The essence and origins of FRP

Conal Elliott

June 13, 2015

What is FRP?

FRP's two fundamental properties

- *Continuous* time. (Natural & composable.)
- Precise, simple denotation. (Elegant & rigorous.)

FRP's two fundamental properties

- *Continuous* time. (Natural & composable.)
- Precise, simple denotation. (Elegant & rigorous.)

Deterministic, continuous "concurrency".

Warning: most modern "FRP" systems have neither property.

Why continuous & infinite (vs discrete/finite) time?

Why continuous & infinite (vs discrete/finite) time?

- Transformation flexibility with simple & precise semantics
- Quality/accuracy
- Efficiency (adapative)
- Modularity/composability:
  - Fewer assumptions, more uses (resolution-independence).
  - More info available for extraction.
  - Same benefits as pure, non-strict functional programming.
    See *Why Functional Programming Matters*.
- Integration and differentiation: natural, accurate, efficient.
- Reconcile differing input sampling rates.

Why continuous & infinite (vs discrete/finite) time?

- Transformation flexibility with simple & precise semantics
- Quality/accuracy
- Efficiency (adapative)
- Modularity/composability:
  - Fewer assumptions, more uses (resolution-independence).
  - More info available for extraction.
  - Same benefits as pure, non-strict functional programming.
    See *Why Functional Programming Matters*.
- Integration and differentiation: natural, accurate, efficient.
- Reconcile differing input sampling rates.

Same issues as for space, hence vector graphics.

Why continuous & infinite (vs discrete/finite) time?

- Transformation flexibility with simple & precise semantics
- Quality/accuracy
- Efficiency (adapative)
- Modularity/composability:
  - Fewer assumptions, more uses (resolution-independence).
  - More info available for extraction.
  - Same benefits as pure, non-strict functional programming.
    See *Why Functional Programming Matters*.
- Integration and differentiation: natural, accurate, efficient.
- Reconcile differing input sampling rates.

Same issues as for space, hence vector graphics.

Principle: Approximations/prunings compose badly, so postpone.

Semantics

Central abstract type: *Behavior a* — a "flow" of values.

Semantics

Central abstract type: $Behavior\ a$ — a "flow" of values.

Precise & simple semantics:

$$\mu :: Behavior\ a \to (T \to a)$$

where $T = R$ (reals).

Semantics

Central abstract type: $Behavior\ a$ — a "flow" of values.

Precise & simple semantics:

$$\mu :: Behavior\ a \to (T \to a)$$

where $T = R$ (reals).

Much of API and its specification follows from this one choice.

Original formulation

API

$$
\begin{aligned}
&\mathit{time} && :: \mathit{Behavior}\ T \\
&\mathit{lift}_0 && :: a \to \mathit{Behavior}\ a \\
&\mathit{lift}_1 && :: (a \to b) \to \mathit{Behavior}\ a \to \mathit{Behavior}\ b \\
&\mathit{lift}_2 && :: (a \to b \to c) \to \mathit{Behavior}\ a \to \mathit{Behavior}\ b \to \mathit{Behavior}\ c \\
&\mathit{timeTrans} && :: \mathit{Behavior}\ a \to \mathit{Behavior}\ T \to \mathit{Behavior}\ a \\
&\mathit{integral} && :: \mathit{VS}\ a \Rightarrow \mathit{Behavior}\ a \to T \to \mathit{Behavior}\ a \\
&\ldots
\end{aligned}
$$

$$\textbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Num}\ (\mathit{Behavior}\ a)\ \textbf{where}\ \ldots$$

...

Reactivity later.

Semantics

$$
\begin{aligned}
\mu\ time &= \lambda t \to t \\
\mu\ (lift_0\ a) &= \lambda t \to a \\
\mu\ (lift_1\ f\ xs) &= \lambda t \to f\ (\mu\ xs\ t) \\
\mu\ (lift_2\ f\ xs\ ys) &= \lambda t \to f\ (\mu\ xs\ t)\ (\mu\ ys\ t) \\
\mu\ (timeTrans\ xs\ tt) &= \lambda t \to \mu\ xs\ (\mu\ tt\ t)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance}\ Num\ a \Rightarrow Num\ (Behavior\ a)\ \textbf{where} \\
&\quad fromInteger = lift_0 \circ fromInteger \\
&\quad (+) = lift_2\ (+) \\
&\quad \ldots
\end{aligned}
$$

Semantics

$$
\begin{aligned}
\mu\ time &= id \\
\mu\ (lift_0\ a) &= const\ a \\
\mu\ (lift_1\ f\ xs) &= f \circ \mu\ xs \\
\mu\ (lift_2\ f\ xs\ ys) &= liftA_2\ f\ (\mu\ xs)\ (\mu\ ys) \\
\mu\ (timeTrans\ xs\ tt) &= \mu\ xs \circ \mu\ tt
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance}\ Num\ a \Rightarrow Num\ (Behavior\ a)\ \textbf{where} \\
&\quad fromInteger = lift_0 \circ fromInteger \\
&\quad (+) = lift_2\ (+) \\
&\quad \ldots
\end{aligned}
$$

Events

Secondary type:

$$\mu :: Event\ a \to [(T, a)] \quad \text{-- non-decreasing times}$$

$$\begin{aligned}
never &:: Event\ a \\
once &:: T \to a \to Event\ a \\
(.|.) &:: Event\ a \to Event\ a \to Event\ a \\
(\Longrightarrow) &:: Event\ a \to (a \to b) \to Event\ b \\
predicate &:: Behavior\ Bool \to T \to Event\ () \\
snapshot &:: Event\ a \to Behavior\ b \to Event\ (a, b)
\end{aligned}$$

Reactivity

Reactive behaviors are defined piecewise, via events.

Reactivity

Reactive behaviors are defined piecewise, via events:

$$switcher :: Behavior\ a \to Event\ (Behavior\ a) \to Behavior\ a$$

Reactivity

Reactive behaviors are defined piecewise, via events:

$$switcher :: Behavior\ a \to Event\ (Behavior\ a) \to Behavior\ a$$

Semantics:

$$\mu\ (b_0\ `switcher`\ e)\ t = \mu\ (last\ (b_0 : before\ (\mu\ e)\ t))\ t$$

$$before :: [(T, a)] \to T \to [a]$$
$$before\ os\ t = [a \mid (t_a, a) \leftarrow os, t_a < t]$$

Reactivity

Reactive behaviors are defined piecewise, via events:

$$switcher :: Behavior\ a \to Event\ (Behavior\ a) \to Behavior\ a$$

Semantics:

$$\mu\ (b_0\ `switcher`\ e)\ t = \mu\ (last\ (b_0 : before\ (\mu\ e)\ t))\ t$$

$$before :: [(T, a)] \to T \to [a]$$
$$before\ os\ t = [a \mid (t_a, a) \leftarrow os, t_a < t]$$

Question: Can occurrences be extracted ("*changes*")?

Modernized formulation

API

Replace several operations with standard abstractions:

instance *Functor Behavior* **where** ...
instance *Applicative Behavior* **where** ...
instance *Monoid a* ⇒ *Monoid* (*Behavior a*) **where** ...

instance *Functor Event* **where** ...
instance *Monoid a* ⇒ *Monoid* (*Event a*) **where** ...

Less learning, more leverage.

Semantic instances

instance $Functor \quad ((\rightarrow)\ z)$ **where** ...
instance $Applicative\ ((\rightarrow)\ z)$ **where** ...

instance $Monoid\ a \Rightarrow Monoid\ (z \rightarrow a)$ **where** ...
instance $Num \quad a \Rightarrow Num \quad (z \rightarrow a)$ **where** ...
...

The *Behavior* instances follow in "precise analogy".

Homomorphisms

A "homomorphism" h is a function that preserves (distributes over) an algebraic structure. For instance, for `Monoid`:

$$\begin{aligned} h\ \varepsilon &\equiv \varepsilon \\ h\ (as \diamond bs) &\equiv h\ as \diamond h\ bs \end{aligned}$$

Homomorphisms

A "homomorphism" h is a function that preserves (distributes over) an algebraic structure. For instance, for `Monoid`:

$$\begin{aligned} h\ \varepsilon &\equiv \varepsilon \\ h\ (as \diamond bs) &\equiv h\ as \diamond h\ bs \end{aligned}$$

Some monoid homomorphisms:

$$\begin{aligned} &length' :: [a] \to Sum\ Int \\ &length' = Sum \circ length \end{aligned}$$

$$\begin{aligned} &log' :: Product\ R \to Sum\ R \\ &log' = Sum \circ log \circ getProduct \end{aligned}$$

More homomorphism properties

Functor:

$$h\ (fmap\ f\ xs) \equiv fmap\ f\ (h\ xs)$$

Applicative:

$$\begin{aligned} h\ (pure\ a) &\equiv pure\ a \\ h\ (fs <\!\!*\!\!> xs) &\equiv h\ fs <\!\!*\!\!> h\ xs \end{aligned}$$

Monad:

$$h\ (m \gg\!= k) \equiv h\ m \gg\!= h \circ k$$

Specification by semantic homomorphism

Specification: μ as homomorphism. For instance,

$$\mu\ (fmap\ f\ \ as) \equiv fmap\ f\ (\mu\ as)$$

$$\begin{aligned} \mu\ (pure\ a) &\equiv pure\ a \\ \mu\ (fs <\!\!*\!\!> xs) &\equiv \mu\ fs <\!\!*\!\!> \mu\ xs \end{aligned}$$

Semantic instances

$$\textbf{instance } \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (z \to a) \textbf{ where}$$
$$\varepsilon \quad = \lambda z \to \varepsilon$$
$$f \diamond g = \lambda z \to f\ z \diamond g\ z$$

$$\textbf{instance } \mathit{Functor}\ ((\to)\ z) \textbf{ where}$$
$$\mathit{fmap}\ g\ f = g \circ f$$

$$\textbf{instance } \mathit{Applicative}\ ((\to)\ z) \textbf{ where}$$
$$\mathit{pure}\ a \quad = \lambda z \to a$$
$$\mathit{ff} <\!\!*\!\!> \mathit{fx} = \lambda z \to (\mathit{ff}\ z)\ (\mathit{fx}\ z)$$

Semantic homomorphisms

Put the pieces together:

$$
\begin{aligned}
& \mu\ (pure\ a) \\
\equiv\ & pure\ a \\
\\
\equiv\ & \lambda t \rightarrow a
\end{aligned}
$$

$$
\begin{aligned}
& \mu\ (fs \mathbin{<\!\!*\!\!>} xs) \\
\equiv\ & \mu\ fs \mathbin{<\!\!*\!\!>} \mu\ xs \\
\\
\equiv\ & \lambda t \rightarrow (\mu\ fs\ t)\ (\mu\ xs\ t)
\end{aligned}
$$

Likewise for *Functor*, *Monoid*, *Num*, etc.

Semantic homomorphisms

Put the pieces together:

$$
\begin{aligned}
& \mu\ (pure\ a) \\
\equiv\ & pure\ a \\
\\
\equiv\ & \lambda t \rightarrow a
\end{aligned}
$$

$$
\begin{aligned}
& \mu\ (fs \mathbin{<\!\!*\!\!>} xs) \\
\equiv\ & \mu\ fs \mathbin{<\!\!*\!\!>} \mu\ xs \\
\\
\equiv\ & \lambda t \rightarrow (\mu\ fs\ t)\ (\mu\ xs\ t)
\end{aligned}
$$

Likewise for *Functor*, *Monoid*, *Num*, etc.

Notes:

- Corresponds exactly to the original FRP denotation.
- Follows inevitably from semantic homomorphism principle.
- Laws hold for free (already paid for).

Laws for free

$$\boxed{\begin{aligned} \mu\ \varepsilon &\equiv \varepsilon \\ \mu\ (a \diamond b) &\equiv \mu\ a \diamond \mu\ b \end{aligned}} \Rightarrow \boxed{\begin{aligned} a \diamond \varepsilon &\equiv a \\ \varepsilon \diamond b &\equiv b \\ a \diamond (b \diamond c) &\equiv (a \diamond b) \diamond c \end{aligned}}$$

where equality is *semantic*.

Laws for free

$$\boxed{\begin{aligned} \mu\ \varepsilon &\equiv \varepsilon \\ \mu\ (a \diamond b) &\equiv \mu\ a \diamond \mu\ b \end{aligned}} \Rightarrow \boxed{\begin{aligned} a \diamond \varepsilon &\equiv a \\ \varepsilon \diamond b &\equiv b \\ a \diamond (b \diamond c) &\equiv (a \diamond b) \diamond c \end{aligned}}$$

where equality is *semantic*. Proofs:

$$\boxed{\begin{aligned} & \mu\ (a \diamond \varepsilon) \\ \equiv\ & \mu\ a \diamond \mu\ \varepsilon \\ \equiv\ & \mu\ a \diamond \varepsilon \\ \equiv\ & \mu\ a \end{aligned}} \quad \boxed{\begin{aligned} & \mu\ (\varepsilon \diamond b) \\ \equiv\ & \mu\ \varepsilon \diamond \mu\ b \\ \equiv\ & \varepsilon \diamond \mu\ b \\ \equiv\ & \mu\ b \end{aligned}} \quad \boxed{\begin{aligned} & \mu\ (a \diamond (b \diamond c)) \\ \equiv\ & \mu\ a \diamond (\mu\ b \diamond \mu\ c) \\ \equiv\ & (\mu\ a \diamond \mu\ b) \diamond \mu\ c \\ \equiv\ & \mu\ ((a \diamond b) \diamond c) \end{aligned}}$$

Works for other classes as well.

Events

```
newtype Event a = Event (Behavior [a])   -- discretely non-empty
  deriving (Monoid, Functor)
```

Events

```
newtype Event a = Event (Behavior [a])   -- discretely non-empty
  deriving (Monoid, Functor)
```

Derived instances:

```
instance Monoid a ⇒ Monoid (Event a) where
  ε = Event (pure ε)
  Event u ⋄ Event v = Event (liftA2 (⋄) u v)

instance Functor Event where
  fmap f (Event b) = Event (fmap (fmap f) b)
```

Events

```
newtype Event a = Event (Behavior [a])   -- discretely non-empty
  deriving (Monoid, Functor)
```

Derived instances:

```
instance Monoid a ⇒ Monoid (Event a) where
  ε = Event (pure ε)
  Event u ⋄ Event v = Event (liftA2 (⋄) u v)

instance Functor Event where
  fmap f (Event b) = Event (fmap (fmap f) b)
```

Alternatively,

```
type Event = Behavior ∘ []
```

Conclusion

- Two fundamental properties:
  - Continuous time. (Natural & composable.)
  - Precise, simple denotation. (Elegant & rigorous.)

Warning: most recent "FRP" systems lack both.

Conclusion

- Two fundamental properties:
  - Continuous time. (Natural & composable.)
  - Precise, simple denotation. (Elegant & rigorous.)

  Warning: most recent "FRP" systems lack both.
- Semantic homomorphisms:
  - Mine semantic model for API.
  - Inevitable API semantics (minimize invention).
  - Laws hold for free (already paid for).
  - No abstraction leaks.
  - Matches original FRP semantics.
  - Generally useful principle for library design.

History

1983–1989 at CMU

- I went for graphics.
- Did program transformation, FP, type theory.
- Class in denotational semantics.

1989 at CMU

- Kavi Arya's visit
  - *Functional animation*
  - Streams of pictures
  - Mostly elegant
- John Reynolds' insight: continuous time. Roughly,

  "You can think of sequences as functions from the natural numbers. Have you thought about functions from the reals instead?

  Doing so might help with the awkwardness of interpolation."
- Finished my dissertation anyway.

1990–93 at Sun: TBAG

- 3D geometry etc as first-class immutable values.
- Optimizing compiler via partial evaluation.
- For animation & interaction, immutable functions of time.
- Multi-way constraints on time-functions.
- Differentiation, integration and ODEs specified via *derivative*. Adaptive Runge-Kutta-5 solver (fast & accurate).
- Efficient multi-user distributed execution for free.
- Reactivity via `assert`/`retract` (high-level but imperative).
- In Common Lisp, C++, Scheme.

1994–1996 at MSR: RBML/ActiveVRML

- Programming model & fast implementation for new 3D hardware.
- TBAG + denotative/functional reactivity.
- Add event algebra to behavior algebra.
- Reactivity via behavior-valued events.
- Drop multi-way constraints “at first”.
- Started in ML as “RBML”.
- Rebranded to “ActiveVRML”, then “DirectAnimation”.

1996–1999 at MSR: RBMH/Fran

- Found Haskell: reborn as "RBMH" (research vehicle).
- Very fast implementation via sprite engine.
- John Hughes suggested using *Arrow*.

2000 at MSR: first try at push-based implementation

- Algebra of imperative event listeners.
- Challenges:
  - Garbage collection & dependency reversal.
  - Determinacy of timing & simultaneity.
  - I doubt anyone has gotten correct.

2009: Push-pull FRP

- Minimal computation, low latency, *provably correct*.
- Push for reactivity and pull for continuous phases.
- "Push" is really blocked pull.
- Modernized API:
  - Standard abstractions.
  - Semantics as homomorphisms.
  - Laws for free.
- Reactive normal form, via equational properties (denotation!).
- Uses *lub* (basis of PL semantics).
- Implementation subtleties & GHC RTS bugs. Didn't quite work.

1997–2014: Paul Hudak / Yale

- Paul Hudak visited MSR in 1996 or so and saw RBMH.
- Encouraged me to publish and suggested collaboration.
- Proposed names “Fran” & “FRP”.
- Many FRP-based papers and theses, drawing much attention.

July 15, 1952 – April 29, 2015