Generating a Functional API for Web Animations

Shane Stephens

Preamble

Two weeks ago:

Hi Guys,

According to the roster:

http://fp-syd.ouroborus.net/wiki/WikiStart

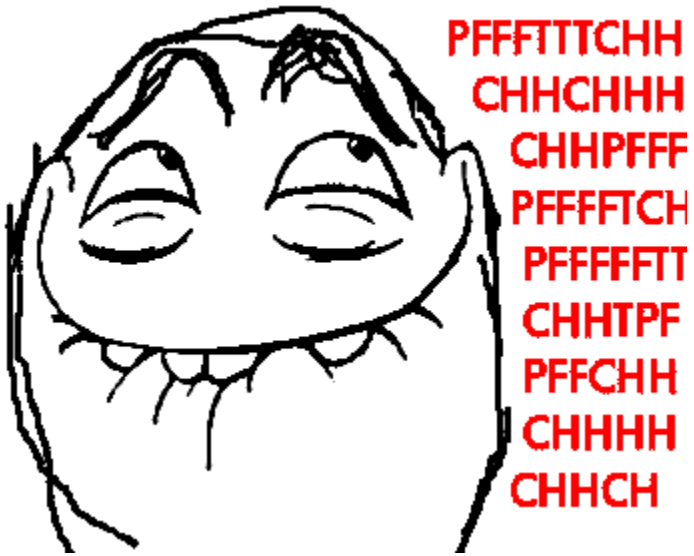
you guys are due to present next month. However, this month's speaker's line up is turning into a bit of a disaster so I was wondering if one or two of you would be available to bring your presentations forward a month.

You still have a little over 2 weeks to prepare.

Cheers,
Erik

Preamble

Me:


PFFFTTTCHH
CHHCHHH
CHHPFFF
PFFFFTCH
PFFFFFTT
CHHTPF
PFFCHH
CHHHH
CHHCH

Preamble

But then:

ICFP 2013

The 18th ACM SIGPLAN International Conference on Functional Programming

Photo credit: Werner Kunz

Boston, Massachusetts; September 25 – 27, 2013

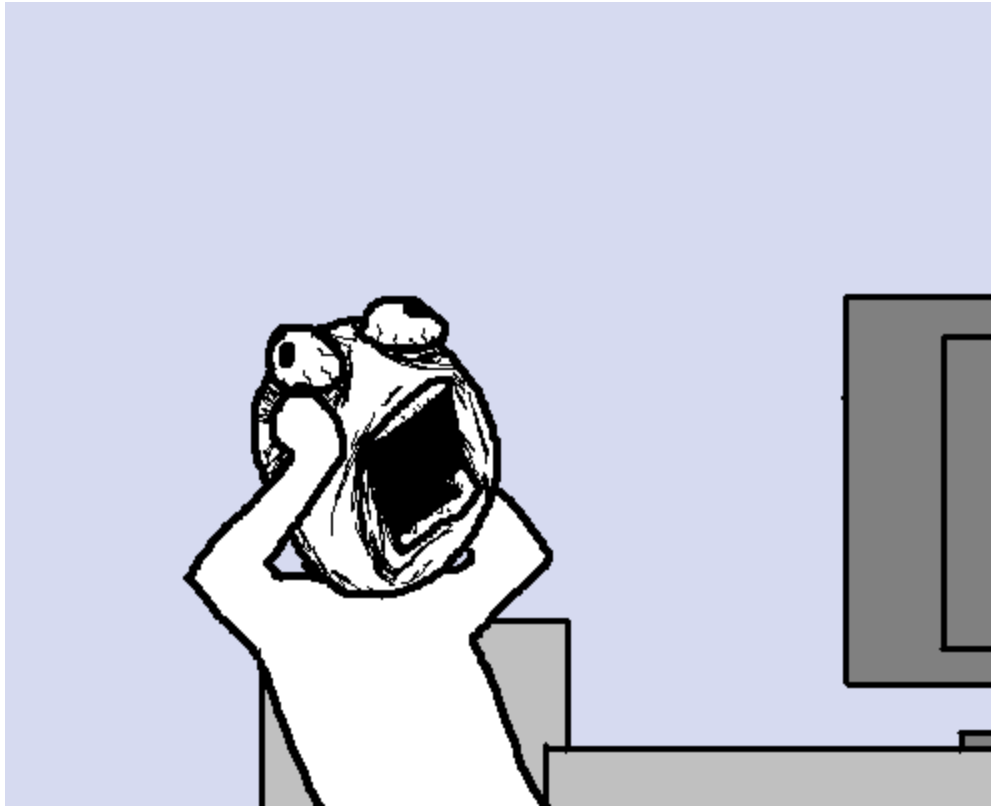
Affiliated events: September 22 – 24 and 28

25th September

- George Roldugin
- Shane Stephens
- Ben Sinclair

Preamble

Me:



Web Animations

- Unify SVG & CSS Animations
- Provide nice JavaScript API
- Spec:
  - http://www.w3.org/TR/css3-animations/
- Polyfill:
  - http://github.com/web-animations/web-animations-js

Demo

Mission

Implement Web Animations bindings for a Haskell → JavaScript compiler

tl;dr

- I failed

tl;dr

- I failed
- I learned two things
  - There are a *lot* of “Haskell” to JavaScript compilers
  - Most of them hate the Web
- I started writing my own language!
- I came up with an interesting idea

UHC-JS

The Utrecht Haskell Compiler JavaScript backend

Design philosophy:

- “Getting rid of programming JavaScript”
- “The JavaScript problem”

Documentation: **very little**

Compiles “most of hackage”

Provides a reasonably good FFI

From

```
new Animation(
  document.querySelector(‘.anim’),
  [
    {left: ‘100px’},
    {left: ‘200px’, width: ‘400px’}
  ],
  1
);
```

To

```
foreign import js "new Animation(%1, %2, %3)" animate_ ::
    Element -> JSList JSMap -> Double -> IO Animation
```

To

```
data JSMap
data JSList a
```

```
foreign import js "new Animation(%1, %2, %3)" animate_ ::
    Element -> JSList JSMap -> Double -> IO Animation
```

To

```
data JSMap
data JSList a

-- utilities for dealing with string maps
foreign import js "newMap()" newMap :: IO JSMap
foreign import js "setMap(%1, %2, %3)" setMap :: JSMap ->
JSString -> JSString -> IO ()

-- utilities for dealing with lists
foreign import js "newList()" newList :: IO (JSList a)
foreign import js "%1.push(%2)" push :: JSList a -> a -> IO ()

foreign import js "new Animation(%1, %2, %3)" animate_ ::
    Element -> JSList JSMap -> Double -> IO Animation
```

(and)

```
<script>
function newMap() {
  return {};
}

function setMap(map, k, v) {
  map[k] = v;
}

function newList() {
  return [];
}
</script>
```

(and)

```
main = do
    doc <- document
    elem <- querySelector doc $ toJS “anim”
    fromMap <- newMap
    setMap fromMap “left” “100px”
    toMap <- newMap
    setMap toMap “left” “200px”
    setMap toMap “width” “400px”
    list <- newList
    push list fromMap
    push list toMap
    animate elem list 1
```

(and)

```
main = do
    doc <- document
    elem <- querySelector doc $ toJS “anim”
    fromMap <- newMap
    setMap fromMap “left” “100px”
    toMap <- newMap
    setMap toMap “left” “200px”
    setMap toMap “width” “400px”
    list <- newList
    push list fromMap
    push list toMap
    animate elem list 1
```

Yeah, *JavaScript* is the horrible language.

ToJS, FromJS

Typeclasses to make crossing the barrier easier

```
animate elem [[(“left”, “100px”)],
   [(“left”, “200px”), (“width”, “400px”)]] 1
```

The Barrier is the Problem

- JavaScript is treated as a poor substitute, to be used only when absolutely required
- All DOM, CSS, and feature access is provided via JavaScript APIs

The Barrier is the Problem

- JavaScript is treated as a poor substitute, to be used only when absolutely required
- All DOM, CSS, and feature access is provided via JavaScript APIs

- Consequence: all features should be avoided as much as possible

Impedance Mismatches

Javascript:

```
a.concat(b, c, d, ...)
```

Haskell:

```
concat1 a b
concat2 a b c
concat3 a b c d
...
```

Impedance Mismatches

Javascript:

```
new Animation(target, keyframes, 1);
new Animation(target, path, 1);
new Animation(target, custom, {..});
```

Haskell:

```
animation target frames 1
pathAnimation target path 1
customAnimationWithTiming
   target custom timing
```

Impedance Mismatches

```
data AnimationEffect
   = Keyframes [[(String, String)]]
   | Path String
   | Custom (IO (Double -> Int))

data Timing
   = Duration Double
   | Dictionary [(String, String)]

animation :: Element -> AnimationEffect ->
                Timing -> IO Animation
```

Elm

http://elm-lang.org

- A ‘functional reactive programming language that compiles to HTML, CSS, and JS’
- good documentation
- lots of examples
- online editor

Elm

```
-- MODEL
data Update = Click (Float,Float) | TimeDelta Time

floatify (x,y) = (toFloat x, toFloat y)
input = let clickPos = floatify <~ sampleOn Mouse.clicks Mouse.position
        in  merge (Click <~ clickPos)
                  (TimeDelta <~ (40 `fpsWhen` (second `since` clickPos)))

-- UPDATE
step inp ((tx,ty),(x,y)) =
    case inp of
      Click t  -> (t, (x,y))
      TimeDelta d -> ((tx,ty), ( x + (tx-x) * (d/100) ,
                                 y + (ty-y) * (d/100) ))
```

Elm

```
-- DISPLAY
greenGrad = radial (0,0) 10 (7,-5) 30
              [(0, rgb 167 211 12), (0.9, rgb 1 159 98), (1, rgba 1 159 98 0)]

follower (w,h) (target,(x,y)) =
  collage w h [ circle 100 |> gradient greenGrad
                                    |> move (x - toFloat w / 2, toFloat h / 2 - y)
]


main = follower <~ Window.dimensions
                 ~ foldp step ((0,0),(0,0)) input
```

JavaScript equivalent

```
document.addEventListener(‘click’, function(e)
{
   document.timeline.play(new Animation(
      target, {left: e.x, top: e.y}, 1));
}
```

On the other hand...

```
<style>
.anim {
    position: absolute;
    top: 0px;
    left: 0px;
    background: radial-gradient(closest-side at 20px 20px,
        0% rgb(167,211,12), 30% rgb(167,211,12),
        75% rgb(1,159,98), 80% rgba(1,159,98,0),
        100% rgba(1,159,98,0));
    width: 100px;
    height: 100px;
    border-radius: 50px;
}
</style>
```

Seems like a good fit

- Provide some nice animations primitives to Elm
- Explore Web Animations in a reactive setting

Seems like a good fit

- Provide some nice animations primitives to Elm
- Explore Web Animations in a reactive setting

However ...

- Elm to JS is message-based
- Elm is another replace-the-world abstraction for the web

Roy

- Much saner approach
  - the language is almost syntactic sugar around JS
  - JS functions directly available
  - Roy types are (almost) JS types
  - lots of nice features

So I can do

```
map console.log (document.querySelectorAll ‘div’)
```

Roy

- But…
  - almost too close to JavaScript
  - no HOF
  - no ADTs
  - no list pattern matching
  - no operators

So I *can't* do

```
let map f [] = []
let map f (h:t) = (f h):(map f t)
```

Some things

- JavaScript is flexible
  - ADTs are possible (ristretto.js)
  - Partial application is possible (ristretto.js)
  - Type inference is (sometimes) possible

- Functional language data representations are flexible
  - are not typically type-tagged
  - are typically fungible

Enter: krazy

Caution: vaporware!

- Functional types directly exposed in JS
  - e.g. lists _are_ arrays
- “Compiler” implemented in JS using PEG and (maybe) esprima
- Full support for ADTs, HOFs, pattern matching etc.
- JS interop constrained through type signatures
- Type inference

Demo

LiNkY

A Krazy Idea

- Lists are ADTs

```
data List a = Cons a (List a) | Nil
[1,2,3] -> Cons 1 (Cons 2 (Cons 3 Nil)))
map f [] -> map f Nil
map f (h:t) -> map f (Cons h@a t@b)
```

- List constructors can be JS functions

```
function Nil() { return []; }
function Cons(a, b) {
   return [a].concat(b);
}
```

A Krazy Idea

- Pattern matches can be JS functions

```
function matchesNil(x) {
   return x.length == 0;
}
function matchesCons(x) {
   if (x.length > 0) {
      return [x[0], x.slice(1)];
   }
   return undefined;
}
```

A Krazy Idea

- We did this with ristretto.js

```
var BTree = D("BTree = Empty | Leaf Int | Node BTree BTree");

function depth (tree) {
    var m = BTree.matcher(depth, "m");
    m.Empty()(function () { return 0; });
    m.Leaf("_")(function () { return 1; });
    m.Node("a", "b")(function() {
        return Math.max(depth(m.a), depth(m.b)) + 1;
    });
    return m(tree);
}
```

A Krazy Idea

Why not open this up to library authors?

e.g. Web Animations has side-effect free constructors for Animations, Effects, TimingGroups, etc.

```
Animation = A Element Effect Timing
TimingGroup = TG Type [TimingGroup] | TGA Animation

play :: TimingGroup -> IO Player
```

Questions?