Representation Independence via the Cubical Structure Identity Principle

Carlo Angiuli[1], Anders Mörtberg[2], and Max Zeuner[2]

[1] Carnegie Mellon University, Pittsburgh, USA
[2] Stockholm University, Stockholm, Sweden

A remarkable consequence of the univalence axiom is the *structure identity principle* (SIP), which states that type-theoretic properties of certain mathematical structures are invariant under isomorphism of structures. There are several variations of the SIP formalized in HoTT/UF [1, 2, 4, 5, 9], which have primarily been applied to algebraic and categorical structures. We consider applications of the SIP to *representation independence*, a central problem in the theory of programming languages: given a program that depends on an abstract data structure, and two implementations of that data structure, does the program return the same results when linked against either implementation [7]?

The SIP answers this question in the affirmative whenever the underlying representation types of the implementations are isomorphic, but this is not the case in standard examples. For instance, we can implement finite multisets as either lists or association lists, which are not in bijection but rather only a *many-to-many* correspondence (e.g., the list $[x, x]$ represents the same finite multiset as the two association lists $[(x, 1), (x, 1)]$ and $[(x, 2)]$ where the numbers represent the multiplicity of x). By taking set quotients of these representation types, we can improve this correspondence to an isomorphism and thereby apply the SIP.

As we are interested in applications to programming, we have formalized our results in a cubical type theory, specifically `Cubical Agda` [10], because it has computational formulations of univalence (hence the SIP) and higher inductive types (hence set quotients). In `Cubical Agda`, we have formalized a variation of Martín Escardó's SIP stated with respect to cubical path types [5]. This version of the SIP is rather algebraic in its character and thus much closer to the presentation of Coquand and Danielsson [4], than to the categorical version in the HoTT Book [9]. Moreover, it is straightforwardly applicable to algebraic and data structures, and Escardó [5] has already applied it to an impressive number of mathematical structures.

Following Escardó, we formalize structures as functions $S : \mathsf{Type}\ \ell \to \mathsf{Type}\ \ell$ and the type of S-structures as

$$\mathsf{TypeWithStr}\ S = \Sigma[\ X \in \mathsf{Type}\ \ell\ ]\ (S\ X)$$

Given two S-structures $A\ B : \mathsf{TypeWithStr}\ S$, we have a notion $\iota : \mathsf{StrIso}\ S$ of which equivalences between the underlying types $\mathsf{fst}\ A \simeq \mathsf{fst}\ B$ are "S-structure-preserving" isomorphisms, where

$$\mathsf{StrIso}\ S = (A\ B : \mathsf{TypeWithStr}\ S) \to \mathsf{fst}\ A \simeq \mathsf{fst}\ B \to \mathsf{Type}\ \ell$$

The type of isomorphisms between A and B is then given by

$$A \cong B = \Sigma[\ e \in \mathsf{fst}\ A \simeq \mathsf{fst}\ B\ ]\ (\iota\ A\ B\ e)$$

A *univalent structure* is a pair (S, ι) for which isomorphisms $e : \mathsf{fst}\ A \simeq \mathsf{fst}\ B$ of underlying types preserve structure exactly when the structures $\mathsf{snd}\ A$ and $\mathsf{snd}\ B$ are homotopic *over* e.

$$\begin{aligned}\mathsf{UnivalentStr}\ S\ \iota = \{A\ B : \mathsf{TypeWithStr}\ S\}\ (e : \mathsf{fst}\ A \simeq \mathsf{fst}\ B)\\ \to (\iota\ A\ B\ e) \simeq \mathsf{PathP}\ (\lambda\ i \to S\ (\mathsf{ua}\ e\ i))\ (\mathsf{snd}\ A)\ (\mathsf{snd}\ B)\end{aligned}$$

Here, PathP is a dependent path type and ua is the function underlying the univalence *theorem* in `Cubical Agda`.

This definition of a univalent structure is equivalent to Escardó's *standard notion of structure*, but interacts better with cubical machinery. Using $\equiv$ for non-dependent path types, we can then prove the (cubical) SIP, which states that we have a term[1]

$$\mathsf{SIP} : \mathsf{UnivalentStr}\ S\ \iota \to (A\ B : \mathsf{TypeWithStr}\ S) \to (A \cong B) \simeq (A \equiv B)$$

Following Escardó, we show that various operations on structures preserve being a UnivalentStr, such as augmenting a structure with proposition-valued axioms, or taking the product of two structures. The cubical proofs of these facts are more direct and shorter than the corresponding ones in HoTT/UF. These two results are what makes Escardó's SIP (and our cubical version) so readily applicable. In order to apply the SIP we always have to prove that the structures we consider are a UnivalentStr, but in Escardó's setting we only have to prove these facts for the most elementary structural components from which we build more complex structures.

To illustrate how the SIP can be applied to concrete data structures we focus on finite multisets. Following Okasaki [8, Fig. 2.7] we define a minimal multiset structure over some fixed type A as

$$\begin{aligned}&\mathsf{multiset\text{-}structure} : \mathsf{Type}\ \ell \to \mathsf{Type}\ \ell\\ &\mathsf{multiset\text{-}structure}\ X = X \times (A \to X \to X) \times (A \to X \to \mathbb{N})\end{aligned}$$

Terms of this signature consist of the empty multiset, an insertion function and a count function that returns the multiplicity of elements in X. This can easily be adapted and extended with other operations, such as union and intersection. We can also easily add axioms, such as requiring all elements to have multiplicity 0 in the empty multiset. However, for simplicity, we focus on this minimal signature in this abstract.

If A has decidable equality we can define a count function on List A and, together with [] and _::_, obtain an implementation of multiset-structure. A more efficient option would be to use *association lists*, represented by List $(A \times \mathbb{N})$ and a count function that interprets $(a\ ,\ n)$ as n copies of a in the multiset. As discussed above, these representations of multisets are not isomorphic, but in a many-to-many correspondence. However, we can add path constructors to refine the types and get two isomorphic HITs[2]

```
data FMSet (A : Type ℓ) : Type ℓ where
  []    : FMSet A
  _::_  : A → FMSet A → FMSet A
  comm  : ∀ x y xs →
    x :: y :: xs ≡ y :: x :: xs
  trunc : isSet (FMSet A)
```

```
data AList (A : Type ℓ) : Type ℓ where
  ⟨⟩ : AList A
  ⟨_,_⟩::_ : A → ℕ → AList A → AList A
  per : ∀ a b m n xs →
    ⟨ a , m ⟩:: ⟨ b , n ⟩:: xs ≡ ⟨ b , n ⟩:: ⟨ a , m ⟩:: xs
  agg : ∀ a m n xs →
    ⟨ a , m ⟩:: ⟨ a , n ⟩:: xs ≡ ⟨ a , m + n ⟩:: xs
  del : ∀ a xs → ⟨ a , 0 ⟩:: xs ≡ xs
  trunc : isSet (AList A)
```

The HIT FMSet has already been formalized in `Cubical Agda` in [3], but AList as well as the equivalence of the two types is novel. The count functions on lists and association lists extend so that we get isomorphic multiset structures on these HITs, which by the SIP implies that they

[1] See https://github.com/agda/cubical/blob/master/Cubical/Foundations/SIP.agda.

[2] See https://github.com/agda/cubical/tree/master/Cubical/HITs/FiniteMultiset and https://github.com/agda/cubical/tree/master/Cubical/HITs/AssocList.

are equal implementations of multiset-structure. However, while adding higher constructors solved the problem of the types not being isomorphic, a natural question to ask is what the relationship is between these HITs and the original ordinary data types?

To answer this question, we demonstrate that we can recover these HITs by identifying two lists (resp., association lists) xs and ys iff they both correspond to a single association list (resp., list). That is, we identify two representations of a finite multiset whenever all elements of A occur with the same multiplicity in each. Because our many-to-many correspondence between lists and association lists is *zigzag-complete* [6], we automatically obtain relations R and R' performing the above identifications, and an isomorphism between the set quotients List A / R and List $(A \times \mathbb{N})$ / R', on the top of the square below[3]

$$\begin{array}{ccc} \mathsf{List}\ A\ /\ R & \xrightarrow{\cong} & \mathsf{List}\ (A \times \mathbb{N})\ /\ R' \\ \downarrow\cong & & \downarrow\cong \\ \mathsf{FMSet}\ A & \xrightarrow{\cong} & \mathsf{AList}\ A \end{array}$$

The right-most isomorphism is just the composition of the other three. In particular, it follows that any two elements in AList A can be identified if they have the same count function, a fact that would be quite hard to prove directly because of the many path constructors in AList.

Using the SIP we hence get paths between all four of the multiset implementations and we can freely transport proofs and programs between them. This means that no matter which implementation we choose to work with the other implementations will automatically share the same operations and properties which ensures representation independence when programming with finite multisets in `Cubical Agda`.

References

[1] Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[2] Benedikt Ahrens, Paige Randall North, Michael Shulman, and Dimitris Tsementzis. A higher structure identity principle. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 53–66, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Vikraman Choudhury and Marcelo Fiore. The finite-multiset construction in hott. `https://hott.github.io/HoTT-2019/conf-slides/Choudhury.pdf`, 2019.

[4] Thierry Coquand and Nils Anders Danielsson. Isomorphism is equality. *Indagationes Mathematicae*, 24(4):1105–1120, 2013.

[5] Martín Hötzel Escardó. Introduction to univalent foundations of mathematics with agda. `https://www.cs.bham.ac.uk/~mhe/HoTT-UF-in-Agda-Lecture-Notes/index.html`, 2019.

[6] Neelakantan R. Krishnaswami and Derek Dreyer. Internalizing Relational Parametricity in the Extensional Calculus of Constructions. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 432–451, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] John C. Mitchell. Representation independence and data abstraction. In *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '86, pages 263–276, New York, NY, USA, 1986. Association for Computing Machinery.

[3] See `https://github.com/agda/cubical/blob/master/Cubical/Relation/ZigZag/Applications/MultiSet.agda`.

[8] Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

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

[10] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: a dependently typed programming language with univalence and higher inductive types. *Proceedings of the ACM on Programming Languages*, 3:1–29, jul 2019.