Functional Reactive Programming

What is reactive programming

Lecture based on:
- 2008 lecture by Lennart Ohlsson
- Edward Amsden: “A Survey of Functional Reactive Programming” (look for a video from September 2012, recommended)
- Conal Elliot's (Yale, Yampa) slides on FRP
- https://github.com/gelisam/frp-zoo

EDAN40: Functional Programming
Functional Reactive Programming

Jacek Malec
Dept. of Computer Science, Lund University, Sweden
May 18th, 2020

Basic Concepts of Reactivity

Reactivity ≡ time-dependent responsiveness
- behaviours (signals, fluents, streams) – functions of time
- occurrences – elements in Val × Time
- events – sets of occurrences (lists in our case)

An interesting issue: continuous vs. discrete time
Approaches to reactivity:
- “embedding” (classical)
- signal-based
- n-ary FRP

Semantics vs. interpretation

Behaviours

newtype Behavior a =
  Behavior { at :: Time -> a }

myName :: Behavior String

myName ‘at‘ yesterday
Behaviours

type Behavior a = Time -> a  -- conceptually
type Time = Float

-- lifting many functions from a to Behavior a
timeTrans :: Behavior Time -> Behavior a
  -> Behavior a
timeTrans f a t = a (f t)

integral :: Behavior Float -> Behavior Float
derivative :: Behavior Float -> Behavior Float

Jacek Malec, http://rss.cs.lth.se

Events

type Event a = [(a, Time)]  -- conceptually

untilB :: Behavior a -> Event (Behavior a)

switch :: Behavior a -> Event (Behavior a)
  -> Behavior a

-- Event mapping
(->>) :: Event a -> b -> Event b
(==>) :: Event a -> (a -> Event b) -> Event b

-- Event choice
(.|.) :: Event a -> Event a -> Event a

Jacek Malec, http://rss.cs.lth.se

Events, cntd

-- Snapshot events
snapshot_ :: Event a -> Behavior b -> Event b

-- Predicate event
when :: Behavior Bool -> Event ()

-- other
step :: a -> Event a -> Behavior a
stepAccum :: a -> Event (a -> a) -> Behavior a
withElem_ :: Event a -> [b] -> Event b

gloss
paddle.hs

Jacek Malec, http://rss.cs.lth.se

A graphics library

gloss
paddle.hs

Jacek Malec, http://rss.cs.lth.se
**Functional Reactive Programming**

### Implementation issues

reactive banana library offers constructors:

- `filter :: (a -> Bool) -> Event a -> Event a`  
- `accumE :: a -> Event (a -> a) -> Event a`  
- `stepper :: a -> Event a -> Behavior a`  
- `apply :: Behavior (a -> b) -> Event a -> Event b`

instance `Functor Event`  
instance `Functor Behavior`  
instance `Applicative Behavior`  
instance `Monoid (Event a)`

---

Jacek Malec, http://rss.cs.lth.se

---

**Functional Reactive Programming**

### Signal functions

**Signal** - primitive concept

**SF** - primitive type:

-- informally  
`SF a b = Signal a -> Signal b`

and

-- informally again  
`Signal a = Time -> a`

Time is considered to be real-valued.

---

Jacek Malec, http://rss.cs.lth.se

---

**Functional Reactive Programming**

### Arrows

Arrow `a b c` represents a process that takes as input something of type `b` and outputs something of type `c`.

`arr` builds an arrow from a function:

```
arr :: (Arrow a) => (b -> c) -> a b c
```

Arrows are composed with `>>>`, while `first` and `second` create new arrows:

```
>>> :: (Arrow a) => a b c -> a d c -> a b d
first :: (Arrow a) => a b c -> a (b, d) (c, d)
second :: (Arrow a) => a b c -> a (d, b) (d, c)
```

---

Jacek Malec, http://rss.cs.lth.se

---

**Functional Reactive Programming**

### Signal function primitives

Point-wise application:

```
arr :: (a -> b) -> SF a b
arr f = \s -> \t -> f (s t)
```

Signal composition:

```
>>> :: SF a b -> SF b c -> SF a c
sf1 >>> sf2 = \s -> \t -> (sf2 (sf1 s)) t = sf2 \ sf1
```

Other compositions:

```
first :: SF a b -> SF (a, c) (b, c)
second :: SF a b -> SF a (b, c)
loop :: SF (a, c) (b, c) -> SF a b
```
Doing something with it:

\[
\text{integral} :: \text{Fractional } a \Rightarrow \text{SF } a \Rightarrow a
\]

is a \textit{stateful primitive} (depends not only on \( t \) but maybe also on \([0, t]\)).

The \text{integral} primitive computes the time integral of its input signal:

\[
\text{localTime} :: \text{SF } a \Rightarrow \text{Time}
\]

\[
\text{localTime} = \text{const } 1.0 \ggg \text{integral}
\]

Then we introduce events … (for more see the AFRP papers).

### Where to go from here?

- Arrows (a generalisation of monads)
  - Hughes@CTH (first paper on Arrows: 2000)
  - AFRP = Arrowized FRP (first paper on AFRP by Hudak et al.: 2002)
- Applicative functors (weaker than monads, no value passing)
- Various signal-functions-semantics implementations (see the survey paper)
- Actively developed libraries: Yampa (unary FRP a la Yale), reactive-banana (1.2.0.0 as of May 15th, 2018)
- Lots to do …