Signals, Not Generators!

Wolfgang Jeltsch

Brandenburgische Technische Universität Cottbus
Cottbus, Germany

Tenth Symposium on Trends in Functional Programming
June 2–4, 2009

Signals

- the heart of Functional Reactive Programming
- describe behavior over time
- three flavors:

discrete	values associated with discrete times
continuous	arbitrary time-varying values (not in this talk)
segmented	time-varying values, changing at discrete times

- examples:

discrete: incoming network packets

DSignal Packet

segmented: amount of network traffic

SSignal Int

Signal combinators (1)

- some combinators:

$$
\begin{aligned}
\mathit{union} &:: \mathit{DSignal\ val} \to \mathit{DSignal\ val} \to \mathit{DSignal\ val} \\
\mathit{scan} &:: \mathit{accu} \to \\
&\quad (\mathit{accu} \to \mathit{val} \to \mathit{accu}) \to \\
&\quad (\mathit{DSignal\ val} \to \mathit{SSignal\ accu})
\end{aligned}
$$

- application:

$$
\begin{aligned}
&\mathit{packets} :: \mathit{DSignal\ Packet} \\
&\mathit{packets} = \mathit{union\ inPackets\ outPackets} \\
&\mathit{amounts} :: \mathit{DSignal\ Packet} \to \mathit{SSignal\ Int} \\
&\mathit{amounts} = \mathit{scan}\ 0\ \mathit{next}\ \textbf{where} \\
&\quad \mathit{next\ amount\ packet} = \mathit{amount} + \mathit{size\ packet}
\end{aligned}
$$

Signal combinators (2)

- switching combinator:

$$
\begin{aligned}
&\mathit{switch} :: (\mathit{Signal}\ \mathit{signal}) \Rightarrow \\
&\qquad \mathit{SSignal}\ (\mathit{signal}\ \mathit{val}) \rightarrow \mathit{signal}\ \mathit{val}
\end{aligned}
$$

instance *Signal DSignal* **where** ...

instance *Signal SSignal* **where** ...

- application:

  showing the amount of either incoming or outgoing traffic, depending on user selection

Push-based evaluation

- event-driven updates
- signal consumers register event handlers
- typical implementation of signals:

discrete signal is registration action:

$$DSignal\ val \cong (val \rightarrow IO\ ()) \rightarrow IO\ (IO\ ())$$

segmented signal is initial value plus update signal:

$$SSignal\ val \cong (val, DSignal\ val)$$

Generators, not signals

- registration actions executed once per consumer
- when using *scan*, every consumer
  - creates a mutable variable holding the accumulated value
  - registers a handler that updates this variable
- two problems:
  1. duplication of computations
  2. signal values depending on consumption time
- explaination:

  signal generators instead of signals

Using native memoization

- discrete signal contains list of value occurences:

$$\mathit{DSignal}\ \mathit{val} \ \cong ([(\mathit{Time}, \mathit{val})], \ldots)$$

- signal union involves union of occurence lists:

$$\begin{aligned} &\mathit{occsUnion}\ ((\mathit{time}_1, \mathit{val}_1) : \mathit{occs}_1)) \\ &\qquad ((\mathit{time}_2, \mathit{val}_2) : \mathit{occs}_2)) = \mathit{occs}' \ \mathbf{where} \\ &\quad \mathit{occs}' = \mathbf{case}\ \mathit{compare}\ \mathit{time}_1\ \mathit{time}_2\ \mathbf{of}\ \ldots \end{aligned}$$

- problem:

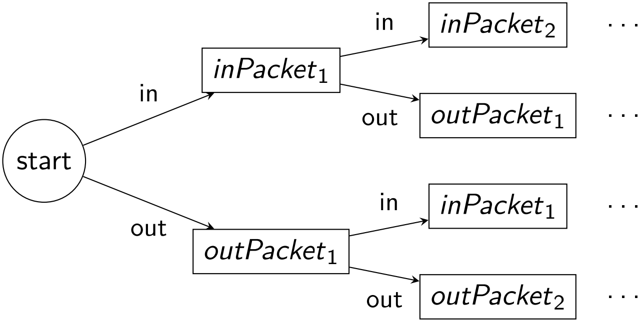
  comparison happens at the earlier one of both times

- our solution:

  postpone the decision

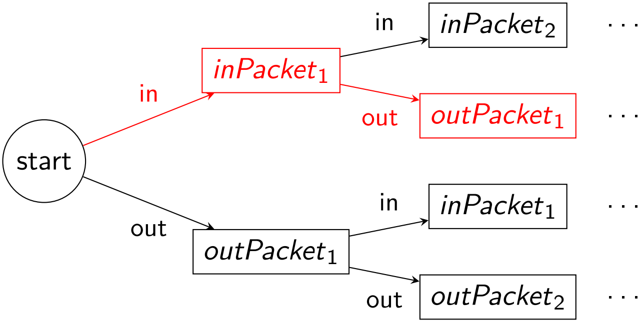
Vistas

- always gives several possible "futures" depending on what event source fires next
- vista for *union inPackets outPackets*:



Consuming vistas

- consumer evaluates only the relevant path:



Fixing start times (1)

- signal types get an extra (phantom) type parameter (similar to *STRef*) that represents their lifetime (era):

  $$DSignal\ era\ val$$

  $$SSignal\ era\ val$$

- signal combinators enforce era equality:

$$
\begin{aligned}
union &:: DSignal\ era\ val \\
&\quad DSignal\ era\ val \rightarrow \\
&\quad DSignal\ era\ val \rightarrow \\
scan &:: accu \rightarrow \\
&\quad (accu \rightarrow val \rightarrow accu) \rightarrow \\
&\quad (DSignal\ era\ val \rightarrow SSignal\ era\ accu)
\end{aligned}
$$

Fixing start times (2)

- reactive actions with era parameters (similar to *ST*):

$$\textbf{newtype}\ \mathit{Reactive}\ \mathit{era}\ \mathit{val} = \mathit{Reactive}\ (\mathit{IO}\ \mathit{val})$$

- signal production and consumption enforce era equality (similar to *newSTRef*, *readSTRef* and *writeSTRef*)
- running reactive actions uses universal quantification (similar to *runST*):

$$\mathit{toIO} :: (\forall \mathit{era}.\mathit{Reactive}\ \mathit{era}\ \mathit{val}) \rightarrow \mathit{IO}\ \mathit{val}$$

Safe switching

- safe switching combinator:

$$\begin{aligned} \mathit{switch} :: (&\mathit{Signal\ signal}) \Rightarrow \\ &\mathit{SSignal\ era}\ (\forall \mathit{era}'.\mathit{signal\ era}'\ \mathit{val}) \rightarrow \\ &\mathit{signal\ era\ val} \end{aligned}$$

- switches only to signals that don't depend on external events:
  - empty discrete signal
  - constant segmented signals
- useless
- idea:

  switching between signal functions instead of signals

Signal functions (1)

- functions over signals with identical era:

$$
\begin{aligned}
\mathit{SignalFun}\ \mathit{era}\ (&\mathit{signal}_1\ \text{`}\mathit{Of}\text{`}\ \mathit{val}_1 \mapsto\\
&\dots \mapsto\\
&\mathit{signal}_n\ \text{`}\mathit{Of}\text{`}\ \mathit{val}_n \mapsto\\
&\mathit{signal}'\ \text{`}\mathit{Of}\text{`}\ \mathit{val}')
\end{aligned}
$$

- empty data types for type indices:

$\textbf{data}\ \mathit{shape} \mapsto \mathit{shape}'$

$\textbf{data}\ \mathit{signal}\ \text{`}\mathit{Of}\text{`}\ \mathit{val}$

Signal functions (2)

- *SignalFun* defined as a GADT:

$$
\begin{aligned}
&\textbf{data}\ \mathit{SignalFun}\ \mathit{era}\ \mathit{shape}\ \textbf{where}\\
&\quad \mathit{OSF} :: (\mathit{Signal}\ \mathit{signal}) \Rightarrow\\
&\qquad\quad \mathit{signal}\ \mathit{era}\ \mathit{val}\\
&\quad \mathit{SSF} :: (\mathit{Signal}\ \mathit{signal}) \Rightarrow\\
&\qquad\quad (\mathit{signal}\ \mathit{era}\ \mathit{val} \rightarrow \mathit{SignalFun}\ \mathit{era}\ \mathit{shape}') \rightarrow\\
&\qquad\quad \mathit{SignalFun}\ \mathit{era}\ (\mathit{signal}\ \text{`}\mathit{Of}\text{`}\ \mathit{val} \mapsto \mathit{shape}')
\end{aligned}
$$

Switching between signal functions

- type of the switching combinator:

$$switch :: SSignal\ era\ (\forall era'.SignalFun\ era'\ shape) \rightarrow SignalFun\ era\ shape$$

- how the combinator works (conceptionally):
  - arguments of the result function are cut up to fit the segments of the argument signal (ageing)
  - each function from the argument signal is applied to its corresponding slice
  - result function combines the results of these applications
- important:

  ageing is enforced

Signals, Not Generators!

Wolfgang Jeltsch

Brandenburgische Technische Universität Cottbus
Cottbus, Germany

Tenth Symposium on Trends in Functional Programming
June 2–4, 2009

Traditional implementation of *scan*

$$
\begin{array}{l}
\mathit{scan} :: \mathit{accu} \rightarrow \\
\qquad (\mathit{accu} \rightarrow \mathit{val} \rightarrow \mathit{accu}) \rightarrow \\
\qquad (\mathit{DSignal}\ \mathit{val} \rightarrow \mathit{SSignal}\ \mathit{accu}) \\
\mathit{scan}\ \mathit{init}\ \mathit{next}\ (\mathit{DSignal}\ \mathit{reg}) = \mathit{SSignal}\ \mathit{init}\ (\mathit{DSignal}\ \mathit{reg}')\ \textbf{where} \\
\quad \mathit{reg}'\ \mathit{hdlr} = \textbf{do} \\
\qquad\qquad \mathit{accuRef} \leftarrow \mathit{newIORef}\ \mathit{init} \\
\qquad\qquad \mathit{reg}\ (\lambda \mathit{val} \rightarrow \textbf{do} \\
\qquad\qquad\qquad \mathit{accu} \leftarrow \mathit{readIORef}\ \mathit{accuRef} \\
\qquad\qquad\qquad \textbf{let} \\
\qquad\qquad\qquad\quad \mathit{accu}' = \mathit{next}\ \mathit{accu}\ \mathit{val} \\
\qquad\qquad\qquad \mathit{writeIORef}\ \mathit{accuRef}\ \mathit{accu}' \\
\qquad\qquad\qquad \mathit{hdlr}\ \mathit{accu}')
\end{array}
$$