Functional reactive Twitter bots

Omar Rizwan

July 22, 2014 at LambdaJam Chicago

What does *this* have to do with functional programming?

Signals

- What is a Signal?

Signals

- What is a Signal?
    - A Signal is a time-varying value.

Signals

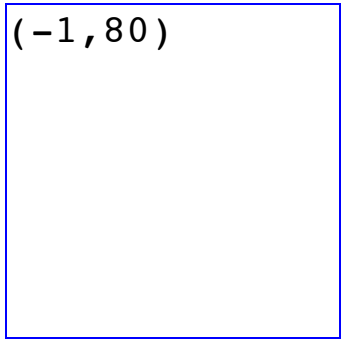
- Suppose we have `Mouse.position` of type `Signal (Int, Int)`.

Signals

- Suppose we have `Mouse.position` of type `Signal (Int, Int)`.
- We can see `Mouse.position` as a *function* from `Time -> (Int, Int)`.

Signals

- Suppose we have `Mouse.position` of type `Signal (Int, Int)`.
- We can see `Mouse.position` as a *function* from `Time -> (Int, Int)`.
- At a certain time, `Mouse.position` gives us a certain value:



Operating on Signals

- One other way to see a Signal, in Elm, is as a *stream of events*.

Operating on Signals

- `Mouse.position` is like a list that extends into the future:

```
(507,83)
(473,71)
(452,67)
(446,63)
(446,63)
(445,61)
(433,63)
```

Operating on Signals: lifting

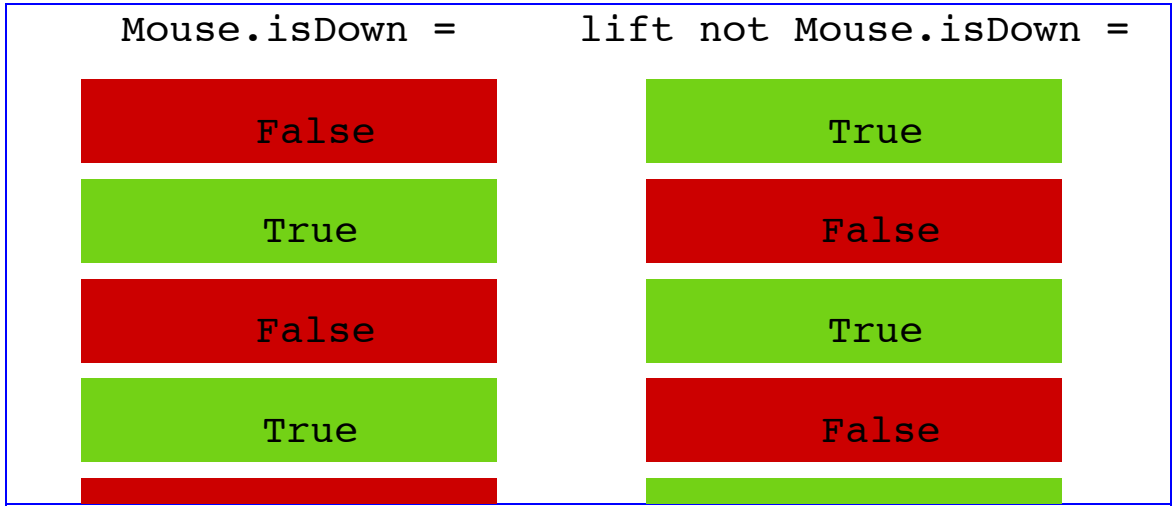
- We use `map` to make a function `a -> b` into a function `[a] -> [b]`
- Similarly, we use `lift` to make a function

```
a -> b
```

into

```
Signal a -> Signal b
```

Operating on Signals: lifting



What does this have to do with Twitter?

Twitter feeds are like Signals!

- Represent an individual tweet as a value of type `StatusUpdate a`

Twitter feeds are like Signals!

- Represent an individual tweet as a value of type `StatusUpdate a`
- Then we'll represent a Twitter feed as `Signal (StatusUpdate a)`

The Counter bot

- We need to generate a Signal of status updates

```
module Bot where

import Birdhouse as BH

port updates : Signal (BH.StatusUpdate {})
port updates = ???
```

The Counter bot

- We need to generate a Signal of status updates
- Each status update has the previous update's number + 1

What are our tools?

Starting with a primitive Signal: a tick

- We want a Signal that ticks every time a fixed time interval passes
- The `every` function

```
every : Time -> Signal Time
```

Takes a time interval t. The resulting signal is the current time, updated every t.

- Pass it a `Time` to get the Signal we'll start with

```
second : Time
```

```
minute : Time
```

```
hour : Time
```

Starting with a primitive Signal: a tick

- `every second` looks like this:

```
1406690897876
1406690896875
1406690895874
1406690894874
1406690893873
```

Start `every second`

Transforming `every second`: ticks to counting

- We need to replace those timestamps with counting!
- The `count` function

```
count : Signal a -> Signal Int
```

Count the number of events that have occurred.

Transforming `every second`: ticks to counting

- Now we have `count (every second)`:

every second =	count (every second) =
1406691171535	492
1406691170090	491
1406691169090	490
1406691168089	489
1406691167088	488

Start `count (every second)`

Transforming `count (every second)`: counting to tweets

- Now we want to turn those counts from `count (every second)` into tweets.

Transforming `count (every second)`: counting to tweets

- Now we want to turn those counts from `count (every second)` into tweets.
- First, lift the `show` function to turn the counts into Strings:

```
show : a -> String

lift show : Signal a -> Signal String
```

Transforming `count (every second)`: counting to tweets

- Now we want to turn those counts from `count (every second)` into tweets.
- Then, lift the `update` function from the Birdhouse library:

```
update : String -> StatusUpdate {}

lift update : Signal String -> Signal (StatusUpdate {})
```

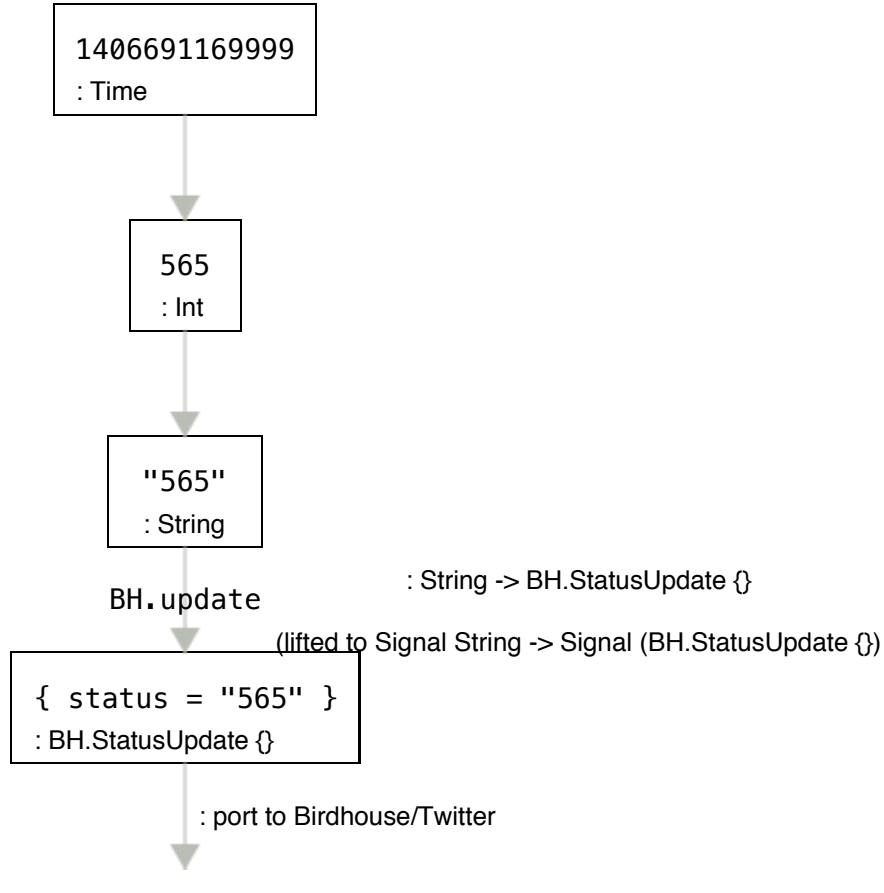
Our final Signal

```
module Bot where

import Birdhouse as BH

port updates : Signal (BH.StatusUpdate {})
port updates = lift BH.update (lift show ((count (every second))))
```

Our final Signal

```
module Bot where

import Birdhouse as BH

port updates : Signal (BH.StatusUpdate {})
port updates = BH.update . show <~ count (every second)
```

```
port updates = BH.update . show <~ count (every second)
```



A more complex bot: @EveryMNLake

List of lakes in Minnesota

From Wikipedia, the free encyclopedia

This is a **list of lakes in Minnesota**. Minnesota is known as *The Land of 10,000 Lakes* and officially there are 11,842 lakes more than ten acres (40,000 m²) in size.[1] The prevalence of lakes has generated many repeat names. For example, there are more than 200 Mud Lakes, 150 Long Lakes, and 120 Rice Lakes.[2] All but four Minnesota counties (Mower, Olmsted, Pipestone and Rock) contain at least one natural lake. Minnesota's lakes and rivers provide more shoreline than there is in California, Florida, and Hawaii combined.[3] [4]

This list is incomplete; you can help by expanding it.

Lake	County	Nearby Town	Size (acres)	Littoral zone (acres)	Max depth (ft)	Water Clarity (ft)
Aaron	Douglas	Millerville	545	417	16	8
Abbey	Becker	Detroit Lakes	269	269	7	N/A
Acorn	Becker	Frazee	154	N/A	N/A	N/A
Ada	Cass	Backus	974	424	60	15
Adley	Otter Tail	Parkers Prairie	239	238	20	2.6
Agnes	St. Louis	Ely	984	453	30	7
Aitkin	Aitkin	Libby	487	453	35	6.3
Albert Lea	Freeborn	Albert Lea	2654	2654	6	1
Alexander	Morrison	Randall	2708.72	842	64	10.9
Alice	Hubbard	Lake George	121	96	21	6.5

Tweets Tweets and replies

Every Minnesota Lake @EveryMNLake · 1h
Lake Rose

Every Minnesota Lake @EveryMNLake · 4h
Lake White Pine

Every Minnesota Lake @EveryMNLake · 7h
Lake Three Island

Every Minnesota Lake @EveryMNLake · 10h
Lake Hattie

Every Minnesota Lake @EveryMNLake · 13h
Lake Fox

A more complex bot: @EveryMNLake

```
port updates = lakesJsonSigResp
             |> lift U.respToMaybe
             |> lift (U.concatMap Json.fromString)
             |> lift (U.map toLakes)
             |> lift (U.map <| map toGeoUpdate . drop pos . fst . PureRandom.shuffle seed)
             |> U.extract []
             |> U.spool (every <| 3 * hour)
```

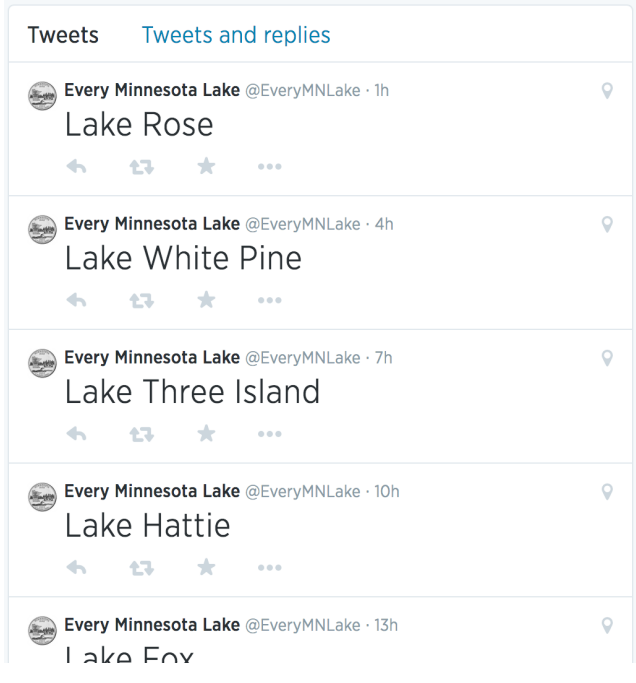
A more complex bot: @EveryMNLake

```
port updates = lakesJsonSigResp
             : Signal (Http.Response String)
             |> lift U.respToMaybe
             : Signal (Maybe String)
             |> lift (U.concatMap Json.fromString)
             : Signal (Maybe Json.Value)
             |> lift (U.map toLakes)
             : Signal (Maybe [Lake])
             |> lift (U.map <| map toGeoUpdate . drop pos . fst . PureRandom.shuffle seed)
             : Signal (Maybe [GeoUpdate (StatusUpdate {})])
             |> U.extract []
             : Signal [GeoUpdate (StatusUpdate {})]
             |> U.spool (every <| 3 * hour)
             : Signal (GeoUpdate (StatusUpdate {}))
```

A more complex bot: @EveryMNLake

```
port updates = lakesJsonSigResp                Http.Success "[{\"county\":\"Douglas\",\"si..."
  : Signal (Http.Response String)
  |> lift U.respToMaybe                        Just "[{\"county\":\"Douglas\",\"si..."
  : Signal (Maybe String)
  |> lift (U.concatMap Json.fromString)        Just (Json.Array [Json.Object {"county": "D..."},
  : Signal (Maybe Json.Value)
  |> lift (U.map toLakes)                      Just [{ name = "Aaron", loc = { lat = ... }}, ...
  : Signal (Maybe [Lake])
  |> lift (U.map <| map toGeoUpdate . drop pos . fst .
                    PureRandom.shuffle seed) Just [{ status = "Lake Elbow", lat = ... }, ...]
  : Signal (Maybe [GeoUpdate (StatusUpdate {})])
  |> U.extract []                              [{ status = "Lake Elbow", lat = ... }, ...]
  : Signal [GeoUpdate (StatusUpdate {})]
  |> U.spool (every <| 3 * hour)               { status = "Lake Elbow", lat = ... }
  : Signal (GeoUpdate (StatusUpdate {}))
```

A more complex bot: @EveryMNLake



Combinators

- Other people's tweet:
  `Tweet`

Combinators

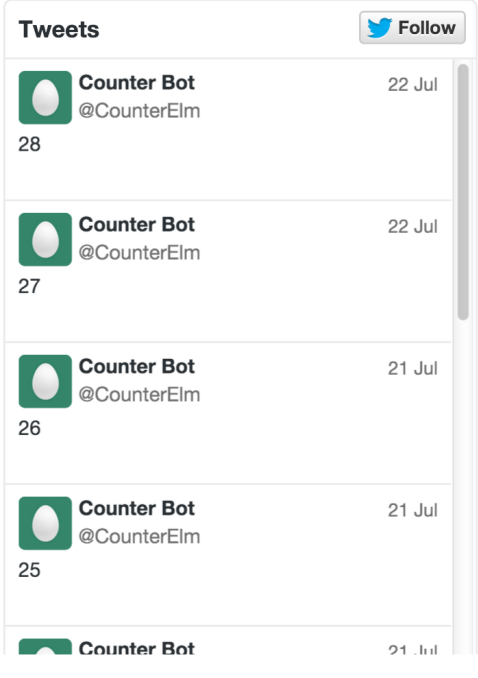
- Other people's tweet:
  `Tweet`
- Other people's Twitter feeds:
  `Signal (Maybe Tweet)`

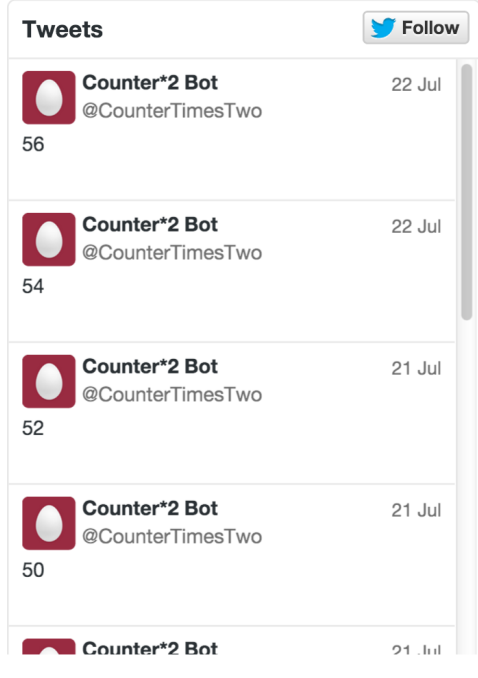
Combinators

- Other people's tweet:
  `Tweet`
- Other people's Twitter feeds:
  `Signal (Maybe Tweet)`
- Map:
  ```
  (String -> String) -> Signal (Maybe (StatusUpdate {}))
  -> Signal (Maybe (StatusUpdate {}))
  ```

Combinators: x -> 2 * x

```
port updates = BH.map (\s -> show <| maybe 0 (\x -> x * 2) <| String.toInt s)
               <| BH.toUpdates counter
```



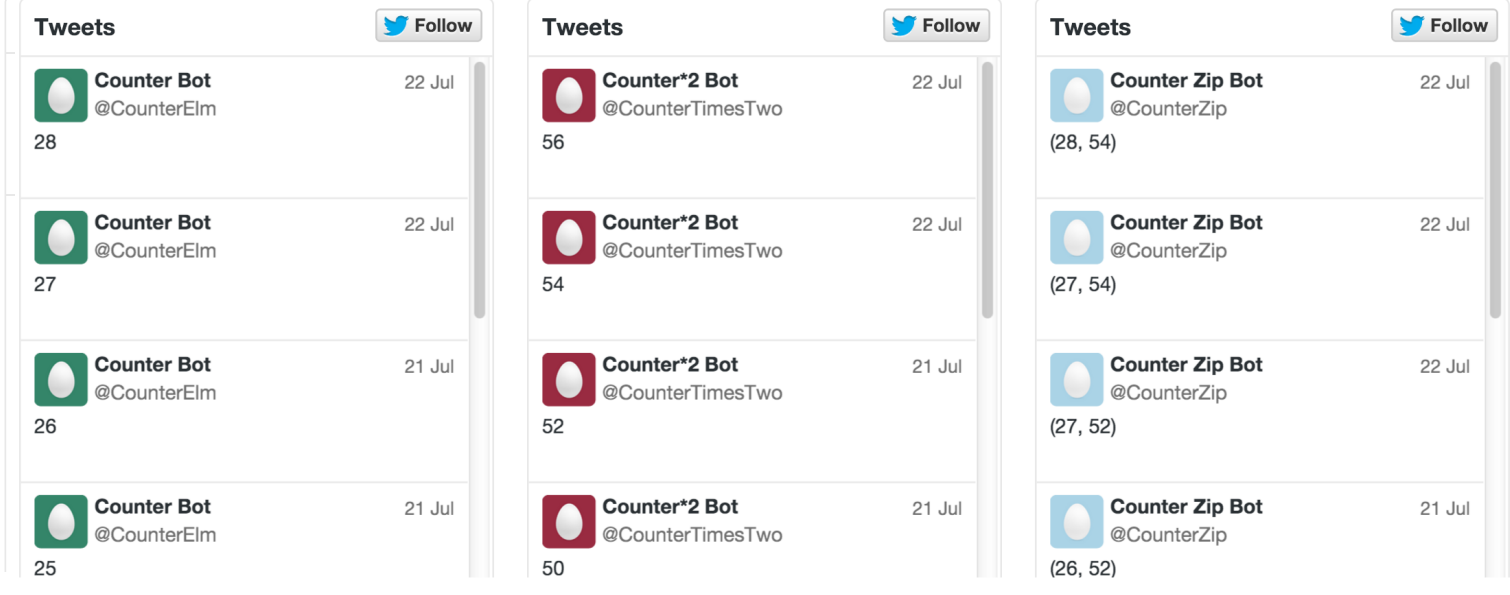


Combinators: x -> 2 * x

```
module Bot where

import Utils as U
import String
import Maybe
import Birdhouse as BH

port getTweetsFrom : Signal [BH.ScreenName]
port getTweetsFrom = sampleOn (every (2 * minute)) <| constant ["CounterElm"]

port tweets : Signal (Maybe (BH.ScreenName, BH.Tweet))

counter : Signal (Maybe BH.Tweet)
counter = tweets `BH.newFromUser` "CounterElm"

port updates : Signal (Maybe { status : String })
port updates = BH.map (\s -> show <| maybe 0 (\x -> x * 2) <| String.toInt s)
               <| BH.toUpdates counter

main = BH.previewStreamM updates
```

Combinators: x y -> (x, y)

```
port updates = (\mc mctt ->
                   case (mc, mctt) of
                     (Just c, Just ct) -> Just { status = "(" ++ c.status ++ ", " ++ ct.status
                     otherwise -> Nothing)
                <~ BH.toUpdates counter ~ BH.toUpdates counterTimesTwo
```



Combinators: x acc -> x + acc

Past-Dependence

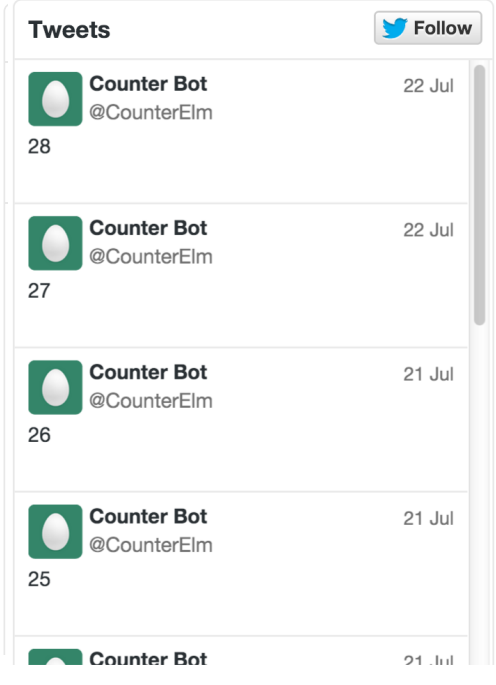
```
foldp : (a -> b -> b) -> b -> Signal a -> Signal b
```

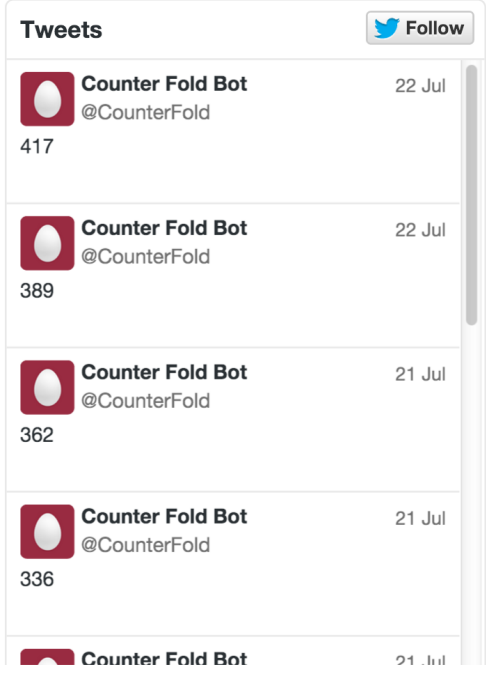
Create a past-dependent signal. Each value given on the input signal will be accumulated, producing a new output value.

For instance, `foldp (+) 0 (fps 40)` is the time the program has been running, updated 40 times a second.

Combinators: x acc -> x + acc

```
port updates = lift (BH.update . show)
               <| BH.fold (\s acc -> maybe 0 (\x -> x + acc) <| String.toInt s) 0
               <| BH.toUpdates counter
```





Problems

- Initial value needs to be defined, forcing us to strew `Maybe` everywhere
- No way to introduce static data at start of program without making HTTP request (a Signal)
- Twitter doesn't push events to us, we have to poll
- Type alias problems in ports
- Difficult to test!

Thanks!

- Omar Rizwan (@rsnous)
- http://rsnous.com
- https://github.com/osnr/Birdhouse