Functional Reactive Programming (FRP)

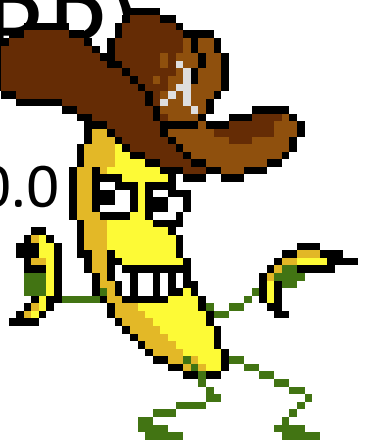
with reactive-banana-0.6.0.0

Heinrich Apfelmus

Translated by Akira Takahashi (faithandbrave@gmail.com)

Functional Reactive Programming (FRP)

with reactive-banana-0.6.0.0



Heinrich Apfelmus

Translated by Akira Takahashi (faithandbrave@gmail.com)

Why?

Functional Ractive Programmingは、命令的プログラムを実装するエレガントな手法である

- GUI
- アニメーション
- デジタルミュージック
- ロボット

How?

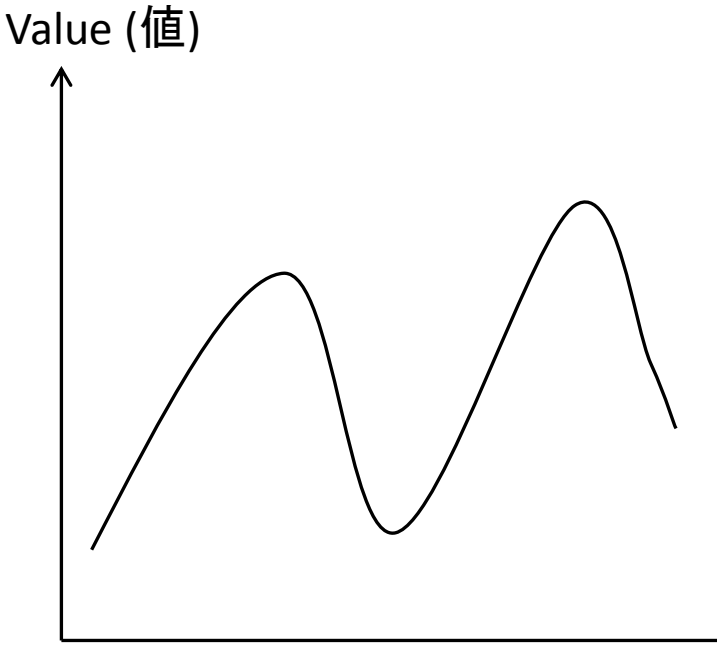
ファーストクラスの値としての時間変化

```
type Behavior a = Time -> a
type Event a    = [(Time, a)]
```

鍵となるデータ型BehaviorとEvent。
Behavior(振る舞い)は、「時間とともに変化する値」に対応する
Event(イベント)は、「特定の時間で呼び出されたイベント」に対応する。
これらを理解する方法を説明するつもりだ。もちろん、実際の実装は抽象的である。

Behavior (振る舞い)

```
type Behavior a = Time -> a
```



- 位置 – アニメーション
- テキスト値 – GUI
- ボリューム – 音楽
- 物理量

$$y(t) = y0 + y0t - g\frac{t^2}{2}$$

Behaviorは時間の各ポイントに値を関連付ける。

Behavior API

```
instance Functor Behavior
```

Functor

```
instance Applicative Behavior
```

Applicative

どのようにしてプログラムでBehaviorを使用するか？
Behavior用のAPIは、実際にはとても簡単：それらはapplicative functorである。

Behavior API

```
(<$>) :: (a -> b)
      -> Behavior a -> Behavior b
```

Functor

```
pure  :: a -> Behavior a
(<*>) :: Behavior (a -> b)
      -> Behavior a -> Behavior b
```

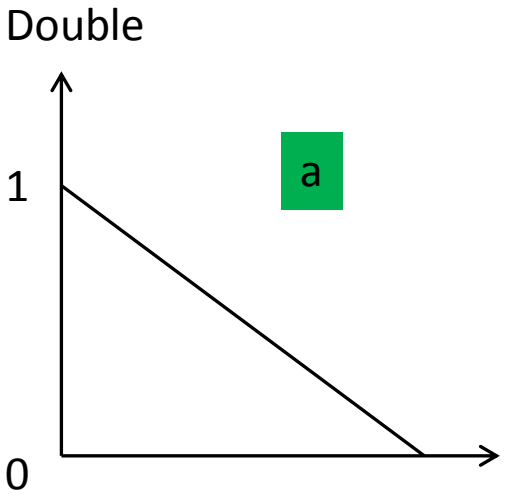
Applicative

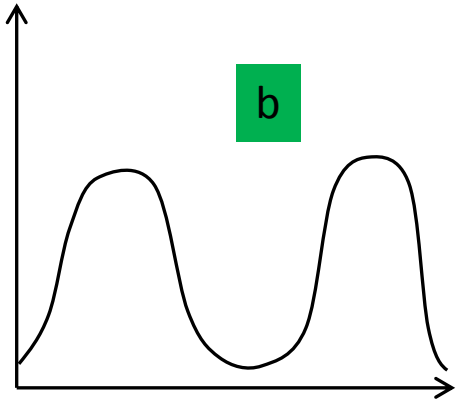
```
bf <*> bx =
   ¥time -> bf time $ bx time
```

時間の各ポイント

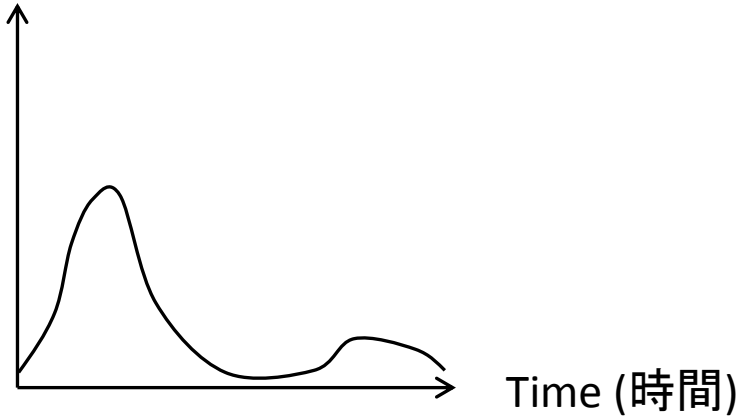
FunctorとApplicativeに関連付けられた関数に対するリマインダ。
最も重要な機能は「apply(適用)」と呼ばれ、単に時間の各ポイントで時間変化の値に時間変化の関数を適用する<*>演算子である。
pure(純粋)な関数で、ある時間における一定の値を構築する

Behavior API





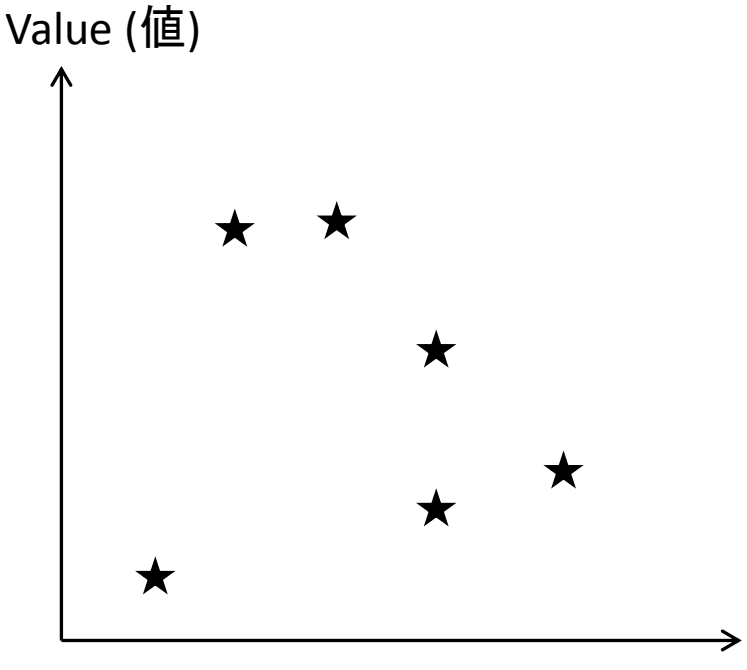

```
(*) <$> a <*> b
```



タスクの例：振動を減衰させる。

Event

```
type Event a = [(Time, a)]
```



- マウスクリック – GUI
- 楽譜 – 音楽
- 衝突 – 物理

Eventは、特定のポイントで「発生」する値のコレクションである。
発生は、同時に起こる可能性があり、最新のreactive-banana 0.6ではイベントの参照もできる。

Event API

```
Instance Functor Event
```

Functor

```
never     :: Event a
unionWith :: (a -> a -> a)
 -> Event a -> Event a -> Event a

filterE :: (a -> Bool)
 -> Event a -> Event a

accumE :: a -> Event (a -> a)
 -> Event a
```

List
[]
zipWith

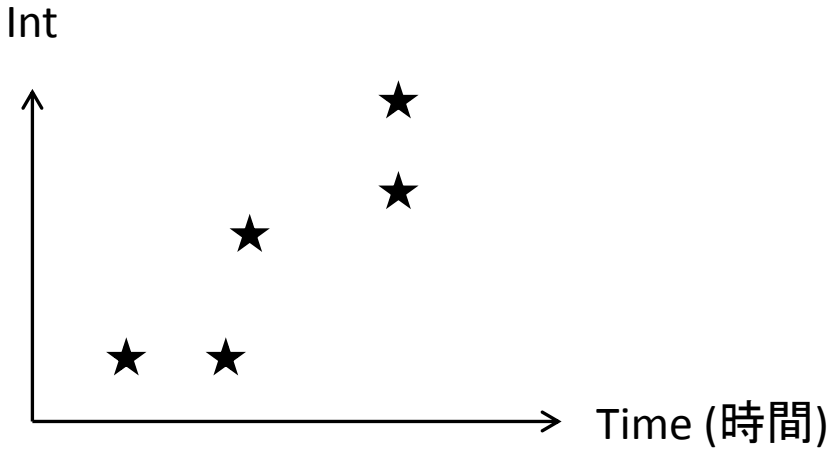
filter

scanl

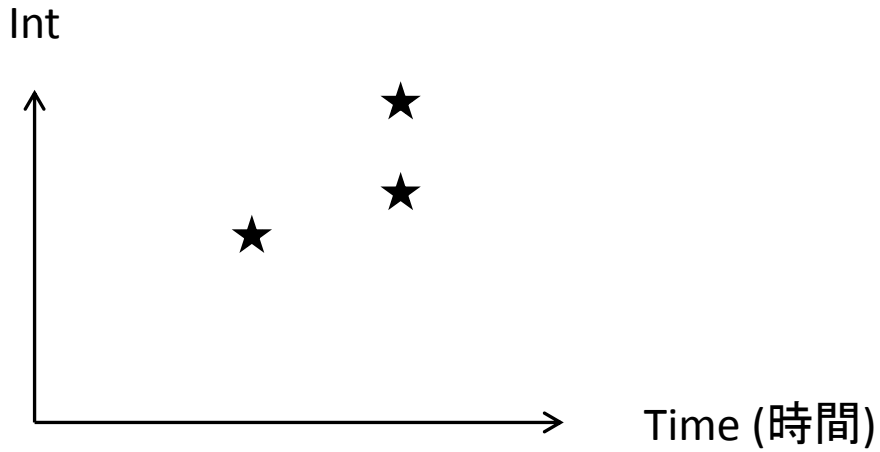
どのようにしてプログラムでEventを使用するか？
Event APIはもう少し手がこんていますが、リスト操作に密接に関係している。

Event API

x



filterE (>5) x



例：filterE

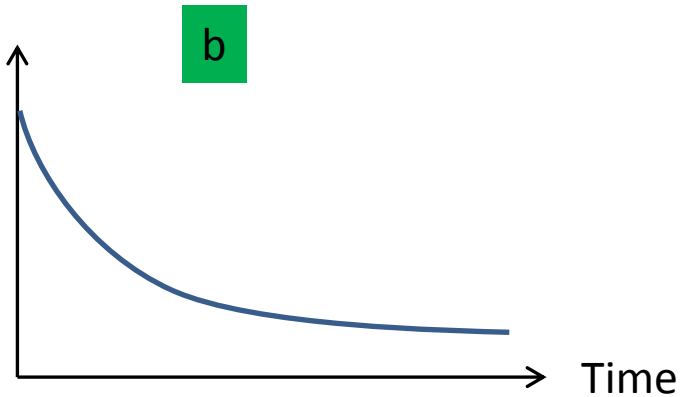
Event & Behavior API

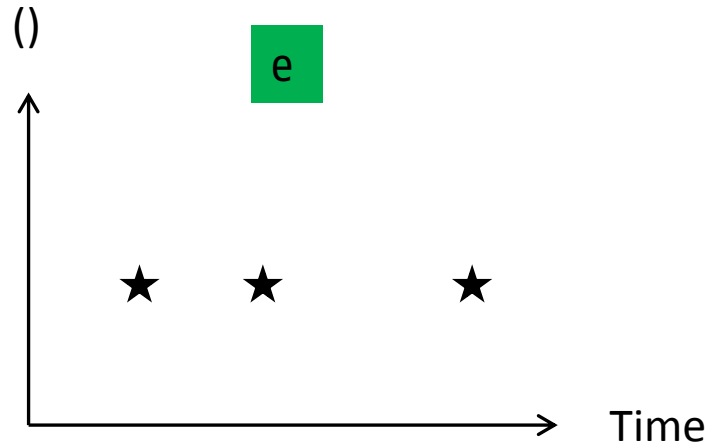
```
(<@>) :: Behavior (a -> b)
      -> Event a -> Event b

(<@) :: Behavior b
     -> Event a -> Event b
```

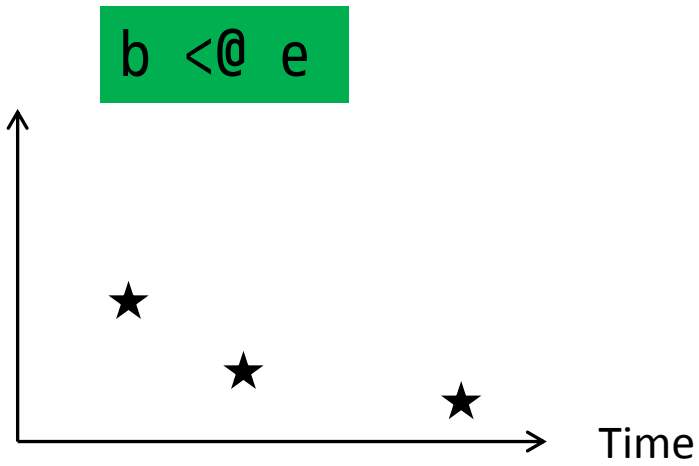
「apply」

<@>演算子は「apply(適用)」と呼ばれ、イベントが発生する時系列関数を適用する。
その弟である<@は、Behaviorから値を持つEventをタグ付けする。これはData.Functorの<$演算子に似ている。

Event & Behavior API







<@演算子の可視化

Frameworks (GUI, ...)

data NetworkDesctiption t a

fromAddHandler	Eventのインポート
fromPoll	Behaviorのインポート
reactimate	Eventのエクスポート
changes	BehaviorからEventを取得

これまで説明したAPIは、既存のEventとBehaviorを新しいものに結合できるが、最初の場所ではそれらを取得する方法を教えてはくれない。そのため、wxHaskellのような外部フレームワークをバインドする必要がある。
Reactive.Banana.FrameworksのNetworkDescriptionモナドはこれができる。これはそれほど興味深いモナドではない。これは単なるdevice for bookkeepingで、いくつかの種類のシンタックスシュガーを使うことをお勧めする。