Dynamic codesign in Poly

Marius Furter

February 6, 2022

Contents

1 Recap of Poly

Recall that a *representable functor* has the form $\mathcal{C}(A,-) : \mathcal{C} \to \mathsf{Set}$. When $\mathcal{C} = \mathsf{Set}$, we will denote $\mathsf{Set}(A,-) =: y^A$ and call this a *monomial*. A *polynomial functor* is any functor that is isomorphic to a sum of monomials

$$p \cong \sum_{i \in I} y^{p[i]},$$

where the $(p[i])_{i\in I}$ is a family of sets, and the sum is taken to be the coproduct in the functor category $[\mathsf{Set}, \mathsf{Set}]$ given by taking the component-wise disjoint union of the sets in question. Observe that if we evaluate p at the singleton set 1, we recover the indexing set of the sum

$$p(1) \cong \sum_{i \in I} 1^{p[i]} \cong \sum_{i \in I} 1 \cong I.$$

Hence we will often write $p \cong \sum_{i \in p(1)} y^{p[i]}$ to avoid always having to name the index set. The elements of $p(1)$ are called the *positions* of p, while the elements of $p[i]$ are called the *directions* at position i.

Since polynomial functors are functors, morphisms between them are given by natural transformations. By using the universal property of coproducts in $[\mathsf{Set}, \mathsf{Set}]$ along with the Yoneda lemma, one can observe that

$$\begin{aligned}\mathsf{Poly}(p, q) &= \mathsf{Poly}(\sum_{i \in p(1)} y^{p[i]}, \sum_{j \in q(1)} y^{q[j]}) \\ &\cong \prod_{i \in p(1)} \mathsf{Poly}(y^{p[i]}, \sum_{j \in q(1)} y^{q[j]}) \\ &\cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]}\end{aligned}$$

The latter expression can be interpreted in terms of dependent products and sums: For every $i \in p(1)$ there exists $j \in q(1)$ with associated function $q[j] \to p[i]$. In other words, a morphism in Poly may be identified with a function $f : p(1) \to q(1)$ on positions, along with a family of functions $(f_i)_{i \in p(1)}$ where $f_i : q[f(i)] \to p[i]$ is a function on directions. This family can be expressed more concisely by assembling it into a dependent function $f^\sharp : (i \in I) \to q[f(i)] \to p[i]$.

1.1 Lenses in Poly

We can express lenses in Set within Poly as maps between special types of polynomials. Let us generalize the term *monomial* to include polynomials of the form $\sum_{i \in S} y^A \cong Sy^A$, where the direction sets are independent of the positions. Now consider a morphism

$$\binom{f^\sharp}{f} : A^+ y^{A^-} \to B^+ y^{B^-}.$$

This consists of an on-positions function $f : A^+ \to B^+$ and an on-directions dependent function $f^\sharp : (a^+ \in A^+) \to B^- \to A^-$. However, since the direction sets are independent of the positions, this is just a regular function and can be curried to $f^\sharp : A^+ \times B^- \to A^-$. We now recognize that this is just a lens

$$\binom{f^\sharp}{f} : \binom{A^-}{A^+} \leftrightarrows \binom{B^-}{B^+}.$$

1.2 Operations in Poly

Poly has four monoidal operations $+, \times, \circ$, and $\otimes$, where $\times$ and $\otimes$ are monoidal closed. The first two are simply the categorical coproduct and product, respectively. For $p = \sum_{i \in p(1)} y^{p[i]}$ and $q = \sum_{j \in q(1)} y^{q[j]}$ these are given by the familiar addition and multiplication of polynomials:

$$p + q = \sum_{i \in p(1)} y^{p[i]} + \sum_{j \in q(1)} y^{q[j]},$$

$$p \times q = \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]+q[j]}.$$

On the other hand, $\circ$ is just functor composition, which corresponds to variable substitution for polynomials. It can be computed as

$$p \circ q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y.$$

Finally, $\otimes$ is given by multiplying both the postitions and directions:

$$p \otimes q = \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]q[j]}.$$

In fact, if we tensor two lenses

$$\binom{f^\sharp}{f} : A_1^+ y^{A_1^-} \to B_1^+ y^{B_1^-}.$$

$$\binom{g^\sharp}{g} : A_2^+ y^{A_2^-} \to B_2^+ y^{B_2^-}.$$

we get a lens

$$(A_1^+ \times A_2^+) y^{A_1^- \times A_2^-} \to (B_1^+ \times B_2^+) y^{B_1^- \times B_2^-}$$

which recovers the usual tensor product of lenses.

1.3 Mode-dependence

We have seen that we can model lenses as polynomial morphisms

$$\binom{f^\sharp}{f} : A^+ y^{A^-} \to B^+ y^{B^-}.$$

Translating a lens representing a dynamical system thus yields a morphism $Sy^S \to Oy^I$. We obtain mode-dependent systems by generalizing the former to morphisms of the type $Sy^S \to q$, where q is an arbitrary polynomial. Let us see how this differs from what we had before.

To specify a map $\sum_{s \in S} y^S \cong Sy^S \to \sum_{j \in q(1)} y^{q[j]}$ we must give a function $f : S \to q(1)$ on positions and a dependent function $f^\sharp : (s \in S) \to q[f(s)] \to S$ on directions. We still think of the elements of $q(1)$ as outputs. The passback function $f^\sharp$ now additionally takes into account what the current output is, since it goes from $q[f(s)] \to S$ for each $s \in S$. In other words, the possible inputs a system can receive now depend on what the current output is.

2 Codesign in Poly

2.1 Preliminaries

Let P be a preorder whose objects we interpret as resources and whose relation $a \leq b$ means that being able to provide b implies being able to provide a. A feasibility relation $\Phi : F \nrightarrow R$ is then a monotone function $F^{\mathrm{op}} \times R \to \mathsf{Bool}$. Observe that

$$\begin{aligned} \hom(F^{\mathrm{op}} \times R, \mathsf{Bool}) &\cong \hom(R, \hom(F^{\mathrm{op}}, \mathsf{Bool})) \\ &\cong \hom(R, \mathsf{L}F) \end{aligned}$$

Consider a monotone map $\varphi : \mathsf{L}R \to \mathsf{L}F$ which preserves unions. This induces a monotone map $\bar{\varphi} : R \to \mathsf{L}F$ by $\bar{\varphi}(r) := \varphi(\downarrow r)$, where $\downarrow r$ denotes the lower closure of $\{r\}$. On the other hand, if we have a monotone map $\psi : R \to \mathsf{L}F$ we can extend it in a unique way to a monotone map $\tilde{\psi} : \mathsf{L}R \to \mathsf{L}F$ which preserves unions by setting $\tilde{\psi}(\downarrow s) := \psi(s)$ on principal lower sets. Then $\tilde{\psi}(S) = \tilde{\psi}(\bigcup_{s \in S} \downarrow s) = \bigcup_{s \in S} \tilde{\psi}(\downarrow s) = \bigcup_{s \in S} \psi(s)$. Since these operations are inverse to one another we have shown that

$$\hom(R, \mathsf{L}F) \cong \{\varphi \colon \mathsf{L}R \to \mathsf{L}F : \varphi \text{ monotone and preserves unions.}\}$$

Combining with the above, this shows that feasibility relation are just union preserving monotone functions $\mathsf{L}R \to \mathsf{L}F$. Moreover, the composition of such functions is again monotone and preserves unions. This justifies restricting out attention to monotone maps with this special property.

2.2 Basic Setup

Let $f_t : A_t \to B_t$ be a family of functions between sets indexed by $t \in T$. Moreover, let $u_t : A_t \to T$ be a family of function indexed by T. We interpret T as a (branching) timeline. If at time $t \in T$ we choose to transform $a \in A_t$ to $f_t(a) \in B_t$, then the timeline advances to $u_t(a) \in T$.

We can view this data as a directed graph with vertices (f_t, t) for $t \in T$ where any vertex (f_t, t) has edges emanating from it indexed by A_t. The vertices of this graph are interpreted as the transformations $f_t : A_t \to B_t$ that are available to us at time t. Using f_t on $a \in A_t$ advances time to $u_t(a)$ where we now have $f_{u_t(a)} : A_{u_t(a)} \to B_{u_t(a)}$ available to us.

Example 2.1: Linear time

To model linear time we set $T := \mathbb{N}$ and put $u_t : A_t \to T$ to be $u_t(a) = t+1$ for all $a \in A_t$ and $t \in T$. The resulting graph looks like this:

Example 2.2: Branching time

Our model for time can accommodate branching which depends on our choices. Let $T := \{(n,i) : n \in \mathbb{N}, 1 \leq i \leq 2^n\}$, and let $A_t := \{0,1\}$ for all $t \in T$. We can now consider the branching tree:

We now encode this data in a mode-dependent system in Poly with states

$$S := \{(f_{u_t(a)}, f_t(a)) : a \in A_t\}.$$

The input set at time t will be A_t and the output set B_t. This means that we are considering the system

$$Sy^S \to \sum_{t \in T} B_t y^{A_t}$$

with on-positions map $(f_{u_t(a)}, f_t(a)) \mapsto f_t(a) \in B_t$ and on directions map $(f_t, x) \mapsto a \mapsto (f_{u_t(a)}, f_t(a))$.

We consider some special cases to illustrate this definition.

Example 2.3: Time-invariant system

If $T := \{*\}$, our system consist of a single function $f : A \to B$ and $u_* : A \times T \to T$ is uniquely defined. Hence our states are simply

$$S := \{(f, f(a)) : a \in A\}.$$

Inputting $a_1 \in A$ into the system outputs $f(a_1)$ and updates the state to $f, f(a_1))$. Now inputting $a_2 \in A$ outputs $f(a_2)$ and updates the state to $f, f(a_2))$. So our system simply behaves like the function f and transforms a stream of inputs $(a_1, a_2, a_3, \ldots) \in A^{\mathbb{N}}$ into the stream $(f(a_1), f(a_2), f(a_3), \ldots) \in B^{\mathbb{N}}$.

Example 2.4: Linear time system with constant interface

Now consider $T := \mathbb{N}$ with $A_t = A$, $B_t = B$ for all t and set $u_t(a) = t+1$ for all $a \in A$. However let $f_t : A \to B$ be potentially different functions. Hence our states are simply

$$S := \{(f_{t+1}, f_t(a)) : t \in \mathbb{N}, a \in A\}.$$

Our systems always accepts inputs from A and outputs into B. Suppose we start in the state (f_0, x), where $x \in B$ is some default initial output. Then inputting $a_0 \in A$ into the systems will update the state to $(f_1, f_0(a_1))$ and cause it to output $f_0(a_0)$. Inputting $a_1 \in A$ will update the state to $(f_2, f_1(a_1))$ and output $f_1(a_1)$. So our system transforms inputs $(a_0, a_1, a_2, \ldots) \in A^{\mathbb{N}}$ into the stream $(f_0(a_0), f_1(a_1), f_2(a_2), \ldots) \in B^{\mathbb{N}}$.

Example 2.5: Linear time system with variable interfaces

We can change what input type the linear time system above accepts by letting the input and output sets A_t, B_t vary over time. Such a system transforms inputs $(a_0, a_1, a_2, \ldots) \in \prod_{t\in\mathbb{N}} A_t$ into the stream $(f_0(a_0), f_1(a_1), f_2(a_2), \ldots) \in \prod_{t\in\mathbb{N}} B_t$.

Example 2.6: Branching time system with constant interfaces

Let $A_t = \{0, 1\}$ and $B_t = B$ for all $t \int T := \{(n, i) : n \in \mathbb{N}, 1 \leq i \leq 2^n\}$. Our available transformations and their transitions are given by the branching graph:
This graph is simultaneously the transition diagram for our system, where the second element in the tuple is what the system is currently outputting. Given a vertex s representing a state, the available inputs correspond to the labeled edges emanating from s. If we input a to the system, following the edge with label a leads to the new state, along with its output.

2.3 Application to codesign

2.4 Recovering codesign problem