Representing Inductive Families

Constantine Theocharis[0000−1111−2222−3333] and
Edwin Brady[0000−1111−2222−3333]

University of St Andrews, UK
{kt81,ecb10}@st-andrews.ac.uk

1 Introduction

Inductive data types allow programmers to define data structures in a declarative manner by specifying the all the constructors which can build up the data. Every inductive definition is equipped with an induction principle that captures the notion of mathematical induction over the data, and in particular, enables structural recursion over the data. This is a powerful tool for programming as well as theorem proving. However, this abstraction often comes at a cost; the data representation of inductive types is a linked tree structure. By default, a list as an inductively defined data structure is stored as a linked list, and a natural number is stored as a unary number where each digit is an empty heap cell. This representation is not always the most efficient for all operations, and often forces users to rely on 'more efficient' machine primitives to achieve good performance. The 'Nat-hack' in languages with dependent types is a prime example of this, where natural numbers are represented as big integers for efficient computation.

In this paper, we propose an extension to a core language with dependent types and inductive families which allows users to define custom representations for inductive types. This is done through a translation of the constructors and eliminators of the inductive type to a concrete implementation, which forms a 'representation'. Representations are defined in the language itself, and come with coherence properties that ensure that the representation is a bijection between the inductive type and its implementation.

2 A tour of data representations

2.1 Natural numbers

2.2 Views on lists

2.3 Reindexing and forgetful maps for free

2.4 Binary data

3 A type system for data representations

In this section, we describe a type system for data representations in a language with dependent types. We start by defining a core language with dependent

types and inductive constructions λ_{IND}. We then extend this language with data representations to form λ_{REP}, which allow users to define custom representations for inductive types and other global symbols. We present these languages with intrinsically well-formed contexts, types, and terms, quotiented by their definitional equality rules [altenkirch].

3.1 A core language with inductive types, λ_{IND}

The core languge we start with is λ_{IND}. It contains Π-types and a single universe $\mathcal{U}$ with $\mathcal{U} : \mathcal{U}$. We are not concerned with universe polymorphism or a sound logical interpretation as this is orthogonal to the main focus of this work. Nevertheless, all the results should be readily extensible to a sound language with universe polymorphism. We follow a similar approach to [1] by packaging named inductive constructions and global function definitions into a signature, and indexing contexts by by signature. A typing judgement looks like

$$\Sigma \mid \Gamma \vdash t : T$$

and is read as "in signature Σ and context Γ, term t has type T". The rules for signatures, contexts and telescopes are given in fig. 1.

$$\textsc{Sig-Empty}\ \frac{}{\cdot\ \text{sig}} \qquad \textsc{Sig-Extend}\ \frac{\Sigma\ \text{sig} \qquad \Sigma \vdash Z}{\Sigma, Z\ \text{sig}}$$

$$\textsc{Con-Empty}\ \frac{\Sigma\ \text{sig}}{\Sigma \vdash \cdot\ \text{con}} \qquad \textsc{Con-Extend}\ \frac{\Sigma \vdash \Gamma\ \text{con} \qquad \Sigma \mid \Gamma \vdash T\ \text{type}}{\Sigma \vdash \Gamma, T\ \text{con}}$$

$$\textsc{Tel-Empty}\ \frac{\Sigma \vdash \Gamma\ \text{con}}{\Sigma \mid \Gamma \vdash \cdot\ \text{tel}} \qquad \textsc{Tel-Extend}\ \frac{\Sigma \mid \Gamma \vdash \Delta\ \text{tel} \qquad \Sigma \mid \Gamma \vdash T\ \text{type}}{\Sigma \mid \Gamma \vdash \Gamma, T\ \text{tel}}$$

Fig. 1. Rules for signatures, contexts and telescopes in λ_{IND}.

Telescopes [deBrujin] are very similar to contexts, but restricted to types from a single stage and well formed with respect to a context Γ, meaning that telescopes can contain open terms. We use the notation $\Delta \to t$ to denote a repeated function type with parameters from Δ and codomain T which may depend on the parameters. Additionally, we will sometimes explicitly bind the names of a telescope such as $(\delta : \Delta) \to T[\delta]$. Similar syntax is used to extend contexts with telescopes: Γ, Δ or $\Gamma, \delta : \Delta$.

Awkward spacing!

Next, the rules for well-formed items in signatures are given in fig. 2.

$$\textsc{Data-Item}\quad \frac{\Sigma \mid \cdot \vdash \Delta \text{ tel} \qquad \Sigma \mid \Delta \vdash \Xi \text{ tel} \qquad \mathsf{D} \text{ label} \notin \Sigma}{\Sigma \vdash \mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U}}$$

$$\textsc{Ctor-Item}\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \mathbf{closed}\ \mathsf{D}, \_ \notin \Sigma \qquad \mathsf{C} \text{ label} \notin \Sigma \qquad \Sigma \mid \Delta \vdash \Pi \text{ tel} \qquad \Sigma \mid \Delta, \Pi \vdash \xi : \Xi}{\Sigma \vdash \mathbf{ctor}\ \mathsf{C}\ \Pi : \mathsf{D}\ \Delta\ \xi}$$

$$\textsc{Closed-Item}\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \forall i \in I.\ \mathbf{ctor}\ \mathsf{C}_i\ \Pi_i : \mathsf{D}\ \Delta\ \xi_i \in \Sigma \qquad \mathbf{closed}\ \mathsf{D}, \_ \notin \Sigma}{\Sigma \vdash \mathbf{closed}\ \mathsf{D}, \mathsf{C}}$$

$$\textsc{Def-Item}\quad \frac{\Sigma \mid \cdot \vdash m : M \qquad \mathsf{f} \text{ label} \notin \Sigma}{\Sigma \vdash \mathbf{def}\ \mathsf{f} : M = m}$$

Fig. 2. Rules for items in signatures in λ_{IND}.

$$\textsc{Data-Form}\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma}{\Sigma \mid \Gamma \vdash \mathsf{D} : \Delta \to \Xi \to \mathcal{U}} \qquad \textsc{Data-Intro}\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \mathbf{ctor}\ \mathsf{C}\ \Pi : \mathsf{D}\ \Delta\ \xi \in \Sigma}{\Sigma \mid \Gamma \vdash \mathsf{C} : (\delta : \Delta) \to (\pi : \Pi[\delta]) \to \mathsf{D}\ \delta\ (\xi[\delta, \pi])}$$

$$\textsc{Data-Case}\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \forall i \in I.\ \mathbf{ctor}\ \mathsf{C}_i\ \Pi_i : \mathsf{D}\ \xi_i \in \Sigma \qquad \mathbf{closed}\ \mathsf{D}\ \mathsf{C} \in \Sigma}{\Sigma \mid \Gamma \vdash \mathsf{case}_{\mathsf{D}} : (\delta : \Delta) \to \mathrm{Case}((\xi : \Xi[\delta], x : \mathsf{D}\ \delta\ \xi), \{(\pi : \Pi_i[\delta]), (\xi_i[\delta], \mathsf{C}_i\ \delta\ \pi)\}_i)}$$

$$\textsc{Data-Case-ID}_j\quad \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \forall i \in I.\ \mathbf{ctor}\ \mathsf{C}_i\ \Pi_i : \mathsf{D}\ \xi_i \in \Sigma \qquad \mathbf{closed}\ \mathsf{D}\ \mathsf{C} \in \Sigma \qquad \Sigma \mid \Gamma \vdash \delta : \Delta}{\Sigma \mid \Gamma \vdash \mathrm{ValidCase}(\mathsf{case}_{\mathsf{D}}\ \delta)}$$

$$\textsc{Def-Intro}\quad \frac{\mathbf{def}\ \mathsf{f} : M = m \in \Sigma}{\Sigma \mid \Gamma \vdash \mathsf{f} : M}$$

Fig. 3. Terms and types associated to items in signatures in λ_{IND}.

The rules for data constructors do not consider the recursive occurences of D explicitly, which means that strict positivity is not ensured. Rather, we assume that a separate check is performed to ensure that the defined types adhere to strict positivity, if necessary. As a result, the eliminator for data types does not provide the inductive hypotheses directly, but we assume that the language

allows general recursion. Similarly to the positivity requirements, we expect that if termination is a desirable property of the system, it is ensured separately to the provided typing rules.

In fig. 3, the type $\Sigma \mid \Gamma \vdash \mathrm{Case}(\Phi, \{\Pi_k, \phi_k\}_k) : \mathcal{U}$ is defined as

$$\frac{\Sigma \mid \Gamma \vdash \Phi \text{ tel} \qquad \forall k \in K.\ \Sigma \mid \Gamma \vdash \Pi_k \text{ tel} \qquad \forall k \in K.\ \Sigma \mid \Gamma, \Pi_k \vdash \phi_k : \Phi}{\mathrm{Case}(\Phi, \{\Pi_k, \phi_k\}_k) := (P : \Phi \to \mathcal{U}) \to \{(\Pi_k \to P\ \phi_k)\}_k \to (\phi : \Phi) \to P\phi}$$

The $\Sigma \mid \Gamma \vdash \mathrm{ValidCase}(c)$ condition is defined inductively as

$$\frac{\begin{array}{c}\Sigma \mid \Gamma \vdash c : \mathrm{Case}(\Phi, \{\Pi_k, \phi_k\}_k) \\ \forall j \in K.\ \Sigma \mid \Gamma, P : \Phi \to \mathcal{U}, \{\kappa_k : \Pi_k \to P\ \phi_k\}_k, \pi : \Pi_j \\ \vdash c\ P\ \{\kappa_k\}_k\ \phi_j[\pi] \equiv \kappa_j\ \pi : P\ \phi_j[\pi]\end{array}}{\Sigma \mid \Gamma \vdash \mathrm{ValidCase}(c)}$$

3.2 Extending λ_{IND} with data representations

We base language λ_{IND} to form λ_{REP}, which allows users to define custom representations for inductive types and global functions.

$$\textsc{Repr-Data}\ \frac{\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \Sigma \mid \Delta, \Xi \vdash A : \mathcal{U}}{\Sigma \vdash \mathbf{repr}\ \mathsf{D}\ \Delta\ \Xi\ \mathbf{as}\ A}$$

$$\textsc{Repr-Ctor}\ \frac{\begin{array}{c}\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \qquad \forall i \in I.\ \mathbf{ctor}\ \mathsf{C}_i\ \Pi_i : \mathsf{D}\ \Delta\ \xi_i \in \Sigma \\ \mathbf{repr}\ \mathsf{D}\ \Delta\ \Xi\ \mathbf{as}\ A \in \Sigma \qquad \forall i < k.\ \mathbf{repr}\ \mathsf{C}_i\ \Pi_i\ \mathbf{as}\ t_i \in \Sigma \qquad \Sigma \mid \Delta, \Pi_k \vdash t_k : A\ \Delta\ \xi_k\end{array}}{\Sigma \vdash \mathbf{repr}\ \mathsf{C}_k\ \Pi_k\ \mathbf{as}\ t_k}$$

$$\textsc{Repr-Case}\ \frac{\begin{array}{c}\mathbf{data}\ \mathsf{D}\ \Delta : \Xi \to \mathcal{U} \in \Sigma \\ \mathbf{repr}\ \mathsf{D}\ \Delta\ \Xi\ \mathbf{as}\ A \in \Sigma \qquad \forall i \in I.\ \mathbf{ctor}\ \mathsf{C}_i\ \Pi_i : \mathsf{D}\ \Delta\ \xi_i \in \Sigma \\ \forall i \in I.\ \mathbf{repr}\ \mathsf{C}_i\ \Pi_i\ \mathbf{as}\ t_i \in \Sigma \qquad \mathbf{closed}\ \mathsf{D}\ \mathsf{C} \in \Sigma \\ \Sigma \mid \delta : \Delta \vdash c : \mathrm{Case}((\xi : \Xi[\delta], x : A\ \delta\ \xi), \{(\pi : \Pi_i[\delta]), (\xi_i[\delta], A\ \delta\ \pi)\}_i) \\ \Sigma \mid \Delta \vdash \mathrm{ValidCase}(c)\end{array}}{\Sigma \vdash \mathbf{repr}\ \mathsf{case}_{\mathsf{D}}\ \Delta\ \mathbf{as}\ c}$$

$$\textsc{Repr-Def}\ \frac{\mathbf{def}\ \mathsf{f} : M = m \in \Sigma \qquad \Sigma \mid \cdot \vdash a : M \qquad \Sigma \mid \cdot \vdash m \equiv a : M}{\Sigma \vdash \mathbf{repr}\ \mathsf{f}\ \mathbf{as}\ a}$$

Fig. 4. Rules for data representations in λ_{REP}.

References

1. Cockx, J., Abel, A.: Elaborating dependent (co)pattern matching. Proc. ACM Program. Lang. **2**(ICFP), 1–30 (Jul 2018), https://doi.org/10.1145/3236770