Normalization by Evaluation with Free Extensions (Extended Abstract)

Nathan Corbyn
Oxford University
England

Ohad Kammar
Sam Lindley
University of Edinburgh
Scotland

Nachiappan Valliappan
Chalmers University of Technology
Sweden

Jeremy Yallop
University of Cambridge
England

1 Introduction

Normalization by Evaluation (NbE), is a normalization technique that normalizes terms by evaluating them in a suitable semantic model. The model identifies all equivalent terms, and is accompanied by a *reification* function which reconstructs normal forms by choosing a canonical representative for each equivalence class of terms.

NbE provides a unified treatment of the implementation and verification of normalization algorithms. NbE is remarkably versatile and modular. However, constructing an NbE model for a given calculus and extending it to support new features requires a certain amount of ingenuity [4, 5, 19]. We report on our ongoing development of a systematic approach to NbE for functional programming languages that is robust to a broad class of modifications and extensions.

Consider, for example, normalization for the following extension of the simply-typed lambda calculus (STLC) with natural numbers and multiplication ($\mathrm{STLC}_{\mathbb{N}*}$):

Numbers $j\ k \in \mathrm{Nat}$ *Types* $a\ b ::= a \to b \mid \mathbb{N}$
Variables $x\ y \in X$ *Terms* $t\ u ::= x \mid \lambda x.t \mid t\ u \mid \underline{n} \mid t * u$

The *lifting* operator $k \mapsto \underline{k}$ embeds a natural number k as a literal, whilst the $*$ operator multiplies two terms of type $\mathbb{N}$. A complete normalization algorithm should simplify expressions such as $\lambda x.(\underline{2} * \underline{3}) * x$ and $\lambda x.(\lambda y.\underline{2} * y)\ (x * \underline{3})$ to $\lambda x.\underline{6} * x$. Whilst a naive normalizer suffices to reduce the former expression, the latter requires some care as the β-reduced expression $\lambda x.\underline{2} * (x * \underline{3})$ must first be reordered to $\lambda x.(\underline{2} * \underline{3}) * x$ before the literals can be multiplied.

Although NbE for STLC is well-established [6, 9], its extension with an interpretation of natural numbers requires a careful consideration of the calculus, its equations, and the desired normal forms. What should the normal forms of terms of type $\mathbb{N}$ be? How do we construct an NbE model that supports reification of these normal forms? How should the model be adapted if we also wish to include addition?

Yallop et al. [20] show that normalization algorithms for a variety of algebraic structures, such as commutative monoids, abelian groups, rings, and distributive lattices, can be constructed systematically as *free extensions* (*frex*) of algebras. Frex's algorithm uses the fact that natural numbers with multiplication form a commutative monoid, and normalizes first-order terms, e.g., $\underline{2} * (x * \underline{3})$ to $\underline{6} * x$.

We present our first steps in achieving NbE using the frex approach. We show how to systematically define a normalization algorithm for a higher-order functional language extended with an algebraic structure decomposing the normalization algorithm into a standard NbE algorithm for the higher-order functional language and a frex for the first-order algebraic structure. We use two concrete examples. In each case we extend a standard NbE algorithm with a frex of an arbitrary commutative monoid and show that normalization is retained regardless of the monoid's instantiation. First we take the higher-order functional language to be simply-typed lambda calculus, generalizing $\mathrm{STLC}_{\mathbb{N}*}$. Second we take the higher-order functional language to be a monadic information-flow security calculus. The NbE algorithm does not rely on the theory of commutative monoids, merely on the defining property of the frex, suggesting a generic NbE algorithm for arbitrary algebraic structures. We conclude by outlining ongoing work reformulating NbE via frex.

2 Frex: Free Extensions of Algebras

When normalizing expressions over an algebraic structure, say the commutative monoid $\mathbb{N}_*$, we can directly evaluate *static* terms, those built entirely from elements of the structure, e.g. $2 * (5 * 3)$. The challenge is to normalize *partially-static* terms, those containing both bound variables and constants, e.g. $2 * (x * 3)$. Whilst valid identities over static terms, e.g. $2 * (5 * 3) = 30$, hold *definitionally* through evaluation, a similar identity of partially-static terms, e.g. $2 * (x * 3) = x * 6$, may not. This difficulty can be removed by avoiding pure syntax trees and instead considering terms modulo provable equivalence in equational logic [3, 11, 20].

Free extensions, originating in universal algebra, capture this situation abstractly. Free extensions characterize normal forms of terms of an algebraic theory up to a universal property, yielding systematic approaches for specifying and constructing NbE models for equational theories.

Free Extensions. The key observation underpinning the abstract characterization of free extensions is that the behaviour of a partially-static term is uniquely determined by a choice of *environment*. Explicitly, given an environment $e : X \to \mathbb{N}$, there is a *unique* way to structurally evaluate terms under this environment. For example, supposing $e(x) = 5$ and $e(y) = 1$, evaluate $2 * (x * 3) * y$ to 30. This situation is similar to the existence of a unique homomorphic extension $\tilde{e} : \mathrm{Free}(X) \to \mathbb{N}$ of e from the free commutative monoid over X into $\mathbb{N}$, but also accounts for the literals $\underline{n}$. This observation generalizes, yielding an abstract definition, applicable to an arbitrary equational theory Θ such as commutative monoids, abelian groups, rings, and so on.

Given a model $A \in \mathrm{Alg}(\Theta)$, e.g. commutative monoid, and a set of variables X, the free extension of A by X, denoted $A[X]$, is a model of Θ equipped with homomorphic insertions $i_A : A \to A[X] \leftarrow \mathrm{Free}(X) : i_X$, with the following property. For every model $W \in \mathrm{Alg}(\Theta)$, homomorphism $h : A \to W$ and environment function $e : X \to W$, there is a unique homomorphism, *match*, which evaluates elements of $A[X]$ in W, and makes the following triangles commute:

$$\begin{tikzcd} A[X] \arrow[dr, dashed, "\exists! match"'] & \mathrm{Free}(X) \arrow[l, "i_X"'] \arrow[d, "\tilde{e}"] \\ A \arrow[u, "i_A"] \arrow[r, "h"'] & W \end{tikzcd}$$

The abstract property postulates that the frex is the category-theoretic coproduct of A with this free algebra.

There is an abstract way to construct the frex $A[X]$, which we denote by $\mathrm{Frex}(A, X)$, by quotienting the term algebra over $A + X$, which adds a constant representing each literal in A. The quotienting equivalence relation is generated by the equations in Θ, such as associativity and commutativity, and additionally all evaluation equations, such as $\underline{n} + \underline{m} = \underline{n + m}$. We cannot use this quotient to directly compute normal forms. However, every two objects possessing the same universal property are canonically isomorphic. Thus there is a canonical isomorphism for *every* frex $A[X]$:

$$\begin{tikzcd} A[X] \arrow[r, shift left, "reify"] \arrow[r, phantom, "\cong"] & \mathrm{Frex}(A, X) \arrow[l, shift left, "eval"] \end{tikzcd}$$

This isomorphism identifies equivalence classes of syntax with semantic objects representing them. If $A[X]$ is effective, meaning $A[X]$ has computable equality, algebraic operations and *match* function, then: both *reify* and *eval* are computable too, are given generically through the frex interface $A[X]$, and moreover *reify* chooses a representative normal form for each semantic representation. The frex mantra is therefore: to obtain an NbE model for the first-order term language of an algebraic structure A, it suffices to focus human ingenuity on obtaining an effective construction of a frex for A.

Example: First-order NbE via Frex. Let Tm be $\mathrm{STLC}_{\mathbb{N}_*}$'s first-order fragment: variables, lifting and $(*)$, e.g. $\underline{2} * (x * \underline{3})$.

First, construct the frex for a commutative monoid $M = (|M|, \epsilon_M, \oplus_M)$, by a set of variables X. This frex, $M[X]$, has as its carrier the set $|M| \times \mathrm{MultiSet}(X)$, its multiplication is $(a, s_1) \oplus (b, s_2) = (a \oplus_M b, s_1 \cup s_2)$, and its unit is $(\epsilon_M, \varnothing)$.

The frex interface, which satisfies the frex axioms, is the following inclusions i_M, i_X and homomorphism *match*:

$$i_M a := (a, \varnothing) \qquad match(h, e)(a, V) :=$$
$$i_X x := (\epsilon_M, \{x\}) \qquad h(a) \oplus_W \left(\bigoplus_{x \in V} e(x)\right)$$

where $\bigoplus$ iterates over the multiset V, reducing in W.

The associated NbE model for Tm normalizes the equivalence generated by equational logic of commutative monoids and the evaluation equations, which we denote by $\approx$. It comprises the equations for associativity, commutativity and unitality of $*$, as well as the two evaluation equations $\underline{n} + \underline{m} = \underline{n + m}$ and $\underline{\epsilon} = \epsilon$. To exhibit such an NbE model:

- construct a commutative monoid N;
- define an effective homomorphism $eval : \mathrm{Tm}/{\approx} \to N$;
- define an effective homomorphism $reify : N \to \mathrm{Tm}/{\approx}$;
- show that *reify* retracts *eval* (i.e., $reify \circ eval \approx 1$).

The normalization homomorphism, $norm : \mathrm{Tm}/{\approx} \to \mathrm{Tm}/{\approx}$ is the composite $reify \circ eval$, congruence means $\approx$-equivalent terms have equal normal forms, and the retraction ensures normalization reflects $\approx$, that is: $norm(t) = norm(u) \Rightarrow t \approx u$.

Taking N to be the frex $\mathbb{N}_*[X]$, we obtain all of the above data immediately from the fact that N is a frex. By construction, $\mathrm{Tm}/{\approx}$ is the abstract frex $\mathrm{Frex}(\mathbb{N}_*, X)$, and therefore the induced canonical isomorphism yields the required maps *eval* and *reify*, noting that each isomorphism is a retraction.

3 Normalization by Evaluation with Frex

We extend NbE for a higher-order language with a first-order algebraic structure's frex. To illustrate this, we generalize our running example from $\mathbb{N}_*$ to an arbitrary commutative monoid M, called STLC_M. We replace the type $\mathbb{N}$ by $\mathbb{M}$, and the set of literals Nat by the carrier set $|M|$. We characterize the normal forms of STLC_M by extending the usual mutually inductive definitions of normal and neutral forms with a new normal form. This normal form, $\underline{k} * n_1 * \ldots * n_j$, represents a multiplication that begins with a literal followed by a sequence of neutrals of type $\mathbb{M}$ ordered by an arbitrary fixed total order on terms.

$$\begin{array}{ll} \textit{Neutrals } n & ::= x \mid n\ m \\ \textit{Normal forms } m & ::= \lambda x.m \mid \underline{k} * n_1 * \ldots * n_j \quad (n_i \le n_{i+1}) \end{array}$$

An NbE model is a suitable model (here, a cartesian-closed category) with *eval* and *reify* such that *reify* retracts *eval*. We extend the standard interpretation of function types in an NbE model for STLC with an interpretation of $\mathbb{M}$.

$$[\![\mathbb{M}]\!] := M[\mathrm{Ne}(\mathbb{M})] \qquad [\![a \to b]\!] := [\![a]\!] \Rightarrow [\![b]\!]$$

Specifically, we interpret $\mathbb{M}$ by the free extension of M with the set $\mathrm{Ne}(\mathbb{M})$ of neutral terms of type $\mathbb{M}$. Taking the free extension with the set of neutrals, as opposed to variables, is the *key* insight that enables NbE for STLC_M using frex. Under this type interpretation, we evaluate STLC terms as usual, and lifting and $*$ using the frex interface.

$$\begin{array}{ll} eval\ x & \gamma := lookup\ x\ \gamma \\ eval\ (\lambda x.t) & \gamma := \lambda v.eval\ t\ (\gamma\ [x \mapsto v]) \\ eval\ (app\ t\ u) & \gamma := (eval\ t\ \gamma)\ (eval\ u\ \gamma) \\ eval\ \underline{n} & \gamma := i_{M[\mathrm{Ne}(\mathbb{M})]}\ n \\ eval\ (t * u) & \gamma := (eval\ t\ \gamma) \oplus_{M[\mathrm{Ne}(\mathbb{M})]} (eval\ u\ \gamma) \end{array}$$

We reify as usual, by mutual type-induction with a *reflect* operation coercing neutral terms into their semantic values:

$$
\begin{array}{ll}
\mathit{reify}_{a \to b} & f := \lambda x.\mathit{reify}_b\,(f\,(\mathit{reflect}_a\,x)) \\
\mathit{reify}_{\mathbb{M}} & v := \mathit{match}\,(\_\,,\mathit{ne})\,v \\
\mathit{reflect}_{a \to b} & n := \lambda v.\,\mathit{reflect}_b\,(n\,(\mathit{reify}_a\,v)) \\
\mathit{reflect}_{\mathbb{M}} & n := i_{\mathrm{Ne}(\mathbb{M})}\,n
\end{array}
$$

For normal forms of type $\mathbb{M}$, we reify using the frex's *match*, the lifting operator $k \mapsto \underline{k}$ that embeds an element k as a literal in normal form, and a function *ne* that embeds neutrals. We define *reflect* using the frex's insertion function $i_{\mathrm{Ne}(\mathbb{M})}$.

To show that *reify* retracts *eval*, however, we are forced into the standard logical relations based argument that is typical in NbE literature [1]. The reason: this frex is unaware of the higher-order constructs, a point revisited in §5.

4 Example: Information-Flow Control

We now extend the NbE algorithm to an information-flow control (IFC) calculus that uses a commutative monoid M of security levels. The unit ϵ_M denotes the least security (or *public*) level and the operation $\oplus_M$ *joins* two levels by computing their least-upper bound. This calculus extends STLC_M with a type constructor T, and the type $\mathbb{M}$ is to be read as the type of security levels. A term of type $T\ a$ represents a computation that associates (or *labels*) a value of type a with a security level and is reminiscent of the monads used for dynamic and staged IFC [16, 18].

The terms and their normal forms are defined as follows.

$$
\begin{array}{lcl}
\mathit{Types}\ a\ b & ::= & \ldots \mid T\ a \\
\mathit{Terms}\ t\ u\ t_l\ t_{l'} & ::= & \ldots \mid \mathit{return}\ t \mid t \gg\!\!= \lambda x.u \mid \mathit{raise}\ t_l\ u \\
\mathit{Normal\ forms}\ m\ m_l & ::= & \ldots \mid \mathit{label}\ m_l\ m \mid n \gg\!\!= \lambda x.m
\end{array}
$$

(the definition of neutral forms is unchanged.) The operation *return* labels a value with public level ϵ_M, and *raise* raises the level of a computation u with level (term) t_l. A term t of type a can be labeled with level l as $\mathit{raise}\ \underline{l}\ (\mathit{return}\ t)$. The term $t \gg\!\!= \lambda x.u$ joins the level of u and t. The monadic normal forms are as usual $n_1 \gg\!\!= \lambda x_1.n_2 \gg\!\!= \ldots \gg\!\!= \lambda x_j.(\mathit{label}\ m_l\ m)$ where $\mathit{label}\ m_l\ m$ is a combination of *return* and *raise* as $\mathit{raise}\ m_l\ (\mathit{return}\ m)$. This shape forces *raise* to be propagated down to the end of the $\gg\!\!=$-chain, where it is fused with other applications of *raise*—as justified by the following equations.

$$
(\mathit{raise}\ l\ t) \gg\!\!= \lambda x.u \ \approx\ t \gg\!\!= \lambda x.(\mathit{raise}\ l\ u)
$$

$$
\mathit{raise}\ t_l\ (\mathit{raise}\ t_{l'}\ u) \ \approx\ \mathit{raise}\ (t_l * t_{l'})\ u \qquad u \ \approx\ \mathit{raise}\ \underline{\epsilon_M}\ u
$$

These equations are imposed in addition to the equations of STLC_M and the standard monadic equations for T.

We inductively define an indexed set $T'\ A$, using the set X_a of variables of type a in the current context:

$$
\frac{p : [\![\mathbb{M}]\!] \quad q : A}{\mathit{label}'\ p\ q\ :\ T'\ A} \qquad \frac{n : \mathrm{Ne}(T\ a) \quad f : X_a \to T'\ A}{\mathit{bind}'\ n\, f\ :\ T'\ A}
$$

The family T' forms a monad, and is akin to the ones used by Ahman and Staton [2] and Tomé Cortiñas and Valliappan [17] to normalize monadic computations. We extend the interpretation of types in STLC_M by $[\![T\ a]\!] := T'\ [\![a]\!]$. Evaluation and reification then extend to the monadic fragment in a straightforward manner [17].

Remarkably, we have extended the NbE algorithm for STLC_M seamlessly to the inclusion of a monad (that interacts with security levels in a meaningful way) using the standard treatment of NbE for monads.

5 Normalization by Evaluation via Frex

Thus NbE can leverage frex productively (§3-4), but for first-order languages, NbE can itself be achieved *via* frex (§2). We are extending this to higher-order languages in two ways.

First, programmatically, we implemented an OCaml frex interface for STLC with sums and products. It exposes the model structure (λ-abstraction; application; case-splitting; etc.) and the insertions and *match* function, combining both NbE [8, 12, 14] and term representation [7, 15] techniques.

Second, semantically, we go beyond equational logic and ordinary algebraic structures, and use *generalized algebraic theories* (GATs) [10]. Ordinarily, terms have simple contexts — sets of variables, partitioned into sorts — while GAT contexts are dependent. GAT models possess a rich semantic structure, including the existence of free models.

For example, the GAT of categories has a simple sort Obj for objects, and a dependent sort $a, b : \mathrm{Obj} \vdash \mathit{Hom}\ (a, b)$. Its algebraic operations include identities and composition:

$$
a : \mathrm{Obj} \vdash \mathit{id}\ a : \mathrm{Hom}(a, a)
$$

$$
a, b, c : \mathrm{Obj}, g : \mathrm{Hom}(b, c), f : \mathrm{Hom}(a, b) \vdash g \circ f : \mathrm{Hom}(a, c)
$$

and further equations for associativity of composition and neutrality of identities. Its models are the (small) categories.

The key is to parameterise the frex by a model and a context, instead of a set of variables. As an example, we extend the category FinSet of hereditarily finite sets and functions with an object $\mathbb{S} : \mathrm{Obj}$ and two morphisms $\top, \bot : \mathrm{Hom}(\mathbb{1}, \mathbb{S})$. In GAT language, take the context $\Gamma := \mathit{one} : \mathrm{Obj}$; the environment $\theta : \mathrm{Free}(\Gamma) \to \mathrm{FinSet}$ mapping *one* to the singleton set $\mathbb{1}$; and Δ the context extension of Γ with $\top, \bot : \mathrm{Hom}(\mathit{one}, \mathbb{S})$. We define the frex $A[\ \Gamma,\ \Delta\ ]\ \theta$ as the push-out:

$$
\begin{tikzcd}
\mathrm{Free}(\Gamma) \arrow[r, "\mathrm{Free}(\mathrm{weaken})"] \arrow[d, "\theta"'] & \mathrm{Free}(\Gamma,\ \Delta) \arrow[d] \arrow[dl, phantom, "\ulcorner", very near end] \\
A \arrow[r] & A[\ \Gamma,\ \Delta\ ]\ \theta
\end{tikzcd}
$$

The idea comes from considering frex as a two-argument functor, compatible with the operations on the extending structure. We have constructed the free extension of a category as alternating composable sequences of freely added and injected morphisms, and proved it satisfies this definition of the generalised frex. We also proved that this frex is not the coproduct of the original category with any other category, and so the GAT frex is a strict generalisation of its equational specialisation. We plan to extend this account to more sophisticated theories: monoidal, cartesian, and cartesian-closed categories, thus expressing NbE via frex [13].

Acknowledgements. Supported by an Engineering and Physical Sciences Research Council (EPSRC UK) Industrial CASE studentship, and a Royal Society University Research Fellowship. We thank James McKinna and Sean K. Moss for interesting discussions and helpful suggestions.

References

[1] Andreas Abel. 2013. Normalization by evaluation: Dependent types and impredicativity. *Habilitation. Ludwig-Maximilians-Universität München* (2013).

[2] Danel Ahman and Sam Staton. 2013. Normalization by Evaluation and Algebraic Effects. In *MFPS (Electronic Notes in Theoretical Computer Science, Vol. 298)*. Elsevier, 51–69.

[3] Guillame Allais, Edwin Brady, Nathan Corbyn, Ohad Kammar, and Jeremy Yallop. 2022. Frex: dependently-typed algebraic simplification. (2022). draft.

[4] Guillaume Allais, Conor McBride, and Pierre Boutillier. 2013. New equations for neutral terms: a sound and complete decision procedure, formalized. In *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-typed Programming*. 13–24.

[5] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. 2001. Normalization by Evaluation for Typed Lambda Calculus with Coproducts. In *LICS*. IEEE Computer Society, 303–310.

[6] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. 1995. Categorical reconstruction of a reduction free normalization proof. In *International Conference on Category Theory and Computer Science*. Springer, 182–199.

[7] Robert Atkey, Sam Lindley, and Jeremy Yallop. 2009. Unembedding domain-specific languages. In *Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell, Haskell 2009, Edinburgh, Scotland, UK, 3 September 2009*, Stephanie Weirich (Ed.). ACM, 37–48. https://doi.org/10.1145/1596638.1596644

[8] Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. 2004. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, Neil D. Jones and Xavier Leroy (Eds.). ACM, 64–76. https://doi.org/10.1145/964001.964007

[9] Ulrich Berger and Helmut Schwichtenberg. 1991. An Inverse of the Evaluation Functional for Typed lambda-calculus. In *LICS*. IEEE Computer Society, 203–211.

[10] John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic* 32 (1986), 209–243. https://doi.org/10.1016/0168-0072(86)90053-9

[11] Nathan Corbyn. 2021. *Proof Synthesis with Free Extensions in Intensional Type Theory*. Technical Report. University of Cambridge. MEng Dissertation.

[12] Marcelo P. Fiore and Ola Mahmoud. 2010. Second-Order Algebraic Theories - (Extended Abstract). In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6281)*, Petr Hlinený and Antonín Kucera (Eds.). Springer, 368–380. https://doi.org/10.1007/978-3-642-15155-2_33

[13] J. M. E. Hyland. 2017. Classical lambda calculus in modern dress. *Math. Struct. Comput. Sci.* 27, 5 (2017), 762–781. https://doi.org/10.1017/S0960129515000377

[14] Chantal Keller and Thorsten Altenkirch. 2010. Hereditary Substitutions for Simple Types, Formalized. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Mathematically Structured Functional Programming, MSFP@ICFP 2010, Baltimore, MD, USA, September 25, 2010*, Venanzio Capretta and James Chapman (Eds.). ACM, 3–10. https://doi.org/10.1145/1863597.1863601

[15] Conor McBride and James McKinna. 2004. Functional pearl: I am not a number-I am a free variable. In *Proceedings of the ACM SIGPLAN Workshop on Haskell, Haskell 2004, Snowbird, UT, USA, September 22-22, 2004*, Henrik Nilsson (Ed.). ACM, 1–9. https://doi.org/10.1145/1017472.1017477

[16] Deian Stefan, Alejandro Russo, John C. Mitchell, and David Mazières. 2011. Flexible dynamic information flow control in Haskell. In *Proceedings of the 4th ACM SIGPLAN Symposium on Haskell, Haskell 2011*. 95–106.

[17] Carlos Tomé Cortiñas and Nachiappan Valliappan. 2019. Simple Noninterference by Normalization. In *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*. 61–72.

[18] Nachiappan Valliappan, Robert Krook, Alejandro Russo, and Koen Claessen. 2020. Towards secure IoT programming in Haskell. In *Haskell@ICFP*. ACM, 136–150.

[19] Nachiappan Valliappan, Alejandro Russo, and Sam Lindley. 2021. Practical normalization by evaluation for EDSLs. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell*. 56–70.

[20] Jeremy Yallop, Tamara von Glehn, and Ohad Kammar. 2018. Partially-static data as free extension of algebras. *Proc. ACM Program. Lang.* 2, ICFP (2018), 100:1–100:30.