Rhine - FRP with type level clocks
A tea tutorial

Manuel Bärenz

15. Januar 2020
cabal update
git clone https://github.com/turion/rhine-tutorial/
cd rhine-tutorial
cabal sandbox init
cabal install --only-dependencies
cabal configure
cabal build
cabal run rhine-tutorial

Read documentation on http://hackage.haskell.org/package/rhine (version 0.1.0.0)!
Dunai (Iván Pérez, Henrik Nilsson, MB)

data MSF m a b = MSF (a -> m (b, MSF m a b))
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

-- Control.Arrow
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(*** ) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

-- Control.Arrow
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(***>) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b

-- only dunai
arrM :: (a -> m b) -> MSF m a b
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

-- Control.Arrow
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(****) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b

-- only dunai
arrM :: (a -> m b) -> MSF m a b

• MSF (Reader Double) is a replacement for FRP.Yampa.SF.
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

--- Control.Arrow

```haskell
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(*** ) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b
```

--- only dunai

```haskell
arrM :: (a -> m b) -> MSF m a b
```

- MSF (Reader Double) is a replacement for FRP.Yampa.SF.
- Other monads allow for concise FRP paradigms:
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

-- Control.Arrow

(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c

(****) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)

arr :: (a -> b) -> MSF m a b

-- only dunai

arrM :: (a -> m b) -> MSF m a b

- MSF (Reader Double) is a replacement for FRP.Yampa.SF.
- Other monads allow for concise FRP paradigms:
  - State, Reader and Writer give global state variables.
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

```haskell
-- Control.Arrow
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(*** ) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b
```

```haskell
-- only dunai
arrM :: (a -> m b) -> MSF m a b
```

- MSF (Reader Double) is a replacement for FRP.Yampa.SF.
- Other monads allow for concise FRP paradigms:
  - State, Reader and Writer give global state variables.
  - List gives branching computations.
**Dunai (Iván Pérez, Henrik Nilsson, MB)**

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

--- *Control.Arrow*

```haskell
(>>>>) :: MSF m a b -> MSF m b c -> MSF m a c

(****) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)

arr :: (a -> b) -> MSF m a b
```

--- *only dunai*

```haskell
arrM :: (a -> m b) -> MSF m a b
```

- MSF (Reader Double) is a replacement for FRP.Yampa.SF.
- Other monads allow for concise FRP paradigms:
  - State, Reader and Writer give global state variables.
  - List gives branching computations.
  - Either gives control flow!
Dunai (Iván Pérez, Henrik Nilsson, MB)

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

-- Control.Arrow

```haskell
(>>>) :: MSF m a b -> MSF m b c -> MSF m a c
(***) :: MSF m a b -> MSF m c d -> MSF m (a,c) (b,d)
arr :: (a -> b) -> MSF m a b
```

-- only dunai

```haskell
arrM :: (a -> m b) -> MSF m a b
```

- MSF (Reader Double) is a replacement for FRP.Yampa.SF.
- Other monads allow for concise FRP paradigms:
  - State, Reader and Writer give global state variables.
  - List gives branching computations.
  - Either gives control flow!
- Support for (entering and leaving) monad transformers.
Synchronous arrowized FRP

Arrow syntax

```haskell
{-# LANGUAGE Arrows #-}

verboseSum :: MSF IO Int Int
verboseSum = proc n -> do
    s <- sumS -< n
    _ <- arrM print -< "The sum is now " ++ show s
    returnA -< s
```
Clock type  All relevant properties of the clock, such as ...
Clock type  All relevant properties of the clock, such as ...
  • When, and how often, the clock should tick
Clock type  All relevant properties of the clock, such as ...

- When, and how often, the clock should tick
- Which monad the clock takes side effects in
Clock type  All relevant properties of the clock, such as ...

- When, and how often, the clock should tick
- Which monad the clock takes side effects in
- What additional data (besides a time stamp) the clock outputs
Clock type  All relevant properties of the clock, such as ...
  - When, and how often, the clock should tick
  - Which monad the clock takes side effects in
  - What additional data (besides a time stamp) the clock outputs

Clock value  All information needed to run the clock
Clock type  All relevant properties of the clock, such as ...
- When, and how often, the clock should tick
- Which monad the clock takes side effects in
- What additional data (besides a time stamp) the clock outputs

Clock value  All information needed to run the clock
- E.g. physical device address, event socket
Clock type  All relevant properties of the clock, such as ...
- When, and how often, the clock should tick
- Which monad the clock takes side effects in
- What additional data (besides a time stamp) the clock outputs

Clock value  All information needed to run the clock
- E.g. physical device address, event socket
- Implementation choice
**Clock type**  All relevant properties of the clock, such as ...  
- When, and how often, the clock should tick  
- Which monad the clock takes side effects in  
- What additional data (besides a time stamp) the clock outputs

**Clock value**  All information needed to run the clock  
- E.g. physical device address, event socket  
- Implementation choice

**Running clock**  Side-effectful stream of *time stamps*, tagged with additional info about the *tick*. 
Quick introduction to Rhine

Let's hack!

After the tutorial

Clocks

Rhine

-- simplified here

class Clock m cl where

  type Time cl -- time stamp

  type Tag cl -- additional information about tick

initClock :: cl -> MSF m () (TimeInfo cl, Tag cl)
Rhine

-- simplified here

class Clock m cl where

  type Time cl -- time stamp

  type Tag cl -- additional information about tick

  initClock :: cl -> MSF m () (TimeInfo cl, Tag cl)

data TimeInfo cl = {...}

  -- absolute and relative time, tag
A clock produces side effects to...

... wait between ticks,
... measure the current time,
... produce additional data (e.g. events).
A clock produces side effects to...

... wait between ticks,

... measure the current time,

... produce additional data (e.g. events).

Examples of clocks

- Fixed sample rate (e.g. Millisecond n)
- Events (e.g. Stdin)
### Clocks

- **Type:**
  ```haskell
type ClSF m cl a b = MSF (ReaderT (TimeInfo cl) m) a b
```
type ClSF m cl a b = MSF (ReaderT (TimeInfo cl) m) a b

Lifting dunai concepts

arrMCl :: (a -> m b) -> SyncSF m cl a b

...
setup

quick introduction to rhine

let's hack!

after the tutorial

Clocks

type ClSF m cl a b = MSF (ReaderT (TimeInfo cl) m) a b

lifting dunai concepts

arrMCl :: (a -> m b) -> SyncSF m cl a b

...

time information

timeInfo :: ClSF m cl () (TimeInfo cl)
type ClSF m cl a b = MSF (ReaderT (TimeInfo cl) m) a b

Lifting dunai concepts

arrMCl :: (a -> m b) -> SyncSF m cl a b
...

Time information

timeInfo :: ClSF m cl () (TimeInfo cl)

Basic signal processing

integral :: VectorSpace v => ClSF m cl v v
...
ExceptT...

data Either e a = Left e | Right a
newtype ExceptT e m a = ExceptT (m (Either e a))
Setup

Quick introduction to Rhine

Let's hack!

After the tutorial

Exceptions and control flow

ExceptT...

data Either e a = Left e | Right a
newtype ExceptT e m a = ExceptT (m (Either e a))

...control flow! (Thanks to Paolo Capriotti)

-- dunai, rhine (simplified)
newtype ClSFExcept m cl a b e
   = ClSFExcept (SyncSF (ExceptT e m) cl a b)
**ExceptT...**

```haskell
data Either e a = Left e | Right a
newtype ExceptT e m a = ExceptT (m (Either e a))
```

**...control flow! (Thanks to Paolo Capriotti)**

```haskell
-- dunai, rhine (simplified)
newtype ClSFExcept m cl a b e
  = ClSFExcept (SyncSF (ExceptT e m) cl a b)

instance Monad m => Monad (ClSFExcept m cl a b)

throwOn' :: ClSF (ExceptT e m) cl (Bool, e) ()
try :: ClSF (ExceptT e m) cl a b
    -> ClSFExcept m cl a b e
safely :: ClSFExcept m cl a b Empty -> SyncSF m cl a b
safe :: ClSF m cl a b -> SyncExcept m cl a b e
```
Hello World!

type SumClock = Millisecond 100

fillUp :: CLSF (ExceptT Double m) SumClock Double ()
fillUp = proc x -> do
  s <- integral <- x
  _ <- throwOn' <- (s > 5, s)
  returnA <- ()

helloWorld :: CLSFEExcept IO SumClock () () Empty
helloWorld = do
  try $ arr (const 1) >>> fillUp
  once_ $ putStrLn "Hello World!"
  helloWorld

main = flow $ safely helloWorld @@ waitClock
Clock safety

```haskell
fastSignal :: ClSF m FastClock () a
slowProcessor :: ClSF m SlowClock a b

clockTypeError = fastSignal >>> slowProcessor
```

PresentationExamples.hs:67:33: error:
  ● Couldn't match type ‘SlowClock’ with ‘FastClock’
Asynchronous FRP

| data Schedule m cl1 cl2 |
Asynchronous FRP

```haskell
data Schedule m cl1 cl2
```

**Binary schedules**

Execute two different clocks simultaneously.
data Schedule m cl1 cl2

Binary schedules

Execute two different clocks simultaneously.

- Can be clock-polymorphic or specific to certain clocks.
data Schedule m cl1 cl2

Binary schedules

Execute two different clocks simultaneously.

- Can be clock-polymorphic or specific to certain clocks.
- (No implementation details here.)
Asynchronous FRP

```haskell
data Schedule m cl1 cl2
```

**Binary schedules**

Execute two different clocks simultaneously.

- Can be clock-polymorphic or specific to certain clocks.
- (No implementation details here.)
- Some examples:
Asynchronous FRP

Setup
Quick introduction to Rhine
Let’s hack!
After the tutorial

Asynchronous FRP

```haskell
data Schedule m cl1 cl2
```

Binary schedules

Execute two different clocks simultaneously.

- Can be clock-polymorphic or specific to certain clocks.
- (No implementation details here.)
- Some examples:
  - `concurrently :: Schedule IO cl1 cl2`

Asynchronous FRP

```haskell
data Schedule m cl1 cl2
```

**Binary schedules**

Execute two different clocks simultaneously.

- Can be clock-polymorphic or specific to certain clocks.
- (No implementation details here.)
- Some examples:
  - `concurrently :: Schedule IO cl1 cl2`
  - `schedule :: Schedule (ScheduleT m) cl1 cl2`
data ResamplingBuffer m cla clb a b = ResamplingBuffer
{ put :: TimeInfo cla -> a
  -> m ( ResamplingBuffer m cla clb a b)
, get :: TimeInfo clb
  -> m (b, ResamplingBuffer m cla clb a b)
}
data ResamplingBuffer m cla clb a b = ResamplingBuffer
    { put :: TimeInfo cla -> a
      -> m ( ResamplingBuffer m cla clb a b)
    , get :: TimeInfo clb
      -> m (b, ResamplingBuffer m cla clb a b)
    }

Resampling buffers
Buffer data at the boundary between two asynchronous systems.
data ResamplingBuffer m cla clb a b = ResamplingBuffer
  { put :: TimeInfo cla -> a
    -> m ( ResamplingBuffer m cla clb a b)
    , get :: TimeInfo clb
       -> m (b, ResamplingBuffer m cla clb a b)
  }

Resampling buffers

Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
data ResamplingBuffer m cla clb a b = ResamplingBuffer
    { put :: TimeInfo cla -> a
        -> m ( ResamplingBuffer m cla clb a b)
    , get :: TimeInfo clb
        -> m (b, ResamplingBuffer m cla clb a b)
    }

Resampling buffers

Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
- Some examples
resampling buffers

Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
- Some examples
  - `collect :: ResamplingBuffer m cl1 cl2 a [a]`
data ResamplingBuffer m cla clb a b = ResamplingBuffer
  { put :: TimeInfo cla -> a
    -> m ( ResamplingBuffer m cla clb a b)
  , get :: TimeInfo clb
    -> m (b, ResamplingBuffer m cla clb a b)
  }

Resampling buffers

Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
- Some examples
  - collect :: ResamplingBuffer m cl1 cl2 a [a]
  - fifo :: ResamplingBuffer m cl1 cl2 a (Maybe a)
data ResamplingBuffer m cla clb a b = ResamplingBuffer
  { put :: TimeInfo cla -> a
      -> m ( ResamplingBuffer m cla clb a b)
    , get :: TimeInfo clb
      -> m (b, ResamplingBuffer m cla clb a b)
  }

Resampling buffers
Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
- Some examples
  - collect :: ResamplingBuffer m cl1 cl2 a [a]
  - fifo :: ResamplingBuffer m cl1 cl2 a (Maybe a)
  - keepLast :: a -> ResamplingBuffer m cl1 cl2 a a
data ResamplingBuffer m cla clb a b = ResamplingBuffer
  { put :: TimeInfo cla -> a
    -> m (  ResamplingBuffer m cla clb a b)
  , get :: TimeInfo clb
    -> m (b,  ResamplingBuffer m cla clb a b)
  }

Resampling buffers
Buffer data at the boundary between two asynchronous systems.

- Can be clock-polymorphic or specific to certain clocks.
- Some examples
  - collect :: ResamplingBuffer m cl1 cl2 a [a]
  - fifo :: ResamplingBuffer m cl1 cl2 a (Maybe a)
  - keepLast :: a -> ResamplingBuffer m cl1 cl2 a a
  - Linear interpolation, combinators to build your own...
Asynchronous FRP

Asynchronous signal functions

```haskell
data SN m cl a b -- Signal network

type family In cl

type family Out cl
```
Asynchronous signal functions

data SN m cl a b -- Signal network

type family In cl

type family Out cl

A clocked reactive program

data Rhine m cl a b

(...basically an SF and a matching clock!)
Asynchronous signal functions

```haskell
data SN m cl a b -- Signal network  
type family In cl  
type family Out cl
```

A clocked reactive program

```haskell
data Rhine m cl a b  
(...basically an SF and a matching clock!)
```

Execution (reactimation)

```haskell
flow :: Rhine m cl () () -> m ()
```
Synchronous subsystems

```haskell
cl :: MyClock
sf :: ClSF m MyClock A B
rhineCl :: Rhine m MyClock A B
rhineCl = sf @@ cl
```
Parallel composition

cL :: MyClockL
cR :: MyClockR
sfL :: ClSF m MyClockL C D
sfR :: ClSF m MyClockR C D
schedPar :: Schedule m MyClockL MyClockR
rhinePar = sfL @@ clL @@ schedPar @@ syncsfR @@ cR
Parallel composition

cL :: MyClockL
clR :: MyClockR
sfL :: ClSF m MyClockL C D
sfR :: ClSF m MyClockR C D
schedPar :: Schedule m MyClockL MyClockR
rhinePar = sfL @@ clL **@ schedPar @** syncsfR @@ clR

Sequential composition

buf :: ResamplingBuffer m MyClock (In (..)) B C
schedSeq :: Schedule m ...
rhineSeq = rhineCl >-- buf -@- schedSeq --> rhineP
The plan: A tea app

- Run several tea timers in parallel
- Reactively read tea requests from the console
The plan: A tea app

- Run several tea timers in parallel
- Reactively read tea requests from the console

Any questions before we start hacking?
Have fun!
What else you could (easily) do with Dunai and Rhine

- Simple arcade games (SDL, Gloss)
- Reactive console apps
What else you could (easily) do with Dunai and Rhine

- Simple arcade games (SDL, Gloss)
- Reactive console apps

What should be doable, but I didn’t do yet because of lazyness

- Webservers, server-side web apps
- Interactive File I/O
- GUI programs
- External devices (e.g. Kinect, Wiimote)
What Rhine can do

What else you could (easily) do with Dunai and Rhine
- Simple arcade games (SDL, Gloss)
- Reactive console apps

What should be doable, but I didn’t do yet because of lazyness
- Webservers, server-side web apps
- Interactive File I/O
- GUI programs
- External devices (e.g. Kinect, Wiimote)

What might eventually be feasible
- Reactive audio synthesis, processing and analysis (performance...)
- Reactive web apps (GHCJS...)
- Android, embedded systems (recent GHC...).
## Comparison to other frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Pro Rhine</th>
<th>Contra Rhine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yampa, dunai</td>
<td>Asynchronicity, clock types</td>
<td>Performance</td>
</tr>
<tr>
<td>Pipes, conduit</td>
<td>FRP, clocks</td>
<td>Performance?</td>
</tr>
<tr>
<td>Most classical FRP</td>
<td>No \textbf{IO} built in, clock types</td>
<td>?</td>
</tr>
<tr>
<td>frameworks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C\lambda SH</td>
<td>General purpose</td>
<td>No compilation to circuits</td>
</tr>
</tbody>
</table>
Dunai
Dunai

- github.com/ivanperez-keera/dunai
Dunai

- github.com/ivanperez-keera/dunai
- There’s a link to the article!
Dunai
- github.com/ivanperez-keera/dunai
- There's a link to the article!

Rhine
Dunai
- [link](github.com/ivanperez-keera/dunai)
- There’s a link to the article!

Rhine
- Article: [github.com/turion/rhine#documentation](github.com/turion/rhine#documentation)
Dunai
- github.com/ivanperez-keera/dunai
- There’s a link to the article!

Rhine
- Article: github.com/turion/rhine#documentation
- This tutorial: github.com/turion/rhine-tutorial/
Dunai
- github.com/ivanperez-keera/dunai
- There’s a link to the article!

Rhine
- Article: github.com/turion/rhine#documentation
- This tutorial: github.com/turion/rhine-tutorial/
- Checkout branch final for solutions
### Dunai
- [github.com/ivanperez-keera/dunai](github.com/ivanperez-keera/dunai)
- There’s a link to the article!

### Rhine
- Article: [github.com/turion/rhine#documentation](github.com/turion/rhine#documentation)
- This tutorial: [github.com/turion/rhine-tutorial/](github.com/turion/rhine-tutorial/)
- Checkout branch `final` for solutions
- Documentation on hackage
Dunai
- github.com/ivanperez-keera/dunai
- There's a link to the article!

Rhine
- Article: github.com/turion/rhine#documentation
- This tutorial: github.com/turion/rhine-tutorial/
- Checkout branch final for solutions
- Documentation on hackage
- Simple examples in github.com/turion/rhine/
Use Rhine at the hackathon and win a nice bar of chocolate!
Use Rhine at the hackathon and win a nice bar of chocolate!

Create issues on github.com/turion/rhine/ and ask for your most needed clocks, schedules, resampling buffers etc.!
Use Rhine at the hackathon and win a nice bar of chocolate!
Create issues on github.com/turion/rhine/ and ask for your most needed clocks, schedules, resampling buffers etc.!
Look at easy to solve issues on github.com/ivanperez-keera/dunai!