Functional Reactive Programming

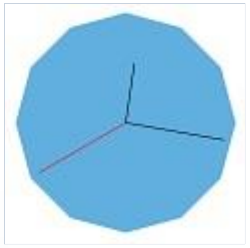
Elegant Interaction with Elm

Evan Czaplicki

Premise

- GUI programming is all about time-varying values
  - Mouse, Keyboard, Touch
  - Time, asynchronous HTTP, and file I/O
- We need a high-level way to talk about time
  - Mouse updates can happen *at the same time* as file I/O
  - This HTTP requests happens *after* that request
  - Only send requests if the query is stable *for half a second*
- The event loop is not the only answer!

Live Examples of Elm



Elm

- Striking a balance between flexibility and structure
  - Reduce program complexity at the level of language design
  - Choose the right abstractions; two good ideas may combine terribly!
- Language features:
  - Functional Reactive Programming
  - Strong / Static Typing
  - Extensible Records with structural typing
  - Purely functional graphics / Markdown support
  - Module system and core libraries
- Social features:
  - An open source project with tons of examples and an online editor
  - Fun and lively community working on a bunch of cool projects
  - “Asynchronous FRP for GUIs” will appear at PLDI 2013

Functional Reactive Programming

The Fundamental Ideas

Signals

- Values that change over time:
  - Mouse, Keyboard, Touch
  - Time, AJAX, Web Sockets, File I/O

- For example:
  - `Mouse.position` is the position of the mouse *right now*.
  - Anything that depends on `Mouse.position` is updated automatically.
  - Let's see it!

Signals

```
Mouse.position : Signal (Int,Int)
```

- Signals always have a current value.
- Signals change discretely, only as needed.
  - If the signals stay the same, there is nothing to compute.
  - Seems obvious, but this is not true in the majority of prior implementations.
- Rules about the order of events:
  - Order is always maintained within a signal.
  - Order does not *need* to be maintained between signals (concurrency!)

Automatic Updates

How do we apply a function to a signal?

```
lift : (a → b) → Signal a → Signal b
```

```
asText : a → Element
```

```
lift asText Mouse.x : Signal Element
asText <~ Mouse.x   : Signal Element
```

An `Element` that changes over time is an animation!

Combining Signals

What if something needs to depend on multiple signals?

```
display : (Int,Int) → (Int,Int) → Element
display (w,h) (x,y) = ...

lift2 : (a → b → c) → Signal a → Signal b → Signal c

main = lift2 display Window.dimensions Mouse.position
main = display <~ Window.dimensions ~ Mouse.position
```

Let's see it!

Stateful Signals

How can a signal depend on the past?

`foldl`	"fold from the left"
`foldr`	"fold from the right"
`foldp`	"fold from the past"

```
foldp : (a → b → b) → b → Signal a → Signal b
```

```
clickCount = foldp (\_ c → c+1) 0 Mouse.clicks
```

Let's see it!

Purely Functional Games

- Any game has four major components:
  - Input
  - Model
  - Update
  - Display
- In Elm, all 4 component *must* be cleanly separated!
  - The game will be unwritable without this structure.
  - Good architecture is easy when there is no other option.

Purely Functional Games

Let’s write one!

Asynchrony

Asynchronous HTTP and WebSockets without callbacks.

```
send : Signal (Request a) → Signal (Response String)


data Response a =
    Success a | Waiting | Failure Int String
```

- The requests are decoupled from the responses.
  - The response updates whenever it is ready.
  - Let’s see it!
  - And a more complex one.

Review of FRP

- Any time-varying value is a `Signal`
- Built around two core functions: `lift` and `foldp`
- Asynchronous signals allow us to escape from Callback Hell
- Allows highly interactive applications with very little code
- Makes web programming enjoyable!

Holy Cow

Looks like we have extra time

Filtering Signals

Sometimes you don’t want every single value.

```
keepIf    : (a → Bool) → a → Signal a → Signal a
keepWhen : Signal Bool → a → Signal a → Signal a

sampleOn : Signal a → Signal b → Signal b

dropRepeats : Signal a → Signal a
```

Time Signals

Working with time is extremely important in games!

```
every : Time   → Signal Time
fps   : Number → Signal Time

timestamp : Signal a → Signal (Time,a)
delay     : Time → Signal a → Signal a
```

Thank you!

And remember to try out Elm!

Questions?