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 ≈ Time → a
Event a ≈ Time × a
```

```haskell
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)*	**Push-based** *(Data-driven)*
Streams	Callbacks
Polling until an event happens	Asynchronous event handling
Latency issues	Instantaneous reactivity
High-level but inefficient	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	FRP
Propositions	Reactive types
$\Box$ modality	Behaviours
$\Diamond$ modality	Events
U modality	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 ◇A = Now A
        | Later ●(◇A)
```

```
case (e :: ◇A) of
  Now   a → …
| Later l → …
```

polling!

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

```
◇A = Σk≥0.●ᵏ A
```

```
case (e :: ◇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 $\Box$

$$\varepsilon_A \colon \Box A \longrightarrow A \qquad \delta_A \colon \Box A \longrightarrow \Box\Box A$$

$$\mathrm{u} \colon \top \longrightarrow \Box\top \qquad \mathrm{m}_{A,B} \colon \Box A \times \Box B \longrightarrow \Box(A \times B)$$

$\Box$-strong monad $\Diamond$

$$\eta_A \colon A \longrightarrow \Diamond A \qquad \mu_A \colon \Diamond\Diamond A \longrightarrow \Diamond A$$

$$\mathrm{st}^{\Box}_{A,B} \colon \Box A \times \Diamond B \longrightarrow \Diamond(\Box A \times B)$$

Category of reactive types

$$\mathbf{Set} \underset{G}{\overset{K}{\rightleftarrows}} \mathbf{Set}^{\mathbb{N}} \quad (K \dashv G)$$

$$\square : \mathbf{Set}^{\mathbb{N}} \longrightarrow \mathbf{Set}^{\mathbb{N}}$$

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

A function from time to types

Box types are always inhabited

$$\diamond : \mathbf{Set}^{\mathbb{N}} \longrightarrow \mathbf{Set}^{\mathbb{N}}$$

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

A pair of a time and delayed value

Diamond types are eventually inhabited

Denotation of types

$$A ::= \mathsf{Unit} \mid A \times B \mid A + B \mid A \to B \mid \mathsf{Stable}\ A \mid \mathsf{Event}\ A$$

$$\begin{aligned}
[\![\mathsf{Unit}]\!] &= \top \\
[\![A \times B]\!] &= [\![A]\!] \otimes [\![B]\!] \\
[\![A + B]\!] &= [\![A]\!] \oplus [\![B]\!] \\
[\![A \to B]\!] &= [\![A]\!] \Rightarrow [\![B]\!] \\
[\![\mathsf{Stable}\ A]\!] &= \Box[\![A]\!] \\
[\![\mathsf{Event}\ A]\!] &= \Diamond[\![A]\!]
\end{aligned}$$

$$\mathsf{handleEvt} \ :\ \mathsf{Event}\ A \to \mathsf{Stable}\ (A \to \mathsf{Event}\ B) \to \mathsf{Event}\ B\ \mathsf{now}$$

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

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

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

$$\begin{aligned} \Diamond A &\approx \neg\Box\neg A \\ &\approx \neg\Box(A \longrightarrow \bot) \\ &\approx \Box(A \longrightarrow \bot) \longrightarrow \bot \end{aligned}$$

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