

Semantics of temporal type systems

Dima Szamozvancev

Downing College
ds709@cam.ac.uk

Supervised by
Dr Neel Krishnaswami

Interactive programming

Event-driven programming

Callbacks

Event listeners

Event loop

Asynchronous programming

Event dispatching thread

Event handlers

Pros

Efficient

Widely used

Cons

Low-level

Complicated and
error-prone

Functional reactive programming

Signal $a \approx \text{Time} \rightarrow a$

Event $a \approx \text{Time} \times a$

```
redblue :: Signal Image
redblue u = withColor c
            (stretch (wiggleRange 0.5 1) circle)
  where c = red `until` lbp u ==> blue
```

Pros

Declarative

Compositional

Cons

Performance issues

Violates causality

Pull vs. push-based FRP

Pull-based
(Demand-driven)

Streams

Polling until an
event happens

Latency issues

High-level but
inefficient

Push-based
(Data-driven)

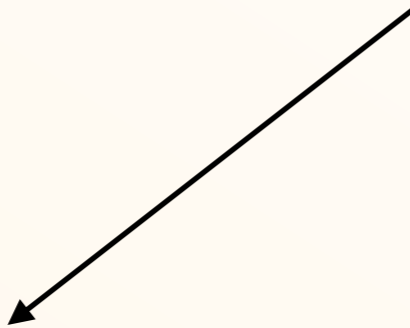
Callbacks

Asynchronous
event handling

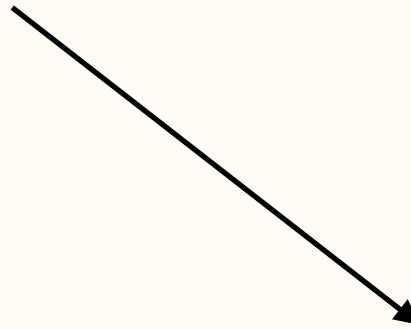
Instantaneous
reactivity

Low-level but
efficient

Can we combine intuitive semantics with performance and correctness?



Efficient FRP
implementations



Theoretical
foundations of FRP

Curry-Howard for FRP

Jeffrey (2012), Jeltsch (2012)

LTL

Propositions

□ modality

◇ modality

U modality

FRP

Reactive types

Behaviours

Events

Processes

Advantages of LTL

Differentiate constant (stable) and time-varying (reactive) values

Restrict event handlers to only use values that are always available

```
let event c = keyPress in
let colour =
  if c = 'r' then red else blue in
let event shape = selectShape in
  withColour colour shape
```


Disadvantages of LTL

Naive inductive implementation of events
(as an infinite sum) leads to polling

*An event happens now, or on the next time step,
or the one after that, ...*

Instead, events should be implemented
as an existential type

An event happens after some unknown delay.

LTL can lead to inefficient implementations

$(\diamond A)_n$ holds iff A_i holds for some $i \geq n$
 iff A_n holds or $(\bullet A)_n$ holds
 or $(\bullet^2 A)_n$ holds...

$$(\diamond A)_n \iff \mu X. A_n \vee (\bullet X)_n$$

```

data  $\diamond A$  = Now A
      | Later  $\bullet(\diamond A)$ 
  
```

```

case (e ::  $\diamond A$ ) of
  Now a → ...
  | Later l → ... polling!
  
```

$$(\diamond A)_n \iff \exists k \geq 0. (\bullet^k A)_n$$

$$\diamond A = \sum_{k \geq 0} \bullet^k A$$

```

case (e ::  $\diamond A$ ) of
  (k, a) → ...
  
```

Contributions

Categorical model of linear temporal logic
with a non-inductive diamond modality

Formalised high-level language
for reactive programming

Sound categorical semantics of the language

Categorical models of constructive temporal logic

Cartesian closed category \mathcal{C}

Cartesian comonad \square

$$\varepsilon_A : \square A \rightarrow A$$

$$\delta_A : \square A \rightarrow \square \square A$$

$$u : \top \rightarrow \square \top$$

$$m_{A,B} : \square A \times \square B \rightarrow \square(A \times B)$$

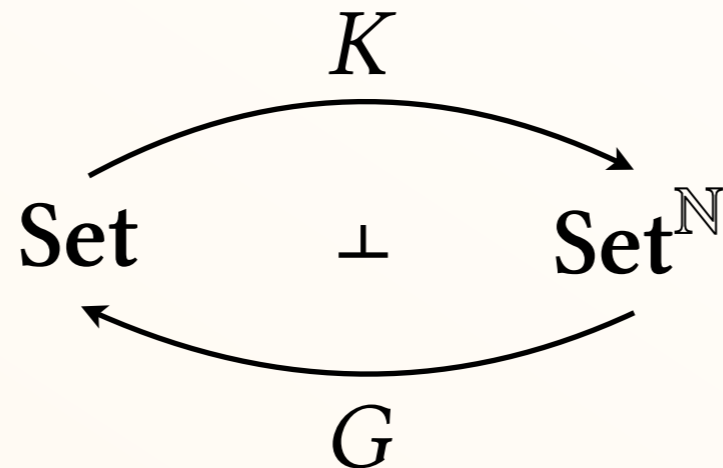
\square -strong monad \diamond

$$\eta_A : A \rightarrow \diamond A$$

$$\mu_A : \diamond \diamond A \rightarrow \diamond A$$

$$st_{A,B}^{\square} : \square A \times \diamond B \rightarrow \diamond(\square A \times B)$$

Category of reactive types



$$\square : \text{Set}^{\mathbb{N}} \rightarrow \text{Set}^{\mathbb{N}}$$

$$\diamond : \text{Set}^{\mathbb{N}} \rightarrow \text{Set}^{\mathbb{N}}$$

$$(\square A)_n = (KGA)_n = \prod_{k \geq 0} A_k$$

$$(\diamond A)_n = \sum_{k \geq 0} (\bullet^k A)_n$$

A function from time to types

A pair of a time and delayed value

Box types are always
inhabited

Diamond types are
eventually inhabited

Denotation of types

$A ::= \text{Unit} \mid A \times B \mid A + B \mid A \rightarrow B \mid \text{Stable } A \mid \text{Event } A$

$$\llbracket \text{Unit} \rrbracket = \top$$

$$\llbracket A \times B \rrbracket = \llbracket A \rrbracket \otimes \llbracket B \rrbracket$$

$$\llbracket A + B \rrbracket = \llbracket A \rrbracket \oplus \llbracket B \rrbracket$$

$$\llbracket A \rightarrow B \rrbracket = \llbracket A \rrbracket \Rightarrow \llbracket B \rrbracket$$

$$\llbracket \text{Stable } A \rrbracket = \square \llbracket A \rrbracket$$

$$\llbracket \text{Event } A \rrbracket = \diamond \llbracket A \rrbracket$$

$\text{handleEvt} : \text{Event } A \rightarrow \text{Stable } (A \rightarrow \text{Event } B) \rightarrow \text{Event } B$ now

$\text{handleEvt} = \lambda x. \lambda y. \text{let stable } f_s = y \text{ in}$

$\text{event } (\mathbf{let\ evt } e = x \mathbf{ in } (\mathbf{let\ evt } e' = \text{extract } f_s e \mathbf{ in pure } e'))$

Future work

Complete categorical semantics

Add temporal recursive types

$$\text{Stream } A = \nu x. A \times \text{Event } x$$

Establish equivalence of \diamond and CPS

$$\begin{aligned}\diamond A &\approx \neg \square \neg A \\ &\approx \neg \square (A \rightarrow \perp) \\ &\approx \square (A \rightarrow \perp) \rightarrow \perp\end{aligned}$$

Implement the language

Summary and conclusions

A high-level reactive language
with events as a primitive type

A concrete categorical model of constructive
temporal logic with an existential \diamond type

A categorical semantics which allows for
an efficient, CPS-like implementation

Combines the abstract semantics of FRP,
temporal properties of LTL and efficiency of CPS

Semantics of temporal type systems

github.com/DimaSamoz/temporal-type-systems

ds709@cam.ac.uk