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
- Event dispatching thread
- Asynchronous programming
- Event handlers

**Pros**
- Efficient
- Widely used

**Cons**
- Low-level
- Complicated and error-prone
Functional reactive programming

\[
\text{Signal } a \approx \text{Time } \rightarrow a \\
\text{Event } a \approx \text{Time } \times a
\]

\[
\text{redblue} :: \text{Signal Image} \\
\text{redblue } u = \text{withColor } c \\
\quad (\text{stretch}\ (\text{wiggleRange } 0.5 1)\ \text{circle}) \\
\quad \text{where } c = \text{red } \text{`until` } \text{lbp } u \rightarrow \text{blue}
\]

**Pros**
- Declarative
- Compositional

**Cons**
- Performance issues
- Violates causality
## Pull vs. push-based FRP

<table>
<thead>
<tr>
<th>Pull-based</th>
<th>Push-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Demand-driven)</td>
<td>(Data-driven)</td>
</tr>
<tr>
<td>Streams</td>
<td>Callbacks</td>
</tr>
<tr>
<td>Polling until an event happens</td>
<td>Asynchronous event handling</td>
</tr>
<tr>
<td>Latency issues</td>
<td>Instantaneous reactivity</td>
</tr>
<tr>
<td>High-level but inefficient</td>
<td>Low-level but efficient</td>
</tr>
</tbody>
</table>
Can we combine intuitive semantics with performance and correctness?

- Efficient FRP implementations
- Theoretical foundations of FRP
Curry–Howard for FRP

*Jeffrey (2012), Jeltsch (2012)*

<table>
<thead>
<tr>
<th>LTL</th>
<th>FRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositions</td>
<td>Reactive types</td>
</tr>
<tr>
<td>□ modality</td>
<td>Behaviours</td>
</tr>
<tr>
<td>◇ modality</td>
<td>Events</td>
</tr>
<tr>
<td>U modality</td>
<td>Processes</td>
</tr>
</tbody>
</table>
Advantages of LTL

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

Restrict event handlers to only use values that are always available

```jsx
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

\[(\Diamond A)_n \text{ holds iff } A_i \text{ holds for some } i \geq n\]

\[\text{iff } A_n \text{ holds or } (\bullet A)_n \text{ holds or } (\bullet^2 A)_n \text{ holds...}\]

\[(\Diamond A)_n \iff \mu X. A_n \lor (\bullet X)_n\]

\[(\Diamond A)_n \iff \exists k \geq 0. (\bullet^k A)_n\]

\[\Diamond A = \Sigma k \geq 0. \bullet^k A\]

\[\text{case } (e :: \Diamond A) \text{ of}\]
\[\text{  Now } a \rightarrow \ldots\]
\[\text{  Later } l \rightarrow \ldots \text{ polling!}\]
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 $\Box$

$\varepsilon_A : \Box A \to A$

$\delta_A : \Box A \to \Box \Box A$

$u : \top \to \Box \top$

$m_{A,B} : \Box A \times \Box B \to \Box (A \times B)$

$\Box$-strong monad $\Diamond$

$\eta_A : A \to \Diamond A$

$\mu_A : \Diamond \Diamond A \to \Diamond A$

$\text{st}^\Box_{A,B} : \Box A \times \Diamond B \to \Diamond (\Box A \times B)$
**Category of reactive types**

\[
\begin{array}{ccc}
K & \downarrow & \text{Set} \\
\text{Set} & \rightsquigarrow & \text{Set}^\mathbb{N} \\
G & \uparrow & 
\end{array}
\]

\[\Box : \text{Set}^\mathbb{N} \rightarrow \text{Set}^\mathbb{N}\]

\[(\Box A)_n = (KGA)_n = \prod_{k \geq 0} A_k\]

A function from time to types

\[\Diamond : \text{Set}^\mathbb{N} \rightarrow \text{Set}^\mathbb{N}\]

\[(\Diamond A)_n = \sum_{k \geq 0} (\bullet^k A)_n\]

A pair of a time and delayed value

Box types are always inhabited

Diamond types are eventually inhabited
Denotation of types

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

\[
\begin{align*}
[\text{Unit}] &= \top \\
[A \times B] &= [A] \otimes [B] \\
[A \rightarrow B] &= [A] \Rightarrow [B] \\
[\text{Stable } A] &= \Box [A] \\
[\text{Event } A] &= \Diamond [A]
\end{align*}
\]

handleEvt : Event A → Stable (A → Event B) → Event B now

handleEvt = λx. λy. let stable \( f_s = y \) in

\[ \text{event (let evt } e = x \text{ in (let evt } e' = \text{extract } f_s e \text{ in pure } e')) \]
Future work

Complete categorical semantics

Add temporal recursive types

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

Establish equivalence of $\Diamond$ and CPS

$\Diamond A \cong \neg \square \neg A$

$\cong \neg \square (A \rightarrow \bot)$

$\cong \square (A \rightarrow \bot) \rightarrow \bot$

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