Synthesizing Functional Reactive Programs

Bernd Finkbeiner[1], Felix Klein[1],
Ruzica Piskac[2], Mark Santolucito[2]

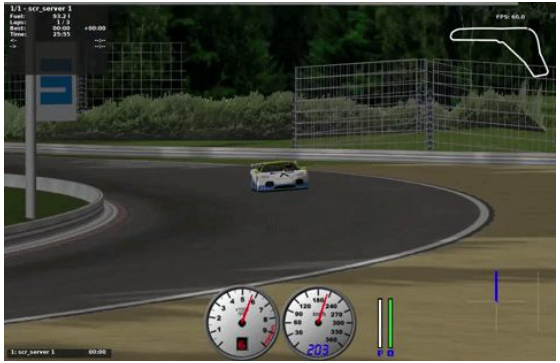
Saarland University[1], Yale University[2]

Reactive Systems

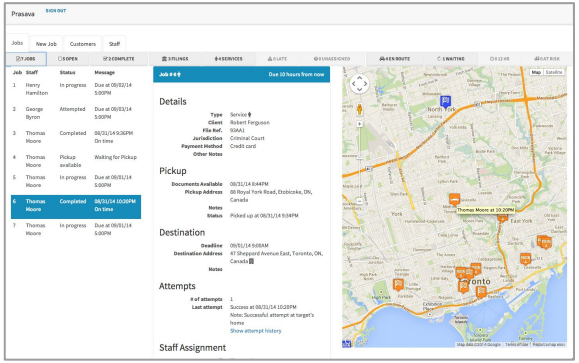
IBM PL Day, Dec 4, 2017

Functional Reactive Programming Applications

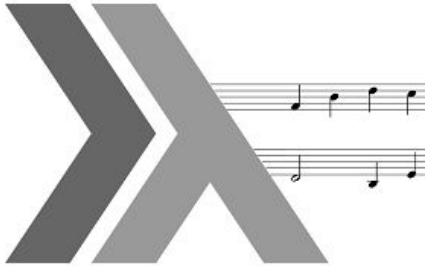
Embedded System

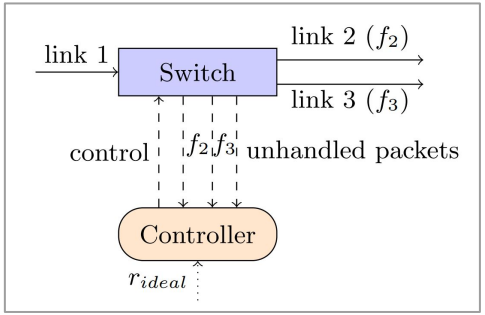


Networking and Web



Interactive Media

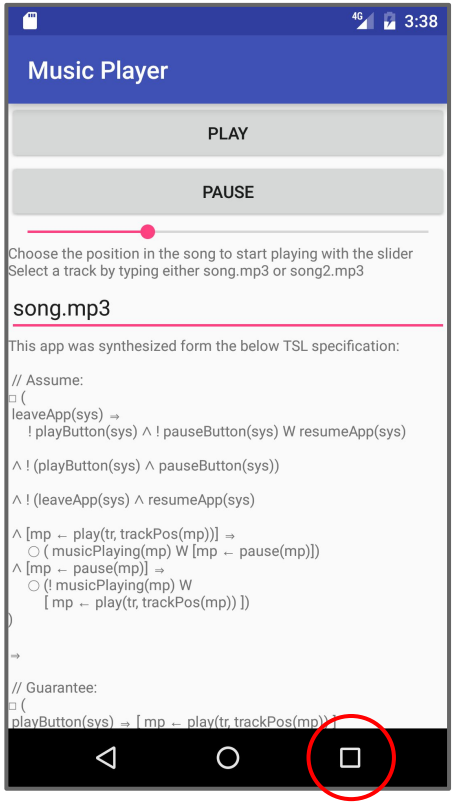




Synthesizing a music player app

```
Sys.leaveApp() {
    if (MP.musicPlaying())
        MP.pause();
}

Sys.resumeApp() {
    pos = MP.trackPos();
    MP.play(Tr,pos);
}
```

Finding resume and restart errors in android applications Shan, Z., Azim, T., Neamtiu, I OOPSLA 2016 / IBM PL Day 2016



Temporal Stream Logic (TSL)

```
Sys.leaveApp() {
    if (MP.musicPlaying())
        MP.pause();
}

Sys.resumeApp() {
    pos = MP.trackPos();
    MP.play(Tr,pos);
}
```

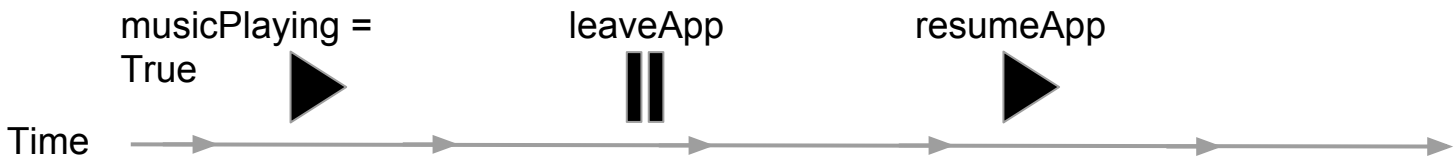
```
ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
        ⇒ [MP ◁ pause(MP))

ALWAYS (resumeApp(Sys)
        ⇒ [MP ◁ play(Tr,trackPos(MP))
```

A music player app in TSL

```
Sys.leaveApp() {
    if (MP.musicPlaying())
        MP.pause();
}

Sys.resumeApp() {
    pos = MP.trackPos();
    MP.play(Tr,pos);
}
```

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{pause}(\mathit{MP}))$$

$$\mathbf{ALWAYS}\ (\mathit{resumeApp}(\mathit{Sys}) \Rightarrow [\mathit{MP} \triangleleft \mathit{play}(\mathit{Tr}, \mathit{trackPos}(\mathit{MP})))$$

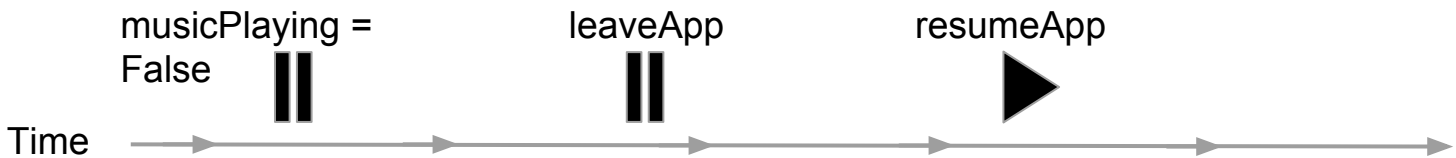


A music player app in TSL

```
Sys.leaveApp() {
    if (MP.musicPlaying())
        MP.pause();
}

Sys.resumeApp() {
    pos = MP.trackPos();
    MP.play(Tr,pos);
}
```

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{pause}(\mathit{MP}))$$

$$\mathbf{ALWAYS}\ (\mathit{resumeApp}(\mathit{Sys}) \Rightarrow [\mathit{MP} \triangleleft \mathit{play}(\mathit{Tr}, \mathit{trackPos}(\mathit{MP})))$$

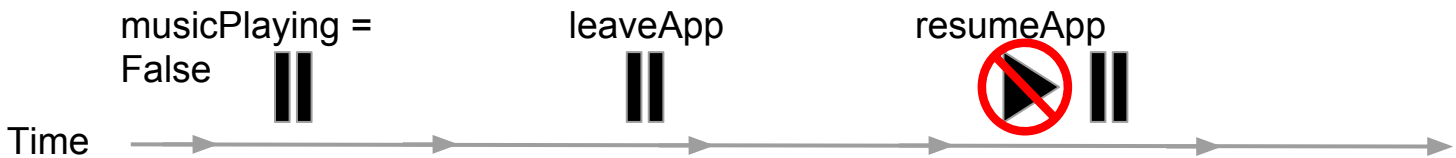


A music player app in TSL

```
Sys.leaveApp() {
    if (MP.musicPlaying())
        MP.pause();
}

Sys.resumeApp() {
    pos = MP.trackPos();
    MP.play(Tr,pos);
}
```

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{pause}(\mathit{MP}))$$

$$\mathbf{ALWAYS}\ (\mathit{resumeApp}(\mathit{Sys}) \Rightarrow [\mathit{MP} \triangleleft \mathit{play}(\mathit{Tr}, \mathit{trackPos}(\mathit{MP})))$$



A music player app in TSL

```
bool wasPlaying = false;

Sys.leaveApp() {
    if (MP.musicPlaying()){
        wasPlaying = true;
        MP.pause();}
    else {
        wasPlaying = false;}
}

Sys.resumeApp() {
    if (wasPlaying) {
        pos = MP.trackPos();
        MP.play(Tr,pos);}
}
```

On resume app, only play music if the music was already playing when paused.

A music player app in TSL

```
bool wasPlaying = false;

Sys.leaveApp() {
    if (MP.musicPlaying()){
        wasPlaying = true;
        MP.pause();}
    else {
        wasPlaying = false;}
}

Sys.resumeApp() {
    if (wasPlaying) {
        pos = MP.trackPos();
        MP.play(Tr,pos);}
}
```

On resume app, only play music if the music was already playing when paused.

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{pause}(\mathit{MP}))$$

$$\mathbf{ALWAYS}\ (\mathit{resumeApp}(\mathit{Sys}) \Rightarrow [\mathit{MP} \triangleleft \mathit{play}(\mathit{Tr}, \mathit{trackPos}(\mathit{MP})))$$

A music player app in TSL

```
bool wasPlaying = false;

Sys.leaveApp() {
    if (MP.musicPlaying()){
        wasPlaying = true;
        MP.pause();}
    else {
        wasPlaying = false;}
}

Sys.resumeApp() {
    if (wasPlaying) {
        pos = MP.trackPos();
        MP.play(Tr,pos);}
}
```

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{pause}(\mathit{MP}))$$

$$\mathbf{ALWAYS}\ (\mathit{leaveApp}(\mathit{Sys}) \wedge \mathit{musicPlaying}(\mathit{MP}) \Rightarrow [\mathit{MP} \triangleleft \mathit{play}(\mathit{Tr}, \mathit{trackPos}(\mathit{MP})]\ \mathbf{AS\_SOON\_AS}\ \mathit{resumeApp}(\mathit{Sys}))$$

$$\mathit{AS\_SOON\_AS} : \varphi\ A\ \psi \equiv \neg\psi\ W(\psi \Box \varphi)$$

Function abstraction

```
ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
        ⇒ [MP ◁ pause(MP))

ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
        ⇒ [MP ◁ play(Tr,trackPos(MP))
            AS_SOON_AS resumeApp(Sys) )
```

Control

```
leaveApp(Sys){ ... }

musicPlaying(MP) { ... }

play(Tr,trackPos(MP)) { ... }

resumeApp(Sys) { ... }
```

Pure Data Transformations

A button counter in Arrowized FRP

```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count    ← init 0    -< newCount
        newCount ← arr f1    -< (click, count)
        pic      ← arr f2    -< count
    returnA -< pic
```

Control

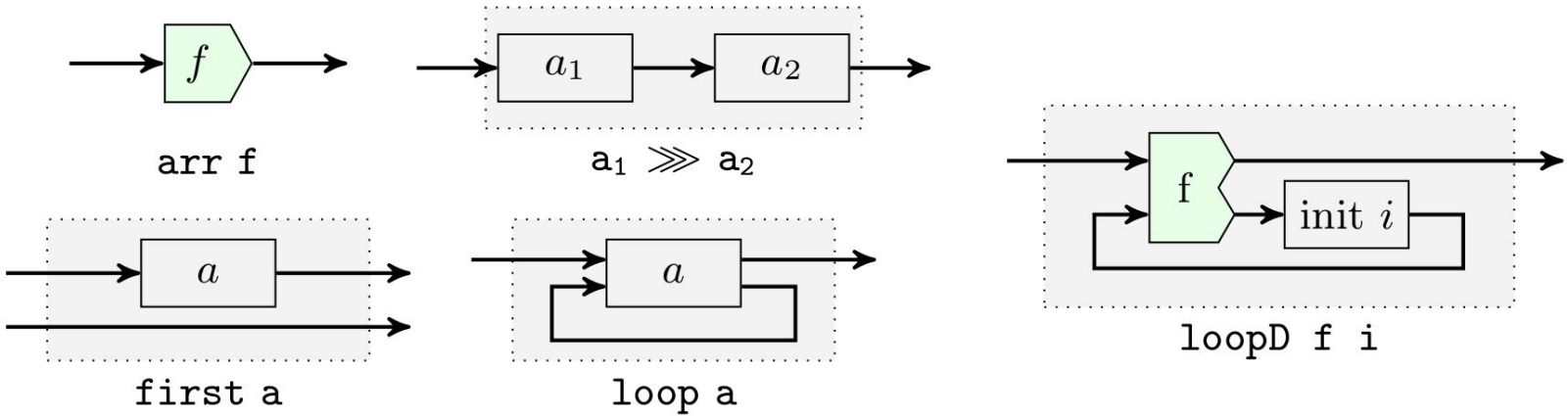
```
f1 :: (Event MouseClick, Int) → Int
f1 (click, count)
    | isEvent click = count + 1
    | otherwise  = count

f2 :: Int → Picture
f2 count = render count
```

Pure Data Transformations

Arrow combinators for control

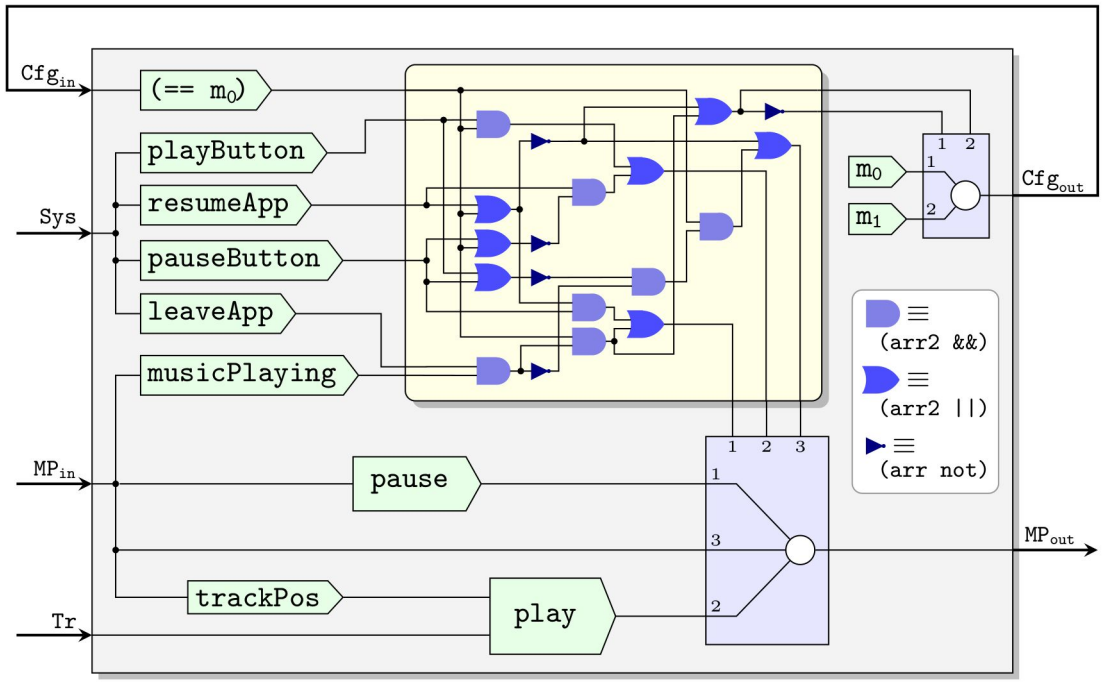
```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count    ← init 0    ⤙ newCount
        newCount ← arr f1    ⤙ (click, count)
        pic      ← arr f2    ⤙ count
    returnA ⤙ pic
```

Control



TSL/FRP synthesis goal

TSL formula $\varphi \Rightarrow$



In summary

A button counter in FRP

```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count     ← init 0    → newCount
        newCount  ← arr f1    → (click, count)
        pic       ← arr f2    → count
    returnA → pic
```

Control

```
f1 :: (Event MouseClick, Int) → Int
f1 (click, count)
    | isEvent click = count + 1
    | otherwise  = count

f2 :: Int → Picture
f2 count = render count
```

Pure Data Transformations

In summary

A button counter in FRP

```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count     ← init 0     → newCount
        newCount  ← arr f1     → (click, count)
        pic       ← arr f2     → count
    returnA → pic
```

Control

```
f1 :: (Event MouseClick, Int) → Int
f1 (click, count)
    | isEvent click = count + 1
    | otherwise  = count

f2 :: Int → Picture
f2 count = render count
```

Pure Data Transformations

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{[MP } \triangleleft \texttt{ pause(MP))}$$

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{[MP } \triangleleft \texttt{ play(Tr,trackPos(MP))} \texttt{ AS\_SOON\_AS resumeApp(Sys))}$$

In summary

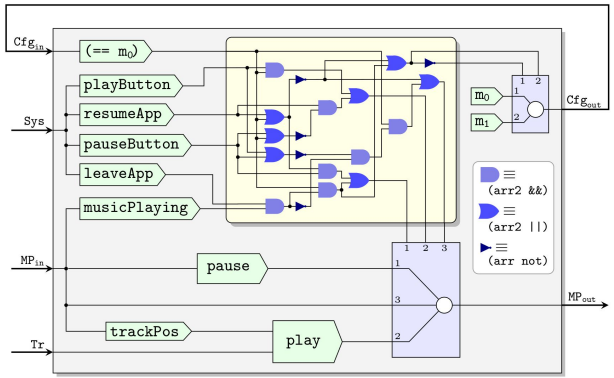
A button counter in FRP

```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count     ← init 0    → newCount
        newCount  ← arr f1    → (click, count)
        pic       ← arr f2    → count
    returnA → pic
```

Control

```
f1 :: (Event MouseClick, Int) → Int
f1 (click, count)
    | isEvent click = count + 1
    | otherwise  = count

f2 :: Int → Picture
f2 count = render count
```

Pure Data Transformations



$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{ [MP } \triangleleft \texttt{ pause(MP))}$$

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{ [MP } \triangleleft \texttt{ play(Tr,trackPos(MP))} \texttt{ AS\_SOON\_AS resumeApp(Sys))}$$

In summary

A button counter in FRP

```
yampaButton :: SF (Event MouseClick) Picture
yampaButton = proc click → do
    rec
        count     ← init 0   → newCount
        newCount  ← arr f1   → (click, count)
        pic       ← arr f2   → count
    returnA → pic
```

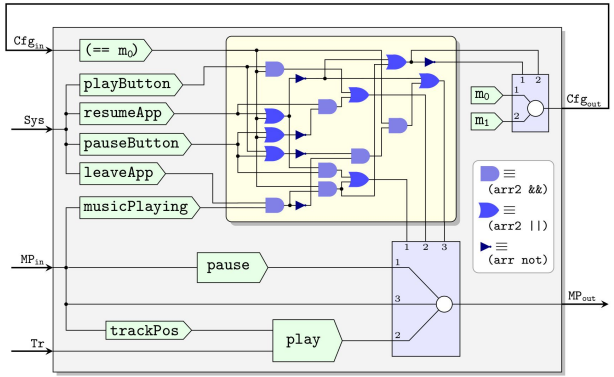
Control

```
f1 :: (Event MouseClick, Int) → Int
f1 (click, count)
    | isEvent click = count + 1
    | otherwise  = count

f2 :: Int → Picture
f2 count = render count
```

Pure Data Transformations

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{[MP} \triangleleft \texttt{ pause(MP))}$$

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow \texttt{[MP} \triangleleft \texttt{ play(Tr,trackPos(MP))} \texttt{ AS\_SOON\_AS resumeApp(Sys))}$$



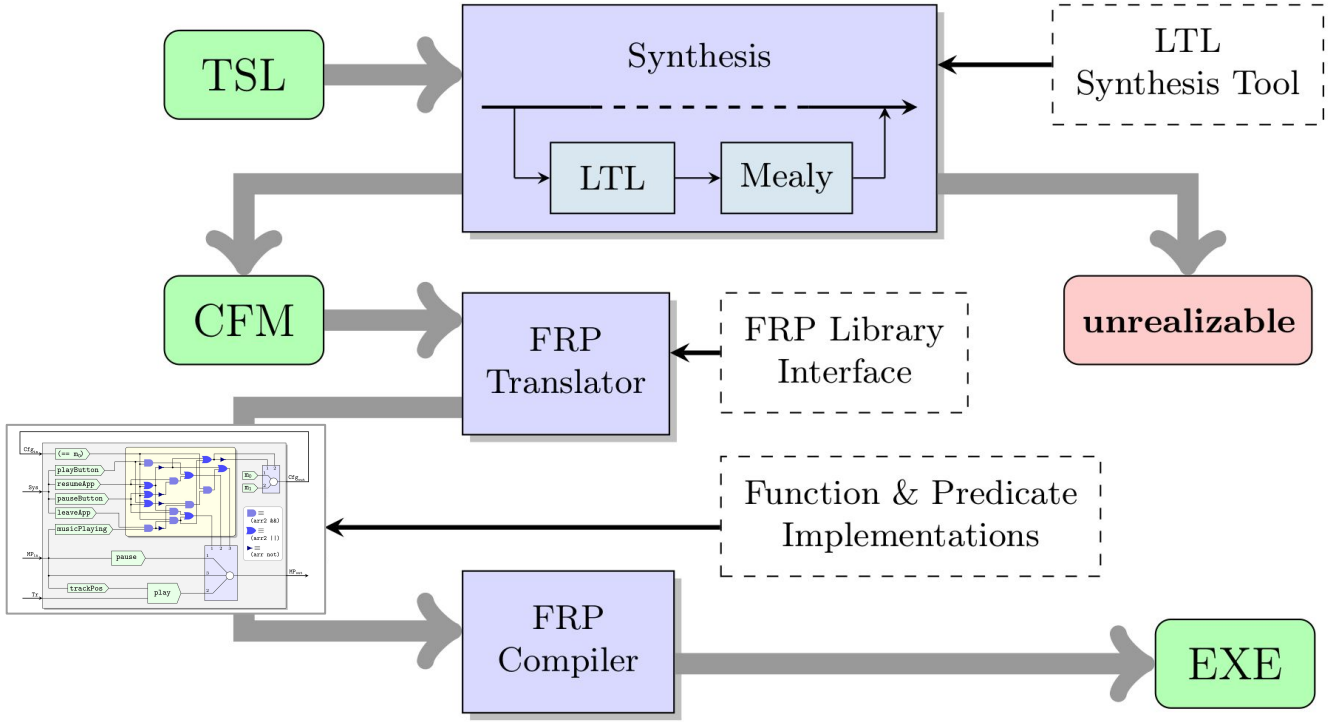
Time in FRP

discrete
Event a :: (Time,a)

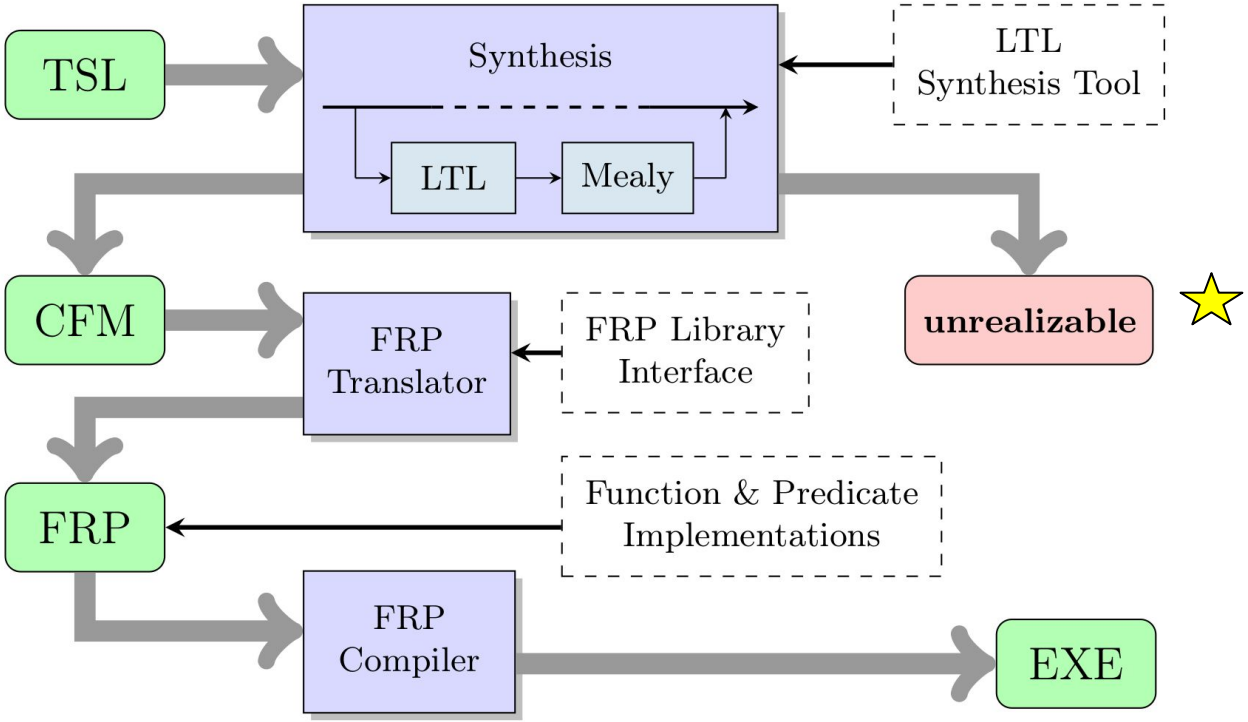
continuous
Signal a :: Time → a

SF a a :: Signal a → Signal a

Overview of synthesis procedure



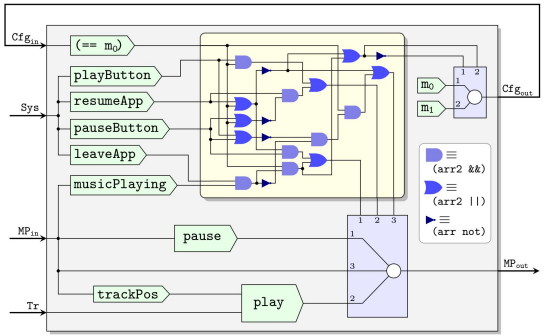
Overview of synthesis procedure



Overview of synthesis procedure

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow [\texttt{MP} \triangleleft \texttt{pause(MP)})$$

$$\texttt{ALWAYS (leaveApp(Sys) } \wedge \texttt{ musicPlaying(MP)} \Rightarrow [\texttt{MP} \triangleleft \texttt{play(Tr,trackPos(MP))}\ \texttt{AS\_SOON\_AS resumeApp(Sys))}$$

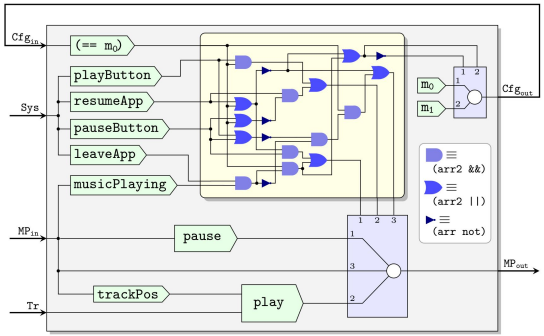


$$\begin{array}{l} \texttt{pauseButton(Sys)} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{pause}(\texttt{MP}_{\texttt{in}})] \\ \texttt{playButton(Sys)} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{play}(\texttt{Tr}, \texttt{trackPos}(\texttt{MP}_{\texttt{in}}))] \\ \texttt{otherwise} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{MP}_{\texttt{in}}] \\ \hline \texttt{musicPlaying}(\texttt{MP}_{\texttt{in}}) \wedge \texttt{leaveApp(Sys)} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{pause}(\texttt{MP}_{\texttt{in}})] \end{array}$$

$$\begin{array}{l} \overline{\texttt{resumeApp(Sys)}} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{MP}_{\texttt{in}}] \\ \hline \texttt{resumeApp(Sys)} \wedge \texttt{pauseButton(Sys)} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{pause}(\texttt{MP}_{\texttt{in}})] \\ \texttt{resumeApp(Sys)} \wedge \overline{\texttt{pauseButton(Sys)}} \rightarrow [\texttt{MP}_{\texttt{out}} \triangleleft \texttt{play}(\texttt{Tr}, \texttt{trackPos}(\texttt{MP}_{\texttt{in}}))] \end{array}$$

Overview of synthesis procedure

```
ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
          ⇒ [MP ◁ pause(MP))

ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
          ⇒ [MP ◁ play(Tr,trackPos(MP))
               AS_SOON_AS resumeApp(Sys))
```



$\overline{\text{(wasPlaying)}}$

$$
\begin{array}{lcl}
\text{pauseButton}(\text{Sys}) & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{pause}(\text{MP}_{\text{in}})\,] \\
\text{playButton}(\text{Sys}) & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{play}(\text{Tr}, \text{trackPos}(\text{MP}_{\text{in}}))\,] \\
\text{otherwise} & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{MP}_{\text{in}}\,] \\
\hline
\text{musicPlaying}(\text{MP}_{\text{in}}) \wedge \text{leaveApp}(\text{Sys}) & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{pause}(\text{MP}_{\text{in}})\,]
\end{array}
$$

(wasPlaying)

$$
\begin{array}{lcl}
\overline{\text{resumeApp}(\text{Sys})} & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{MP}_{\text{in}}\,] \\
\hline
\text{resumeApp}(\text{Sys}) \wedge \text{pauseButton}(\text{Sys}) & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{pause}(\text{MP}_{\text{in}})\,] \\
\text{resumeApp}(\text{Sys}) \wedge \overline{\text{pauseButton}(\text{Sys})} & \rightarrow & [\,\text{MP}_{\text{out}} \lhd \text{play}(\text{Tr}, \text{trackPos}(\text{MP}_{\text{in}}))\,]
\end{array}
$$

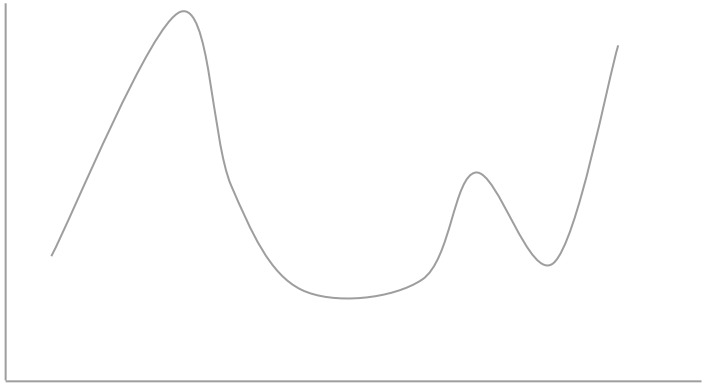
Time in FRP

discrete

Event a :: (Time,a)

continuous

Signal a :: Time → a



SF a a :: Signal a → Signal a

A discrete button-continuous counter in FRP

```
yampaButton :: SF (Event MouseClick, Event MouseClick) Picture
yampaButton = proc (click, toggle) → do
    rec
        countMode ← accumHold True     ⤙ isEvent toggle
        count     ← init 0             ⤙ newCount

        t         ← init 0             ⤙ t'
        t'        ← time               ⤙ ()

        newCount  ← if countMode
                      then integral    ⤙ (t'-t)
                      else arr id      ⤙ count

        pic       ← arr render ⤙ count
    returnA ⤙ pic
```

// Constant sum over time,
// regardless of sample rate

A bouncing ball in FRP

```
fallingBall :: Pos -> Vel -> SF () (Pos, Vel)
fallingBall y0 v0 =
    proc _ -> do
        v ← integral >>^ (+ v0)        -≺ (-9.81)
        y ← integral >>^ (+ y0)        -≺ v
        returnA -≺ (y, v)

bouncingBall :: Pos -> Vel -> SF () (Pos, Vel)
bouncingBall y0 v0 = switch (bb y0 v0) bswitch
    where
        bb y0' v0' = proc input -> do
            (pos, vel)    ← fallingBall y0' v0'          -≺ input
            event'        ← edge                         -≺ pos <= 0
            returnA -≺ ((pos, vel), event' `tag` (pos, vel))
        bswitch (pos, vel)
            | abs vel < 1.0 = constant (0.0, 0.0)
            | otherwise     = bouncingBall pos (-vel * 0.6)
```