Coinductive Program Synthesis for the Masses

Summary of: Higher-Order Causal Stream Functions in Sig from First Principles [5]

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

[1] Technische Universität Ilmenau
[2] semantics GmbH Berlin

The Sig language for purely functional high-level online synchronous data-flow programming (stream programming) is founded on a core with compositional formal semantics [4]. The key construction is the representation of Mealy-type stream transducers as coalgebras for the family of **Set**-endofunctors $T_{AB} = (B \times -)^A$, for any input type A and output type B, whose final coalgebras are the respective spaces of *causal* stream functions $(A \overset{\omega}{\rightsquigarrow} B) \subseteq (A^\omega \to B^\omega)$ with an operation $\phi : (A \overset{\omega}{\rightsquigarrow} B) \to (B \times (A \overset{\omega}{\rightsquigarrow} B))^A$ that sends a causal stream function and first input element to the first output element and continuation function (cf. [1, 3]).

Sig programs are reduced to automata $(X, f : X \times A \to B \times X)$; the user is only concerned with elementwise computation $(A \to B)$, and the state space X and transition are inferred from the use of *delay* operators. By Currying, a T_{AB}-coalgebra $(X, \lambda f)$ is obtained, and hence an anamorphism $[\![\lambda f]\!] : X \to (A \overset{\omega}{\rightsquigarrow} B)$ that sends initial states to coinductive stream functions, the denotational semantics of a program specified by running f in an *infinite reactive* loop.

We have recently discovered that, apart from the *descriptive* use of the coinductive semantics, there is also a promising *prescriptive* side: the specification of stream computations as anamorphisms gives rise to a method for the formal calculation of implementations. This serves the triple goals of explicating the semantics of Sig, demonstrating coalgebraic techniques to a wider audience in terms of widely understood programming concepts, and exploring the rather novel field of coinductive stream program synthesis (cf. [2]).

As a typical example, consider the calculation of a fundamental ingredient of stream processing algorithms, the single-step delay operator δ, from its intended semantics p, that prepends an initial element to its input stream; $p : A \to (A \overset{\omega}{\rightsquigarrow} A)$ with $p(a_0)(\alpha) = \mathit{cons}(a_0, \alpha)$. The equivalent coinductive form of this behavior is $\phi(p(a_0))(a_1) = (a_0, p(a_1))$, where a_1 is the first element of α. (Prepending a_0 starts with a_0 instead of a_1, but continues by re-prepending a_1 to the rest.)

By comparing type signatures, we find the underlying automaton structure $(A, \delta : A \times A \to A \times A)$ and calculate δ, assuming the goal $[\![\lambda\delta]\!] = p$, simply:

$$\begin{aligned} \phi \circ p &= T_{AB}(p) \circ \lambda\delta \\ \phi\big(p(a_0)\big)(a_1) &= T_{AB}(p)\big(\lambda\delta(a_0)\big)(a_1) \\ &= (\mathrm{id}_A \times p)\big(\lambda\delta(a_0)(a_1)\big) \\ \big(a_0, p(a_1)\big) &= (\mathrm{id}_A \times p)\big(\delta(a_0, a_1)\big) \qquad (*) \end{aligned}$$

Evidently, even without knowledge about the kernel of p, the equation (*) can be satisfied by putting $\delta = \mathrm{id}_{A\times A} = \mathrm{id}_A \times \mathrm{id}_A$. This canonical, 'maximum-entropy' solution also turns out to be the only one, since p is in fact injective.

On the one hand, this calculation justifies the intuitive implementation of the delay operation as a sub-automaton that, at each step, outputs its pre-state (first component), and simultaneously stores its input as post-state (second component), to be retrieved in the next step, and so forth. Coinduction comes in precisely as the formal specification of 'and so forth'. On the other hand, it justifies the treatment of delay operations as pairs of *independent* identities, whose propagation is at the heart of the SIG code optimizer and causality checker.

In [5] we give an introduction to coinductive streams and stream transducers entirely in the classical language of set theory, without reference to categorial notation. We demonstrate the power of coinductive synthesis in this more verbose but less hermetic style, not only for the delay operator (which accounts for the streaming properties of SIG), but also for sequential and parallel composition (which account for compositionality) in the style of Z schema composition, and for dynamic function application (which accounts for higher-orderness) in an object-oriented style. We effectively illustrate the nature of stream coinduction, the use of delay in functional stream programming, and the relationship to low-level imperative styles, with a family of pseudo-random number generators.

References

[1] P. Caspi and M. Pouzet. "A Co-iterative Characterization of Synchronous Stream Functions". In: *Electronic Notes in Theoretical Computer Science* 11 (1998), pp. 1–21. DOI: 10.1016/S1571-0661(04)00050-7.

[2] H. H. Hansen and J. J. M. M. Rutten. "Symbolic Synthesis of Mealy Machines from Arithmetic Bitstream Functions". In: *Sci. Ann. Comp. Sci.* 20 (2010), pp. 97–130. URL: http://www.infoiasi.ro/bin/Annals/Article?v=XX&a=3.

[3] H. Nilsson, A. Courtney, and J. Peterson. "Functional Reactive Programming, Continued". In: *Proc. Haskell Workshop*. ACM, 2002, pp. 51–64. DOI: 10.1145/581690.581695.

[4] B. Trancón y Widemann and M. Lepper. "Foundations of Total Functional Data-Flow Programming". In: *Proceedings 5th International Workshop on Mathematically Structured Functional Programming (MSFP 2014)*. Ed. by P. B. Levy and N. Krishnaswamy. Vol. 154. Electronic Proceedings in Theoretical Computer Science. 2014, pp. 143–167. DOI: 10.4204/EPTCS.153.10.

[5] B. Trancón y Widemann and M. Lepper. "Higher-Order Causal Stream Functions in Sig from First Principles". In: *Proceedings Software Engineering Workshops (SE-WS 2016)*. CEUR Workshop Proceedings. 2016, pp. 25–39. URL: http://ceur-ws.org/Vol-1559/paper03.pdf.