Functional Hybrid Modeling from an Object-Oriented Perspective

Henrik Nilsson (University of Nottingham),
John Peterson (Western State College),
and Paul Hudak (Yale University)

Background (1)

- **Functional Reactive Programming** (FRP) integrates notions suitable for *causal* hybrid modelling with functional programming.
- *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.

Background (2)

- Additional interesting aspects:
  - full power of a modern functional language available;
  - polymorphic type system;
  - well-understood underlying semantics.

Functional Hybrid Modelling (1)

- 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.

**Signal functions**

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

![Diagram](x \rightarrow f \rightarrow y)

Intuition:
- Signal $\alpha \approx \text{Time} \rightarrow \alpha$
- $x :: \text{Signal } T1$
- $y :: \text{Signal } T2$
- $\text{SF } \alpha \beta \approx \text{Signal } \alpha \rightarrow \text{Signal } \beta$
- $f :: \text{SF } T1 \text{ } T2$

Additionally, *causality* required: output at time $t$ must be determined by input on interval $[0,t]$.

**Signal functions and state**

Alternative view:

Signal functions can encapsulate *state*.

![Diagram](x(t) \rightarrow f_{\text{state}(t)} \rightarrow y(t))

$\text{state}(t)$ 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.

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.
### Programming with signal functions

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

For example, serial composition:

\[
\begin{array}{c}
\fbox{f} \\
\rightarrow
\fbox{g}
\end{array}
\]

A *combinator* can be defined that captures this:

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

Note: plain function operating on first-class signal function.

---

### The Arrow framework (1)

These diagrams convey the general idea:

- \(\text{arr}\; f\)
- \(f \gg g\)
- \(\text{first}\; f\)
- \(\text{loop}\; f\)

\[
\begin{align*}
\text{first} & :: SF\; a\; b \rightarrow SF\; (a, c)\; (b, c) \\
\text{loop} & :: SF\; (a, c)\; (b, c) \rightarrow SF\; a\; b
\end{align*}
\]

---

### The Arrow framework (2)

Some derived combinators:

- \(\text{second}\; f\)
- \(f \gg\gg g\)
- \(f \&\&\& g\)

---

### Example: Constructing a network
The Arrow notation

Switching

Some switching combinators:

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

- $\text{pSwitchB} :: \text{Functor } \text{col} \Rightarrow \text{col} \ (\text{SF } a \ b) \rightarrow \text{SF } (a, \text{col } b) \ (\text{Event } c) \rightarrow (\text{col} \ (\text{SF } a \ b) \rightarrow c \rightarrow \text{SF } a \ (\text{col } b)) \rightarrow \text{SF } a \ (\text{col } b)$

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:

Example: Space Invaders
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 Signal $\alpha$:

$$SR \alpha$$

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

$$\text{der} :: SR (\text{Real}, \text{Real})$$

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:

$$\text{sigrel} \ \text{pattern} \ \text{where} \ \text{equations}$$

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

Given $p :: t$, we have:

$$\text{sigrel} \ p \ \text{where} \ldots :: SR \ t$$

Equations

Let $e_1 :: t_i$ 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:

$$sr \odot e_3$$

Here, $sr$ is an expression having type $SR \ t_3$. 
Equations: examples

Consider a differential equation like \( x' = f(x, y) \). This equation could be written:

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

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

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

Here, \( der \) 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.

\[
twoPin :: \text{SR (Pin, Pin, Voltage)}
twoPin = \text{sigrel } (p, n, v) \text{ where } v = p.v - n.v \quad p.i + n.i = 0
\]

Modeling electrical components (2)

\[
resistor :: \text{Resistance} \rightarrow \text{SR (Pin, Pin)}
resistor(r) = \text{sigrel } (p, n) \text{ where } twoPin \circ (p, n, v)
\]

\[
\quad r \cdot p.i = v
\]

\[
inductor :: \text{Inductance} \rightarrow \text{SR (Pin, Pin)}
inductor(l) = \text{sigrel } (p, n) \text{ where } twoPin \circ (p, n, v)
\]

\[
\quad l \cdot \text{der}(p.i) = v
\]

Modeling electrical components (3)

\[
capacitor :: \text{Capacitance} \rightarrow \text{SR (Pin, Pin)}
capacitor(c) = \text{sigrel } (p, n) \text{ where } twoPin \circ (p, n, v)
\]

\[
\quad c \cdot \text{der}(v) = p.i
\]
Modeling an electrical circuit (1)

\[
\text{simpleCircuit} :: \text{SR Current}
\]
\[
\text{simpleCircuit} = \text{sigrel i where}
\]
\[
\text{resistor}(1000) \diamond (r1p, r1n)
\]
\[
\text{resistor}(2200) \diamond (r2p, r2n)
\]
\[
\text{capacitor}(0.00047) \diamond (cp, cn)
\]
\[
\text{inductor}(0.01) \diamond (lp, ln)
\]
\[
\text{vSourceAC}(12) \diamond (acp, acn)
\]
\[
\text{ground} \diamond \text{gp}
\]

... 

Modeling an electrical circuit (2)

\[
\text{\ldots connect acp, r1p, r2p}
\]
\[
\text{connect r1n, cp}
\]
\[
\text{connect r2n, lp}
\]
\[
\text{connect acn, cn, ln, gp}
\]
\[
i = r1p.i + r2p.i
\]

Central Research Questions

- Adapting 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.