Foundations of Total Functional Data-Flow Programming, Coinductively
(Cross-Summary from MSFP 2014 [3])

Baltasar Trancón y Widemann[1,2] and Markus Lepper[2]

[1] Ilmenau University of Technology, DE
[2] <semantics/> GmbH, DE
baltasar.trancon@tu-ilmenau.de

Abstract. The field of declarative data stream programming (discrete time, clocked synchronous, modular, data-centric) is divided between the data-flow graph paradigm favored by domain experts, and the functional reactive paradigm favored by academics. In the summarized paper, we describe the foundations of a framework for unifying functional and data-flow styles that differs from FRP proper in significant ways: It draws on semantic objects from plain set theory to match the expectations of domain experts, and the two paradigms are reduced symmetrically to a low-level middle ground, with strongly compositional semantics. The design of the framework is derived from mathematical first principles, in particular coalgebraic coinduction and a standard relational model of stateful computation. Abstract syntax and semantics are given and constitute the full core of a novel stream programming language.

Our goal is a formal foundation for the declarative programming of stream processing software, that appeals to mathematically educated domain experts in engineering, scientific modeling and digital arts. Thus we readily sacrifice general recursion and Turing completeness, in order to eliminate domain theory from the semantical picture, such that object level functions and the concept of ordinary total mathematical functions, as such held by the intended users, coincide.

We derive our framework, which is very much work in progress, from a number of disparate low-level formal tools that are already available: As suggested by Turner [4], coinductive datatypes and their structural recursion schemes, such as coiteration, are required. On the other hand, our approach to the unification of control flow and data flow introduces local nondeterminism; hence we employ a well-known monadic generalization of coiteration. The coinductive core of semantics is complemented with connections to actual language: Firstly, a program representation inspired by the state of the art in sequential languages, namely static single-assignment form, adapted to a data-flow scenario. Secondly program transformations, both from higher-level programs written in either functional or visual flow graph style, and to a lower-level state transition form that is then amenable to coiteration.

The present short paper focuses only on the role of coinductive techniques within the framework.

Arguably the most basic and well-understood final coalgebra is that of the functor $A \times -$. Its carrier A^ω contains precisely the countably infinite sequences (streams) of elements of A, with the operation $\langle \mathit{head}, \mathit{tail} \rangle = \mathit{cons}^{-1} : A^\omega \to A \times A^\omega$. A straightforward, but less well-known extension concerns the functor $T = (B \times -)^A$. The carrier $(A \xrightarrow{!} B)$ of its final coalgebra contains precisely the *causal* stream functions from A to B, that is, the maps $f : A^\omega \to B^\omega$ such that each finite prefix of the result is determined uniquely by the corresponding prefix of the argument of the same length. The operation is an appropriate lifting of *head* and *tail*. Causal stream functions are of particular interest because they can be posed as real-time computation problems. The isomorphism $(A \xrightarrow{!} B) \cong T(A \xrightarrow{!} B)$ is the basis of continuation-based implementations of functional reactive programming [1].

Ordinary coinduction assign to every F-coalgebra $(X, f : X \to FX)$ a unique homomorphism $[\![f]\!]$ into the final F-coalgebra $(\nu F, \mathrm{out}_F)$, that is, a map $[\![f]\!] : X \to \nu F$ such that $\mathrm{out}_F \circ [\![f]\!] = F[\![f]\!] \circ f$. Monadic coinduction [2] reads the same commuting square in the Kleisli category of a monad M, namely as $M\,\mathrm{out}_F \circ [\![f]\!]^M = \widehat{F}([\![f]\!]^M)^* \circ f$ where $\widehat{F}$ is the Kleisli lifting of F, and * is Kleisli extension of arrows. This equation need not have a unique solution. The Tarski-style argument of [2] for canonical fixpoint solutions mostly carries over from the original setting in **CPO** to our setting in **Set**, except that only the greatest fixpoint is useful.

Following the tradition of the Z notation for formal specification, and for various reasons out of scope here, we model building blocks of stream computations as quaternary relations between pre-states, inputs, outputs and post-states; that is, in the form $R : S \times A \to \mathcal{P}(B \times S)$. By means of currying and the distributive law $\lambda : (-^A)\mathcal{P} \Rightarrow \mathcal{P}(-^A)$, we obtain a monadic coalgebra operation $r : S \to \mathcal{P}TS$, namely $r = \lambda_{B \times S} \circ \mathit{curry}(R)$. This in turn induces the desired, monadic coinductive semantics $[\![r]\!]^{\mathcal{P}} : S \to \mathcal{P}(A \xrightarrow{!} B)$, a relation between initial states and possible behavior in terms of causal stream functions.

References

1. Nilsson, H., Courtney, A., Peterson, J.: Functional reactive programming, continued. In: Haskell Workshop. pp. 51–64. ACM (2002)
2. Pardo, A.: Monadic corecursion – definition, fusion laws and applications. ENTCS 11, 105–139 (1998)
3. Trancón y Widemann, B., Lepper, M.: Foundations of total functional data-flow programming. In: Krishnaswami, N., Levy, P.B. (eds.) Mathematically Structured Functional Programming. EPTCS (2014), to appear
4. Turner, D.A.: Total functional programming. Univ. Comput. Sci. 10(7), 751–768 (2004)