Functional Hybrid Modeling from an Object-Oriented Perspective

Henrik Nilsson (University of Nottingham),
John Peterson (Western State College),
and Paul Hudak (Yale University)
• *Functional Reactive Programming* (FRP) integrates notions suitable for *causal* hybrid modelling with functional programming.
Functional Reactive Programming (FRP) integrates notions suitable for causal hybrid modelling with functional programming.

Yampa is an instance of FRP embedded in Haskell.
**Background (1)**


- *Yampa* is an instance of FRP embedded in Haskell.

- One central idea: *first-class* reactive components (or models):
  - enables highly structurally dynamic systems to be described declaratively;
  - opens up for meta-modelling without additional language layers.
Additional interesting aspects:
- full power of a modern functional language available;
- polymorphic type system;
- well-understood underlying semantics.
Our goal with *Functional Hybrid Modelling* (FHM) is to combine an FRP-approach with non-causal modelling yielding:
Our goal with *Functional Hybrid Modelling* (FHM) is to combine an FRP-approach with non-causal modelling yielding:

- a powerful, fully-declarative, non-causal modelling language supporting highly structurally dynamic systems;
Our goal with *Functional Hybrid Modelling* (FHM) is to combine an FRP-approach with non-causal modelling yielding:

- a powerful, fully-declarative, non-causal modelling language supporting highly structurally dynamic systems;
- a semantic framework for studying modelling and simulation languages supporting structural dynamism.
The idea of FHM goes back a few years (PADL 2003). UK research funding (EPSRC) secured very recently. Thus still work in very early stages.
The Rest of the Talk

• A brief introduction to FRP/Yampa as a background.

• Sketch the key ideas of how this may be generalized to a non-causal setting.
Signal functions

Key concept: *functions on signals* (first class).

![Diagram](image)
Signal functions

Key concept: **functions on signals** (first class).

Intuition:

\[
\text{Signal } \alpha \approx \text{Time } \rightarrow \alpha \\
x :: \text{ Signal } T1 \\
y :: \text{ Signal } T2 \\
\text{SF } \alpha \beta \approx \text{Signal } \alpha \rightarrow \text{Signal } \beta \\
f :: \text{ SF } T1 \text{ T2}
\]
Key concept: *functions on signals* (first class).

Intuition:

\[ \text{Signal } \alpha \approx \text{Time } \rightarrow \alpha \]
\[ x :: \text{Signal T1} \]
\[ y :: \text{Signal T2} \]
\[ \text{SF } \alpha \beta \approx \text{Signal } \alpha \rightarrow \text{Signal } \beta \]
\[ f :: \text{SF T1 T2} \]

Additionally, *causality* required: output at time \( t \) must be determined by input on interval \([0, t]\).
Signal functions and state

Alternative view:
Signal functions and state

Alternative view:

Signal functions can encapsulate \textit{state}.

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
  \node (input) {$x(t)$};
  \node (state) [right of=input] {$\text{state}(t)$};
  \node (output) [right of=state] {$y(t)$};

  \draw[->] (input) -- (state);
  \draw[->] (state) -- (output);

  \node[below of=state] {$f$};
\end{tikzpicture}
\end{center}

$\text{state}(t)$ summarizes input history $x(t')$, $t' \in [0, t]$. 
Signal functions and state

Alternative view:

Signal functions can encapsulate state.

\[ \text{state}(t) \text{ summarizes input history } x(t'), t' \in [0, t] \].

From this perspective, signal functions are:

- **stateful** if \( y(t) \) depends on \( x(t) \) and \( \text{state}(t) \)
- **stateless** if \( y(t) \) depends only on \( x(t) \)

Integral is an example of a stateful signal function.
Programming with signal functions

In Yampa, systems are described by combining signal functions (forming new signal functions).
Programming with signal functions

In Yampa, systems are described by combining signal functions (forming new signal functions).

For example, serial composition:

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>g</td>
</tr>
</tbody>
</table>
```

Functional Hybrid Modeling from an Object-Oriented Perspective – p.9/27
Programming with signal functions

In Yampa, systems are described by combining signal functions (forming new signal functions).

For example, serial composition:

A *combinator* can be defined that captures this:

\[
(\gg\gg) :: SF \ a \ b \rightarrow SF \ b \ c \rightarrow SF \ a \ c
\]

Note: plain function operating on first-class signal function.
The Arrow framework (1)

These diagrams convey the general idea:

\[
\text{arr } f \\
\]

\[
\text{first } f \\
\]

\[
\text{loop } f \\
\]

\[
\text{first } :: \ SF \ a \ b \ \rightarrow \ SF \ (a, c) \ (b, c) \\
\text{loop } :: \ SF \ (a, c) \ (b, c) \ \rightarrow \ SF \ a \ b
\]
The Arrow framework (2)

Some derived combinators:

- $\text{second } f$
- $f \bullet\bullet g$
- $f \&\& g$
Example: Constructing a network
Example: Constructing a network
Example: Constructing a network

\[
\text{loop (arr (λ( x, y) \rightarrow ((x, y), x)))}
\]
\[
\ggg (\text{fst } f)
\]
\[
\ggg (\text{arr (λ( x, y) \rightarrow (x, (x, y))))} \ggg (g \star h)))
\]
The Arrow notation

\[
\text{f} \rightarrow \text{g} \\
\text{h} \rightarrow \text{g}
\]
The Arrow notation
The Arrow notation

\[
\text{proc } x \rightarrow \text{ do }
\]

\[
\text{rec}
\]

\[
u \leftarrow f \leftarrow (x, v)
\]

\[
y \leftarrow g \leftarrow u
\]

\[
v \leftarrow h \leftarrow (u, x)
\]

\[
\text{return } A \leftarrow y
\]
Some switching combinators:

- $\text{switch} :: SF a (b, \text{Event } c) \rightarrow (c \rightarrow SF a b) \rightarrow SF a b$

- $p\text{SwitchB} :: \text{Functor } col \Rightarrow$
  $col (SF a b) \rightarrow SF (a, col b) (\text{Event } c) \rightarrow (col (SF a b) \rightarrow c \rightarrow SF a (col b)) \rightarrow SF a (col b)$
What makes Yampa different?

- First class reactive components (signal functions).
What makes Yampa different?

- First class reactive components (signal functions).
- Supports hybrid (mixed continuous and discrete time) systems: option type $Event$ represents discrete-time signals.
What makes Yampa different?

- First class reactive components (signal functions).
- Supports hybrid (mixed continuous and discrete time) systems: option type `Event` represents discrete-time signals.
- Supports dynamic system structure through `switching combinators`:

![Diagram of switching combinators]

---

Functional Hybrid Modeling from an Object-Oriented Perspective – p.15/27
Example: Space Invaders
Functional Hybrid Modeling

Same conceptual structure as Yampa, but:
Functional Hybrid Modeling

Same conceptual structure as Yampa, but:

• First-class *relations* on signals instead of functions on signals to enable non-causal modeling.
Functional Hybrid Modeling

Same conceptual structure as Yampa, but:

- First-class *relations* on signals instead of functions on signals to enable non-causal modeling.
- Employ state-of-the-art symbolic and numerical methods for sound and efficient simulation.
Functional Hybrid Modeling

Same conceptual structure as Yampa, but:

- First-class *relations* on signals instead of functions on signals to enable non-causal modeling.
- Employ state-of-the-art symbolic and numerical methods for sound and efficient simulation.
- Adapted switch constructs.
First class signal relations

The type for a relation on a signal of type \( \text{Signal} \ \alpha \):

\[ \text{SR} \ \alpha \]

Specific relations use a more refined type; e.g. the derivative relation:

\[ \text{der} :: \text{SR} (\text{Real, Real}) \]

Since a signal carrying pairs is isomorphic to a pair of signals, \( \text{der} \) can be understood as a binary relation on two signals.
Defining relations

The following tentative construct denotes a signal relation:

\[
\text{sigrel } \text{pattern where equations}
\]

The pattern introduces \textit{signal variables} which at each point in time are going to be bound to a “sample” of the corresponding signal.

Given \( p :: t \), we have:

\[
\text{sigrel } p \text{ where } \ldots :: \text{SR } t
\]
Equations

Let $e_i :: t_i$ be non-relational expressions possibly introducing new signal variables.

Point-wise equality; the equality must hold for all points in time:

$$e_1 = e_2$$

Relation “application”; the relation must hold for all points in time:

$$sr \diamond e_3$$

Here, $sr$ is an expression having type $SR t_3$. 
Consider a differential equation like \( x' = f(x, y) \). This equation could be written:

\[
der (x, f(x, y))
\]

For convenience, *syntactic sugar* closer to standard mathematical notation could be considered:

\[
der(x) = f(x, y)
\]

Here, \( \text{der} \) is *not* a pure function operating only on instantaneous signal values since it depends on the history of the signal.
Modeling electrical components (1)

The type $\text{Pin}$ is assumed to be a record type describing an electrical connection. It has fields $v$ for voltage and $i$ for current.

\[
twoPin :: \text{SR} (\text{Pin}, \text{Pin}, \text{Voltage})
\]

\[
twoPin = \text{sigrel} (p, n, v) \text{ where }
\]

\[
v = p.v - n.v
\]

\[
p.i + n.i = 0
\]
**Modeling electrical components (2)**

resistor :: Resistance → SR(Pin, Pin)
resistor(r) = \textbf{sigrel} (p, n) \textbf{where}
\hspace{1cm} \textit{twoPin} \diamond (p, n, v) \\
\hspace{1cm} r \cdot p.i = v

inductor :: Inductance → SR(Pin, Pin)
inductor(l) = \textbf{sigrel} (p, n) \textbf{where}
\hspace{1cm} \textit{twoPin} \diamond (p, n, v) \\
\hspace{1cm} l \cdot \textbf{der}(p.i) = v
Modeling electrical components (3)

\[ \text{capacitor} :: \text{Capacitance} \rightarrow \text{SR} (\text{Pin, Pin}) \]
\[ \text{capacitor}(c) = \text{sigrel} (p, n) \text{ where} \]
\[ \text{twoPin} \diamond (p, n, v) \]
\[ c \cdot \text{der}(v) = p.i \]
Modeling an electrical circuit (1)

\[
\text{simpleCircuit } :: \text{ SR Current}
\]

\[
\text{simpleCircuit} = \text{sigrel i where}
\]

\[
\begin{align*}
\text{resistor}(1000) & \diamond (r1p, r1n) \\
\text{resistor}(2200) & \diamond (r2p, r2n) \\
\text{capacitor}(0.00047) & \diamond (cp, cn) \\
\text{inductor}(0.01) & \diamond (lp, ln) \\
\text{vSourceAC}(12) & \diamond (acp, acn) \\
\text{ground} & \diamond gp
\end{align*}
\]

\ldots
Modeling an electrical circuit (2)

\[ i = r_{1p} \cdot i + r_{2p} \cdot i \]

connect \( acp, r_{1p}, r_{2p} \)
connect \( r_{1n}, cp \)
connect \( r_{2n}, lp \)
connect \( acn, cn, ln, gp \)
Central Research Questions

- Adapting Yampa’s switching constructs, including handling initialization issues.
Central Research Questions

- Adaptating Yampa’s switching constructs, including handling initialization issues.
- Adapting non-causal modelling and simulation methods to a setting with first class signal relations: causality analysis, symbolic processing code generation after each switch.
Central Research Questions

- Adaptating Yampa’s switching constructs, including handling initialization issues.
- Adapting non-causal modelling and simulation methods to a setting with first class signal relations: causality analysis, symbolic processing, code generation after each switch.
- Guaranteeing compositional correctness statically through the type system to the extent possible; e.g. employing dependent types to keep track of variable/equation balance.