = cg \circledast a = ca * g^{-1}$ and by cancelling ca on the left, we get $e = g$. ■

The final result concerns inheriting strict equivariance and immediateness from a canoniser:

Proof of Lemma 2.22

To show strict equivariance, we have that the projection is an equivariant map $\pi_1 : A \times B \circledast\!\!\longrightarrow A$, and equivariant maps compose. For immediateness, we calculate

$$\begin{aligned} c \circ \pi_1 \left((c \circ \pi_1 \langle a, b\rangle) * \langle a, b\rangle\right) &= c \circ \pi_1 \left(ca * \langle a, b\rangle\right) = c \circ \pi_1 \left(\langle ca * a, ca * b\rangle\right) \\ &= c \left(ca * a\right) = e \end{aligned}$$

and so $c \circ \pi_1$ is immediate when c is. ∎

D Dependent types: further details

We begin by establishing the correctness of the action over the Grothendieck construction $\coprod_{\underline{\Gamma}} A$, as we will use its properties pervasively, e.g., when extending contexts.

Proof of Proposition 3.7

Take a Γ-indexed action A and define an action on $\coprod_{\underline{A}} \underline{A}$ by:

$$g \circledast \left\langle \gamma : \underline{\Gamma}, a : \underline{A}_{\gamma} \right\rangle \coloneqq \left\langle g \circledast \gamma, g \circledast_{\gamma} a : \underline{B}(g \circledast \gamma) \right\rangle$$

It is easy to compute that this makes π_1 equivariant.

Conversely, given an action $\coprod_{\Gamma} A$ as in the assumption, define $g \circledast_{\gamma} a \coloneqq \pi_2(g \circledast \langle \gamma, a\rangle)$ and easily verify that this is indeed well-typed and does satisfy the two conditions for an indexed action. ∎

The following example lets us work more formally with concrete examples:

Example D.1. The Grothendieck construction extends equivariantly to equivariant terms and maps. It sends equivariant terms $\Gamma \vdash t : A$ to equivariant sections $(\gamma \mapsto \langle \gamma, t_{\gamma}\rangle) : \Gamma \circledast\!\!\longrightarrow \coprod_{\Gamma} A$. It sends a dependent equivariant function $\Gamma \vdash f : (x : A) \to B$ to the following equivariant map $\left(\langle \gamma, a\rangle \mapsto \left\langle \gamma, x{\mapsto}a, f_{\gamma} a\right\rangle\right) : \coprod_{\Gamma} A \circledast\!\!\longrightarrow \coprod_{\Gamma, x:A} B$. □

Next, we establish the indexed action axioms for the family of dependent functions, and the indexed equivariance of related operations.

Lemma D.2. *The indexed action structure over the family of dependent functions is an action, evaluation is equivariant, and currying and uncurrying preserve equivariance.*

Proof

Indeed, the action axioms hold by calculating with the correct indices as in the ordinary case. Similarly for evaluation:

$$\mathrm{eval}(g \circledast_{\gamma} f, x)_{g \circledast \gamma, x \mapsto g \circledast_{\gamma} a} = g \circledast f(g^{-1} \circledast g \circledast a) = g \circledast fa = g \circledast \mathrm{eval}(f, x)_{\gamma, x \mapsto a}$$

The bijective correspondence is given as in the set-theoretic case. If t is equivariant, then:

$$\begin{aligned} (x : A \mapsto t)_{g \circledast \gamma} &= (a \mapsto g \circledast t_{g \circledast \gamma, x \mapsto a}) = (a \mapsto g \circledast t_{g \circledast \gamma, x \mapsto (g \circledast (g^{-1} \circledast a))}) \\ &\quad \text{equivariance} \\ &\quad \downarrow \\ &= (a \mapsto g \circledast g \circledast t_{\gamma, x \mapsto (g^{-1} \circledast a)}) = g \circledast (x : A \mapsto t)_{\gamma} \end{aligned}$$

Conversely, if $x : A \mapsto t$ is equivariant, then since $\mathrm{eval}(-, x)$ is equivariant, t is. ∎

We now turn to prove the indexed counterpart to the w.l.o.g. Representation Theorem 2.13:

Proof of Theorem 3.12

We can prove this theorem directly, but it may be more interesting to reduce it to the ordinary representation theorem. Applying the Grothendieck construction to the situation, we have:

- G-actions $\coprod_\Gamma A$ and $\coprod_{\Gamma,x:A} B$;
- canoniser $c' \coloneqq \langle \gamma, a\rangle \mapsto c_\gamma a : \underline{\coprod_\Gamma A} \to \underline{G}$;
- we have that $\mathsf{Can}\, c' = \{\langle \gamma, a\rangle \mid a \in \mathsf{Can}\, c, \gamma\} =: \coprod_\Gamma \mathsf{Can}\, c$ (see below);
- core function $f' \coloneqq \left(\langle \gamma, a\rangle \mapsto \left\langle (\gamma, x{\mapsto}a), f_\gamma a\right\rangle\right) : \mathsf{Can}\, c' \to \underline{\coprod_{\Gamma,x:A} B}$;
- we have that $\mathsf{wlog}(c', f'; \langle \gamma, a\rangle) = \left\langle (\gamma, x{\mapsto}a), \mathsf{wlog}(c, f)_\gamma a\right\rangle$;
- the core function f' is equivariant iff f is (by Example D.1);
- we have that $\mathsf{wlog}(c', f')$ is equivariant iff $\mathsf{wlog}(c, f)$ is equivariant (see below).
- the function term $\underline{\Gamma} \vdash g : \left(x : \underline{A}\right) \to \underline{B}$ extends f along $\Gamma \vdash \mathsf{Can}\, c \subseteq A$ iff the function $\langle \gamma, a\rangle \mapsto \langle (\gamma, x{\mapsto}a), g_\gamma a\rangle : \coprod_{\underline{\Gamma}} \underline{A} \to \coprod_{\underline{\Gamma}, x:\underline{A}} \underline{B}$ extends f' along $\mathsf{Can}\, c' \subseteq \coprod_{\underline{\Gamma}} \underline{A}$ (see below).

The theorem follows by unpacking and re-packing definitions in both directions. Let's discharge the outstanding proof obligations.

Let $\langle \gamma, a\rangle$ be canonical for c', so $\langle \gamma, a\rangle = (c_\gamma a) \circledast \langle \gamma', a'\rangle$. Then $\gamma' = (ca)^{-1} \circledast \gamma$ and $a' \in A_{(ca)^{-1} \circledast \gamma}$, so $a \in \mathsf{Can}\, c, \gamma$, and so $\langle \gamma, a\rangle \in \coprod_\Gamma \mathsf{Can}\, c$. These implications are invertible, proving the converse.

Next, we show that the Grothendieck construction preserves the w.l.o.g. extension:

$$\begin{aligned}
\mathsf{wlog}(c', f'; \langle \gamma, a\rangle) &= (c'\, \langle \gamma, a\rangle)^{-1} \circledast f'((c'\, \langle \gamma, a\rangle) \circledast \langle \gamma, a\rangle) \\
&= (c_\gamma a)^{-1} \circledast f'((c_\gamma a) \circledast \langle \gamma, a\rangle) \\
&= (c_\gamma a)^{-1} \circledast f' \left\langle (c_\gamma a) \circledast \gamma, (c_\gamma a) \circledast a\right\rangle \\
&= (c_\gamma a)^{-1} \circledast \left\langle \left((c_\gamma a) \circledast \gamma, x{\mapsto}((c_\gamma a) \circledast a)\right), f_{(c_\gamma a) \circledast \gamma}(c_\gamma a) \circledast a\right\rangle \\
&= \left\langle (\gamma, x{\mapsto}a), (c_\gamma a)^{-1} \circledast f_{(c_\gamma a) \circledast \gamma}(c_\gamma a) \circledast a\right\rangle \\
&= \left\langle (\gamma, x{\mapsto}a), \mathsf{wlog}(c, f)_\gamma a\right\rangle
\end{aligned}$$

In addition, by Example D.1, one w.l.o.g. extension is equivariant iff the other is equivariant.

The final bullet point follows by functoriality of the Grothendieck construction. ∎

We now establish the bread-and-butter properties facilitating smooth dependently-typed symmetric programming and symmetric propositional reasoning. We start with dependent-function abstraction:

Lemma D.3 (abstraction). *Function abstraction of an equivariant term $\Gamma, x : A \vdash t : B$ is both:*

- *an equivariant term $\Gamma \vdash x : A \mapsto t : (x : A) \to B$.*
- *an equivariant function $\Gamma \vdash x : A \mapsto t : (x : A) \circledast\!\!\longrightarrow B$.*

Proof

Calculate, ignoring indices (as those type-check):

$$\begin{aligned}
(g \circledast (x : A \mapsto t))_\gamma\, a &\overset{\text{action on dep. functions}}{=} g \circledast ((x : A \mapsto t)(g^{-1} \circledast a)) \overset{\text{abstraction def}}{=} g \circledast t_{\gamma, x\mapsto(g^{-1} \circledast a)} \overset{t \text{ equivariant}}{=} t_{g \circledast \gamma, x\mapsto(g \circledast g^{-1} \circledast a)} \\
&= t_{g \circledast \gamma, x\mapsto a} = (x : A \mapsto t)_{g \circledast \gamma} a
\end{aligned}$$

showing abstraction is an equivariant term. It is also an equivariant function, for:

$$(x : A \mapsto t)_{g \circledast \gamma}(g \circledast a) \overset{\text{abstraction def}}{=} t_{g \circledast \gamma, x \mapsto g \circledast a} \overset{t \text{ equivariant}}{=} g \circledast t_{\gamma, x \mapsto a}$$

as we wanted. ∎

Substitution using an equivariant substitution preserves equivariance:

Lemma D.4 (reindexing and substitution). *Reindexing by an equivariant function $\theta : \Gamma \to \Delta$ sends equivariant terms $\Delta \vdash s : B$ to equivariant terms $\Gamma \vdash s\,[\theta] : B\,[\theta]$, and moreover every equivariant term $\Gamma \vdash t : B\,[\theta]$ induces an equivariant substitution $(\theta, y \mapsto t) : \Gamma \circledast\!\!\to (\Delta, y : B)$.*

Proof

Calculate:

$$g \circledast s\,[\theta]_\gamma \overset{\text{reindexing def}}{=} g \circledast s_{\theta\gamma} \overset{s \text{ equivariant}}{=} s_{g \circledast \theta\gamma} \overset{\theta \text{ equivariant}}{=} s_{\theta(g \circledast \gamma)} = s\,[\theta]_{g \circledast \gamma}$$

and so $s\,[\theta]$ is equivariant. Next, calculate:

$$\begin{aligned}(\theta, y \mapsto t)(g \circledast \gamma) &\overset{\text{substitution def}}{=} (\theta(g \circledast \gamma), y \mapsto t_{g \circledast \gamma}) \overset{t \text{ equivariant}}{=} (\theta(g \circledast \gamma), y \mapsto g \circledast t_\gamma) \overset{\text{action on context extension}}{=} g \circledast (\theta\gamma, y \mapsto t_\gamma) \\ &= g \circledast ((\theta, y \mapsto t)\gamma)\end{aligned}$$

and so $(\theta, y \mapsto t) : \Gamma \circledast\!\!\to (\Delta, y : B)$. ∎

Dependent pairing also preserves equivariant terms:

Lemma D.5 (dependent pairing). *Two equivariant terms $\Gamma \vdash t : A$ and $\Gamma \vdash s : B\,[x \mapsto t]$ induce an equivariant term $\Gamma \vdash t\,, s : (x : A) \times B$.*

Proof

Calculate:

$$(g \circledast (t\,, s))_\gamma = (g \circledast t_\gamma\,, g \circledast s_{\gamma, x \mapsto t_\gamma}) \overset{t, s \text{ equivariant}}{=} (t_{g \circledast \gamma}\,, s_{g \circledast \gamma, x \mapsto (g \circledast t_\gamma)})$$

as we wanted. ∎

We now turn to propositions:

Lemma D.6 (subfamilies). *Each equivariant proposition $\Gamma, x : A \vdash \varphi : \mathsf{Prop}$ induces an indexed action on the subfamily $\Gamma \vdash \{x : A \mid \varphi\}$ that lifts A along the subset inclusions. Equivariant terms $\Gamma \vdash t : A$ satisfying φ lift to equivariant terms inhabiting the associated subspace i.e., $\Gamma \vdash \mathsf{lift}\ t : \{x : A \mid \varphi\}$.*

Proof

Equivariance means that if $a \in \underline{A}_\gamma$ satisfies φ, then so goes $g \circledast a$. Therefore $g \circledast_\gamma a \in \{x : A \mid \varphi\}$, and the action lifts. For equivariance of the lifted term, use injectivity of the coercion function $\Gamma \vdash \mathsf{coerce} := a \mapsto a : \{x : A \mid \varphi\} \circledast\!\!\to A$:

$$\mathsf{coerce}_{g \circledast \gamma}(g \circledast (\mathsf{lift}\ t)_\gamma) \overset{\text{coerce equivariance}}{=} g \circledast \mathsf{coerce}_\gamma\ \mathsf{lift}\ t_\gamma \overset{\text{coercion}}{=} g \circledast t_\gamma \overset{t \text{ equivariant}}{=} t_{g \circledast \gamma} = \mathsf{coerce}_{g \circledast \gamma}(\mathsf{lift}\ t)_{g \circledast \gamma}$$

and so $g \circledast (\mathsf{lift}\ t)_\gamma = (\mathsf{lift}\ t)_{g \circledast \gamma}$. ∎

We use these facts implicitly when conducting reasoning:

Lemma D.7. *The logical connectives are equivariant* $(\wedge), (\vee) : \mathsf{Prop}^2 \circledast\!\!\longrightarrow \mathsf{Prop}$, $(\neg) : \mathsf{Prop} \circledast\!\!\longrightarrow \mathsf{Prop}$, *as well as the quantifiers* $\forall, \exists : (A \to \mathsf{Prop}) \circledast\!\!\longrightarrow \mathsf{Prop}$.

Proof

The logical connectives operate between trivial actions hence equivariant. For the quantifiers, assume $\forall t = \mathsf{True}$, and so, for each $a \in \underline{A}_\gamma$, we have $t(g^{-1} \circledast a) = \mathsf{True}$, i.e.:

$$\forall(g \circledast t) = \forall x.t(g^{-1} \circledast x) = \mathsf{True}$$

Conversely, if $\forall(g \circledast t)$, then by instantiating with $g \circledast a$ we have that $ta = t(g^{-1} \circledast g \circledast a) = \mathsf{True}$. Therefore $\forall t = \forall g \circledast t$. A similar argument holds for $\exists$, or by use of de Morgan. ■

We conclude our treatment of propositions by explicating how symmetric reasoning w.l.o.g. often manifests—via a property invariant under the symmetry of interest:

Proof of Lemma 3.17

Take canonical a and $g \circledast_\gamma a$. By type-checking, $g \circledast f_\gamma a, f_\gamma(g \circledast a)$ prove the same proposition, namely $\varphi_{g \circledast \gamma, x \mapsto (g \circledast a)}$. Therefore, $g \circledast f_\gamma a = f_\gamma(g \circledast a)$. ■

In some concrete cases, working with the entire group of permutations incurs a combinatorial explosion. Group theorists often generate the group from a smaller set of symmetries, and then only work with those. Recall that a group G is *generated* by a subset $S \subseteq \underline{G}$ when G is the smallest subgroup of G containing S, equivalently, when every $\underline{G}$-element is the multiplication of a sequence of $\underline{G}$-elements or their inverses. For example, the symmetry group S_3 is generated by the two transpositions $(1\,2)$ and $(2\,3)$. We have an indexed variation of a well-known lemma:

Lemma D.8. *Let* $\Gamma \vdash A$ *and* $\Gamma, x : A \vdash B$ *be indexed* G*-actions. If* S *generates* G *and* $\underline{\Gamma} \vdash f : (x : \underline{A}) \to \underline{B}$ *is equivariant w.r.t. symmetries in* S*, then* f *is equivariant* $\Gamma \vdash f : (x : A) \circledast\!\!\longrightarrow B$*. When* $B = \mathsf{Prop}$ *and* $[S]^{-1} \subseteq S$*, it suffices to show lax/oplax equivariance, i.e.:*

$$(\forall g \in S, \forall x.f(g \circledast x) \implies fx) \vee (\forall g \in S, \forall x.f(g \circledast x) \impliedby fx)$$

Proof of Lemma D.8

Let $H \subseteq \underline{G}$ be the set of symmetries that f respects. Then $S \subseteq H$. Since $f(e \circledast a) = fa = e \circledast (fa)$, we have: $e \in H$. Take $h \in H$ we have:

$$f(h \circledast a) = h \circledast (fa) \implies f(h^{-1} \circledast a) = h^{-1} \circledast f(h \circledast (h^{-1} \circledast a)) = h^{-1} \circledast (fa)$$

so $h^{-1} \in H$. Take $g, h \in H$ and then:

$$f((g * h) \circledast a) = f(g \circledast (h \circledast a)) = g \circledast (f(h \circledast a)) = g \circledast (h \circledast (fa)) = (g * h) \circledast (fa)$$

so $g * h \in H$. Thus $H \subseteq \underline{G}$ is a subgroup containing S, and so $H = \underline{G}$.

When $B = \mathsf{Prop}$ and $[S]^{-1} \subseteq S$, by substituting $x \mapsto g^{-1} \circledast a$ we have:

$$fx \iff f(g^{-1} \circledast (g \circledast x)) \overset{[S]^{-1} \subseteq S}{\implies} f(g \circledast x)$$

For the left disjunct, f is in fact equivariant for symmetries in S. Similarly for the right disjunct. ■

We now turn to study equivariant canonisers. It turns out that the indexed setting is fruitful for this purpose. We first show that the indexed action of *distinguishable symmetries* G/A from Definition 3.18 is well-defined:

Lemma D.9. *Let* A *be a* G*-action. The action over distinguishable symmetries* G/A *is congruent w.r.t. indistinguishability, hence is well-defined.*

Proof
Assume $s_1 \overset{a}{\sim} s_2$. Calculate:

$$(s_1 * g^{-1}) \circledast (g \circledast a) = s_1 \circledast a = s_2 \circledast a = (s_2 * g^{-1}) \circledast (g \circledast a)$$

and so $s_1 * g^{-1} \overset{g \circledast a}{\sim} s_2 * g^{-1}$, and the action is well-defined. ■

In §3.4 we mentioned in passing the stabiliser family alongside the indexed actions of the orbits and the indistinguishable symmetries. The stabilisers also form an indexed action:

Example D.10. The *stabiliser* family has all the symmetries fixing an index $a \in \underline{A}$ as the fibre over it: $a : \underline{A} \vdash G_a \coloneqq \{g \in \underline{G} \mid g \circledast a = a\}$. The stabiliser has an A-indexed action:

$$g \circledast_a s \coloneqq s * g^{-1} \in G_{g \circledast a} \quad \text{since} \quad s * g^{-1} \circledast (g \circledast a) = s \circledast a = a$$ □

We now prove our reformulation of the famous Orbit-Stabiliser Theorem 3.20:

Proof of Theorem 3.20
The bijectivity of φ is the well-known theorem. We show equivariance:

$$\begin{aligned} \varphi_{g \circledast x}(g \circledast_x [h]_{\sim x}) &\overset{\text{action on } G/A}{=} \varphi_{g \circledast x}([h * g^{-1}]_{g \circledast x}) = (h * g^{-1}) \circledast (g \circledast x) = h \circledast x \overset{\text{action on } G \circledast [x]}{=} g \circledast_x (h \circledast x) \\ &= g \circledast_x \varphi_x [h]_{\sim x} \end{aligned}$$

and we conclude as inverses of equivariant functions are equivariant. ■

Next, we show characterise equivariant canonisers as those for which stable and equivariant core functions coincide. First, an observation:

Lemma D.11. *Let $c : \underline{A} \to \underline{G}$ be an equivariant canoniser for a G-action A. For every canonical element $a \in \mathsf{Can}\, c$, we have $ca \overset{a}{\sim} e$.*

Proof
Since a is canonical, there is some $b \in \underline{A}$ such that $a = (cb) \circledast b$. Since c is equivariant, we have:

$$ca = c((cb) \circledast b) \overset{a}{\sim} (cb) * (cb)^{-1} = e$$
$$\uparrow \text{ equivariance } (cb) \circledast b = a$$

as we wanted. ■

Now, for the proof:

Proof Theorem 3.25
First, assume $c : (a : A) \circledast\!\!\to (G/A)a$ is equivariant, and consider any stable core function $f : \mathsf{Can}\, c \to B$. We show that f is equivariant.

Consider any canonical $g \circledast a, a \in \mathsf{Can}\, c$. Since c is equivariant, by Lemma D.11, we have $ca \overset{a}{\sim} e$ and so $ca \in G_a$. By equivariance, we have:

$$g \circledast^{G/A} [ca]_a = [c(g \circledast a)]_{g \circledast a} \overset{\text{Lemma D.11 for } g \circledast a}{=} [e]_{g \circledast a} \implies [ca]_a = g^{-1} \circledast [e]_{g \circledast a} = [g]_a$$

and so $g \overset{a}{\sim} ca \in G_a$ and so $g \in G_a \subseteq G_{fa}$. Therefore:

$$f(g \circledast a) \overset{g \in G_a}{=} fa \overset{g \in G_{fa}}{=} g \circledast fa$$

and so f is equivariant.

Conversely, assume that every stable core function is equivariant. We need to show the canoniser is equivariant. The proof adapts the argument in Theorem 2.17. Since we now have the indexed action vocabulary, the proof simplifies somewhat.

Take the A-indexed action $B \coloneqq G/A$, and the following core function:

$$f : (x : \mathsf{Can}\, c) \to (G/A)x \qquad fa \coloneqq [e]_a$$

It is stable. Indeed, take any canonical $a \in \mathsf{Can}\, c$ and a stabilising symmetry $g \in G_a$. Since G_a is a group, $g^{-1} \in G_a$, and so $[g^{-1}]_a = [e]_a$. Calculate:

$$g \circledast_a^{G/A} (fa) = g \circledast_a^{G/A} [e]_a = \left[e * g^{-1}\right]_{g \circledast a} \overset{g \in G_a}{=} \left[g^{-1}\right]_a = [e]_a = fa$$

Therefore $g \in G_{fa}$, and so $G_a \subseteq G_{fa}$, and so f is stable.

By assumption, f is equivariant, and by the Indexed Representation Theorem 3.12 we have an equivariant $\mathrm{wlog}(c, f) : (x : A) \circledast\!\!\longrightarrow (G/A)x$. We will show that $\mathrm{wlog}(c, f) = [c-]_-$ and therefore c is equivariant.

Take any $a \in \underline{A}$ and calculate:

$$\mathrm{wlog}(c, f)a = (ca)^{-1} \circledast f(ca \circledast a) = (ca)^{-1} \circledast [e]_{ca \circledast a} = \left[\left((ca)^{-1}\right)^{-1}\right]_{((ca)^{-1}) \circledast ca \circledast a} = [ca]_a$$

Therefore $\mathrm{wlog}(c, f) = [c-]_-$, as we wanted. ∎

We conclude this section by proving that every action has an equivariant canoniser. Recall that a *transversal* of an action is a (dependent) function $t : (a : \underline{A}) \to G \circledast [a]$ that picks a representative for the orbit of its input: $ta = tb$ iff $G \circledast [a] = G \circledast [b]$.

Proof Theorem 3.26

Consider the family of orbits $x : A \vdash G \circledast [x]$, and use the axiom of choice to choose a transversal: $t : (x : A) \to G \circledast [x]$. For each $a \in \underline{A}$, a and $t\, a \in G \circledast [a]$ are symmetric, and so there is some symmetry $g \in \underline{G}$ such that $g \circledast a = t\, a$. Use the axiom of choice to choose a canoniser $c : A \to \underline{G}$ such that $ca \circledast a = t\, a$. We will show this canoniser is equivariant.

Take any element $a \in \underline{A}$ and symmetry $g \in \underline{G}$. Then $g \circledast a$ and a are in the same orbit, and therefore $t(g \circledast a) = t\, a$. Calculate:

$$(c(g \circledast a) * g) \circledast a = c(g \circledast a) \circledast (g \circledast a) = t(g \circledast a) = t\, a = ca \circledast a$$

Therefore, $c(g \circledast a) * g \overset{a}{\sim} ca$, and so $[c(g \circledast a) * g]_a = [ca]_a$. Calculate:

$$[c(g \circledast a)]_{g \circledast a} = \left[c(g \circledast a) * g * g^{-1}\right]_{g \circledast a} = g \circledast [c(g \circledast a) * g]_a = g \circledast [ca]_a$$

and so c is equivariant. ∎

E Algebraic data types: further details

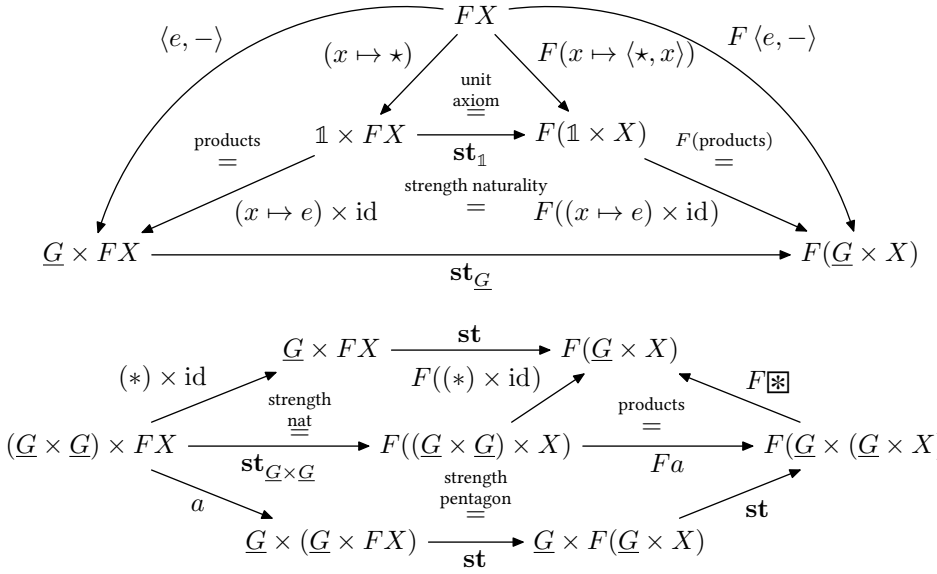
In some sense, the theoretical development in this section is a special case of the proceedings two sections. However, it requires some additional abstractions that make the material less accessible: indexed families (Appendix F), or monadic adjunctions and distributive laws (Appendix G). Therefore, we decided to include more specialised proofs for the simply-typed case, that use less categorical reasoning and mostly concern sets and functions.

We start by proving that every Cartesian strength is a G-strength (cf. Remark 4.3).

Lemma E.1. *Let* $\mathrm{st} : X \times FY \to F(X \times Y)$ *be a Cartesian strength for a functor* $F : \mathsf{Set} \to \mathsf{Set}$. *Then* $\mathrm{st}_{\underline{G},X} : \underline{G} \times FX \to F(\underline{G} \times X)$ *is a G-strength.*

Proof

It's more natural to use diagrammatic reasoning here:



as we wanted. ∎

Next, we prove that a G-strength for F and an G-action A let us equip $F\underline{A}$ with a G-action:

Proof of Lemma 4.4

This definition is a G-action by straightforward calculation:

$$e \circledast t = F(\circledast)(\mathrm{st}(e,t)) \overset{\text{first axiom}}{=} F(\circledast)(F(x \mapsto \langle e, x\rangle)t) \overset{\text{map fusion}}{=} F(x \mapsto \langle e, \circledast\rangle\, x)t = t$$

$$\begin{aligned}
g \circledast (h \circledast t) &= F(\circledast)(\mathrm{st}(g, F(\circledast)(\mathrm{st}(h,t)))) \\
&\overset{\text{strength nat.}}{=} F(\circledast)(F(\langle g', h', x\rangle \mapsto \langle g', h' \circledast x\rangle)(\mathrm{st}(g, (\mathrm{st}(h,t))))) \\
&\overset{\text{map fusion}}{=} F(\langle g', h', x\rangle \mapsto g' \circledast h' \circledast x)(\mathrm{st}(g, (\mathrm{st}(h,t)))) \\
&= F(\langle g', h', x\rangle \mapsto (g' * h') \circledast x)(\mathrm{st}(g, (\mathrm{st}(h,t)))) \\
&\overset{\text{map fusion}}{=} F(\circledast)(F(\langle g', h', x\rangle \mapsto \langle g' * h', x\rangle)(\mathrm{st}(g, (\mathrm{st}(h,t))))) \\
&\overset{\text{second axiom}}{=} F(\circledast)(\mathrm{st}(g * h, t)) =: (g * h) \circledast t
\end{aligned}$$

and so we have an action.

Let $h : A \circledast\!\!\longrightarrow B$ be equivariant. Calculate:

$$F\,h(g \circledast t) = F\,h(F(\circledast)(\mathrm{st}(g,t))) \overset{\text{map fusion}}{=} F(\langle g', x\rangle \mapsto h(g' \circledast x))(\mathrm{st}(g,t))$$

$$= F(\langle g', x\rangle \mapsto g' \circledast h(x))(\mathrm{st}(g,t)) \overset{\text{map fusion}}{=} F(\circledast)\,F(\mathrm{id} \times h)(\mathrm{st}(g,t))$$

$$\overset{\text{strength nat.}}{=} F(\circledast)\,\mathrm{st}(g, F\,ht) = g \circledast F\,ht$$

and so we have an equivariant map $F\,h : FA \circledast\!\!\longrightarrow FB$. ■

Next, we prove that G-strengths for each component functor induce a G-strength for the coproduct functor:

Proof Proposition 4.7

Follows by Set's distributivity: the natural isomorphism $\underline{G} \times (X + Y) \cong (\underline{G} \times X) + (\underline{G} \times Y)$. ■

The following technical result is the workhorse that allows us to both derive the action on the data type from its strength, and to reason about properties of parameterised maps. It is prevalent in Fiore et al.'s work on abstract syntax [10, 11, 13], originally proposed by Bird and Paterson [5]. Here we use the following special case.

Theorem E.2 (parameterised initiality). *Let A be a set, $F : \mathrm{Set} \to \mathrm{Set}$ a functor, $\mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ an initial algebra, and $\mathrm{st} : A \times FX \to F(A \times X)$ a natural transformation in X. For every F-algebra $f : FX \to X$ there is a unique A-*parameterised *homomorphism $\mathrm{pfold}\,f : \langle \boldsymbol{\mu}F, \mathrm{roll}\rangle \to \langle X, f\rangle$, i.e., a function $\mathrm{pfold}\,f : A \times \boldsymbol{\mu}F \to X$ satisfying:*

$$\mathrm{pfold}\,f\,\langle a, \mathrm{roll}\,t\rangle = f(F(\mathrm{st}(a,t)))$$

Proof

Using both the given strength and the Cartesian-closed structure, define the following morphism:

$$\begin{array}{ccc} A \times F(X^A) & \xrightarrow{\ \mathbf{st}\ } & F(A \times X^A) \\ {\scriptstyle g}\downarrow & \coloneqq & \downarrow{\scriptstyle F\,\mathrm{eval}} \\ X & \xleftarrow[\ f\ ]{} & FX \end{array}$$

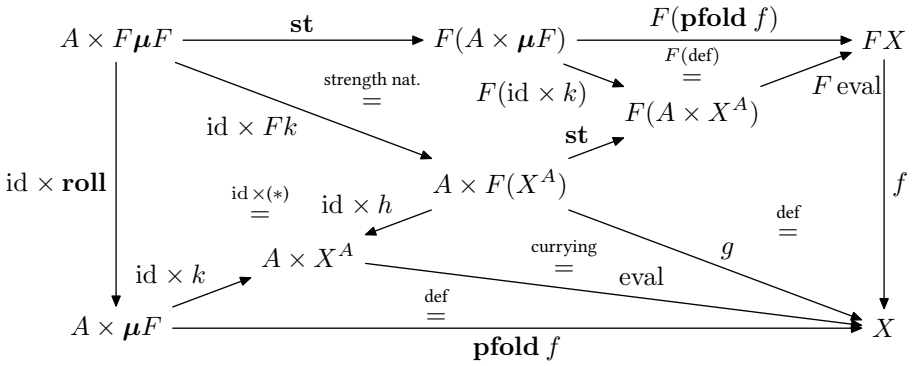
Currying, we have $h \coloneqq \mathrm{curry}\,g : F(X^A) \to X^A$, and so by ordinary initiality, there is a unique homomorphism into X^A:

$$\begin{array}{ccc} F\boldsymbol{\mu}F & \xrightarrow{\ Fk\ } & F(X^A) \\ {\scriptstyle \mathbf{roll}}\downarrow & \overset{(*)}{=} & \downarrow{\scriptstyle h} \\ \boldsymbol{\mu}F & \xrightarrow[\ k\ ]{} & X^A \end{array}$$

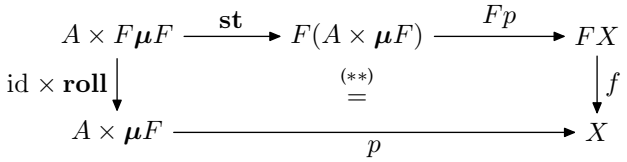
We defined the required parameterised traversal as the uncurried function:

$$\mathrm{pfold}\,f \coloneqq \mathrm{uncurry}\,k : A \times \boldsymbol{\mu}F \xrightarrow{\mathrm{id}\times k} A \times X^A \xrightarrow{\mathrm{eval}} X$$

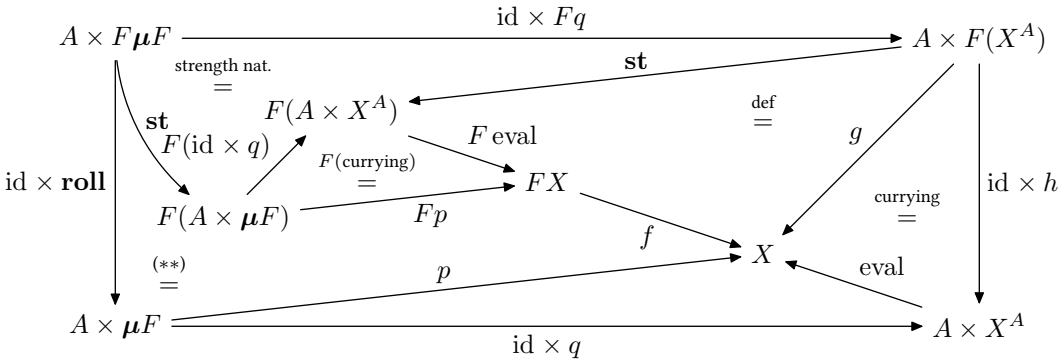
We need to show that it satisfies the required equation, and that it is the unique such function. Calculate:



Now, take any map $p : A \times \boldsymbol{\mu}F \to X$ that satisfies:



Let $q \coloneqq \mathrm{curry}\ p : \boldsymbol{\mu}F \to X^A$. Calculate:



and so, by the universal property of the exponential, q satisfies $(*)$, and so $q = k$, hence:

$$p = \mathrm{uncurry}\ q = \mathrm{uncurry}\ k = \mathrm{pfold}\ f$$

and we proved uniqueness. ■

We now lay the capstone in the theory of data types, the Equivariant Initiality Theorem 4.8:

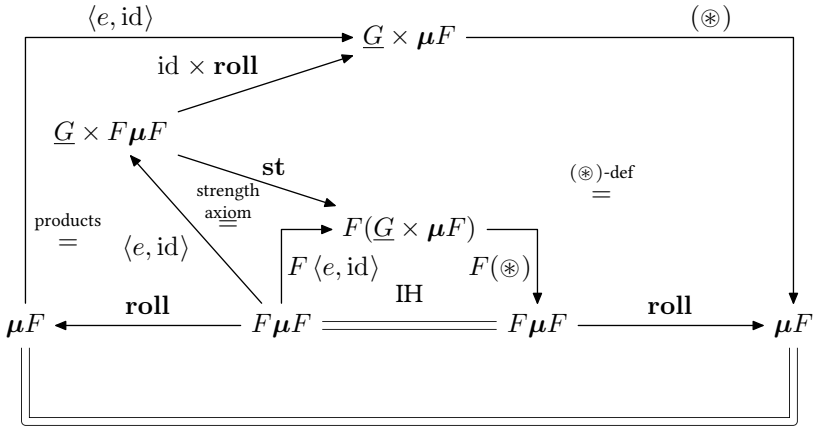
Proof of Theorem 4.8

We have three things to do:

- show the unit axiom for $(\circledast)$.
- show the associativity axiom for $(\circledast)$.
- show that fold f is equivariant when f is equivariant.

All three are straightforward calculations, appealing to initiality or parameterised initiality.

For the unit axiom, appeal to initiality:



and so, by initiality, $e \circledast t = t$.

For associativity, we appeal to parameterised initiality, for the composite strength:

$$\begin{array}{ccc} (\underline{G} \times \underline{G}) \times FX & \xrightarrow{a} & \underline{G} \times (\underline{G} \times X) \\ \Big\downarrow \mathbf{st}_2 & & \Big\downarrow \mathrm{id} \times \mathbf{st} \\ & \coloneqq & \underline{G} \times F(\underline{G} \times X) \\ & & \Big\downarrow \mathbf{st} \\ F((\underline{G} \times \underline{G}) \times X) & \xleftarrow[Fa^{-1}]{} & F(\underline{G} \times (\underline{G} \times X)) \end{array}$$

via the calculation in Fig. 4. Therefore, $(g * h) \circledast t = g \circledast (h * t)$, and we have an action. Note that the defining property of $(\circledast)$ makes the initial algebra structure equivariant roll : $F\boldsymbol{\mu}F \circledcirc\!\!\longrightarrow \boldsymbol{\mu}F$.

Finally, take any equivariant algebra $f : FA \circledcirc\!\!\longrightarrow A$, and consider the unique F-homomorphism fold $f : \boldsymbol{\mu}F \to \underline{A}$. We appeal to parametrised initiality w.r.t. the G-strength, and calculate as in Fig. 5. Therefore fold $f(g \circledast t) = g \circledast (\text{fold } f)$, as we wanted. ■

The proofs from §4.3 are identical and special cases to their indexed variants, and do not use abstract categorical concepts. We therefore only prove them in the indexed case (cf. Appendix F).

F Indexed data types

The theory of indexed data types has a few more moving parts. So in this case we do most of our proofs more abstractly in Appendix G. In this appendix we establish the assumptions needed for that development, prove a result about canonisers, who do not feature in the abstract account, and state the indexed version of the Parameterised Initiality Theorem.

Proof Theorem 4.16

Let's start with the left adjoint. It is an action by direct calculation:

$$e \circledast_\gamma \langle g, x\rangle \coloneqq \langle e * g, x\rangle = \langle g, x\rangle \qquad g \circledast h \circledast \langle s, x\rangle \coloneqq \langle g * (h * s), x\rangle = \langle (g * h) * s, x\rangle = (g * h) \circledast \langle s, x\rangle$$

Given any Γ-indexed G-action A and a vertical map $h : X \to \underline{A}$, define:

$$h^\dagger : (g : G) \times g \circledast X \circledcirc\!\!\longrightarrow A \qquad h^\dagger_\gamma \langle g, x\rangle \coloneqq \left(g \circledast_{g^{-1} \circledast \gamma} x\right) : X_\gamma$$

Indeed:

$$h^\dagger_{g \circledast \gamma}(g \circledast \langle s, x\rangle) = h^\dagger_{g \circledast \gamma}(\langle g * s, x\rangle) = (g * s) \circledast (hx) = g \circledast (s \circledast (hx)) = g \circledast h^\dagger_\gamma \langle s, x\rangle$$

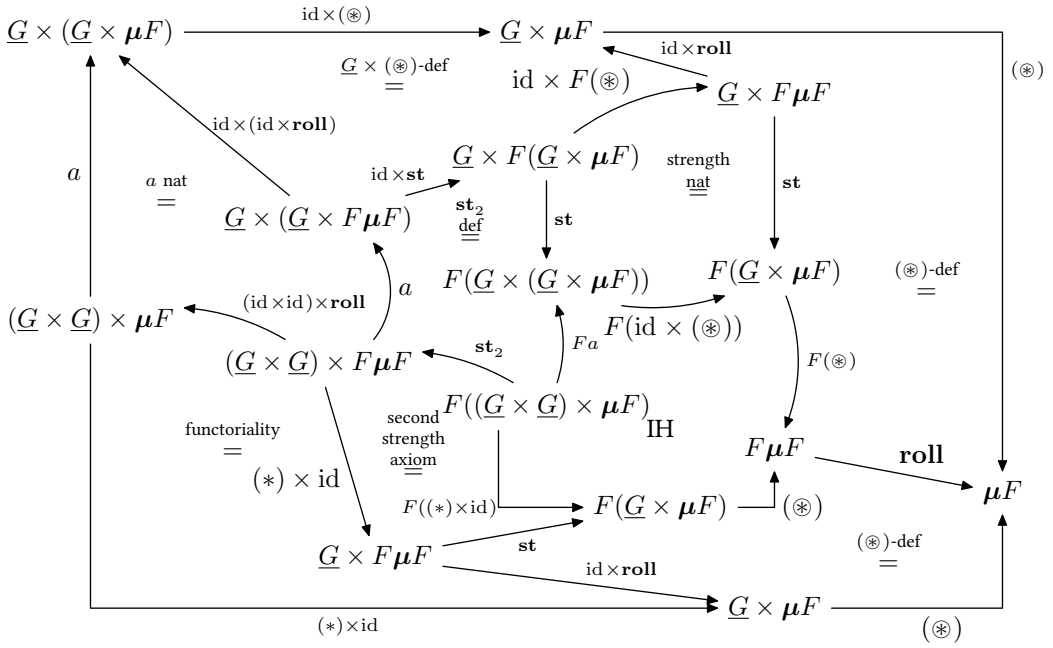


Fig. 4. associativity of the action over the initial algebra

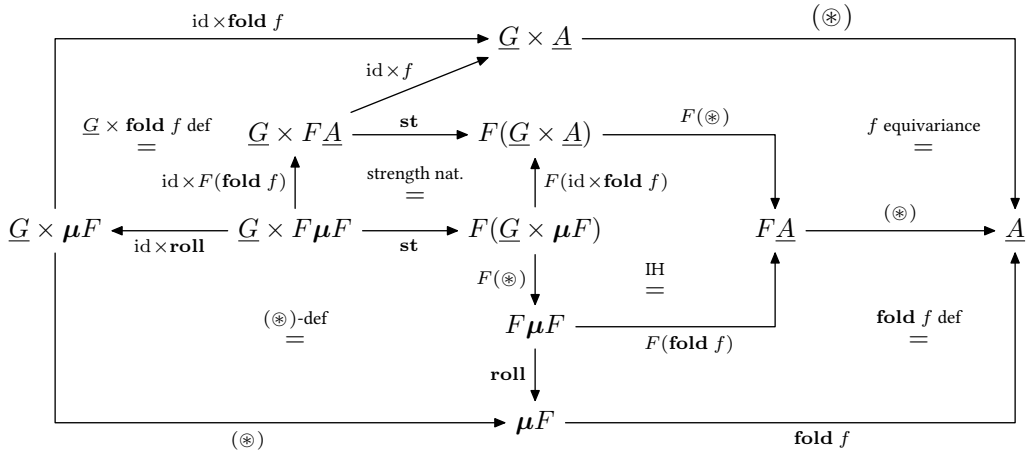


Fig. 5. equivariance of the homomorphism into an equivariant algebra

Now, $h^\dagger$ extends h along the unit, since:

$$h^\dagger \langle e, x \rangle = e \circledast hx = hx$$

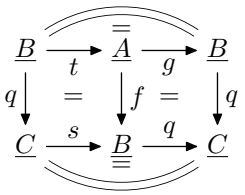
If $\overline{h} : (g : G) \times g \circledast X \mathrel{\circledast\!\!\longrightarrow} A$ is any other such extension, we have:

$$\overline{h}_\gamma \langle g, x\rangle = \overline{h}_\gamma(g \circledast \langle e, x\rangle) \overset{\text{equivariance}}{=} g \circledast \overline{h}_{g^{-1}\circledast\gamma} \langle e, x\rangle = g \circledast h_{g^{-1}\circledast\gamma} x = h^\dagger_\gamma \langle g, x\rangle$$

($\overline{h}$ extends the unit)

and so $\overline{h} = h^\dagger$, as we wanted.

The monadicity result follows by a simple application of Beck's monadicity theorem [20, Thm. VI.7.1] (in fact, the vulgar tripleability theorem of Ex. VI.7.8 applied here, too). Start with a split coequalier diagram, in which A and B are Γ-indexed actions, and $f, g : A \mathrel{\circledast\!\!\longrightarrow} B$ are equivariant:



First, equip $\underline{C}$ with an indexed action: $h \circledast_\gamma c \coloneqq q_{g\circledast\gamma}(h \circledast_\gamma s_\gamma c)$. Indeed: $e \circledast_\gamma c = q(e \circledast_\gamma sc) = q(sc) = c$ and:

$$h \circledast k \circledast c = q(h \circledast s(q(k \circledast sc))) = q(h \circledast (k \circledast_\gamma sc)) = q((h * k) \circledast sc) = (h * k) \circledast c$$

Now we need to show the maps are equivariant.Let's start with showing $q : B \mathrel{\circledast\!\!\longrightarrow} C$:

$$h \circledast (q\, b) = q(h \circledast (s(q\, b))) = q(h \circledast (f(t\, b))) = q(f(h \circledast (t\, b))) = q(g(h \circledast (t\, b))) = q(h \circledast g(t\, b)) = q(h \circledast b)$$

Now $t : B \mathrel{\circledast\!\!\longrightarrow} A$,

$$t(h \circledast b) = t(h \circledast (g(t\, b))) = t(g(h \circledast (t\, b))) = h \circledast (t\, b)$$

Finally, $s : C \mathrel{\circledast\!\!\longrightarrow} B$:

$$s(h \circledast c) = s(q(h \circledast s\, c)) = f(t(h \circledast s\, c)) = h \circledast (f(t(s\, c))) = h \circledast (s(q(s\, c))) = h \circledast (s\, c)$$

therefore $\underline{-}$ lifts split coequalisers of parallel pairs.

Now, if in the diagram C already had an indexed A-action structure and q was equivariant, then necessarily $h \circledast c = h \circledast q(s\, c) = q(h \circledast (s\, c))$, and so the action on $\underline{C}$ is the one we defined, making s and t equivariant.

Turning to the right adjoint, we first prove it's an action:

$$e \circledast_\gamma \varphi \coloneqq (h \mapsto \varphi(h * e)) = (h \mapsto \varphi h) = \varphi$$

and:

$$(g \circledast k \circledast \varphi)h = (k \circledast \varphi)(h * g) = \varphi((h * g) * k) = \varphi(h * (g * k)) = ((g * k) \circledast \varphi)h$$

and so $g \circledast k \circledast \varphi = (g * k) \circledast \varphi$.

Next, take any Γ-indexed G-action A, and a vertical map $\underline{\Gamma} \vdash f : \underline{A} \to X$. Define $\Gamma \vdash f^\dagger : A \mathrel{\circledast\!\!\longrightarrow} (g : G) \to g^{-1} \circledast X$ by:

$$f^\dagger_\gamma a \coloneqq g \mapsto f(g \circledast a) \in \left(g : \underline{G}\right) \to X_{g\circledast\gamma}$$

It is equivariant:

$$f^\dagger_{h\circledast\gamma}(h \circledast a) = g \mapsto f(g \circledast (h \circledast a)) = g \mapsto f((g * h) \circledast a) = h \circledast (g \mapsto f(g \circledast a)) = h \circledast f^\dagger_\gamma a$$

It lifts f along eval $\langle -, e\rangle$:

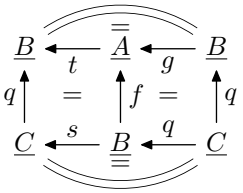
$$\text{eval}\left\langle f^\dagger a, e\right\rangle = (f^\dagger a)e = f(e \circledast a) = f\,a$$

Now if $\overline{f} : A \circledcirc\!\!\rightarrow (g : G) \to g^{-1} \circledast X$ lifts f along eval $\langle -, e\rangle$, then:

$$(\overline{f}a)g = (\overline{f}a)(e * g) = (g \circledast (\overline{f}a))(e) = \text{eval}\left\langle g \circledast (\overline{f}a), e\right\rangle = \text{eval}\left\langle \overline{f}(g \circledast a), e\right\rangle = f(g \circledast a)$$

and so $\overline{f} = f^\dagger$, and we have a right adjoint.

We use Beck's theorem in its dual formulation. Consider a split equaliser, with A, B actions:



Equip $\underline{C}$ with the following indexed G-action structure: $k \circledast_\gamma c \coloneqq s(k \circledast_\gamma (q\,c))$. It is indeed an action, for $e \circledast c = s(e \circledast (q\,c)) = s(q\,c) = c$ and:

$$k \circledast h \circledast c = s(k \circledast (q(s(h \circledast (q\,c))))) = s(k \circledast h \circledast (q\,c)) = s((k * h) \circledast (q\,c)) = (k * h) \circledast c$$

We need to show the structure maps q and s (but not t!) are equivariant. For q:

$$q(k \circledast c) = q(s(k \circledast (q\,c))) = k \circledast (q\,c)$$

For s:

$$k \circledast sb = s(k \circledast (q(s\,b))) = s(k \circledast b)$$

Now, given an equalising equivariant map $\Gamma e : X \circledcirc\!\!\rightarrow B$, by moving to families, we know the mediating map is $u : X \overset{e}{\circledcirc\!\!\rightarrow} B \overset{s}{\circledcirc\!\!\rightarrow} C$, and therefore equivariant. Therefore $\underline{\_}$ lifts equalisers of split equalisers. Now, equip C with any action that makes $q : C \to B$ equivariant. We then have:

$$k \circledast c = s(q(k \circledast c)) = s(k \circledast (q\,c))$$

and so the action on C is the lifted one, making $q : C \to B$ a coequaliser, i.e., $\underline{\_}$ reflects equalisers that split in $\underline{\Gamma}$-families. ■

Next, we justify not checking the second axiom of G-strengths in the cases $g = e$ or $h = e$:

Proof Lemma B.3

For $h = e$:

$$\begin{aligned}
F(\boxplus)(\text{st}(g, \text{st}(e, t))) &\overset{\text{first axiom}}{=} F(\boxplus)(\text{st}_X(g, F(x \mapsto \langle e, x\rangle)t)) \\
&\overset{\text{naturality}}{=} F(\boxplus)(F(\langle g', x\rangle \mapsto \langle g', e, x\rangle)\,\text{st}(g, t)) \\
&\overset{\text{map fusion}}{=} (F(\langle g', x\rangle \mapsto \langle g' * e, x\rangle)\,\text{st}(g, t)) \\
&= \text{st}(g, t) = \text{st}(g * e, t)
\end{aligned}$$

Next, for $g = e$:

$$\begin{aligned} F(\circledast)(\mathrm{st}(e, \mathrm{st}(h,t))) &\overset{\text{first axiom}}{=} F(\circledast)(F(\langle h', x\rangle \mapsto \langle e, h', x\rangle)(\mathrm{st}(h,t))) \\ &\overset{\text{map fusion}}{=} F(\langle h', x\rangle \mapsto \langle e * h', x\rangle)(\mathrm{st}(h,t)) \\ &= \mathrm{st}(h,t) = \mathrm{st}(e * h, t) \end{aligned}$$

as we wanted. ■

Next, we state the indexed analogue to the Parameterised Initiality Theorem E.2 for the indexed setting:

Theorem F.1 (indexed parameterised initiality)**.** *Let Γ be a G-action, $F : \mathrm{Fam}_{\underline{\Gamma}} \to \mathrm{Fam}_{\underline{\Gamma}}$ an endofunctor equipped with a symmetry strength, and $\underline{\Gamma} \vdash \mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ an initial F-algebra. For every F-algebra $f : FX \to X$, there is a unique* parameterised homomorphism *into $\langle X, f\rangle$, i.e., a map* $\mathrm{pfold}\ f : G \circledast \boldsymbol{\mu}F \to X$ *satisfying:* $(\mathrm{pfold}\ f)(g, \mathrm{roll}\ u) = f \circ F(\mathrm{pfold}\ f)\ \mathrm{st}(g,u)$.

We will prove this theorem and the Indexed Equivariant Initiality Theorem B.5 using the more abstract tools of the next section.

The remainder of this section establishes the results concerning canonisers for indexed G-strong functors, which do not have a general counterpart. By ignoring the indices, they also generalise the corresponding results from §4.3.

Lemma F.2. *Let Γ be a G-action, $F : \mathrm{Fam}_{\underline{\Gamma}} \to \mathrm{Fam}_{\underline{\Gamma}}$ be a functor equipped with a symmetry strength* $\mathrm{st} : G \circledast FX \to F(G \circledast X)$*, and $c : F \to \underline{G}$ an indexed canoniser for F. The canonical elements extend to a functor* $\mathrm{Can}\ c : G\text{-}\mathrm{Act}_{\underline{\Gamma}} \to \mathrm{Fam}_{\underline{\Gamma}}$ *given for* $\Gamma \vdash h : A \circledast\!\!\longrightarrow B$ *by:*

$$(\mathrm{Can}\ c)\ h\ (g \circledast_\delta u) \coloneqq g \circledast_\delta (Fhu) \qquad (g = c_{\underline{A};\delta} u, g \circledast \delta = \gamma, u \in (F\underline{A})_\delta)$$

Consequently, we have an equivariant function $\Gamma \vdash ((\mathrm{Can}\ c)\ h : (\mathrm{Can}\ c)A \circledast\!\!\longrightarrow (\mathrm{Can}\ c)A$. The inclusions $\mathrm{Can}\ c \hookrightarrow F$ *are natural.*

Proof

The crux of the proof is that, by c's naturality, $c_{\underline{B};\delta}(F\,hu) = c_{\underline{A};\delta}(u) = g$. Therefore, $g \circledast_\delta (F\,hu)$ is canonical, and $(\mathrm{Can}\ c)h$ is well-defined. Functoriality follows by routine calculation. For equivariance, take:

$$\begin{aligned} g \circledast_\delta u &\in ((\mathrm{Can}\ c)A)_\gamma && (g = c_{\underline{A};\delta} u, g \circledast \delta = \gamma, u \in (F\underline{A})_\delta) \\ p \circledast_\xi v &\in ((\mathrm{Can}\ c)A)_\gamma && (p = c_{\underline{A};\xi} v, p \circledast \xi = \gamma, v \in (F\underline{A})_\xi) \end{aligned}$$

and $r \in G_\gamma \subseteq \underline{G}$ such that:

$$r \circledast_\gamma g \circledast_\delta u = p \circledast_\xi v$$

We then have:

$$\begin{aligned} (\mathrm{Can}\ c)\ h\ (p \circledast_\xi v) = p \circledast_\xi (F\,h\,v) &\overset{Fh : FA \circledast\!\!\longrightarrow FB}{=} F\,h\,(p \circledast_\xi v) = F\,h\,(r \circledast_\gamma g \circledast_\delta u) = r \circledast_\gamma g \circledast_\delta (F\,h\,u) \\ &= r \circledast_\gamma ((\mathrm{Can}\ c)\ h\ (g \circledast_\delta u)) \end{aligned}$$

and so $(\mathrm{Can}\ c)\ h$ is equivariant.

The naturality of the inclusions $\mathrm{Can}\ c \hookrightarrow F$ amounts to the calculation:

$$(\mathrm{Can}\ c)h(g \circledast_\delta u) \coloneqq g \circledast_\delta (Fhu) = Fh(g \circledast_\delta u)$$

which holds by equivariance of $Fh : FA \circledast\!\!\longrightarrow FB$. ■

The following transformation lets us simultaneously work with canonical elements in a symmetric fibre while carrying around their canonising symmetry.

Proposition F.3. *The transformation* $\gamma : \underline{\Gamma} \vdash \mathsf{canonise}_c := u \mapsto \langle (cu)^{-1}, cu \circledast_\gamma u \rangle : F \Rightarrow G \boxtimes (\mathsf{Can}\, c)$ *is natural.*

Proof

More generally, for every $g \in \underline{G}$, we have transformation:

$$\gamma : \underline{\Gamma} \vdash (g \circledast_\gamma \eta -) : \underline{A} \to G \boxtimes \underline{A} \qquad g \circledast_\gamma \eta a := \langle g, g^{-1} \circledast_\gamma a \rangle = g \overset{G \boxtimes \underline{A}}{\circledast_{g^{-1} \circledast \gamma}} \langle e, g^{-1} \circledast_\gamma a \rangle$$

Given an equivariant family map $\Gamma \vdash f : A \overset{\circledast}{\longrightarrow} B$, we indeed have:

$$\begin{array}{ccc} \underline{A} \ni a & \xmapsto{(g \circledast_\gamma)} & \langle g, g^{-1} \circledast_\gamma a \rangle \in G \boxtimes \underline{A} \\ \Big\downarrow f_\gamma & f \text{ equivariant} = & \Big\downarrow (G \boxtimes f)_\gamma \\ \underline{B} \ni f_\gamma a & \xmapsto[(g \circledast_\gamma)]{} & \langle g, g^{-1} \circledast_\gamma f a \rangle = \langle g, f(g^{-1} \circledast_\gamma a) \rangle \in G \boxtimes \underline{B} \end{array}$$

We now use the canoniser's naturality:

$$\begin{array}{l} \mathsf{canonise}(F f u) := \left\langle (c(F f u))^{-1}, (c(F f u)) \circledast (F f u) \right\rangle \overset{c : F \to \underline{G} \text{ natural}}{=} \left\langle (cu)^{-1}, (cu) \circledast (F f u) \right\rangle \\ \qquad \Big\downarrow F f : FA \overset{\circledast}{\longrightarrow} FB \qquad\qquad \Big\downarrow ((cu)^{-1} \circledast \eta -) : \underline{C} \to G \boxtimes \underline{C} \text{ natural} \\ \qquad = \left\langle (cu)^{-1}, F f \left((cu) \circledast (u) \right) \right\rangle \overset{}{=} F f \left(\left\langle (cu)^{-1}, (cu) \circledast u \right\rangle \right) \\ \qquad =: F f (\mathsf{canonise}\, u) \end{array}$$

as we wanted. ∎

We can use this transformation to express the w.l.o.g. construction:

Lemma F.4. *We have:*

$$\underline{\Gamma} \vdash \mathsf{wlog}(c, f) : F\underline{A} \xrightarrow{\mathsf{canonise}_c} \underline{G \boxtimes ((\mathsf{Can}\, c) A)} \xrightarrow{G \boxtimes f} \underline{G \boxtimes \underline{A}} \xrightarrow{(\circledast)} \underline{A}$$

Proof

Calculate:

$$u \xmapsto{\mathsf{canonise}_c} \left\langle (cu)^{-1}, (cu) \circledast u \right\rangle \xmapsto{G \boxtimes f} \left\langle (cu)^{-1}, f\left((cu) \circledast u\right) \right\rangle \xmapsto{(\circledast)} (cu)^{-1} \circledast f\left((cu) \circledast u\right) = \mathsf{wlog}(c, f)$$

as we wanted. ∎

The next result establishes a universal property for the intermediate step in the flog construction: the extension of a core algebra to an F-algebra.

Theorem F.5 (indexed equivariant algebra representation)**.** *Let* Γ *be a* G*-action,* F Γ*-indexed* G*-strong functor, and* $c : F \Rightarrow \underline{G}$ *an indexed canoniser for* F*, and consider an indexed equivariant core-*F *algebra* $\Gamma \vdash f : \mathsf{Can}\, cA \overset{\circledast}{\longrightarrow} A$*. Then the w.l.o.g. construction is the unique equivariant* F*-algebra* $\Gamma \vdash \mathsf{wlog}(c, f) : FA \overset{\circledast}{\longrightarrow} A$ *extending* f *along the inclusion* $\mathsf{Can}\, c \hookrightarrow F$*.*

Given two equivariant F*-algebras* (A, f) *and* (B, g) *and an equivariant function* $\Gamma \vdash h : A \to B$*, it is an equivariant* F*-homomorphism* $\Gamma \vdash h : (A, f) \overset{\circledast}{\longrightarrow} (B, g)$ *iff it is an equivariant core-*$\mathsf{Can}\, c$ *algebra homomorphism.*

Proof

Assume $f : (\mathsf{Can}\ c)A \circledast\!\!\longrightarrow A$ is a core algebra. By the naturality of Can c, f is a core function for the canoniser $c_A : F\underline{A} \to \underline{G}$. This function is equivariant w.r.t. c_A in the sense of the main paper iff it is equivariant w.r.t. the canoniser c for F in the sense of Definition B.10.

If f is equivariant, then by the Indexed Representation Theorem 3.12, $\mathsf{wlog}(c, f) = \mathsf{wlog}(c_{\underline{A}}, f)$ is equivariant extending f. Covnersely, if $h : FA \circledast\!\!\longrightarrow A$ is an equivariant algebra, then by the representation theorem, $h = \mathsf{wlog}(c_{\underline{A}}, f) = \mathsf{wlog}(c, f)$, as we wanted.

Given another such equivariant algebra $g : FB \circledast\!\!\longrightarrow B$, and an equivariant map $h : A \to B$. The naturality and injectivity of the embedding Can $c \hookrightarrow F$ show that h is an F-algebra homomorphism iff it is a Can c-algebra homomorphism. Indeed, consider the composite square:

$$\begin{array}{ccccc} (\mathsf{Can}\ c)A & \hookrightarrow & FA & \xrightarrow{\ f\ } & A \\ {\scriptstyle (\mathsf{Can}\ c)h}\downarrow & \overset{\text{naturality}}{=} & {\scriptstyle Fh}\downarrow & (**) & \downarrow{\scriptstyle h} \\ (\mathsf{Can}\ c)B & \hookrightarrow & FB & \xrightarrow[\ g\ ]{} & B \end{array} \qquad (*)$$

Our goal is to show that the inner square $(**)$ commutes—h is an F-algebra homomorphism—iff the outer square $(*)$ commutes—h is a Can c-algebra homomorphism. If $(**)$ commutes, the whole diagram commutes. Assume $(*)$ commutes. We have that f, g, and h are equivariant, and the latter also implies that Fh is equivariant. Therefore, the two sides of $(**)$ are equivariant extensions of the outer face $(*)$ along the embedding $(\mathsf{Can}\ c)A \hookrightarrow FA$. By the indexed representation theorem, they coincide, i.e., $(**)$ commutes. ■

We conclude by proving the representation theorem for the indexed flog construction:

Proof

The proof assembles the previous results. First, by the indexed equivariant initiality theorem, the initial algebra is equivariant $\mathsf{roll} : F\boldsymbol{\mu}F \circledast\!\!\longrightarrow \boldsymbol{\mu}F$. By the algebra representation Theorem F.5, its restriction along the embedding Can $c \hookrightarrow F$ is an equivariant algebra $\mathsf{roll} : (\mathsf{Can}\ c)\boldsymbol{\mu}F \to \boldsymbol{\mu}F$.

Now, given any equivariant core algebra $f : (\mathsf{Can}\ c)A \circledast\!\!\longrightarrow A$, by the algebra representation Theorem F.5, its w.l.o.g. construction is an equivariant algebra $g \coloneqq \mathsf{wlog}(c, f) : FA \circledast\!\!\longrightarrow A$. By the equivariant initiality theorem, the unique F-homomorphism out of the initial algebra is equivariant $\mathsf{flog}\ f \coloneqq \mathsf{fold}\ g : \boldsymbol{\mu}F \circledast\!\!\longrightarrow A$. By the algebra representation Theorem F.5, it is also a $(\mathsf{Can}\ c)$-homomorphism.

Conversely, if $h : \boldsymbol{\mu}F \circledast\!\!\longrightarrow A$ is any equivariant core algebra homomorphism, then by Theorem F.5, it is an equivariant F-homomorphism. By the equivariant initiality theorem, $h = \mathsf{fold}\ g = \mathsf{flog}\ f$. ■

G Category theoretic development

The amount of details in the theory of indexed symmetric data types is high, but a few categorical abstractions keep the development relatively clean. We start by recasting the parameterised initiality theorems in this setting. Much of this is known but appears in different places in the work of Fiore et al. [12], and especially in the more recent work with Szamozvancev [10, 11, 28]. We provide references where available.

Lemma G.1. *Let $P, F : C \to C$ be two endofunctors, and $\mathsf{st} : PF \to FP$ a natural transformation. Then F lifts along the forgetful functor $\langle x, a\rangle \mapsto x$ to an endofunctor over P-algebras:*

$$\langle x, a : Px \to x\rangle \mapsto \left\langle Fx, a_F : PFx \xrightarrow{\mathsf{st}} FPx \xrightarrow{Fa} Fx \right\rangle$$

$$\left(\langle x, a\rangle \xrightarrow{h} \langle y, b\rangle\right) \mapsto \left(\langle Fx, a_F\rangle \xrightarrow{Fh} \langle Fy, b_F\rangle\right)$$

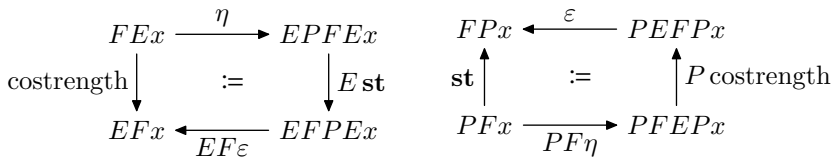
The proof is by direct calculation, see Prop. 3.1.2 in Szamozvancev's thesis. This type of result is known from the early days of the concept, as in Beck's work [4].

When P is a left adjoint $P \dashv E$, we can swap P-strengths for an endofunctor F with E-costrengths for the same endofunctor[8]:

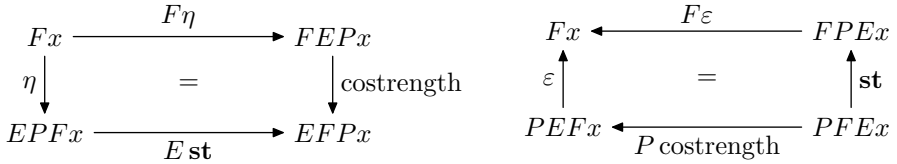
Lemma G.2 (strength-costrength)**.** *Consider endofunctor $P, E, F : C \to C$. Let $P \dashv E$ with unit $\eta : \mathrm{Id} \to EP$ and counit $\varepsilon : PE \to \mathrm{Id}$. TFAE:*

- *P-strengths for F, i.e., natural transformations* $\mathrm{st} : PF \to FP$.
- *E-costrengths for F, i.e., natural transformations* $\mathrm{costrength} : FE \to EF$.

The equivalence is given by the following relationships:



and moreover satisfy:



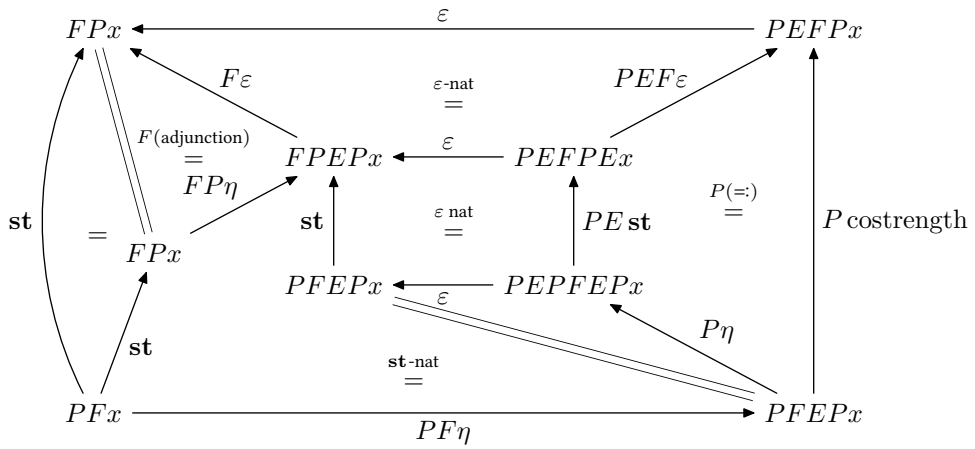
Proof

By duality, we only need to show:

- That costrength is natural.
- That the strength associated with costrength is st.
- The diagram involving the units only (η and $F\eta$).

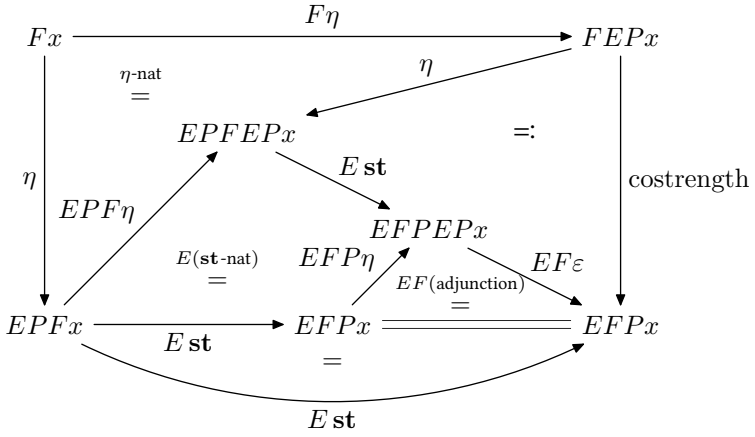
The costrength is natural: it is a composition of natural transformations.

We substitute the definition of costrength into the definition of st and show it recovers st:



The diagram involving units only amounts to straightforward calculation:

[8]We learned this fact from Marcelo P. Fiore in private communication.



■

We now prove a general version of parameterised initiality, already prevalent in Fiore et al.'s work on abstract syntax [10, 11, 13], originally proposed by Bird and Paterson [5].

Definition G.3. Let $P, F : C \to C$ be endofunctors. An initial F-algebra $\mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ is *P-initial* when every F-algebra $f : Fx \to x$ has a unique morphism $\mathrm{pfold}\ f : P\boldsymbol{\mu}F \to x$ satisfying:

$$\begin{array}{ccc} PF\boldsymbol{\mu}F & \xrightarrow{\ \mathbf{st}\ } & FP\boldsymbol{\mu}F \xrightarrow{\ F(\mathbf{pfold}\ f)\ } Fx \\ {\scriptstyle P\,\mathbf{roll}}\downarrow & = & \downarrow{\scriptstyle f} \\ P\boldsymbol{\mu}F & \xrightarrow[\mathbf{pfold}\ f]{} & x \end{array}$$

Theorem G.4 (generalised parameterised initiality)**.** *Let $P \dashv (E : C \to C)$ be an adjunction over a category C, with unit $\eta : \mathrm{Id} \to EP$ and counit $\varepsilon : PE \to \mathrm{Id}$. Every initial algebra $\mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F$, for an endofunctor $F : C \to C$ equipped with a natural transformation $\mathrm{st} : PF \to FP$, is P-initial.*

For completeness, we include the proof of this result in this formulation.

Proof

Let $f : Fx \to x$ be any F-algebra. We equip Ex with an F-algebra structure. It is more direct to work with the costrength:

$$g : FEx \xrightarrow{\ \mathrm{costrength}\ } EFx \xrightarrow{\ Ef\ } Ex$$

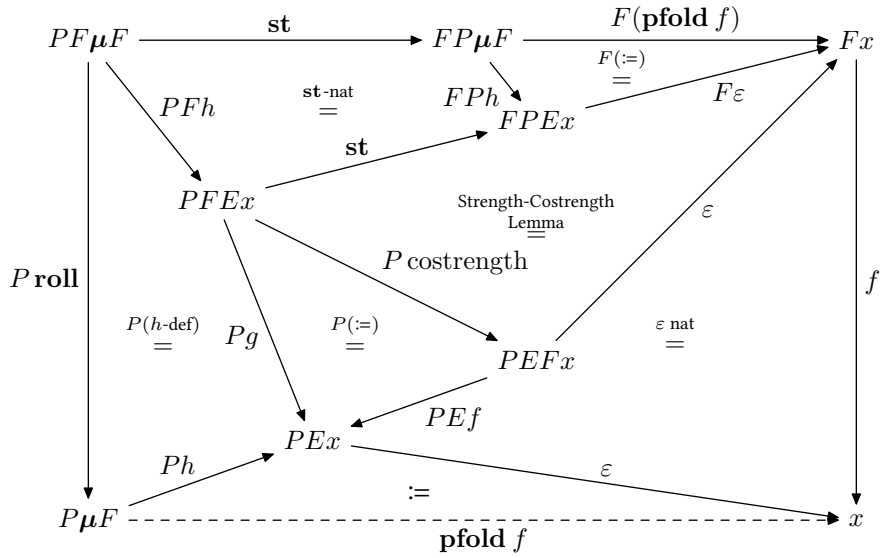
and so, by initiality, we have a unique morphism $h \coloneqq \mathrm{fold}\ g : \boldsymbol{\mu}F \to Ex$ satisfying:

$$\begin{array}{ccc} F\boldsymbol{\mu}F & \xrightarrow{\ Fh\ } & FEx \\ {\scriptstyle \mathbf{roll}}\downarrow & = & \downarrow{\scriptstyle g} \\ \boldsymbol{\mu}F & \xrightarrow[h]{} & Ex \end{array}$$

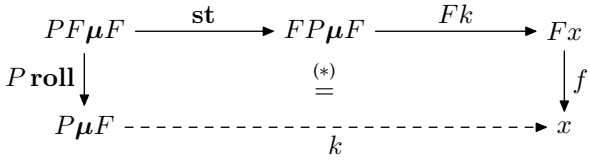
Now take the mate of this morphism:

$$\mathrm{pfold}\ f : P\boldsymbol{\mu}F \xrightarrow{\ Ph\ } PEx \xrightarrow{\ \varepsilon\ } x$$

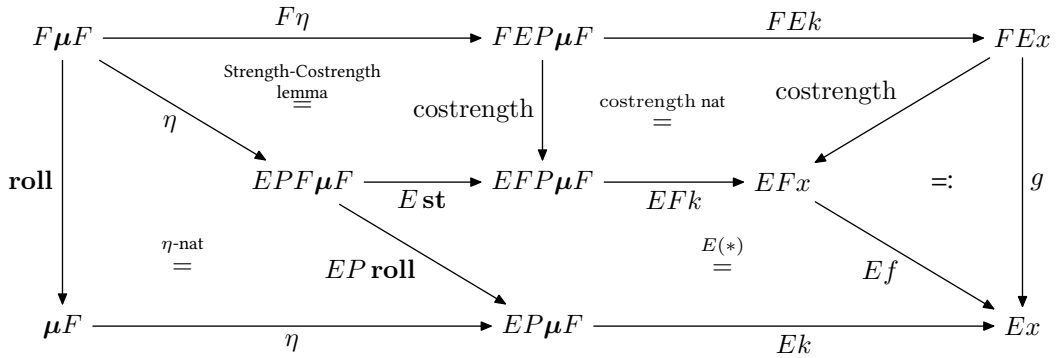
This morphism satisfies the required property:



Now consider any morphism $k : P\boldsymbol{\mu}F \to x$ making the following diagram commute:



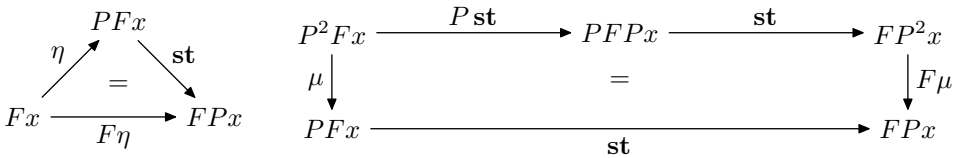
We will show that its mate must be h, i.e.: $h : \boldsymbol{\mu}F \xrightarrow{\eta} EP\boldsymbol{\mu}F \xrightarrow{Ek} Ex$. Calculate:



By universality, h is the mate of k, and so $k = \mathbf{pfold}\ f$. ■

Therefore, if an endofunctor F over $\underline{\Gamma}$-indexed families has a $(G \circledast -)$-strength, then its initial algebras are both $(G \circledast -)$-initial and $(G \circledast G \circledast -)$-initial.

Definition G.5. Let $P = \langle P : C \to C, \eta, \mu \rangle$ be a monad, and $F : C \to C$ an endofunctor over C. A *P-strength for F* is a natural transformation $\mathrm{st} : PF \to FP$ satisfying:



This notion is sometimes called: a *distributive law* between a monad and a functor; a *module*; or a *monad endomorphism.*

The following notion generalises equivariant F-algebras as algebras of the lifting of F to P-algebras:

Definition G.6. Let $P = \langle P : C \to C, \eta, \mu \rangle$ be a monad over C, and $F : C \to C$ an endofunctor equipped with a P-strength $\mathrm{st}^P : PF \to FP$. A *$P$-linear F-algebra* $\langle x, f, a \rangle$ consists of:

- an F-algebra $f : Fx \to x$; and
- a P-algebra $a : Pa \to a$;

such that f is P-linear $f : \langle Fx, a_F \rangle \to \langle x, a \rangle$. A *homomorphism* $h : \langle x, f, a \rangle \to \langle y, g, b \rangle$ of P-linear F-algebras is a morphism $h : x \to y$ that is simultaneously an F-algebra homomorphism and a P-algebra homomorphism.

Since a $(G \boxtimes -)$-algebra is precisely a Γ-indexed G-action and a $(G \boxtimes -)$-linear map between them is precisely an equivariant function, we have that the $(G \boxtimes -)$-linear F-algebras are precisely the equivariant F-algebras. The capstone of this development is the following theorem:

Theorem G.7 (general representation theorem)**.** *Let $P = \langle P : C \to C, \eta, \mu \rangle$ be a monad over C, and $F : C \to C$ an endofunctor equipped with a P-strength* $\mathrm{st} : PF \to FP$*. If an initial F-algebra* $\mathrm{roll} : F\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ *is both P-initial and P^2-initial, then it is also the initial P-linear F-algebra. Its P-algebra structure is given by P-initiality as* $\mathrm{alg} \coloneqq \mathrm{pfold}\,\mathrm{roll} : P\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ *and the unique F-homomorphism* $\mathrm{fold}\, f : \boldsymbol{\mu}F \to x$ *into any other P-linear F-algebra $\langle x, f, a \rangle$ is P-linear.*

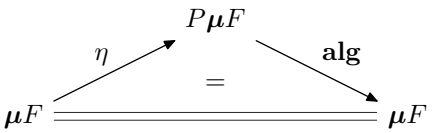
We chose this formulation as its proof follows our proof in the simply-typed case more closely, which itself follows the argument in abstract syntax with binding and substitution [12]. Our situation is semantically well-behaved, and there are alternative theorems we could use. For example, since Γ-indexed G-actions are $E \coloneqq (G \multimap -)$-algebras, we could use Szamozvancev's Prop. 3.3.1 instead [11]. Whichever formulation we use, it is clear it proves the Indexed Representation Theorem B.5.

Proof

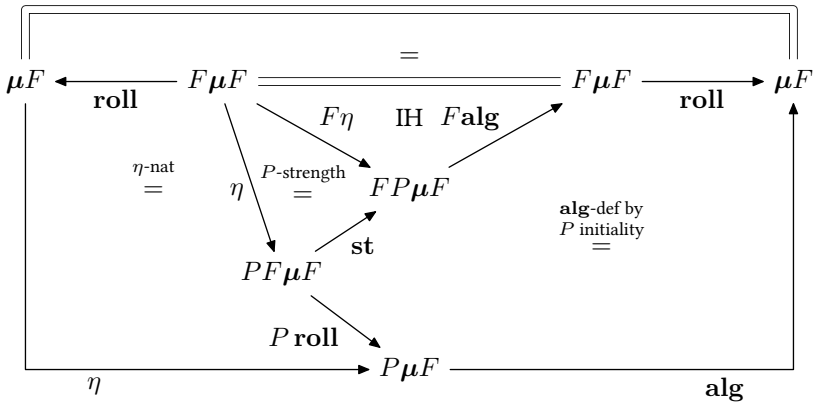
We show that:

- $\mathrm{alg} : P\boldsymbol{\mu}F \to \boldsymbol{\mu}F$ is a P-algebra;
- roll is P-linear by the defining property of P-initiality;
- each fold f is P-linear.

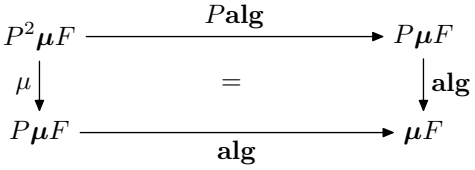
We show the P-algebra unit axiom:



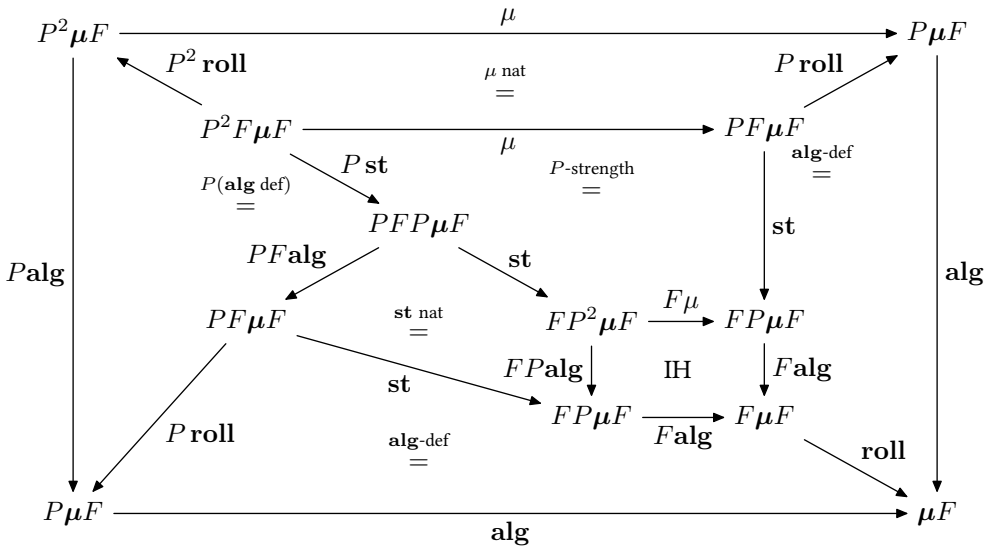
by recourse to mere initiality:



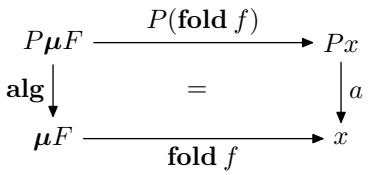
We show the P-algebra multiplication axiom:



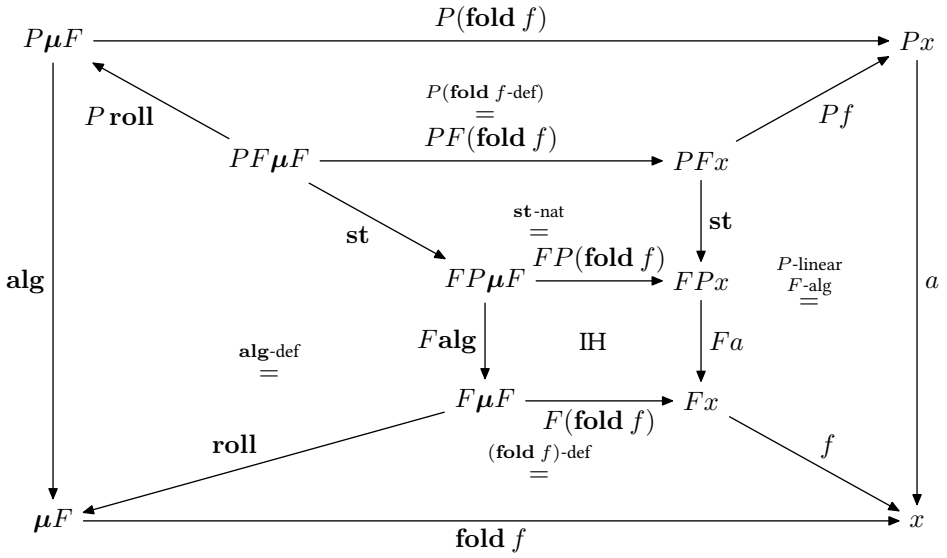
by recourse to P^2 initiality:



Finally, take any P-linear F-algebra $\langle x, f, a\rangle$. By initiality there is at most one P-linear F-homomorphism, and exactly one F-homomorphism, into it. We show:



by recourse to P-initiality:



concluding the proof. ■