Containers of Applications and Applications of Containers

Malin Altenmüller and Conor McBride

University of Strathclyde, Glasgow, United Kingdom

We use the notion of indexed containers to define a datatype of *applications* in dependent type theory. Applications are server-like programs, indexed by their current status. The basis for the definition is the type of *interface* of an application, using the intuition that indexed containers are interaction structures [HH06]. The type of applications itself results from building cofree comonads on these containers. We take a comonadic view (applications provide a service) rather than a monadic view (applications do side effects), but Hancock, Setzer and Hyvernat's analysis of interaction structures [HS00, HSCS05] keep guiding our design. With indexed containers we additionally get both a notion of how applications display themselves to a user and combinators which enable modular construction of complex applications.

What is an Application? The applications we describe are *programs* which deliver some functionality to a user. The user sends requests to the application and the applications reacts accordingly. Applications act in a server-like way: they lazily await commands from their user, updating their internal state when receiving one. In addition they are always ready to display themselves to the user. Defining the type of applications consists of two parts: The type of specification (or interface) for applications contains the commands and responses an application can receive and send. The type of an application itself is based on an interface and coinductively defines its display type and its reaction to a command.

Indexed Containers as Templates. Hancock and Hyvernat [HH06] introduced indexed containers as *interaction structures* — protocols of communication between client and server. They consist of fields for commands and responses as well as for the next state the server will reach after a command-response pair has been sent. Indexed containers also represent indexed functors on the underlying indexed sets as defined by Altenkirch et al.[AGH+15]. Our type of specification for applications is a modified version of the description of interaction structure. Instead of computing the next status with an indexed function (depending on the response), we will use the contravariant powerset notion via a predicate on the type of status. This notion emphasises that only *some* properties of the server's next status might be known to the client, but not all of it. In Agda, the type of specification for applications looks like this:

```
record _◁_ (Now Next : Set) : Set₁ where
  field Commands : Now → Set
        Response : (now : Now) → Commands now → (Next → Set)
```

Here *Now* and *Next* are the types of status an application can be in, the Commands an application can receive depend on its current status and the Response depends on the current status and the issued command and returns a property of the next status of the application. The closure properties of _◁_ are more flexible if *Now* and *Next* are separated, but we take fixpoints only when they coincide, i.e. when the container represents an *endofunctor*.

The Type of Application. With indexed containers as a notion of interface, we use them as the basis to define the type of applications themselves:

```
record App {S : Set} (spec : S ◁ S) (D : S → Set) (s : S) : Set where
  coinductive
  field display : D s
        react   : (c : spec .Commands s)
                → (s : S) × (spec .Response s c s′ × App spec D s′)
```

We distinguish three types of *state* of applications: The static status S, the public display and a private internal state, which is the carrier of the coalgebra. When it receives a command, the application reacts by returning a Response and updating its internal state.

Having containers defining the interface of applications, we can use product operations on them to build higher dimensional structures and describe more complex applications. Examples include a text editor which is built from one-dimensional line editing applications and a window manager, controlling multiple overlapping windows on a screen.

The Type of Display. We also use indexed containers to define the type of *displays* of applications (called D in the above definition). This two-dimensional structure is indexed by its size and the underlying indexed container defines how to partition it. Using the size as index type makes the *pieces* (which result from partitioning) fit together properly. As we are describing structures which take up a limited amount of space, the underlying container is restricted to only have a *finite* amount of positions at which we can store data. Given a type of *tiles* (also indexed by their size) we can build interiors by plugging in the tiles into holes they fit in. This plugging-in operation is building the free monads of the underlying containers and gives us the type of display of an application. Again, combinators of the underlying containers let us build multi-dimensional structures. We start with notion of cutting a one-dimensional structure (i.e. dividing its length) and then use a product operation to get higher-dimensional structures. Partitioning a higher-dimensional structure consists of a choice of dimension, in which the split is performed, the other dimensions stay unchanged.

Summary. With these definitions of applications and displays we are building a library to construct complex applications from low dimensional ones. Using containers as the underlying structures results in a notion of combining containers by using product operations. We are currently implementing this library and aim for a sufficient collection of tools for building applications compositionally.

References

[AGH+15] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. Indexed containers. *Journal of Functional Programming*, 25:e5, 2015.

[HH06] Peter Hancock and Pierre Hyvernat. Programming interfaces and basic topology. *Annals of Pure and Applied Logic*, 137(1):189 – 239, 2006.

[HS00] Peter Hancock and Anton Setzer. Interactive programs in dependent type theory. In Peter G. Clote and Helmut Schwichtenberg, editors, *Computer Science Logic*, pages 317–331, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[HSCS05] Peter Hancock, Anton Setzer, L Crosilla, and P Schuster. Interactive programs and weakly final coalgebras in dependent type theory. *From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics*, 48:115–134, 2005.