Deriving Correct and Efficient FRP Implementations from Categorical Denotations

Edward Amsden

October 2, 2013

1 Introduction

Functional Reactive Programming (FRP) is a promising concept which enables the compositional and equational construction and analysis of reactive systems. Recent work has applied basic concepts from category theory to show a Curry-Howard correspondence between Linear Temporal Logic and Functional Reactive Programming. The categorical equations used in these demonstrations capture precise notions of the modality of propositions in LTL and of types in FRP.

FRP has been said to have "resisted efficient implementation [1]." In particular, most extant FRP implementations inhabit a spectrum in which pure semantic correctness and efficiency of implementation are opposed, and very few inhabit the parts of the spectrum nearest semantic correctness.

The dichotomy may be falsified by demonstrating a way to derive efficient implementations rigorously from the semantic definitions of FRP. Therefore, once the semantics for FRP have been reviewed (Section 2), a procedure for deriving implementations from the semantics is given (Section 3), shown to produce efficient implememementations and justify optimizations (Section 4) and shown to be correct (Section 5).

2 Review of Categorical Semantics for FRP

There have been several presentations of categorical semantics for FRP [2, 4]. The clearest such explication [3] denotes FRP types with objects in an exponential category from a simple category representing time to a Cartesian Closed Category with Coproducts (CCCC) representing the underlying functional language semantics. We denote this category $\mathcal{B}^{\mathcal{T}}$, where $\mathcal{T}$ is the category representing time, and $\mathcal{B}$ is the CCCC representing non-temporal propositions or types. Objects from $\mathcal{B}$ can be lifted pointwise to objects (functors) in $\mathcal{B}^{\mathcal{T}}$. We can then define more interesting functors using these pointwise functors and products and coproducts from $\mathcal{B}$.

3 Derivation of an Implementation From Semantics

Given a categorical equation for a family of temporal objects (LTL propositions or FRP types), the next step is to derive an implementation. In this case, we show how to embed these objects in $\mathcal{B}$ by showing how to use the definition of the family of objects in $\mathcal{B}^{\mathcal{T}}$ to guide the construction of objects in $\mathcal{B}$. (We will prove the correctness of this derivation in Section 5.)

The key intuition to this derivation is that series products across time intervals (which represent continuities in time) can be projected from (sampled or pulled) but not injected into (notified or pushed). Conversely, series coproducts across time intervals (which represent discontinuities in time) can be injected into but not projected from. The relationship between products as functions and co-products as continutations is suggestive here, since pulling can be thought of as the consumer of a reactive value calling the function which produces it, while pushing can be thought of as the producer calling a continuation of the consumer.

4 Efficiency of Derived Implementations

TODO

5 Correctness of Derived Implementations

TODO

References

[1] Elliott, C. M. Push-pull functional reactive programming. In *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell* (New York, NY, USA, 2009), Haskell '09, ACM, pp. 25–36.

[2] Jeffrey, A. LTL types FRP: linear-time temporal logic propositions as types, proofs as functional reactive programs. In *Proceedings of the sixth workshop on Programming languages meets program verification* (New York, NY, USA, 2012), PLPV '12, ACM, pp. 49–60.

[3] Jeltsch, W. 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* (New York, NY, USA, 2013), PLPV '13, ACM, pp. 69–78.

[4] Krishnaswami, N. R., and Benton, N. Ultrametric semantics of reactive programs. In *Proceedings of the 2011 IEEE 26th Annual Symposium on*

Logic in Computer Science (Washington, DC, USA, 2011), LICS '11, IEEE Computer Society, pp. 257–266.