# The encode-decode method, relationally

# The Encode-Decode Method, Relationally

James McKinna[1] and Fredrik Nordvall Forsberg[2]

[1] School of Informatics, University of Edinburgh
[2] Dept. of Comput. and Inform. Sci., University of Strathclyde
james.mckinna@ed.ac.uk, fredrik.nordvall-forsberg@strath.ac.uk

**Abstract**

In Homotopy Type Theory (HoTT), the ‘encode-decode’ method makes heavy use of explicit recursion over higher inductive types to construct, and prove properties of, homotopy equivalences. We argue for the classical separation between specification and implementation, and hence for using relations to track the graphs of encode/decode functions. Our aim is to isolate the technicalities of their definitions, arising from higher path constructors, from their properties. We describe our technique in the calculation of $\pi_1(\mathbb{S}^1)$, and comment on its applicability in the current AGDA implementation of HoTT.

**Introduction** Burstall’s seminal analysis [2] of how to prove properties of fold (defined by structural *recursion* on lists) proceeds by first deriving a proof principle for fold by structural *induction*, and then applying it. This avoids relying on the *definition* of fold to achieve coincidences in proof by list induction, by encapsulating the recursive call structure once and for all. This achieves separation of concerns between a *concrete* implementation and its *abstract* specification in terms of the induction principle for its *graph.* This insight has been rediscovered, applied, and engineered many times over; in type theory most recently in the `Function` [1] and `Program` [8] extensions to COQ, and in the design and implementation of *views* in EPIGRAM [6].

The *encode-decode* method, pioneered by Licata and Shulman in their proof of $\pi_1(\mathbb{S}^1) \simeq \mathbb{Z}$ [4], heavily exploits definition by structural recursion and proof by induction on the newly-introduced Higher Inductive Types (HITs) of Homotopy Type Theory (HoTT [9]), where inductive types introduce not only new *term* constructors, but also those of (higher) *paths.* Below we explore Burstall’s technique in this context, and its implementation in systems such as AGDA.

**Graphs of (recursively-defined) functions** A function $f : \Pi_{x:A}B(x)$ gives rise to a *graph* relation $F : \Pi_{x:A}\Pi_{y:B(x)}\mathsf{Type}$ trivially via $Fxy \equiv y = f(x)$. But if $f$ is defined *recursively*, then $F$ may be given *inductively*, with base cases of the definition of $f$ corresponding to axioms in that of $F$, and step cases to inference rules, with recursive calls of $f$ tracked by inductive premises involving $F$. Such an $F$ then yields an induction principle for $f$, and proofs that $F$ is *sound*, $\mathsf{snd}_f(F) : \Pi_{x:A}\Pi_{y:Bx}Fxy \to y = f(x)$, and *complete*, $\mathsf{cmp}_f(F) : \Pi_{x:A}Fx(f(x))$, for the graph relation. Turning this around, *any* choice of (inductive) family $G$ satisfying $\mathsf{snd}_f(G)$ and $\mathsf{cmp}_f(G)$ constitutes an adequate representation of the graph of $f$, and may be used in proofs about $f$, by virtue of establishing that $F$ and $G$ are *extensionally equivalent* ($\Leftrightarrow$) as relations.

**The encode-decode equivalence, revisited** The circle $\mathbb{S}^1$ is given as a HIT with 0-constructor $\mathsf{base} : \mathbb{S}^1$ and 1-constructor $\mathsf{loop} : \mathsf{base} = \mathsf{base}$. Its *covering space* $C(x)$ is defined by higher recursion on $x : \mathbb{S}^1$, by $C(\mathsf{base}) \equiv \mathbb{Z}$ and a path $\mathbb{Z}{=}\mathbb{Z}$ obtained via the Univalence Axiom from the automorphism of $\mathbb{Z}$ induced by $\mathsf{succ} : \mathbb{Z} \to \mathbb{Z}$. Then the path space $P(x) \equiv \mathsf{base} = x$ over $x : \mathbb{S}^1$ is shown homotopy equivalent to $C(x)$ via functions $\mathsf{encode} : \Pi_{x:\mathbb{S}^1}\Pi_{p:P(x)}C(x)$ (given by the action $p^\star(0) \equiv \mathsf{transport}_{C(\_)}p(0)$ of paths $p$ on $0 : \mathbb{Z}$) and $\mathsf{decode} : \Pi_{x:\mathbb{S}^1}\Pi_{c:C(x)}P(x)$, defined by higher induction on $\mathbb{S}^1$: the function $\mathsf{decode}_{\mathsf{base}}$ maps $z : \mathbb{Z}$ to the iterated path $\mathsf{loop}^z$, and one has to show this definition respects the action $\mathsf{loop}^\star$ of $\mathsf{loop}$.

Our approach requires the choice of two relations, $\mathsf{Encode}_x\, p\, c$ and $\mathsf{Decode}_x\, p\, c$, which we show sound and complete for $\mathsf{encode}_x$ and $\mathsf{decode}_x$, together with a proof of the homotopy equivalence, which amounts to proving: $\Pi_{x:\mathbb{S}^1}\Pi_{p:P(x)}\Pi_{c:C(x)}\mathsf{Encode}_x\, p\, c \Leftrightarrow \mathsf{Decode}_x\, c\, p$ (†). For $\mathsf{Encode}_x\, p\, c$ we take simply the graph of $\mathsf{encode}_x$, while for $\mathsf{Decode}_{\mathsf{base}}\, z\, p$ we take the inductively defined graph of $z \mapsto \mathsf{loop}^z$. The main subtlety lies in proving $\mathsf{cmp}_{\mathsf{decode}_x}(\mathsf{Decode}_x)$. As above, one must show that $\mathsf{Decode}_{\mathsf{base}}$ respects the $\mathsf{loop}^\star$ action in an appropriate sense. We then have that $\mathsf{decode}_x$ and $\mathsf{encode}_x$ are inverse: by $\mathsf{cmp}_{\mathsf{encode}_x}(\mathsf{Encode}_x)$, we have $\mathsf{Encode}_x\, p\, (\mathsf{encode}_x(p))$, hence $\mathsf{Decode}_x\, (\mathsf{encode}_x(p))\, p$ by (†), and finally $\mathsf{decode}_x(\mathsf{encode}_x(p)) = p$ by $\mathsf{snd}_{\mathsf{decode}_x}(\mathsf{Decode}_x)$. The other direction is similar.

Now, the argument may be further streamlined: since we *choose* relations $\mathsf{Encode}_x$ and $\mathsf{Decode}_x$, subject to the equivalence (†), we might as well take $\mathsf{Encode}_x\, p\, c \equiv \mathsf{Decode}_x\, c\, p$, and thus must show $\mathsf{Decode}_x\, c\, p$ sound and complete for $\mathsf{encode}_x$, or else $\mathsf{Decode}_x\, c\, p \equiv \mathsf{Encode}_x\, p\, c$, and show that $\mathsf{Encode}_{\mathsf{base}}$ is closed under $\mathsf{loop}^\star$, and $\mathsf{Encode}_x$ sound and complete for $\mathsf{decode}_x$.

**Work in progress!** Towards validating the above approach, and making comparisons with Licata-Shulman, we have made some progress on an AGDA formalisation [7], subject to some wrinkles: firstly, inductive families such as those considered above appear *not* well-tolerated by AGDA, and in particular, its pattern-matching algorithm; instead, one must give *equivalent* formulations, where the conclusions are explicitly described by equational premises [5]. Secondly, the proofs of soundness and completeness are made far heavier by the explicit appeal to higher induction on $\mathbb{S}^1$, in particular when showing closure of $\mathsf{Encode}_{\mathsf{base}}$, $\mathsf{Decode}_{\mathsf{base}}$ under $\mathsf{loop}^\star$.

**Conclusions and future work** We have only had time (and space!) to explore one of the simplest instances of the encode-decode method. We hope to extend our approach to examples such as Brunerie's Flattening Lemma [9, Lemmas 8.1.12, 6.12.2], the Freudenthal suspension theorem [*ibid.*, Theorem 8.6.4], or Cavallo's analysis of the Mayer-Vietoris sequence [3], and to consider how systems such as AGDA might better support the methods described here.

**Acknowledgments** We are grateful for feedback from the TYPES referees and the recent Strathclyde HoTT workshop, as well as EPSRC support (EP/K020218/1 and EP/K023837/1).

## References

[1] G. Barthe, J. Forest, D. Pichardie, and V. Rusu. Defining and reasoning about recursive functions: A practical tool for the Coq proof assistant. In *Proc. of FLOPS 2006*, v. 3945 of *Lect. Notes in Comput. Sci.*, pp. 114–129. Springer, 2006.

[2] R. M. Burstall. Proving Properties of Programs by Structural Induction. *Comput. J.*, 12(1):41–48, 1969.

[3] E. Cavallo. Exactness of the Mayer-Vietoris sequence in homotopy type theory. Draft, 2015. `http://www.contrib.andrew.cmu.edu/~ecavallo/works/mayer-vietoris.pdf`

[4] D. Licata and M. Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *Proc. of LICS '13*, pp. 223–232. IEEE CS, 2013.

[5] C. McBride. Inverting inductively defined relations in LEGO. In *Selectd Papers from TYPES '96*, v. 1512 of *Lect. Notes in Comput. Sci.*, pp. 236–253. Springer, 1998.

[6] C. McBride and J. McKinna. The view from the left. *J. of Funct. Program.*, 14(1):69–111, 2004.

[7] J. McKinna and F. Nordvall Forsberg. `http://homepages.inf.ed.ac.uk/jmckinna/LoopSpaceCircleREL.agda`

[8] M. Sozeau. *Un environnement pour la programmation avec types dépendants.* PhD thesis, Université Paris 11, 2008.

[9] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Inst. for Advanced Study, 2013. `http://homotopytypetheory.org/book`