Stark typisierte und effiziente Funktionale Reaktive Programmierung

Wolfgang Jeltsch

Dissertationsverteidigung

Fakultät für Mathematik, Naturwissenschaften und Informatik
Brandenburgische Technische Universität Cottbus

8. Dezember 2011

Überblick

Überblick

Funtionale Reaktive Programmierung

- das ideale reaktive System:
  - kontinuierliche Zustandsänderungen
  - sofortige, atomare Reaktionen auf Ereignisse
- spiegelt sich nicht in gängigen Implementierungen wider:
  - Diskretisierung sichtbar
  - inkonsistente Zwischenzustände sichtbar
- Programmierer mit technischen Details konfrontiert:
  - Polling-Schleifen
  - Event-Handler
- Ziel der funktionalen Programmierung:

  Problembeschreibung statt Ausführungsplan
- Funktionale Reaktive Programmierung (FRP):

  Anwendung dieses Prinzips auf reaktive Systeme
- in diesem Vortrag Umsetzung von FRP
  als Haskell-Bibliothek

Signale

- das Herzstück der FRP
- beschreiben zeitliches Verhalten
- drei Arten:

  diskret Werte an diskreten Zeitpunkten:

  $$[\![\mathit{DSignal}]\!]\ \alpha \approx [(\mathit{Time}, \alpha)]$$

  kontinuierlich beliebige zeitveränderliche Werte:

  $$[\![\mathit{CSignal}]\!]\ \alpha \approx \mathit{Time} \to \alpha$$

  segmentiert Treppenfunktionen über der Zeit:

  $$[\![\mathit{SSignal}]\!]\ \alpha = (\alpha, [\![\mathit{DSignal}]\!]\ \alpha)$$

- Beispielsignale:

  diskret eingehende Netzwerkpakete:

  DSignal Packet

  segmentiert bisheriges Netzwerk-Datenvolumen:

  SSignal Integer

Signalkombinatoren

- Funktionen, welche Signale konstruieren
- einige Beispiele:

$$\begin{aligned} \mathit{union} &:: \mathit{DSignal}\ \alpha \to \mathit{DSignal}\ \alpha \to \mathit{DSignal}\ \alpha \\ \mathit{filter} &:: (\alpha \to \mathit{Bool}) \to \mathit{DSignal}\ \alpha \to \mathit{DSignal}\ \alpha \\ \mathit{scanl} &:: (\beta \to \alpha \to \beta) \to \beta \to \mathit{DSignal}\ \alpha \\ &\quad \to \mathit{SSignal}\ \beta \end{aligned}$$

- Anwendung dieser Kombinatoren:

$$\begin{aligned} &\ddot{p} :: \mathit{DSignal\ Packet} \\ &\ddot{p} = \mathit{union}\ \ddot{p}_{\mathit{In}}\ \ddot{p}_{\mathit{Out}} \\ &\ddot{p}_{\mathit{TCP}} :: \mathit{DSignal\ Packet} \\ &\ddot{p}_{\mathit{TCP}} = \mathit{filter}\ \mathit{isTCPPacket}\ \ddot{p} \\ &\bar{v} :: \mathit{SSignal\ Integer} \\ &\bar{v} = \mathit{scanl}\ (\lambda v\ p \to v + \mathit{size}\ p)\ 0\ \ddot{p} \end{aligned}$$

Switching

- Typklasse *Signal* aller Signaltypen
- Switching-Kombinator:

  $$switch :: (Signal\ \sigma) \Rightarrow SSignal\ (\sigma\ \alpha) \rightarrow \sigma\ \alpha$$

- mögliche Anwendung:
  - zwei segmentierte Signale, die den Umfang des eingehenden bzw. ausgehenden Netzwerkverkehrs darstellen:

    $$\bar{v}_{In}, \bar{v}_{Out} :: SSignal\ Integer$$

  - segmentiertes Signal, dessen Wert gemäß Benutzerwahl zwischen $\bar{v}_{In}$ und $\bar{v}_{Out}$ wechselt:

    $$\bar{\bar{v}} :: SSignal\ (SSignal\ Integer)$$

  - Switching erzeugt Signal, welches immer das entsprechende Datenvolumen liefert:

    $$\bar{v}_{Sel} :: SSignal\ Integer$$
    $$\bar{v}_{Sel} = switch\ \bar{\bar{v}}$$

  - $\bar{v}_{Sel}$ als Eingabe für eine Anzeigekomponente nutzbar

Überblick

Naive Implementierung

- Signalkonsumenten registrieren Event-Handler
- diskretes Signal kann durch Registrierungsaktion repräsentiert werden:

  type *Handler* $\alpha = \alpha \to IO\ ()$

  type *DSignal* $\alpha =$ *Handler* $\alpha \to IO\ ()$
- Implementierung von *SSignal* direkt aus Semantik ableitbar:

  type *SSignal* $\alpha = (\alpha, DSignal\ \alpha)$

Generatoren statt Signale

- Ausführung der Registrierungsaktionen einmal pro Signalkonsument
- bei Nutzung von *scanl*:
  - Erzeugen einer veränderlichen Variable, die zum Konsumierungszeitpunkt initialisiert wird
  - Registrierung eines Event-Handlers, der diese Variable aktualisiert
- zwei Probleme:
  1. Mehrfachberechnungen
  2. Signalwerte abhängig vom Konsumierungszeitpunkt
- Intuition:
  - Werte von Signaltypen sind tatsächlich Signalgeneratoren
  - Generator erzeugt beim Konsumieren ein neues Signal
  - keine direkte Unterstützung von Signalen mehr

Überblick

Nutzung von Haskells Caching-Unterstützung

- Haskell speichert berechnete Teile einer Datenstruktur, wenn eine Variable an diese Struktur gebunden ist
- Problem:

  Werte von *DSignal* enthalten die Ereigniswerte nicht
- Änderung der Datenstruktur:

  type $\mathit{DSignal}\ \alpha = [(\mathit{Time}, \alpha)]$
- Ereignisströme müssen beim Berechnen von Signalvereinigungen verzahnt werden:

$$\mathit{union}\ ((t_1, x_1) : \ddot{x}_1)\ ((t_2, x_2) : \ddot{x}_2) \mid t_1 < t_2 = \cdots \mid t_1 \equiv t_2 = \cdots \mid t_1 > t_2 = \cdots$$

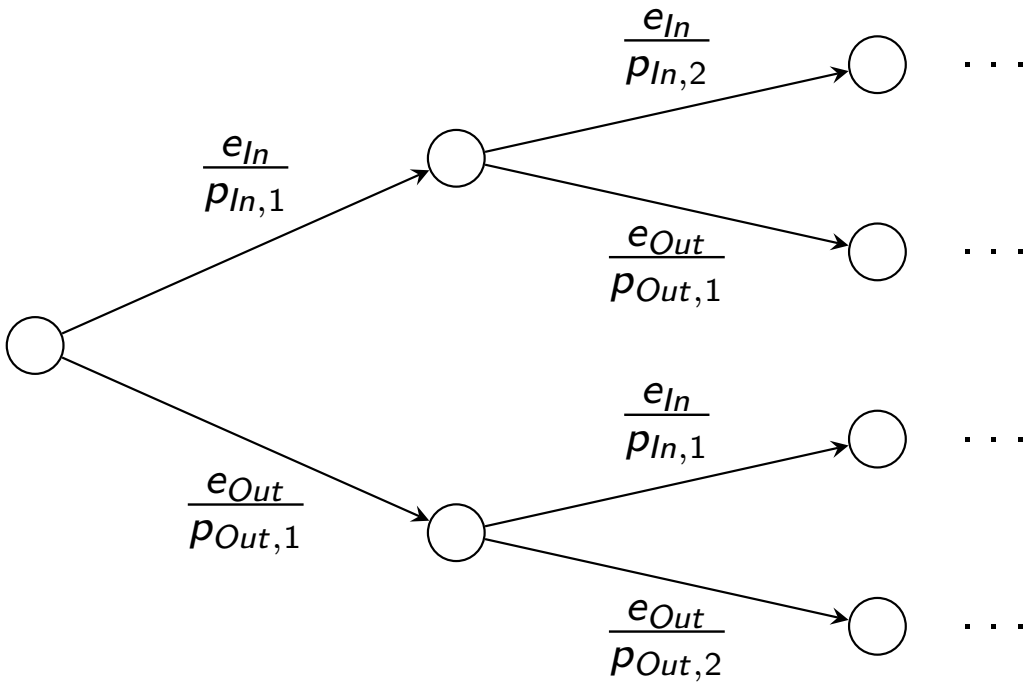
- Problem:

  Vergleich der initialen Ereigniszeitpunkte muss erfolgen, wenn das erste Ereignis auftritt
- unsere Lösung:

  Verzahnung von Konsumenten ermitteln lassen

Darstellung diskreter Signale durch Vistas

- Vista beinhaltet jede mögliche Verzahnung der entsprechenden Ereignisströme
- Vista für *union* $\ddot{p}_{In}$ $\ddot{p}_{Out}$:

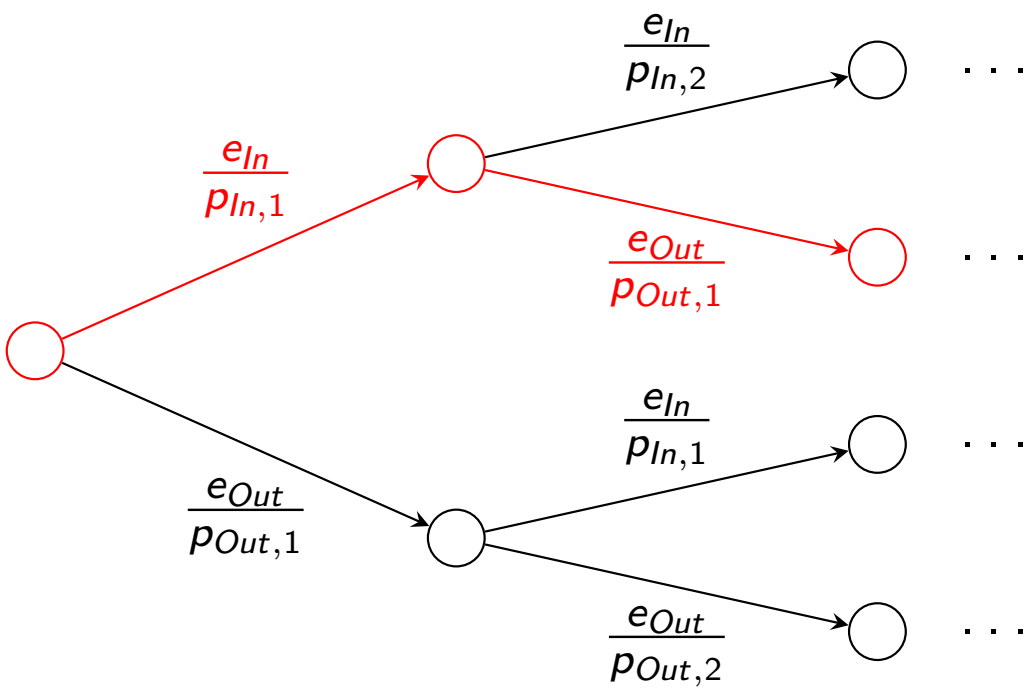


- Implementierung:

$$\textbf{type}\ \mathit{Vista}\ \alpha = \mathit{Map}\ \mathit{EventSrc}\ (\alpha, \mathit{Vista}\ \alpha)$$

Konsumieren von Vistas

- Konsument kennt die Reihenfolge, in der die Ereignisquellen feuern
- wertet nur den relevanten Pfad aus:



Überblick

Signalsuffixe

- Unterstützung von Signalsuffixen
- Umschalten und Konsumieren soll nur zur Startzeit des jeweiligen Suffixes möglich sein
- Typsystem wird genutzt, um dies zur Compilierzeit sicher zu stellen
- Suffixtypen erhalten zusätzlichen Typparameter, der die Startzeiten der Suffixe repräsentiert
- Suffixkombinatoren erzwingen Startzeitgleichheit:

$$\begin{aligned} \mathit{union} &:: \mathit{DSuffix}\ \tau\ \alpha \to \mathit{DSuffix}\ \tau\ \alpha \\ &\to \mathit{DSuffix}\ \tau\ \alpha \end{aligned}$$

$$\begin{aligned} \mathit{scanl} &:: (\beta \to \alpha \to \beta) \to \beta \to \mathit{DSuffix}\ \tau\ \alpha \\ &\to \mathit{SSuffix}\ \tau\ \beta \end{aligned}$$

Sicheres Switching

- Switching-Kombinator arbeitet mit Funktionen über Suffixen mit identischer Startzeit:

$$\mathit{SuffixFun}\ \tau\ (\sigma_1\ \text{`}\mathit{Of}\text{`}\ \alpha_1 \mapsto \cdots \mapsto \sigma_n\ \text{`}\mathit{Of}\text{`}\ \alpha_n \mapsto \sigma\ \text{`}\mathit{Of}\text{`}\ \alpha)$$
$$\cong$$
$$\sigma_1\ \tau\ \alpha_1 \to \cdots \to \sigma_n\ \tau\ \alpha_n \to \sigma\ \tau\ \alpha$$

- Typ des Switching-Kombinators:

$$\begin{aligned}\mathit{switch}\ ::\ & \mathit{SSuffix}\ \tau\ (\forall\tau'.\mathit{SuffixFun}\ \tau'\ \varphi) \\ & \to \mathit{SuffixFun}\ \tau\ \varphi\end{aligned}$$

- Funktionsweise des Kombinators (konzeptionell):
  - Argumente der resultierenden Funktion werden sukzessive gekürzt
  - bei Änderung des Argumentsignals wird die neue Signalfunktion auf die gekürzten Argumente angewendet
  - aus resultierenden Suffixen wird finaler Suffix erzeugt

Überblick

Weitere Beiträge der Dissertation

- detailierter Vergleich verschiedener herkömmlicher FRP-Implementierungen
- Implementierung eines Record-Systems mit Unterstützung für statisch typisierte, generische Kombinatoren:
  - Eigenschaften und Aktivitäten eines Objekts durch Eingabesignale gesteuert
  - Ausgabesignale liefern Informationen zu einem Objekt
  - Folge:

    Objekte haben i.d.R. viele Eingabe- und Ausgabesignale
  - Bündelung dieser Signale in Records sinnvoll
  - generische Record-Kombinatoren wichtig, um Boilerplate-Code zu vermeiden

Überblick

Zusammenfassung und Ausblick

- ereignisgetriebene Implementierung mit Signalen statt Generatoren:
  - Caching von Signalwerten durch Vista-Datenstruktur
  - Sichern von Startzeitkonsistenz durch Verwalten der Startzeiten auf der Typebene
- Vergleich herkömmlicher FRP-Implementierungen
- statisch typisierte, generische Record-Kombinatoren
- einige offene Probleme:
  - Implementierung von kontinuierlichen Signalen
  - inkrementelle segmentierte Signale
  - Beziehung zwischen FRP und temporaler Logik

Stark typisierte und effiziente Funktionale Reaktive Programmierung

Wolfgang Jeltsch

Dissertationsverteidigung

Fakultät für Mathematik, Naturwissenschaften und Informatik
Brandenburgische Technische Universität Cottbus

8. Dezember 2011

scanl mit Registrierungsaktionen

$$
\begin{aligned}
&\mathit{scanl} :: (\beta \to \alpha \to \beta) \to \beta \to \mathit{DSignal}\ \alpha \to \mathit{SSignal}\ \beta \\
&\mathit{scanl}\ f\ y_0\ \ddot{x} = (y_0, \ddot{y})\ \textbf{where} \\
&\quad \ddot{y} = \lambda h \to \textbf{do} \\
&\qquad\qquad \vec{y} \leftarrow \mathit{newIORef}\ y_0 \\
&\qquad\qquad \ddot{x}\ (\lambda x \to \textbf{do} \\
&\qquad\qquad\qquad\qquad y \leftarrow \mathit{readIORef}\ \vec{y} \\
&\qquad\qquad\qquad\qquad \textbf{let} \\
&\qquad\qquad\qquad\qquad\quad y' = f\ y\ x \\
&\qquad\qquad\qquad\qquad \mathit{writeIORef}\ \vec{y}\ y' \\
&\qquad\qquad\qquad\qquad h\ y')
\end{aligned}
$$

Kombinatoren vista-basiert

- funktionale Repräsentation diskreter Signale führt zu funktionalen Implementierungen von Kombinatoren
- Implementierung von *scanl*:

$$\begin{aligned} &\mathit{scanl} :: (\beta \to \alpha \to \beta) \to \beta \to \mathit{DSignal}\ \alpha \\ &\qquad \to \mathit{SSignal}\ \beta \\ &\mathit{scanl}\ f\ y_0\ \ddot{x} = (y_0, a\ y_0\ \ddot{x})\ \textbf{where} \\ &\quad a\ y = \mathit{fmap}\ (\lambda(x, \ddot{x}) \to \textbf{let} \\ &\qquad\qquad\qquad\qquad y' = f\ y\ x \\ &\qquad\qquad\qquad\qquad \textbf{in}\ (y', a\ y'\ \ddot{x})) \end{aligned}$$

- Problem mit *filter*:

  Entfernen von Knoten zerstört Vista-Struktur

- Lösung:

  Ereigniswerte optional machen

- modifizierter *Vista*-Typ:

  $\textbf{type}\ \mathit{Vista}\ \alpha = \mathit{Map}\ \mathit{EventSrc}\ (\mathit{Maybe}\ \alpha, \mathit{Vista}\ \alpha)$

Definition von *SuffixFun*

- leere Datentypen für Typindizes:

  data $\varphi \mapsto \varphi'$

  data σ ‘*Of*‘ α

- *SuffixFun* definiert als GADT:

  data *SuffixFun* τ φ **where**

  $$\begin{aligned} \mathit{SuffixFun}_0 \quad &:: (\mathit{Suffix}\ \sigma) \\ &\Rightarrow \sigma\ \tau\ \alpha \\ &\rightarrow \mathit{SuffixFun}\ \tau\ (\sigma\ \text{‘}\mathit{Of}\text{‘}\ \alpha) \\ \mathit{SuffixFun}_{\mathit{succ}} \quad &:: (\mathit{Suffix}\ \sigma) \\ &\Rightarrow (\sigma\ \tau\ \alpha \rightarrow \mathit{SuffixFun}\ \tau\ \varphi') \\ &\rightarrow \mathit{SuffixFun}\ \tau\ (\sigma\ \text{‘}\mathit{Of}\text{‘}\ \alpha \mapsto \varphi') \end{aligned}$$

Netzwerkbeispiel mit Signalfunktionen

- Typ zweistelliger Suffixfunktionen über einem Suffixtyp:

$$\textbf{type } \mathit{BinSuffixFun}\ \tau\ \sigma\ \alpha = \mathit{SuffixFun}\ \tau\ (\sigma\ \text{`}\mathit{Of}\text{`}\ \alpha \mapsto \sigma\ \text{`}\mathit{Of}\text{`}\ \alpha \mapsto \sigma\ \text{`}\mathit{Of}\text{`}\ \alpha)$$

- Projektionsfunktionen:

$$\begin{aligned}
&\pi_1, \pi_2 :: \mathit{BinSuffixFun}\ \tau\ \sigma\ \alpha \\
&\pi_1 = \mathit{SuffixFun}_{succ}\ \$\ \lambda s_1 \to \\
&\qquad \mathit{SuffixFun}_{succ}\ \$\ \lambda_ \to \mathit{SuffixFun}_0\ s_1 \\
&\pi_2 = \mathit{SuffixFun}_{succ}\ \$\ \lambda_ \to \\
&\qquad \mathit{SuffixFun}_{succ}\ \$\ \lambda s_2 \to \mathit{SuffixFun}_0\ s_2
\end{aligned}$$

- segmentiertes Signal, welches zwischen diesen Funktionen wechselt:

$$\bar{f} :: \mathit{SSuffix}\ \tau\ (\forall \tau'.\mathit{BinSuffixFun}\ \tau'\ \sigma\ \alpha)$$

- Switching liefert zeitveränderliche Projektion:

$$\begin{aligned}
&f :: \mathit{BinSuffixFun}\ \tau\ \sigma\ \alpha \\
&f = \mathit{switch}\ \bar{f}
\end{aligned}$$

- Entpacken und Anwenden auf $\bar{v}_{In}$ und $\bar{v}_{Out}$ liefert $\bar{v}_{Sel}$

Konsumieren aller Signale eines Signal-Records

- Records als Listen von Name-Wert-Paaren:

$$\begin{aligned} X \;:\&\; \mathit{Traffic} &::: \mathit{SSignal}\ \ \mathit{Integer} \\ :\&\; \mathit{Visible} &::: \mathit{SSignal}\ \ \mathit{Bool} \\ :\&\; \mathit{Close} &::: \mathit{DSignal}\ () \end{aligned}$$

- Eingabe einer Komponente:

$$X \;:\&\; \nu_1 ::: \sigma_1 \;:\&\; \ldots \;:\&\; \nu_n ::: \sigma_n$$

- Record von Signalkonsumierungsaktionen:

$$X \;:\&\; \nu_1 ::: \sigma_1 \to \mathit{IO}\ () \;:\&\; \ldots \;:\&\; \nu_n ::: \sigma_n \to \mathit{IO}\ ()$$

- gewünscht:

  Record-Konsumierungsfunktion, die mit beliebigen Signal-Records arbeiten kann

- Probleme:
  - Induktion über Record-Typen notwendig
  - Beziehungen zwischen Record-Typen müssen spezifizierbar sein

Record-Typfamilien

- allgemeine Lösung für alle derart gelagerten Probleme
- Record-Typ ist jetzt aus zwei Teilen aufgebaut:

  Schema eine Liste von Paaren aus je einem Namen und einer sogenannten Sorte:

  $$X :\& \nu_1 ::: \varsigma_1 :\& \ldots :\& \nu_n ::: \varsigma_n$$

  Stil eine Typfunktion
- Feldtypen entstehen durch Anwenden des Stils auf die Sorten
- Familien verwandter Record-Typen durch Kombination eines Schemas mit verschiedenen Stilen
- im Falle der Record-Konsumierung:

  Schema enthält die Signaltypen:

  $$X :\& \nu_1 ::: \sigma_1 :\& \ldots :\& \nu_n ::: \sigma_n$$

  Stile $\lambda\sigma \to \sigma$ und $\lambda\sigma \to (\sigma \to IO\ ())$
- induktive Definitionen durch Faltungsfunktion für Record-Schemata