A more elegant specification for FRP

Conal Elliott

LambdaJam 2015

The story so far

FRP's two fundamental properties

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

FRP *is not* about:

FRP's two fundamental properties

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

FRP *is not* about:

- graphs,
- updates and propagation,
- streams,
- doing

Semantics

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

Semantics

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

Precise & simple semantics:

$$\mu :: \mathit{Behavior}\ a \to (T \to a)$$

where $T = \mathbb{R}$ (reals).

Semantics

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

Precise & simple semantics:

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

where $T = \mathbb{R}$ (reals).

Much of API and its specification can follow from this one choice.

Original formulation

API

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

$$
\begin{array}{l}
\mathbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Num}\ (\mathit{Behavior}\ a)\ \mathbf{where} \ldots \\
\ldots
\end{array}
$$

Reactivity later.

Semantics

$$
\begin{aligned}
\mu\ \mathit{time} &= \lambda t \to t \\
\mu\ (\mathit{lift}_0\ a) &= \lambda t \to a \\
\mu\ (\mathit{lift}_1\ f\ \mathit{xs}) &= \lambda t \to f\ (\mu\ \mathit{xs}\ t) \\
\mu\ (\mathit{lift}_2\ f\ \mathit{xs}\ \mathit{ys}) &= \lambda t \to f\ (\mu\ \mathit{xs}\ t)\ (\mu\ \mathit{ys}\ t) \\
\mu\ (\mathit{timeTrans}\ \mathit{xs}\ \mathit{tt}) &= \lambda t \to \mu\ \mathit{xs}\ (\mu\ \mathit{tt}\ t)
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Num}\ (\mathit{Behavior}\ a)\ \mathbf{where} \\
&\quad \mathit{fromInteger} = \mathit{lift}_0 \circ \mathit{fromInteger} \\
&\quad (+) \qquad\qquad\quad = \mathit{lift}_2\ (+) \\
&\quad \ldots
\end{aligned}
$$

Semantics

$$
\begin{aligned}
\mu\ \mathit{time} &= \mathit{id} \\
\mu\ (\mathit{lift}_0\ a) &= \mathit{const}\ a \\
\mu\ (\mathit{lift}_1\ f\ \mathit{xs}) &= f \circ \mu\ \mathit{xs} \\
\mu\ (\mathit{lift}_2\ f\ \mathit{xs}\ \mathit{ys}) &= \mathit{liftA}_2\ f\ (\mu\ \mathit{xs})\ (\mu\ \mathit{ys}) \\
\mu\ (\mathit{timeTrans}\ \mathit{xs}\ \mathit{tt}) &= \mu\ \mathit{xs} \circ \mu\ \mathit{tt}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Num}\ (\mathit{Behavior}\ a)\ \mathbf{where} \\
&\quad \mathit{fromInteger} = \mathit{lift}_0 \circ \mathit{fromInteger} \\
&\quad (+) \phantom{\mathit{fromInteger}} = \mathit{lift}_2\ (+) \\
&\quad \ldots
\end{aligned}
$$

Events

Secondary type:

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

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

Exercise: define semantics of these operations.

A more elegant specification

API

Replace several operations with standard abstractions:

$$\textbf{instance}\ \mathit{Functor}\ \mathit{Behavior}\ \textbf{where} \ldots$$
$$\textbf{instance}\ \mathit{Applicative}\ \mathit{Behavior}\ \textbf{where} \ldots$$
$$\textbf{instance}\ \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Behavior}\ a)\ \textbf{where} \ldots$$

$$\textbf{instance}\ \mathit{Functor}\ \mathit{Event}\ \textbf{where} \ldots$$
$$\textbf{instance}\ \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Event}\ a)\ \textbf{where} \ldots$$

Why?

API

Replace several operations with standard abstractions:

$$\textbf{instance } \mathit{Functor\ Behavior} \textbf{ where } \ldots$$
$$\textbf{instance } \mathit{Applicative\ Behavior} \textbf{ where } \ldots$$
$$\textbf{instance } \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Behavior}\ a) \textbf{ where } \ldots$$

$$\textbf{instance } \mathit{Functor\ Event} \textbf{ where } \ldots$$
$$\textbf{instance } \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Event}\ a) \textbf{ where } \ldots$$

Why?

- Less learning, more leverage.
- Specifications and laws for free.

Semantic instances

$$
\begin{array}{l}
\textbf{instance } \mathit{Functor} \quad\quad ((\rightarrow)\ z)\ \textbf{where} \ldots \\
\textbf{instance } \mathit{Applicative}\ ((\rightarrow)\ z)\ \textbf{where} \ldots \\
\\
\textbf{instance } \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (z \rightarrow a)\ \textbf{where} \ldots \\
\textbf{instance } \mathit{Num} \quad\ \ a \Rightarrow \mathit{Num} \quad\ \ (z \rightarrow a)\ \textbf{where} \ldots \\
\ldots
\end{array}
$$

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

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:

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

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

More homomorphism properties

Functor:

$$h\ (\mathit{fmap}\ f\ \mathit{xs}) \equiv \mathit{fmap}\ f\ (h\ \mathit{xs})$$

Applicative:

$$\begin{aligned} h\ (\mathit{pure}\ a) &\equiv \mathit{pure}\ a \\ h\ (\mathit{fs} \circledast \mathit{xs}) &\equiv h\ \mathit{fs} \circledast h\ \mathit{xs} \end{aligned}$$

Monad:

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

Specification by semantic homomorphism

Specification: μ as homomorphism. For instance,

$$\mu\ (\mathit{fmap}\ f\ \mathit{as}) \equiv \mathit{fmap}\ f\ (\mu\ \mathit{as})$$

$$\begin{aligned} \mu\ (\mathit{pure}\ a) &\equiv \mathit{pure}\ a \\ \mu\ (\mathit{fs} \circledast \mathit{xs}) &\equiv \mu\ \mathit{fs} \circledast \mu\ \mathit{xs} \end{aligned}$$

Semantic instances

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

$$
\begin{aligned}
&\textbf{instance } \mathit{Functor}\ ((\to)\ z)\ \textbf{where}\\
&\quad \mathit{fmap}\ g\ f = g \circ f
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance } \mathit{Applicative}\ ((\to)\ z)\ \textbf{where}\\
&\quad \mathit{pure}\ a \quad = \lambda z \to a\\
&\quad \mathit{ff} \circledast \mathit{fx} = \lambda z \to (\mathit{ff}\ z)\ (\mathit{fx}\ z)
\end{aligned}
$$

Semantic homomorphisms

Put the pieces together:

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

$$\begin{aligned} & \mu\ (\mathit{fs} \circledast \mathit{xs}) \\ \equiv\ & \mu\ \mathit{fs} \circledast \mu\ \mathit{xs} \\ \equiv\ & \lambda t \rightarrow (\mu\ \mathit{fs}\ t)\ (\mu\ \mathit{xs}\ t) \end{aligned}$$

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

Semantic homomorphisms

Put the pieces together:

$$
\begin{aligned}
& \mu\ (\mathit{pure}\ a) \\
\equiv\ & \mathit{pure}\ a \\
\\
\equiv\ & \lambda t \rightarrow a
\end{aligned}
\qquad\qquad
\begin{aligned}
& \mu\ (\mathit{fs} \circledast \mathit{xs}) \\
\equiv\ & \mu\ \mathit{fs} \circledast \mu\ \mathit{xs} \\
\\
\equiv\ & \lambda t \rightarrow (\mu\ \mathit{fs}\ t)\ (\mu\ \mathit{xs}\ t)
\end{aligned}
$$

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

$$\textbf{newtype}\ \mathit{Event}\ a = \mathit{Event}\ (\mathit{Behavior}\ [a]) \quad \text{-- discretely non-empty}$$
$$\quad \textbf{deriving}\ (\mathit{Monoid}, \mathit{Functor})$$

Events

$$\textbf{newtype}\ \mathit{Event}\ a = \mathit{Event}\ (\mathit{Behavior}\ [a]) \quad \text{-- discretely non-empty}$$
$$\quad \textbf{deriving}\ (\mathit{Monoid}, \mathit{Functor})$$

Derived instances:

$$\textbf{instance}\ \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Event}\ a)\ \textbf{where}$$
$$\quad \varepsilon = \mathit{Event}\ (\mathit{pure}\ \varepsilon)$$
$$\quad \mathit{Event}\ u \diamond \mathit{Event}\ v = \mathit{Event}\ (\mathit{liftA_2}\ (\diamond)\ u\ v)$$

$$\textbf{instance}\ \mathit{Functor}\ \mathit{Event}\ \textbf{where}$$
$$\quad \mathit{fmap}\ f\ (\mathit{Event}\ b) = \mathit{Event}\ (\mathit{fmap}\ (\mathit{fmap}\ f)\ b)$$

Events

$$\textbf{newtype}\ \mathit{Event}\ a = \mathit{Event}\ (\mathit{Behavior}\ [\,a\,])\quad \text{-- discretely non-empty}$$
$$\quad\textbf{deriving}\ (\mathit{Monoid}, \mathit{Functor})$$

Derived instances:

$$\textbf{instance}\ \mathit{Monoid}\ a \Rightarrow \mathit{Monoid}\ (\mathit{Event}\ a)\ \textbf{where}$$
$$\quad \varepsilon = \mathit{Event}\ (\mathit{pure}\ \varepsilon)$$
$$\quad \mathit{Event}\ u \diamond \mathit{Event}\ v = \mathit{Event}\ (\mathit{liftA_2}\ (\diamond)\ u\ v)$$

$$\textbf{instance}\ \mathit{Functor}\ \mathit{Event}\ \textbf{where}$$
$$\quad \mathit{fmap}\ f\ (\mathit{Event}\ b) = \mathit{Event}\ (\mathit{fmap}\ (\mathit{fmap}\ f)\ b)$$

Alternatively,

$$\textbf{type}\ \mathit{Event} = \mathit{Behavior} \circ [\,]$$

Conclusion

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

  Warning: most recent "FRP" systems lack both.

Conclusion

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

  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.