A linear temporal linear type system for GUIs

Christian Uldal Graulund [1]   Neelakantan Krishnaswami [2]   **Dmitrij Szamozvancev** [2]

[1]IT University of Copenhagen   [2]University of Cambridge

Denotational semantics

Formal reasoning via a sound categorical interpretation in a model of linear and temporal logic

$$\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{Rel}_{\widehat{\mathbb{N}_\top}} \arrow[l, bend left, "G"] \end{tikzcd}$$

Adjoint model of mixed linear-nonlinear logic (Benton, 1995)

Functional sublanguage

- Regular functional programming
- Cartesian closed category of sets

$$X \xrightarrow{f} Y \in \mathbf{Set}$$

Linear sublanguage

- Layout of GUI and reactivity
- Monoidal closed category of relations between time-indexed sets with a top element

$$A \xrightarrow{r} B \in \mathbf{Rel}_{\widehat{\mathbb{N}_\top}}$$
$$\parallel$$
$$A \otimes B \xrightarrow{r} \Omega \in \widehat{\mathbb{N}_\top}$$

Features

- **Presheaf model** Time-indexed sets A_k with a "global" object $A_\top$
  - Natural model of time-dependence of temporal logic
  - Distinguish between dynamic behaviour and global properties
- **Internal relations** Category of relations on the presheaf topos $\widehat{\mathbb{N}_\top}$

$$\begin{tikzcd} A_\top \times B_\top \arrow[rrr, "r_\top"] \arrow[dr, bend right] \arrow[ddr, bend right] \arrow[dddr, bend right] & & & \mathcal{P}(\mathbb{N})+1 \arrow[dl, bend left] \arrow[ddl, bend left] \arrow[dddl, bend left] \\ & A_0 \times B_0 \arrow[r, "r_0"] & \{0,1\} & \\ & A_1 \times B_1 \arrow[r, "r_1"] & \{0,1\} & \\ & A_k \times B_k \arrow[r, "r_k"] & \{0,1\} & \end{tikzcd}$$

  - Relational model of linear logic for unambiguous treatment of widgets
  - Express *nondeterminism* of GUI interaction: listeners relate a widget to every possible event that can occur on it (e.g. *onKeyPress* relates a widget to all possible events of any key press at any time step)
- **Event monad** Interpret eventuality as a monad (Kobayashi, 1997)
  - An *existential* type rather than a coinductive one – implementation via continuation-passing style, not polling (busy-waiting)
  - Compare the occurrence of two events (temporal linearity) and select the earlier one – related to *derivatives* (Paykin et al., 2016)

$$select : \diamond A \otimes \diamond B \multimap \diamond((A \otimes \diamond B) \oplus (\diamond A \otimes B))$$

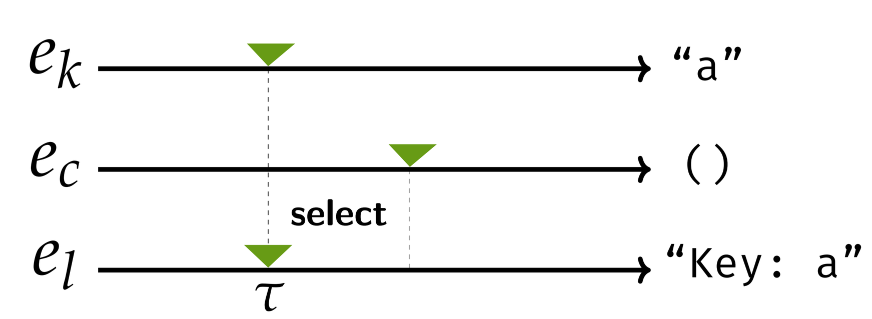
- **Hyperdoctrines** Universal and existential quantification over time steps
  - Precise control over the temporal behaviour of widgets via *split* and *join*

Programming model

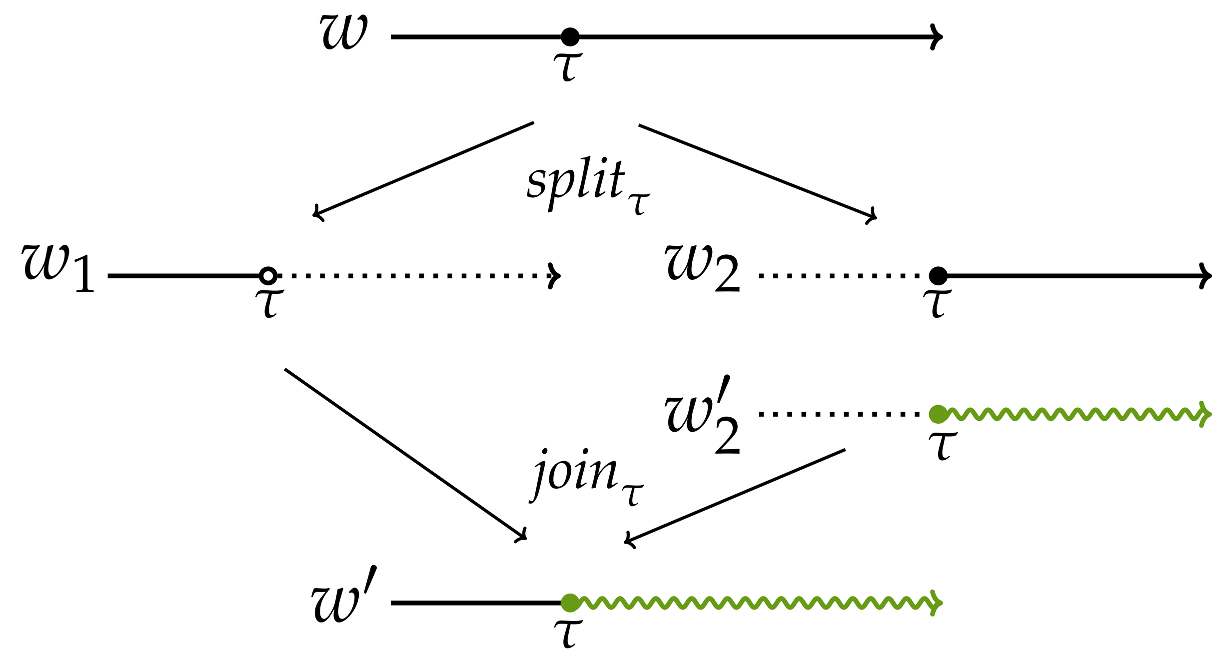
Functional reactive programming with monadic events, linearity, and time step quantification

$$\begin{array}{l}
\textbf{val}\ setLabel : \mathsf{Widget} \multimap \mathsf{Widget} \\
\textbf{fun}\ setLabel(w) = \\
\quad \textbf{let}\ (w, e_k : \diamond(\mathsf{F\ Char})) = onKeyPress\ w \\
\qquad (w, e_c : \diamond\mathsf{Unit}) = onClick\ w \\
\qquad e_l : \diamond(\mathsf{F\ String}) = (\textbf{select}\ [e_k\ \textbf{as}\ (\mathsf{F}\ k) \to \mathsf{F}(\text{“Key:”} +\!\!+\ toString\ k)] \\
\qquad\qquad\qquad\qquad\qquad \parallel [e_c\ \textbf{as}\ _ \quad \to \mathsf{F}\ \text{“Click”}]) \\
\qquad \{\tau, (\mathsf{F}\ lab)@\tau\} = time\ e_l \\
\qquad (w_1, w_2@\tau) = split_\tau\ w \\
\qquad w_2' = setLabel\ (w_2, lab) \\
\quad \textbf{in}\ join_\tau\ (w_1, w_2'@\tau)
\end{array}$$

Events



Widgets



- Linear functions to construct GUI elements and register event handlers

$$onKeyPress : \mathsf{Widget} \multimap \mathsf{Widget} \otimes \diamond(\mathsf{F\ Char})$$

- Select between two events based on their occurrence time

$$\frac{\Theta \mid \Gamma; \Delta_1 \vdash_\ell e_a : \diamond A \qquad \Theta \mid \Gamma; a : A, b : \diamond B \vdash_\ell g : \diamond C \qquad \Theta \mid \Gamma; \Delta_2 \vdash_\ell e_b : \diamond B \qquad \Theta \mid \Gamma; a : \diamond A, b : B \vdash_\ell h : \diamond C}{\Theta \mid \Gamma; \Delta_1, \Delta_2 \vdash_\ell \textbf{select}\ [e_a\ \textbf{as}\ a \to g] \parallel [e_b\ \textbf{as}\ b \to h] : \diamond C}$$

- Convert between monadic and existential events

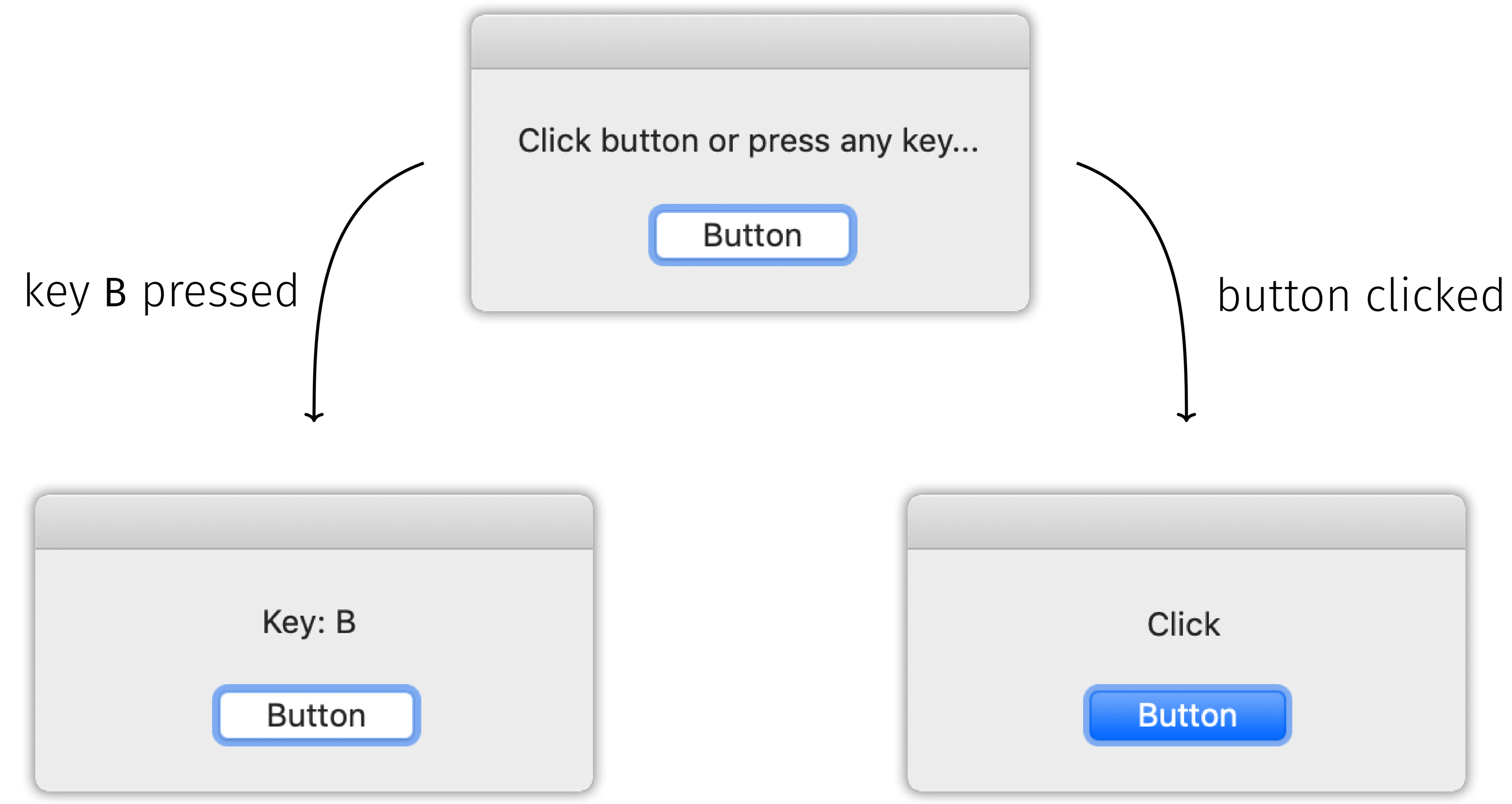
$$time, event^{-1} : \diamond A \multimap (\exists \tau.\ A@\tau)$$

- Separate and rejoin the history of a widget at a timestep

$$split, join^{-1} : \forall \tau.\ \mathsf{Widget} \multimap \mathsf{Widget}_\tau \otimes \mathsf{Widget}@\tau$$

Implementation (in progress)

Standard implementation strategy via callbacks and continuation-passing style



Efficiency and correctness

- Semantics designed with an efficient and statically correct implementation in mind (Krishnaswami, 2013)
  - Novel semantics, but standard implementation strategy
  - Eliminate space- and time leaks, and causality violations
- **Linearity** Treatment of graphical elements as discrete resources
  - Handle stateful nature of interfaces in a purely functional setting
  - Two widgets are never "collapsed", even if their behaviour is identical
- **Events** A "type delayed by some time"
  - Monadic program structure without explicit wait instructions
  - Efficient implementation using callbacks (*push-based FRP*)

$$\diamond A \cong \neg\Box\neg A \cong \Box(A \to \bot) \to \bot$$

Syntactic extensions

- Type inference for time indices and the F and G constructors
- Syntactic sugar for registering and handling events on widgets

```
fun setLabel(w) =
    let [ek, ec] = listen w [onClick, onKeyPress]
        el = select ek as k -> "Key: " ++ toString k
               | ec as _ -> "Click"
    in when (w, el) ~> (w', lab) do
         return setLabel (w', lab)
```

References

Benton, P. N. (1995). "A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models". In: *Computer Science Logic*. Vol. 933. Springer Berlin Heidelberg, pp. 121–135. DOI: 10.1007/BFb0022251.

Kobayashi, Satoshi (1997). "Monad as Modality". In: *Theoretical Computer Science* 175.1, pp. 29–74.

Krishnaswami, Neelakantan R. (2013). "Higher-Order Functional Reactive Programming without Spacetime Leaks". In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, p. 221.

Paykin, Jennifer, Neelakantan R. Krishnaswami, and Steve Zdancewic (2016). *The Essence of Event-Driven Programming*. Available at https://www.cl.cam.ac.uk/~nk480/essence-of-events.pdf.