Rhine, FRP with type-level clocks

Functional Reactive Programming for Zurihac '24

Manuel Bärenz

June 8, 2024

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)

Plan for this session

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)
- 13:20 If you haven't already, you'll clone the rhine-koans repo

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)
- 13:20 If you haven't already, you'll clone the rhine-koans repo
- 13:20 I'll show you how it works & walk you through the first koan

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)
- 13:20 If you haven't already, you'll clone the rhine-koans repo
- 13:20 I'll show you how it works & walk you through the first koan
- 13:25 You'll solve the basic track, I'll come around and answer your questions

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)
- 13:20 If you haven't already, you'll clone the rhine-koans repo
- 13:20 I'll show you how it works & walk you through the first koan
- 13:25 You'll solve the basic track, I'll come around and answer your questions
- 14:00 We'll collect & discuss the biggest questions & hardest problems that occurred so far

Functional Reactive Programming with Rhine

What's this now?

- I'm Manuel Bärenz (he/him)
- We'll do https://github.com/turion/rhine-koans/

Plan for this session

- 13:05 I'll briefly talk about FRP & Rhine (<15 minutes)
- 13:20 If you haven't already, you'll clone the rhine-koans repo
- 13:20 I'll show you how it works & walk you through the first koan
- 13:25 You'll solve the basic track, I'll come around and answer your questions
- 14:00 We'll collect & discuss the biggest questions & hardest problems that occurred so far
- 14:10 You'll start on the UI track, or on your own little app, I'll come around and answer questions

Let me tell you a tale...

...but don't worry!

You don't need to memorise everything. Lean back & relax :)

About Functional Reactive Programming

What is Functional Reactive Programming?

- Ivan Perez: FRP is about time.

About Functional Reactive Programming

What is Functional Reactive Programming?

- Ivan Perez: FRP is about time.
- Use awareness of time in the program.

About Functional Reactive Programming

What is Functional Reactive Programming?

- Ivan Perez: FRP is about time.
- Use awareness of time in the program.
- *When* do computations & effects happen?

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects
- "Synchronous": One output per input

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects
- "Synchronous": One output per input
  - dunai, automaton, essence-of-live-coding, machines, varying, netwire, ...

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects
- "Synchronous": One output per input
  - dunai, automaton, essence-of-live-coding, machines, varying, netwire, ...
  - **Monadic stream function**:

```
data MSF m a b = MSF (a -> m (b, MSF m a b))
```

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects
- "Synchronous": One output per input
  - dunai, automaton, essence-of-live-coding, machines, varying, netwire, ...
  - **Monadic stream function**:

    ```
    data MSF m a b = MSF (a -> m (b, MSF m a b))
    ```
- "Asynchronous": Many outputs per many inputs

Monadic/effectful streaming

- Centered around a "main loop" which constantly consumes & produces data, and performs side effects
- "Synchronous": One output per input
  - dunai, automaton, essence-of-live-coding, machines, varying, netwire, ...
  - **Monadic stream function**:

    ```
    data MSF m a b = MSF (a -> m (b, MSF m a b))
    ```
- "Asynchronous": Many outputs per many inputs
  - pipes, conduit, streamly, streaming, machines, ...

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**
  - Morally `type SF a b = Behaviour a -> Behaviour b`

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**
  - Morally `type SF a b = Behaviour a -> Behaviour b`
  - Yampa, dunai, bearriver, Rhine

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**
  - Morally `type SF a b = Behaviour a -> Behaviour b`
  - Yampa, dunai, bearriver, Rhine
- Effectful FRP: "Monadic signal functions"

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**
  - Morally `type SF a b = Behaviour a -> Behaviour b`
  - Yampa, dunai, bearriver, Rhine
- Effectful FRP: "Monadic signal functions"
  - `type SF m a b = MSF (ReaderT Time m) a b`

Functional reactive programming paradigms

- "Classic" FRP: **Behaviours** & **events**
  - Morally `type Behaviour a = Time -> a`
    - Computations happen all the time!
  - Morally `type Event a = [(Time, a)]`
    - A computation happens on every event
  - FRAN, frpnow, reactive-banana, reflex, ...
- Arrowized FRP: **Signal functions**
  - Morally `type SF a b = Behaviour a -> Behaviour b`
  - Yampa, dunai, bearriver, Rhine
- Effectful FRP: "Monadic signal functions"
  - `type SF m a b = MSF (ReaderT Time m) a b`
  - bearriver, Rhine

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times
  - E.g. game simulation at one frame rate, video and audio at other rates, user input as an event source

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times
  - E.g. game simulation at one frame rate, video and audio at other rates, user input as an event source
- Make these differences visible as *type level clocks*

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times
  - E.g. game simulation at one frame rate, video and audio at other rates, user input as an event source
- Make these differences visible as *type level clocks*
- Compose synchronous (1-1) and asynchronous (many-many) components safely

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times
  - E.g. game simulation at one frame rate, video and audio at other rates, user input as an event source
- Make these differences visible as *type level clocks*
- Compose synchronous (1-1) and asynchronous (many-many) components safely
- Accidental synchronisation becomes a type error

Rhine: Arrowized FRP with type level clocks

How to organize bigger FRP applications?

- Different components are activated at different times
  - E.g. game simulation at one frame rate, video and audio at other rates, user input as an event source
- Make these differences visible as *type level clocks*
- Compose synchronous (1-1) and asynchronous (many-many) components safely
- Accidental synchronisation becomes a type error
- Framework to answer the question "*When* do computations & effects happen?"

Rhine concepts

Clock types and values

```haskell
class Clock m cl where
  type Time cl -- The type of timestamps
  type Tag cl -- Additional info about the tick
  ...

-- Ticks every 10 milliseconds.
waitClock :: Millisecond 10
waitClock = ...
instance Clock IO (Millisecond 10) where ...

-- Ticks for every line entered on stdin. (An "event")
data StdinClock = StdinClock
instance Clock IO StdinClock where
  type Tag StdinClock = Text
  ...
```

Rhine concepts

Running example

github.com/turion/rhine/blob/master/rhine-examples/src/Ball.hs

Clocked signal functions (`ClSF`)

```haskell
type Ball = (Double, Double, Double)
type BallVel = (Double, Double, Double)

startVel :: ClSF IO StdinClock () BallVel
startVel = arrMCl $ const $ do
  velX <- randomRIO (-10, 10)
  velY <- randomRIO (-10, 10)
  velZ <- randomRIO (3, 10)
  return (velX, velY, velZ)
```

Rhine concepts

Behaviours: Clock-independent signal functions

```
freeFall :: (Monad m) => BallVel ->
  BehaviourF m UTCTime () Ball
freeFall v0 =
  arr (const (0, 0, -9.81))
    >>> integralFrom v0
    >>> integral
```

Arrow syntax

```
height :: (Monad m) => BallVel ->
  BehaviourF m UTCTime () Double
height v0 = proc _ -> do
  pos <- freeFall v0 -< ()
  let (_, _, height) = pos
  returnA -< height
```

Rhine concepts

```
throwMaybe :: (Monad m) =>
  ClSF (ExceptT e m) cl (Maybe e) (Maybe a)
```

Throwing exceptions

```
falling :: (Monad m) => BallVel -> ClSF (ExceptT () m)
  (Millisecond 10) (Maybe BallVel) Ball
falling v0 = proc _ -> do
  pos <- freeFall v0 -< ()
  let (_, _, height) = pos
  throwMaybe -< guard $ height < 0
  returnA -< pos

waiting :: (Monad m) => ClSF (ExceptT BallVel m)
  (Millisecond 10) (Maybe BallVel) Ball
waiting = throwMaybe >>> arr (const zeroVector)
```

Rhine concepts

```
data ClSFExcept clock input output monad exception
```

Handling exceptions

```
ballModes :: ClSFExcept (Millisecond 10)
  (Maybe BallVel) Ball IO void
ballModes = do
  v0 <- try waiting
  once_ $ putStrLn "Catch!"
  try $ falling v0
  once_ $ putStrLn "Caught!"
  ballModes

ball :: ClSF IO (Millisecond 10) (Maybe BallVel) Ball
ball = safely ballModes
```

Rhine concepts

Top level programs: `Rhine`

```
startVelRh :: Rhine IO StdinClock () BallVel
startVelRh = startVel @@ StdinClock

resample :: ResamplingBuffer IO
  StdinClock (Millisecond 10) BallVel (Maybe BallVel)
resample = fifoUnbounded

ballRh :: Rhine IO (Millisecond 10) (Maybe BallVel) Ball
ballRh = ball @@ waitClock

mainRhine :: Rhine IO
  (SeqClock StdinClock (Millisecond 10)) () ()
mainRhine = startVelRh >-- resample --> ballRh

main = flow mainRhine
```

Basic track (until 14:00)

Let's get it off the ground!

```
git clone git@github.com:turion/rhine-koans.git
cabal update
cabal run basic-1-1-hello-rhine
cabal test basic-1-1-hello-rhine-test
```

Slides

github.com/turion/rhine-koans/blob/main/presentation/presentation.pdf

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Advanced track

Let's dive in!

```
cabal test ui-1-gloss-1-circle-test
```

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Some project ideas

- Websocket clock: `https://hackage.haskell.org/package/wuss`

Advanced track

Let's dive in!

```
cabal test ui-1-gloss-1-circle-test
```

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Some project ideas

- Websocket clock: `https://hackage.haskell.org/package/wuss`
- Webserver: `https://hackage.haskell.org/package/wai`

Advanced track

Let's dive in!

```
cabal test ui-1-gloss-1-circle-test
```

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Some project ideas

- Websocket clock: `https://hackage.haskell.org/package/wuss`
- Webserver: `https://hackage.haskell.org/package/wai`
- Machine learning: `https://hackage.haskell.org/package/rhine-bayes`

Advanced track

Let's dive in!

```
cabal test ui-1-gloss-1-circle-test
```

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Some project ideas

- Websocket clock: `https://hackage.haskell.org/package/wuss`
- Webserver: `https://hackage.haskell.org/package/wai`
- Machine learning: `https://hackage.haskell.org/package/rhine-bayes`
- Port the snake to `rhine-terminal`

Advanced track

Let's dive in!

```
cabal test ui-1-gloss-1-circle-test
```

Ask me anything :)

Manuel (he/him), turion on Discord/Discourse/Github/..., turion@types.pl on Mastodon

Some project ideas

- Websocket clock: `https://hackage.haskell.org/package/wuss`
- Webserver: `https://hackage.haskell.org/package/wai`
- Machine learning: `https://hackage.haskell.org/package/rhine-bayes`
- Port the snake to `rhine-terminal`
- Challenge: Rhine entry in https://github.com/gelisam/frp-zoo