# Pairing Dynamic Systems

Solomon Bothwell

March 2024

"An $Ay^B$ Mealy Machine is the 'universal thing' that interacts with a $By^A$ Moore Machine. It is the universal thing that can be put together with a $By^A$ Moore Machine. They're not just two different definitions, they are dual in a certain sense." David Spivak

In past work encoding state machines as Co-Algebras in Haskell I have leveraged this idea to create protocol agnostic programs as Mealy Machines which can then be paired with a Moore Machine which manages a particular protocol.

```
annihilate :: Moore o i -> Mealy i o -> Fix Identity
annihilate (MooreM' moore) (MealyM' mealy) = Fix $ do
  (i, nextMoore) <- moore
  (o, mealy') <- mealy i

  pure $ annihilate' moore' mealy'
```

Using the tools presented in the Poly Workshop I wanted to understand more precisely what is going on with the idea of Pairings and how they might be applied more generally.

Given: 1. A Moore Machine with input set $A$ and output set $B$ $Sy^S \Rightarrow B^A$ 2. A Mealy Machine with input set $B$ and output set $A$ $(SB)y^S \Rightarrow Ay^1$.

We can create a pairing by the following transformations:

$$
\begin{aligned}
TBy^T &\Rightarrow Ay^1 && && (1)\\
Ty^T &\Rightarrow [By, Ay] && (P \otimes Q \to R \cong P \Rightarrow [Q, R]) && (2)\\
Ty^T &\Rightarrow [By^A, y] && ([Ay, By] \Rightarrow [Ay^B, y]) && (3)\\
STy^{ST} &\Rightarrow By^A \otimes [By^A, y] && (\text{The functorial action of} \otimes) && (4)\\
STy^{ST} &\Rightarrow y && (P \otimes [P, Q] \Rightarrow Q) && (5)
\end{aligned}
$$

This gives us a closed system where the two machines propagate messages back and forth indefinitely which we can demonstrate with Agda:

```
pair-machines : ∀{S T A B : Set} →S y^ S ⇒B y^ A →(T × B) y^ T ⇒A y^ ⊤→(S y^ S) ⊗(T y^ T) ⇒
    (B y^ A) ⊗[ B y^ A , Y]
pair-machines moore mealy =
  moore ⊗⇒⊗-to-hom (compute-tensor ;ₚ drop⊤--fiber ;ₚ mealy) ;ₚ ⊗-second hom-to-y

annihilate : ∀{S T A B : Set} →S y^ S ⇒B y^ A →(T × B) y^ T ⇒A y^ ⊤→S y^ S ⊗T y^ T ⇒Y
annihilate moore mealy = pair-machines moore mealy ;ₚ eval
```

If we wish to observe the interactions of our closed system we can substite some other polynomial for $y$ when "annihilating" our machines:

```
focus : ∀{A B C : Set} →(B × A →C) →[ B y^ A , Y} ] ⇒[ B y^ A , C y^ ⊤]
map-base (map-base (focus observe) p) b with p .map-fiber b tt
... | a = observe (b , a)
map-fiber (map-base (focus _) p) b tt = p .map-fiber b tt

witness : ∀{S T A B C : Set} →(B × A →C) →S y^ S ⇒B y^ A →(T × B) y^ T ⇒A y^ ⊤→S y^ S ⊗T y^
    T ⇒C y^ ⊤
witness observe moore mealy =
  pair-machines moore mealy ;ₚ ⊗-second (focus observe) ;ₚ eval
```

*witness* asks us to provide an observation function $B \times A \to C$ to monitor the interactions of the pairing. The observation function is used to generate a polynomial $\mathrm{Cy}^1$ which used as the interface to the dynamic system.

It turns out that this behavior is not particular to Mealy and Moore but rather that for any dynamic system $Sy^S \Rightarrow P$ we can create a paired dynamic system of the form $Sy^S \Rightarrow [P, y]$ which can then be annihilated via *eval*.

```
annihilate' : ∀{S T : Set} {P : Poly} →S y^ S ⇒P →T y^ T ⇒[ P , Y] →S y^ S ⊗T y^ T ⇒y
annihilate' machine dual =
  machine ⊗⇒dual ;ₚ eval
```

This allows us to create arbitrary closed loop dynamic systems whose interactions can then be extracted via an observation function and *witness*. Leaking observations could allow communication between multiple isolated dynamic systems.

As a further exploration I would like to look into the use of effects inside these closed loop systems. By leveraging the state management inherent in these machines we can ensure the sequencing of IO effects ordinarily done via monads in pure functional languages. This would allow us to create closed loop dynamic systems which can perform real work in the world.