A Categorical Foundation of Functional Reactive Programming with Mutable State

Wolfgang Jeltsch

TTÜ Küberneetika Instituut, Tallinn, Estonia
wolfgang@cs.ioc.ee

Abstract

Ordinary functional programming deals with values, which can be duplicated and discarded at will. Linear functional programming, on the other hand, deals with state, which can neither be duplicated nor discarded. Both paradigms can be combined within a single language, and their interaction can be modeled by a lax symmetric monoidal adjunction (LSMA).

In this paper, we show that a similar situation holds regarding functional reactive programming (FRP): we can combine ordinary functional programming and FRP and model their interaction by an adjunction that interacts with a cartesian endofunctor. Based on this observation, we develop a categorical structure that models the interaction of functional programming with linear functional programming and FRP at the same time. This structure enables us to obtain categorical models of an FRP variant with mutable state as pushouts in a suitable category.

1 Functional Reactive Programming

Functional reactive programming (FRP) is a programming approach that deals with temporal aspects in a declarative way. It is the Curry–Howard correspondent of an intuitionistic temporal logic.

We have developed several kinds of categorical models of FRP in our earlier work [2, 3, 4]. These assume that time is linear, but put no further restrictions on the time scale. In this paper, we additionally assume that time is discrete. This enables us to use a much simpler categorical semantics, which we describe in the remainder of this section.

A categorical model of FRP contains a cartesian closed category (CCC), which we call $\mathcal{T}$ here. The objects of $\mathcal{T}$ model FRP types. Type inhabitance in FRP is time-dependent. If objects A and B model FRP types τ_1 and τ_2, a morphism $f : A \to B$ denotes an operation that turns any value that inhabits τ_1 at some time into a value that inhabits τ_2 at the same time. Finite products and exponentials in $\mathcal{T}$ model product types and function types, respectively.

Besides the CCC $\mathcal{T}$, a categorical model of FRP contains a cartesian endofunctor $\bigcirc$. If an object A models a type τ, $\bigcirc A$ models a type whose inhabitants at a time t are the inhabitants of τ at time $t + 1$.

2 Linear Functional Programming

Linear functional programming (LFP) does not deal with ordinary values but with the state of mutable objects. It uses a linear type system, which prevents state from being duplicated or discarded. LFP is the Curry–Howard correspondent of intuitionistic linear propositional logic.

We can model LFP by a symmetric monoidal closed category (SMCC). In an SMCC $(\mathcal{L}, \otimes, I, \multimap)$, the tensor $\otimes$ and the identity I model the type constructors for binary and nullary tuples, and the right adjoint $\multimap$ of $\otimes$ models the function type constructor.

Ordinary functional programming and LFP can be combined and the resulting system can be given a categorical semantics based on lax symmetric monoidal adjunctions (LSMAs) [1]. A categorical model according to this semantics consists of the following components:

- a CCC $\mathcal{C}$, which models the non-linear part
- an SMCC $(\mathcal{L}, \otimes, I, \multimap)$, which models the linear part
- an LSMA $(F, \varphi, \psi) \dashv (G, \upsilon, \nu)$ between $(\mathcal{C}, \times, 1)$ and $(\mathcal{L}, \otimes, I)$, which models the interaction between the two parts

The functor F models a type constructor whose inhabitants are values treated as state, while the functor G models a type constructor whose inhabitants are values that describe the generation of mutable objects.

Note the general idea behind this approach of modeling functional programming, LFP, and their interaction:

- We model both functional programming and LFP by a category with additional structure (an SMCC structure).
- For the model of functional programming, we require that the additional structure is of a specific kind (a CCC is a specific kind of SMCC).
- We model the interaction between functional programming and LFP by an adjunction that interacts with the additional structure of the two categories (by being an LSMA).

3 Interaction between Functional Programming and FRP

We can combine functional programming and FRP in a way analogous to combining functional programming and LFP. Let $(\mathcal{T}, \bigcirc)$ be a categorical model of FRP. Its "additional structure" is the cartesian endofunctor $\bigcirc$. We take a category $\mathcal{C}$ for modeling functional programming and give it a cartesian endofunctor $\bigcirc$ as well. However we choose a specific cartesian endofunctor structure: we define $\bigcirc$ to be the identity functor.

We model the interaction between functional programming and FRP by an adjunction $F \dashv G$. The functor F models a constructor of FRP types whose inhabitants are those of an ordinary type, independently of time. The functor G models a constructor of ordinary types whose inhabitants are those that inhabit an FRP type at every time.

We let the adjunction $F \dashv G$ interact with the cartesian endofunctors by requiring that there exist natural transformations of types $F\bigcirc \to \bigcirc F$ and $\bigcirc G \to G\bigcirc$ that fulfill obvious axioms. Since $\bigcirc = \mathrm{Id}$ for the category $\mathcal{C}$, this boils down to having natural transformations of types $F \to \bigcirc F$ and $G \to G\bigcirc$. The first of them models an operation that stores a value of an ordinary type until the next time; the second of them establishes that time-universal values are valid at every time $t + 1$.

4 FRP with Mutable State

Now we integrate the structures of Sects. 2 and 3:

- We model each of the paradigms functional programming, LFP, and FRP by an SMCC with a symmetric monoidal endofunctor (SME), which we name $\bigcirc$. We call the underlying categories $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{T}$.
- We require that the SMCC structures of $\mathcal{C}$ and $\mathcal{T}$ are CCC structures, and that the functor $\bigcirc$ of $\mathcal{C}$ and $\mathcal{L}$ is the identity functor.
- We require the existence of two LSMAs that interact with $\bigcirc$: one between $(\mathcal{C}, \times, 1, \Rightarrow, \mathrm{Id})$ and $(\mathcal{L}, \otimes, I, \multimap, \mathrm{Id})$, another between $(\mathcal{C}, \times, 1, \Rightarrow, \mathrm{Id})$ and $(\mathcal{T}, \times, 1, \Rightarrow, \bigcirc)$.

An implication of the above requirements is that the adjunction between $\mathcal{C}$ and $\mathcal{T}$ preserves products, something that we did not enforce before.

Now consider the category whose objects are SMCCs with an SME, and whose morphisms are LSMAs that interact with the SMEs through appropriate natural transformations. The constructs described above form a span in this category. A pushout of this span models a linear form of FRP, that is, an FRP variant that can deal with mutable state.

Note that there is an LSMA between $(\mathcal{T}, \times, 1, \Rightarrow, \bigcirc)$ and the pushout that has two associated natural transformations of types $F\bigcirc \to \bigcirc F$ and $\bigcirc G \to G\bigcirc$. We assume that an object $\bigcirc A$ of the pushout models a type whose inhabitants generally do not just denote future state, but may denote effectful computations that take one time step to produce a state. Note that even under this assumption, the types of the abovementioned two natural transformations make sense:

1. A future value (treated as a state) can be turned into a computation that delivers this value in the future (again treated as a state). This computation is special in that it does not have any actual effect.
2. A future generator of mutable objects can be turned into a generator of computations that deliver mutable objects in the future. The generated computations just wait until the next time and then run the original generator. They are also special in that they do not have any actual effect.

References

[1] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. Technical Report UCAM-CL-TR-352, University of Cambridge, Cambridge, England, October 1994.

[2] Wolfgang Jeltsch. Towards a common categorical semantics for linear-time temporal logic and functional reactive programming. *Electronic Notes in Theoretical Computer Science*, 286:229–242, September 2012.

[3] Wolfgang Jeltsch. Temporal logic with “until”, functional reactive programming with processes, and concrete process categories. In *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification (PLPV ’13)*, pages 69–78, New York, 2013. ACM.

[4] Wolfgang Jeltsch. An abstract categorical semantics for functional reactive programming with processes. Accepted for PLPV ’14, January 2014.