The encode-decode method, relationally

Citation for published version:
McKinna, J & Forsberg, FN 2015, The encode-decode method, relationally. in Abstracts of the 21st International Conference on Types for Proofs and Programs, TYPES 2015. Institute of Cybernetics at Tallinn University of Technology, pp. 63-64.

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Abstracts of the 21st International Conference on Types for Proofs and Programs, TYPES 2015

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
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  
\(\text{james.mckinna@ed.ac.uk, fredrik.nordvall-forsberg@strath.ac.uk}\)

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 \(\pi_1(S^1)\), 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 \(^5\) 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 \(\pi_1(S^1) \simeq \mathbb{Z}\) \(^4\), 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 \(F : \Pi_{x:A}\Pi_{y:B(x)} \text{Type} \) 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, \(\text{snd}_f(F) : \Pi_{x:A}\Pi_{y:B(x)} Fxy \to y = f(x)\), and complete, \(\text{cmp}_f(F) : \Pi_{x:A}Fx(f(x))\), for the graph relation. Turning this around, any choice of (inductive) family \(G\) satisfying \(\text{snd}_f(G)\) and \(\text{cmp}_f(G)\) 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 \(S^1\) is given as a HIT with 0-constructor base : \(S^1\) and 1-constructor loop : base = base. Its covering space \(C(x)\) is defined by higher recursion on \(x : S^1\), by \(C(\text{base}) \equiv \mathbb{Z}\) and a path \(Z = \mathbb{Z}\) obtained via the Univalence Axiom from the automorphism of \(\mathbb{Z}\) induced by \(\text{succ} : \mathbb{Z} \to \mathbb{Z}\). Then the path space \(P(x) \equiv \text{base} = x\) over \(x : S^1\) is shown homotopy equivalent to \(C(x)\) via functions encode : \(\Pi_{x:S^1}\Pi_{p:P(x)} C(x)\) (given by the action \(p^*(0) \equiv \text{transport}_{C(x)}p(0)\) of paths \(p\) on \(0 : \mathbb{Z}\)) and decode : \(\Pi_{x:S^1}\Pi_{p:C(x)} P(x)\), defined by higher induction on \(S^1\): the function \(\text{decode}_\text{base} \) maps \(z : \mathbb{Z}\) to the iterated path \(\text{loop}^z\), and one has to show this definition respects the action \(\text{loop}^\ast\) of \(\text{loop}\).
The encode-decode method, relationally

McKinna and Nordvall Forsberg

Our approach requires the choice of two relations, $\text{Encode}_x : p \leftrightarrow \text{Decode}_x : p$, which we show sound and complete for $\text{encode}_z \land \text{decode}_z$, together with a proof of the homotopy equivalence, which amounts to proving: $\Pi_{x:Sn} \Pi_{\tilde{p}:\text{pc}(x)} \Pi_{y:C(x)} \text{Encode}_z : x \leftrightarrow \text{Decode}_z : y \Leftrightarrow \text{loop}^*$. The main subtlety lies in proving $\text{cmp}_{\text{Decode}_z} (\text{Decode}_z)$.

Towards validating the above approach, and making comparisons with Agda Licata-Shulman, we have made some progress on an $\text{Encode}_z$ and show that $\text{Decode}_z$, together with a proof of the homotopy equivalence, which amounts to proving: $\Pi_{x:Sn} \Pi_{\tilde{p}:\text{pc}(x)} \Pi_{y:C(x)} \text{Encode}_z : x \leftrightarrow \text{Decode}_z : y \Leftrightarrow \text{loop}^*$. The main subtlety lies in proving $\text{cmp}_{\text{Decode}_z} (\text{Decode}_z)$. As above, one must show that $\text{Decode}_z$ respects the loop$^*$ action in an appropriate sense. We then have that $\text{decode}_z$ and $\text{encode}_z$ are inverse: by $\text{cmp}_{\text{Encode}_z} (\text{Encode}_z)$, we have $\text{Encode}_z : p \leftrightarrow \text{decode}_z (\text{encode}_z (p))$, hence $\text{Decode}_z (\text{encode}_z (p))$ by (†), and finally $\text{decode}_z (\text{encode}_z (p)) = p$ by $\text{snd}_{\text{decode}_z} (\text{Decode}_z)$. The other direction is similar.

Work in progress! Towards validating the above approach, and making comparisons with Licata-Shulman, we have made some progress on an Agda formalisation, 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. Secondly, the proofs of soundness and completeness are made far heavier by the explicit appeal to higher induction on $\text{Ex}_1$, in particular when showing closure of $\text{Encode}_z \land \text{Decode}_z$ under loop$^*$.

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


