ITU FRP 2010

Lecture 1: Introduction, Classic FRP

Henrik Nilsson

School of Computer Science

University of Nottingham, UK

Overview

- Lectures and practical exercises
- Course web page:

  `http://www.cs.nott.ac.uk/~nhn/ITU-FRP2010`
- Outline is tentative:
  - Hard to know how long the the practical bits will take: should not rush unduly.
  - Happy to adapt.

This Lecture

- Brief introduction to FRP:
  - Central ideas
  - Key notions
  - Applications
  - FRP variants
- Classical FRP
  - Basic combinators
  - Semantics

Reactive Programming

Reactive systems:

Reactive Programming

Reactive systems:

- Input arrives ***incrementally*** while system is running.

Reactive Programming

Reactive systems:

- Input arrives ***incrementally*** while system is running.
- Output is generated in response to input in an interleaved and ***timely*** fashion.

Reactive Programming

Reactive systems:

- Input arrives ***incrementally*** while system is running.
- Output is generated in response to input in an interleaved and ***timely*** fashion.

Contrast ***transformational systems***.

Reactive Programming

Reactive systems:

- Input arrives ***incrementally*** while system is running.
- Output is generated in response to input in an interleaved and ***timely*** fashion.

Contrast ***transformational systems***.

The notions of

- time
- time-varying values, or ***signals***

are inherent and central for reactive systems.

Functional Reactive Programming

What is Functional Reactive Programming (FRP)?

- Paradigm for reactive programming in a functional setting.

Functional Reactive Programming

What is Functional Reactive Programming (FRP)?

- Paradigm for reactive programming in a functional setting.
- Typically realised as an ***Embedded Domain-Specific Language (EDSL)***. The host langage is often Haskell. But also Scheme (FrTime) (and Java, and C++, and . . .)

Functional Reactive Programming

What is Functional Reactive Programming (FRP)?

- Paradigm for reactive programming in a functional setting.
- Typically realised as an ***Embedded Domain-Specific Language (EDSL)***. The host langage is often Haskell. But also Scheme (FrTime) (and Java, and C++, and . . .)
- Originated from Functional Reactive Animation (Fran) (Elliott & Hudak).

Functional Reactive Programming

What is Functional Reactive Programming (FRP)?

- Paradigm for reactive programming in a functional setting.
- Typically realised as an ***Embedded Domain-Specific Language (EDSL)***. The host langage is often Haskell. But also Scheme (FrTime) (and Java, and C++, and . . .)
- Originated from Functional Reactive Animation (Fran) (Elliott & Hudak).
- Has evolved in a number of directions and into different concrete implementations.

FRP Applications (1)

Some domains where FRP or FRP-inspired approaches have been used:

- Graphical Animation (Fran: Elliott, Hudak)
- Robotics (Frob: Peterson, Hager, Hudak, Elliott, Pembeci, Nilsson)
- Vision (FVision: Peterson, Hudak, Reid, Hager)
- GUIs (Fruit: Courtney; Grapefruit: Jeltsch)
- Games (Courtney, Nilsson, Peterson, Cheong, . . .)

FRP Applications (2)

- Virtual Reality Environments (Blom)
- Sound synthesis (Giorgidze, Nilsson)
- (Non-causal) modeling and simulation (Nilsson, Hudak, Peterson, Giorgidze)
- Experiment descriptions (Nielsen, Matheson, Nilsson)

Key FRP Features

- First class reactive entities.

Key FRP Features

- First class reactive entities.
- Synchronous: all system parts operate in synchrony.

Key FRP Features

- First class reactive entities.
- Synchronous: all system parts operate in synchrony.
- Support for hybrid (mixed continuous and discrete time) systems.

Key FRP Features

- First class reactive entities.
- Synchronous: all system parts operate in synchrony.
- Support for hybrid (mixed continuous and discrete time) systems.
- Allows dynamic system structure.

Related Languages and Paradigms

FRP related to:

- Synchronous languages, like Esterel, Lucid Synchrone.

Related Languages and Paradigms

FRP related to:

- Synchronous languages, like Esterel, Lucid Synchrone.
- Modeling languages, like Simulink, Modelica.

Central Notions (1)

Central Notions (1)

- Time-varying value or ***Signal***. Intuition:

  $\texttt{Signal}\ \alpha \approx \texttt{Time} \rightarrow \alpha$

Central Notions (1)

- Time-varying value or ***Signal***. Intuition:

  $$\texttt{Signal}\ \alpha \approx \texttt{Time} \rightarrow \alpha$$

- ***Signal Generator***: maps a ***start time*** to a signal. Intuition:

  $$\texttt{SG}\ \alpha \approx \texttt{Time} \rightarrow \texttt{Signal}\ \alpha$$

Central Notions (1)

- Time-varying value or ***Signal***. Intuition:

  $\texttt{Signal}\ \alpha \approx \texttt{Time} \rightarrow \alpha$

- ***Signal Generator***: maps a ***start time*** to a signal. Intuition:

  $\texttt{SG}\ \alpha \approx \texttt{Time} \rightarrow \texttt{Signal}\ \alpha$

- ***Signal Function***: maps a signal to a signal. Intuition:

  $\texttt{SF}\ \alpha\ \beta \approx \texttt{Signal}\ \alpha \rightarrow \texttt{Signal}\ \beta$

Central Notions (2)

Additionally, general ***causality*** requirement: output at time t must be determined by input on interval $[0, t]$.

Central Notions (2)

Additionally, general ***causality*** requirement: output at time t must be determined by input on interval $[0, t]$.

Signal functions are said to be

- ***pure*** or ***stateless*** if output at time t only depends on input at time t

Central Notions (2)

Additionally, general ***causality*** requirement: output at time t must be determined by input on interval $[0, t]$.

Signal functions are said to be

- ***pure*** or ***stateless*** if output at time t only depends on input at time t
- ***impure*** or ***stateful*** if output at time t depends on input over the interval $[0, t]$.

Central Notions (2)

Additionally, general ***causality*** requirement: output at time t must be determined by input on interval $[0, t]$.

Signal functions are said to be

- ***pure*** or ***stateless*** if output at time t only depends on input at time t
- ***impure*** or ***stateful*** if output at time t depends on input over the interval $[0, t]$.

Generally also a notion of ***discrete time***.

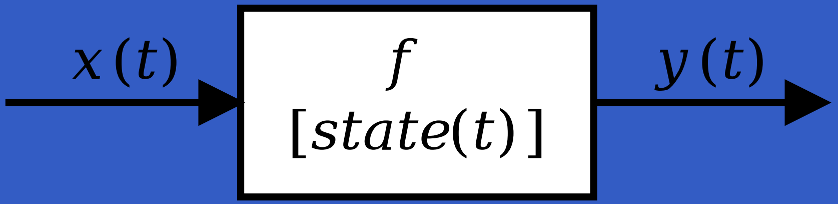
Signal Functions and State

Alternative view:

Signal Functions and State

Alternative view:

Signal functions can encapsulate ***state***.

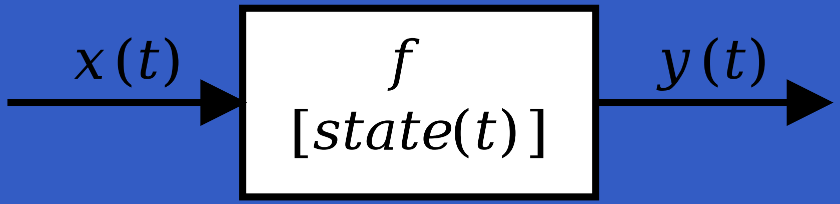


$state(t)$ summarizes input history $x(t')$, $t' \in [0, t]$.
Thus, really a kind of ***process***.

Signal Functions and State

Alternative view:

Signal functions can encapsulate ***state***.



$state(t)$ summarizes input history $x(t')$, $t' \in [0, t]$. Thus, really a kind of ***process***.

From this perspective, signal functions are:

- ***stateful*** if $y(t)$ depends on $x(t)$ and $state(t)$
- ***stateless*** if $y(t)$ depends only on $x(t)$

FRP Variants

A number of FRP variants have emerged. Key differences include what the central abstractions are. Some examples:

FRP Variants

A number of FRP variants have emerged. Key differences include what the central abstractions are. Some examples:

- Classic FRP: First class signal generators.

FRP Variants

A number of FRP variants have emerged. Key differences include what the central abstractions are. Some examples:

- Classic FRP: First class signal generators.
- Extended Classic FRP: First class signal generators and signals.

FRP Variants

A number of FRP variants have emerged. Key differences include what the central abstractions are. Some examples:

- Classic FRP: First class signal generators.
- Extended Classic FRP: First class signal generators and signals.
- Yampa: First class signal functions, signals a secondary notion.

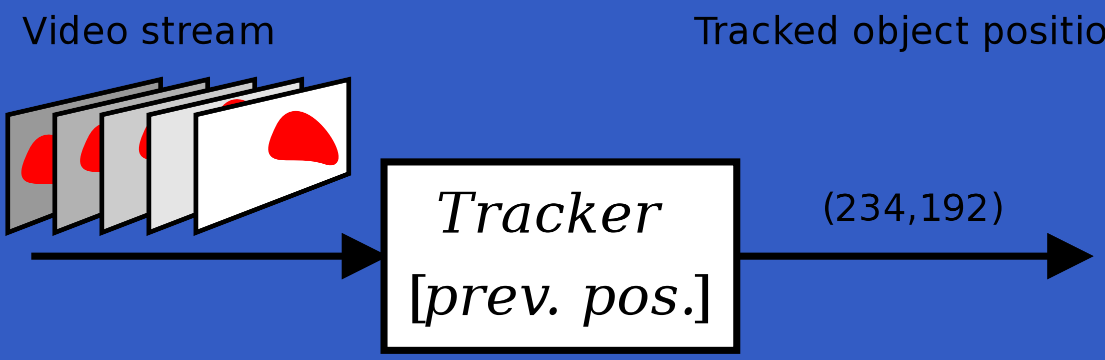
FRP Variants

A number of FRP variants have emerged. Key differences include what the central abstractions are. Some examples:

- Classic FRP: First class signal generators.
- Extended Classic FRP: First class signal generators and signals.
- Yampa: First class signal functions, signals a secondary notion.
- Elerea: First class signals and signal generators.

Example: Video Tracker

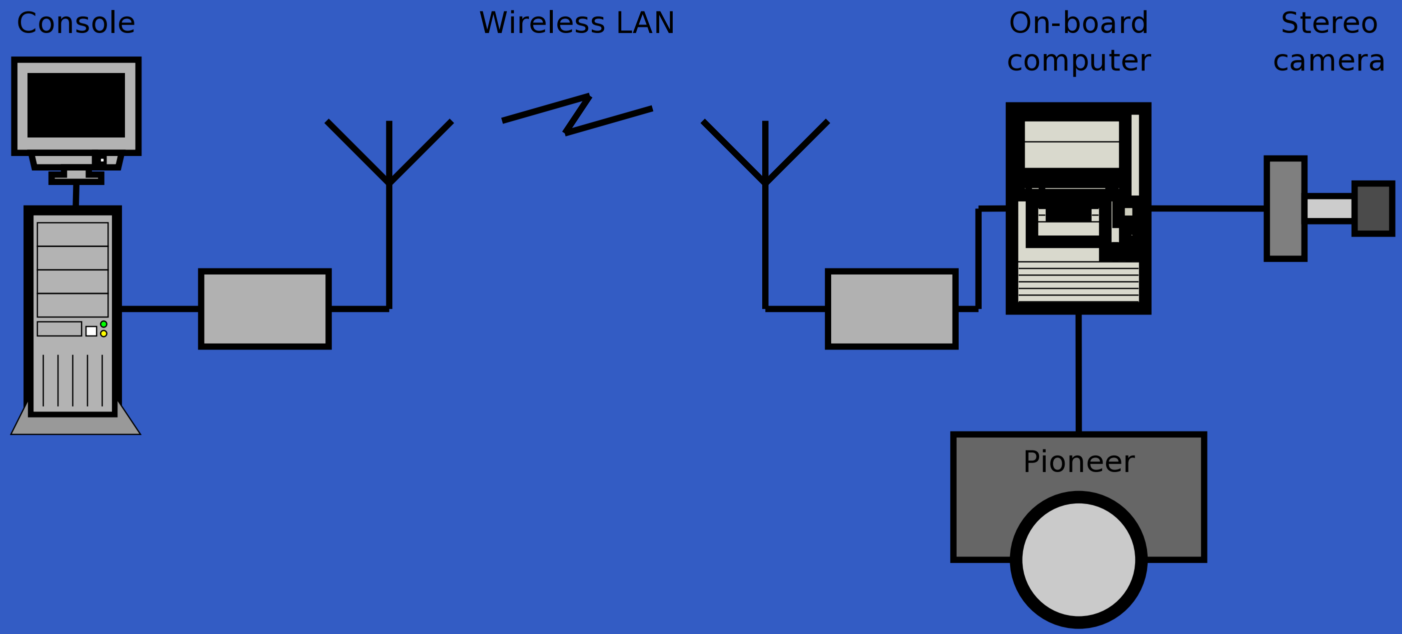
Video trackers are typically stateful signal functions:



Example: Robotics (1)

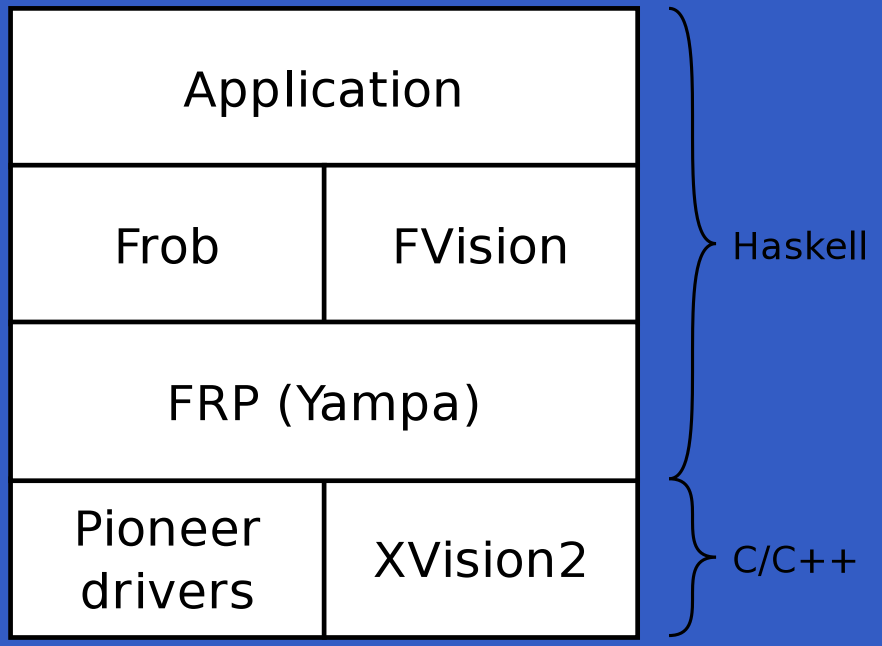
[PPDP'02, with Izzet Pembeci and Greg Hager, Johns Hopkins University]

Hardware setup:



Example: Robotics (2)

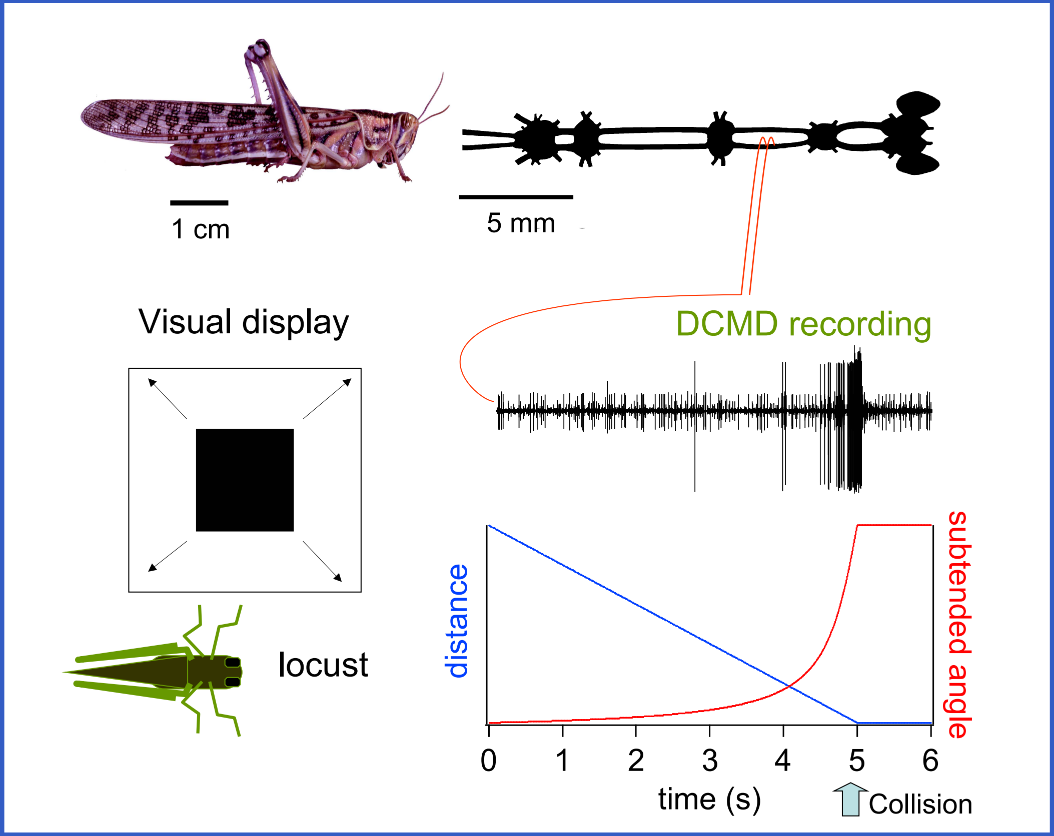
Software architecture:



Example: Robotics (3)

Example: Neuroscience Experiments

[TFP'09, Tom Nielsen, Tom Matheson, Henrik Nilsson]



Classic FRP (1)

Classic FRP (CFRP): Fran derivative. Central abstractions:

- ***Behavior***:
  - Polymorphic, (conceptually) continuous-time, signal generator.
  - Type constructor: $\mathsf{B}\ \ \alpha$
- ***Event***:
  - Polymorphic, discrete-time, signal generator.
  - Type constructor: $\mathsf{E}\ \ \alpha$

Classic FRP (2)

Examples:

```
7         :: B Real
time      :: B Time
(+)       :: B Real → B Real → B Real
lift1     :: (α → β) →(B α → B β)
integral  :: B Real → B Real
```

Classic FRP (3)

Some more examples:

```
never       :: E α
now         :: E ()
after       :: Time → E ()
repeatedly  :: Time → E ()
edge        :: B Bool → E ()
hold        :: α → E α → B α
lbp         :: E ()
key         :: E Char
```

Classic FRP (4)

Switching and event mapping:

$\texttt{until} \;::\; \texttt{B}\ \alpha \rightarrow \texttt{E}\ (\texttt{B}\ \alpha) \rightarrow \texttt{B}\ \alpha$

$\texttt{==>} \;::\; \texttt{E}\ \alpha \rightarrow (\alpha \rightarrow \beta) \rightarrow \texttt{E}\ \beta$

$\texttt{-=>} \;::\; \texttt{E}\ \alpha \rightarrow \beta \rightarrow \texttt{E}\ \beta$

Typical CFRP Snippets (1)

```
color :: B Color
color = red ‘until‘ lbp -=> blue

ball :: B Picture
ball = paint color circ

circ :: B Region
circ = translate (cos time, sin time)
                 (circle 1)
```

Typical CFRP Snippets (2)

```
color2 = red ‘until‘
                  (lbp -=> blue)
                  .|.
                  (key -=> yellow)

color3 = red ‘until‘
              (edge (time >* 5) -=> blue)
```

Semantic Functions (1)

$$\begin{aligned} \textbf{at} &: \langle \mathrm{B}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow \alpha \\ \textbf{occ} &: \langle \mathrm{E}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow [\mathit{Time} \times \alpha] \end{aligned}$$

Semantic Functions (1)

$$\begin{aligned} \textbf{at} &: \langle \mathrm{B}_\alpha \rangle \to \mathit{Time} \to \mathit{Time} \to \alpha \\ \textbf{occ} &: \langle \mathrm{E}_\alpha \rangle \to \mathit{Time} \to \mathit{Time} \to [\mathit{Time} \times \alpha] \end{aligned}$$

Intuitively, $\mathbf{at}$ maps a behavior to a function from a ***start time*** and a ***time of interest*** to a value at that time.

Semantic Functions (1)

$$\begin{aligned} \mathbf{at} &: \langle \mathrm{B}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow \alpha \\ \mathbf{occ} &: \langle \mathrm{E}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow [\mathit{Time} \times \alpha] \end{aligned}$$

Intuitively, $\mathbf{at}$ maps a behavior to a function from a ***start time*** and a ***time of interest*** to a value at that time.

Note that the type of $\mathbf{at}$ can be parenthesized:

$$\langle \mathrm{B}_\alpha \rangle \rightarrow (\mathit{Time} \ \rightarrow \ (\mathit{Time} \ \rightarrow \ \alpha))$$

Thus, $\mathbf{at}$ maps a behavior to a ***signal generator***.

Semantic Functions (2)

$$\begin{aligned} \mathbf{at} &: \langle \mathrm{B}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow \alpha \\ \mathbf{occ} &: \langle \mathrm{E}_\alpha \rangle \rightarrow \mathit{Time} \rightarrow \mathit{Time} \rightarrow [\mathit{Time} \times \alpha] \end{aligned}$$

The function **occ** gives meaning to events in a similar way, but the result is a finite list of ***time-ascending*** event occurrences from the start time to the time of interest.

Semantics (1)

Time, liftings, integration:

$$
\begin{aligned}
\mathbf{at}[\![\texttt{time}]\!]\;T\;t &= t \\
\mathbf{at}[\![\texttt{lift0}\;c]\!]\;T\;t &= \lfloor c \rfloor \\
\mathbf{at}[\![\texttt{lift1}\;f\;b]\!]\;T\;t &= \lfloor f \rfloor\;(\mathbf{at}[\![b]\!]\;T\;t) \\
\mathbf{at}[\![\texttt{lift2}\;f\;b\;d]\!]\;T\;t &= \lfloor f \rfloor\;(\mathbf{at}[\![b]\!]\;T\;t)\;(\mathbf{at}[\![d]\!]\;T\;t) \\
\mathbf{at}[\![\texttt{integral}\;b]\!]\;T\;t &= \int_T^t (\mathbf{at}[\![b]\!]\;T\;\tau)\mathrm{d}\tau
\end{aligned}
$$

Semantics (2)

Basic events:

$$
\begin{aligned}
\mathbf{occ}[\![\texttt{never}]\!]\, T\; t &= [\,]\\
\mathbf{occ}[\![\texttt{now}]\!]\, T\; t &= [\,(T, ())\,]\\
\mathbf{occ}[\![\texttt{after}\ \tau]\!]\, T\; t &= \begin{cases} [\,] & T + \tau < t \\ [\,(T + \tau, ())\,] & \text{otherwise} \end{cases}
\end{aligned}
$$

Semantics (3)

$$\mathbf{occ}[\![\texttt{repeatedly}\ \tau]\!]\ T\ t = \begin{cases} [\,] & n = 0 \\ [\,(T+\tau, ()),\ (T+2\tau, ()), \\ \ \dots,\ (T+n\tau, ())\,] & \text{otherwise} \end{cases}$$

where $n \in \mathbb{N}$ is the largest number such that $T + n\tau \leq t$.

Semantics (4)

Intuitively, the predicate event:

$$\text{edge} :: \texttt{B Bool} \longrightarrow \texttt{E} \; ()$$

occurrs whenever the argument behavior changes from `False` to `True`.

However, surprisingly hard to characterize exactly (and, of course, not computable).

Semantics (5)

Semantics of `until`. Recall:

$$\texttt{until :: B } \alpha \rightarrow \texttt{E (B } \alpha\texttt{)} \rightarrow \texttt{B } \alpha$$

If

$$\mathbf{occ}[\![\mathrm{e}]\!]\ T\ t = [\,(t_1, \lfloor b_1 \rfloor),\ \ldots,\ (t_n, \lfloor b_n \rfloor)\,]$$

then, for any $\tau \in [T, t]$:

$$\mathbf{at}[\![b\ \texttt{until}\ e]\!]\ T\ t = \begin{cases} \mathbf{at}[\![b]\!]\ T\ \tau & n = 0 \text{ or } \tau < t_1 \\ \mathbf{at}[\![b_1]\!]\ t_1\ \tau & \text{otherwise} \end{cases}$$

Implementation

Using infinite lists as ***streams***, stream-based versions of the central CFRP abstractions can be realised as follows:

```
B a = [Time] -> [a]
E a = [Time] -> [Maybe a]
```

Note that this corresponds to ***signal generators***: A prefix of `[Time]` is a discretized approximation of an interval from the start time to the current time.

Faithfulness (1)

Of course, we can only hope to approximate the ideal, continuous semantics.

Faithfulness (1)

Of course, we can only hope to approximate the ideal, continuous semantics.

But, then, what is a ***faithful*** implementation?

Faithfulness (1)

Of course, we can only hope to approximate the ideal, continuous semantics.

But, then, what is a ***faithful*** implementation?

- Wan and Hudak (2000) adapts the notion of ***uniform convergence*** to the setting of CFRP.

Faithfulness (1)

Of course, we can only hope to approximate the ideal, continuous semantics.

But, then, what is a ***faithful*** implementation?

- Wan and Hudak (2000) adapts the notion of ***uniform convergence*** to the setting of CFRP.
- They then show that the stream-based semantics of the CFRP converges to the ideal semantics in the limit as the maximal sampling interval tends to 0, establishing necessary side conditions where needed.

Faithfulness (2)

- Wan and Hudak still assume real reals and exact functions on the reals. Floating point arithmetic adds another level of difficulty.

Reading

- Zhanyong Wan and Paul Hudak. Functional reactive programming from first principles. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’00)*, Canada, June, 2000.