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.

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.

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:

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;

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.

Functional Hybrid Modelling (2)

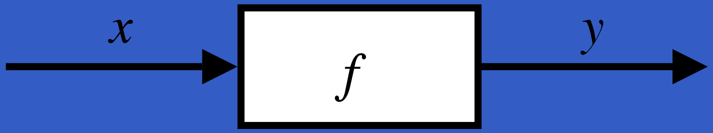
- The idea of FHM goes back a few years (PADL 2003). UK research funding (EPSRC) secured very recently. Thus still work in very early stages.

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.

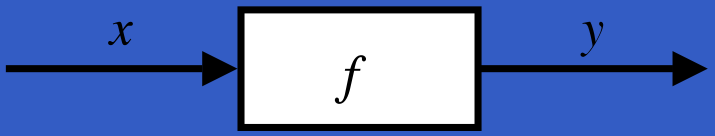
Signal functions

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



Signal functions

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



Intuition:

$$\texttt{Signal}\ \alpha \approx \texttt{Time} \rightarrow \alpha$$

$$x :: \texttt{Signal T1}$$

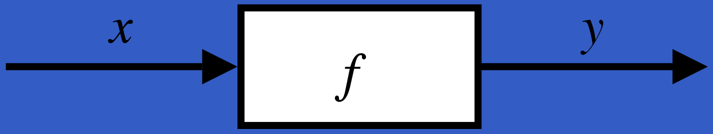
$$y :: \texttt{Signal T2}$$

$$\texttt{SF}\ \alpha\ \beta \approx \texttt{Signal}\ \alpha \rightarrow \texttt{Signal}\ \beta$$

$$f :: \texttt{SF T1 T2}$$

Signal functions

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



Intuition:

$\texttt{Signal}\ \alpha \approx \texttt{Time} \rightarrow \alpha$

x :: Signal T1

y :: Signal T2

$\texttt{SF}\ \alpha\ \beta \approx \texttt{Signal}\ \alpha \rightarrow \texttt{Signal}\ \beta$

f :: SF T1 T2

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

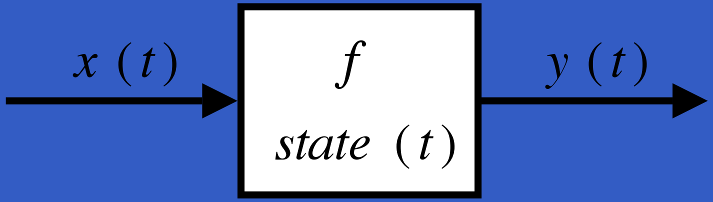
Signal functions and state

Alternative view:

Signal functions and state

Alternative view:

Signal functions can encapsulate ***state***.

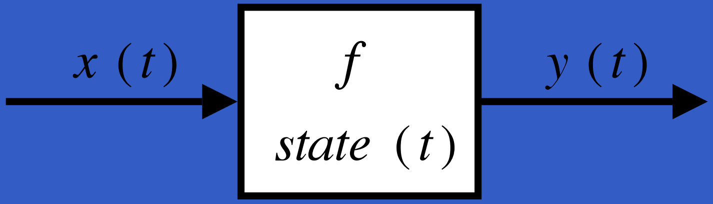


$state(t)$ summarizes input history $x(t')$, $t' \in [0, t]$.

Signal functions and state

Alternative view:

Signal functions can encapsulate ***state***.



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

From this perspective, signal functions are:

- ***stateful*** if $y(t)$ depends on $x(t)$ and $state(t)$
- ***stateless*** if $y(t)$ depends only on $x(t)$

Integral is an example of a stateful signal function.

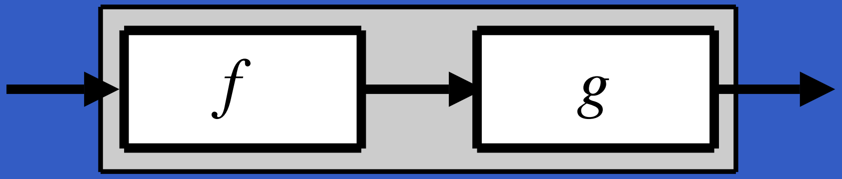
Programming with signal functions

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).

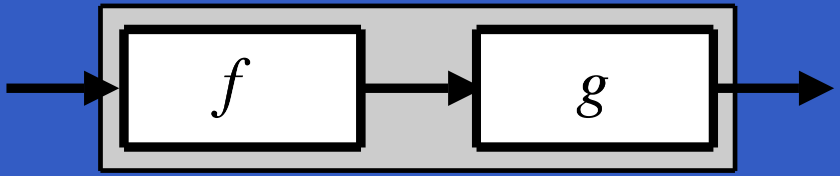
For example, serial composition:



Programming with signal functions

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

For example, serial composition:



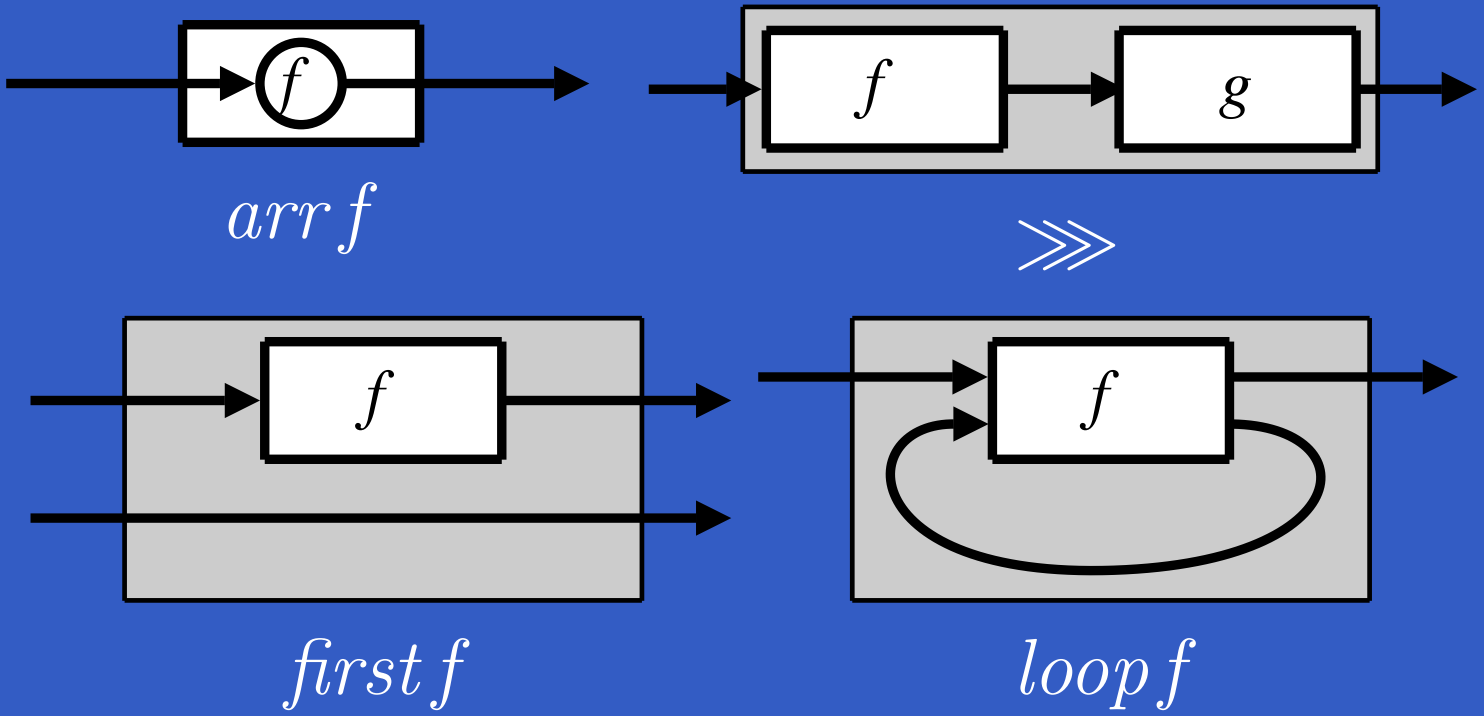
A *combinator* can be defined that captures this:

$$(\ggg) :: SF\ a\ b \rightarrow SF\ b\ c \rightarrow SF\ a\ c$$

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

The Arrow framework (1)

These diagrams convey the general idea:



$arr\, f$

$\ggg$

$first\, f$

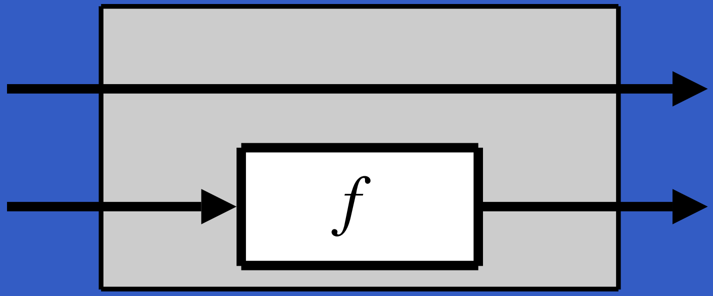
$loop\, f$

$$first :: SF\ \ a\ \ b \longrightarrow SF\ \ (a, c)\ \ (b, c)$$

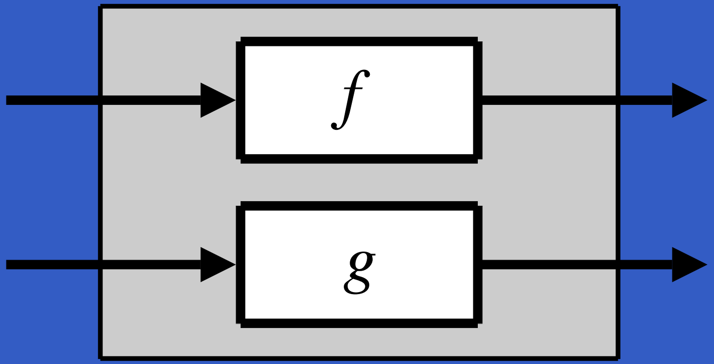
$$loop :: SF\ \ (a, c)\ \ (b, c) \longrightarrow SF\ \ a\ \ b$$

The Arrow framework (2)

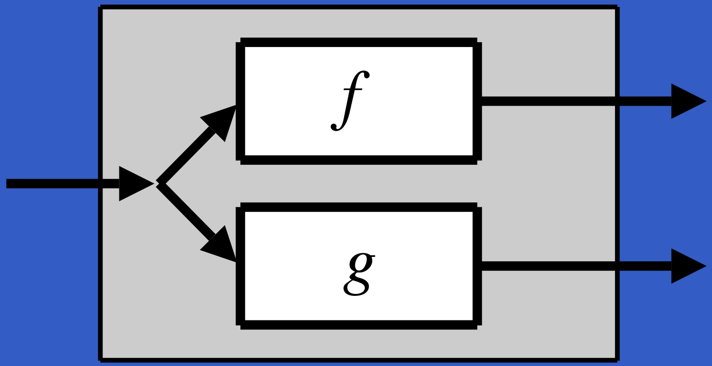
Some derived combinators:



$second\, f$

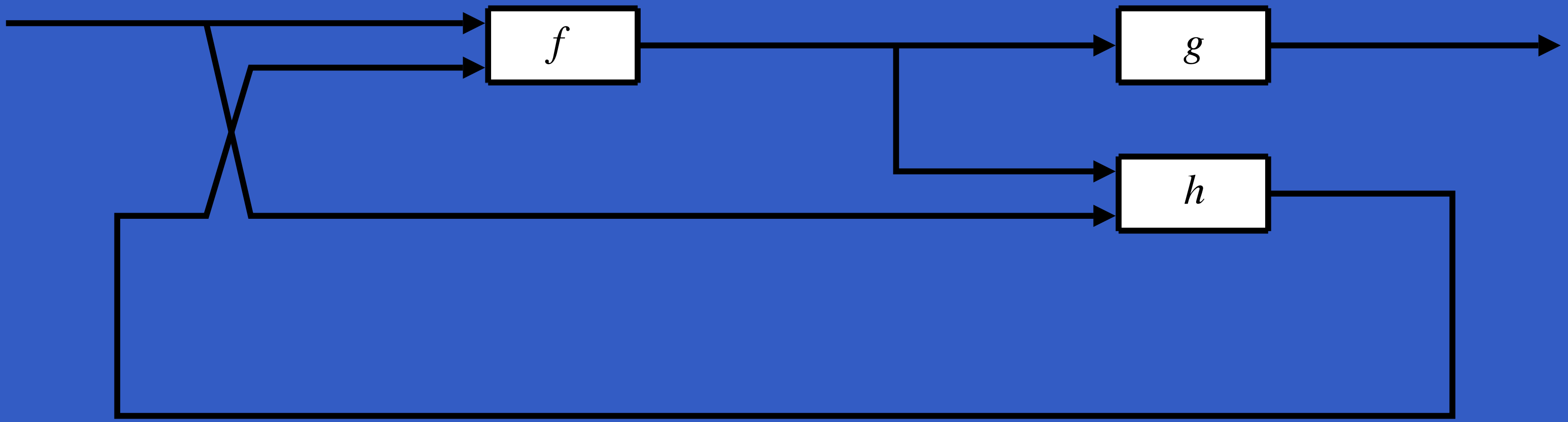


$f *** g$

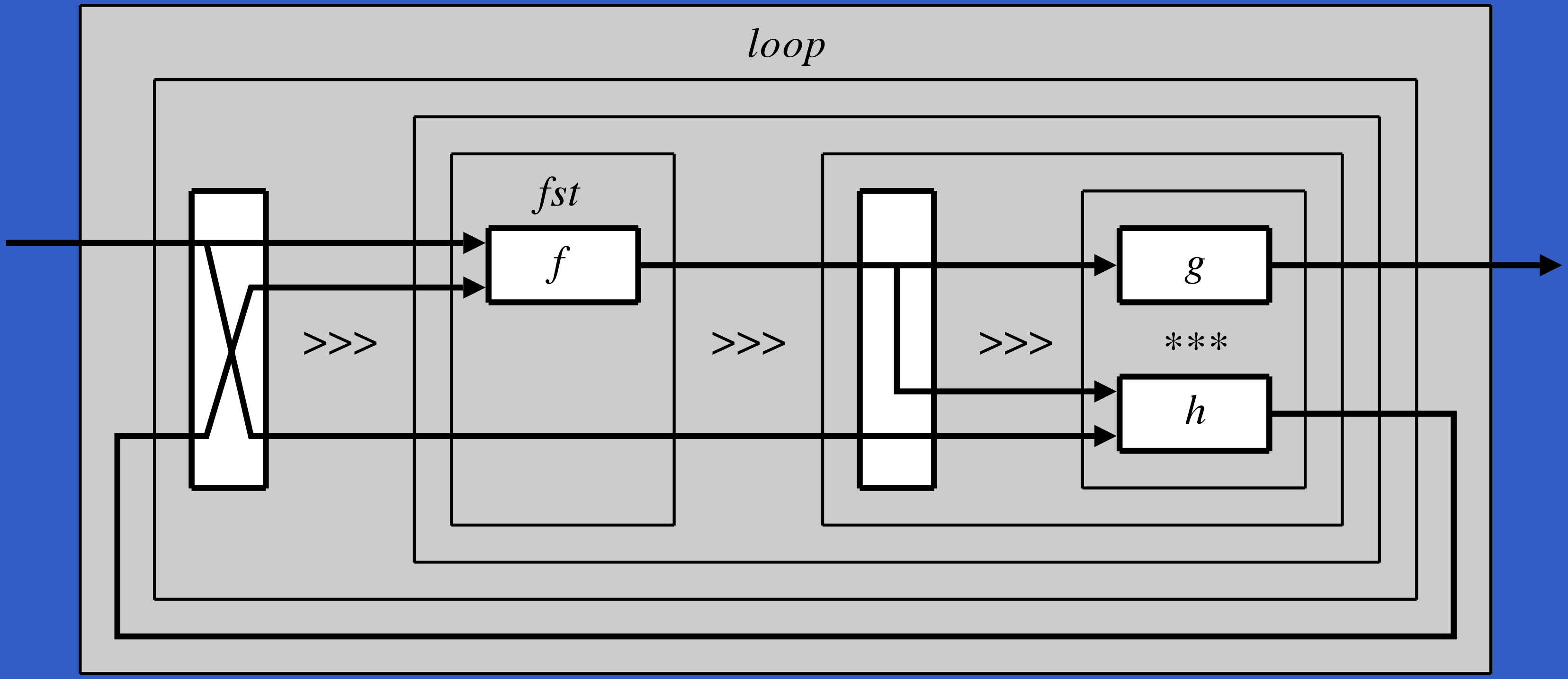


$f \&\&\& g$

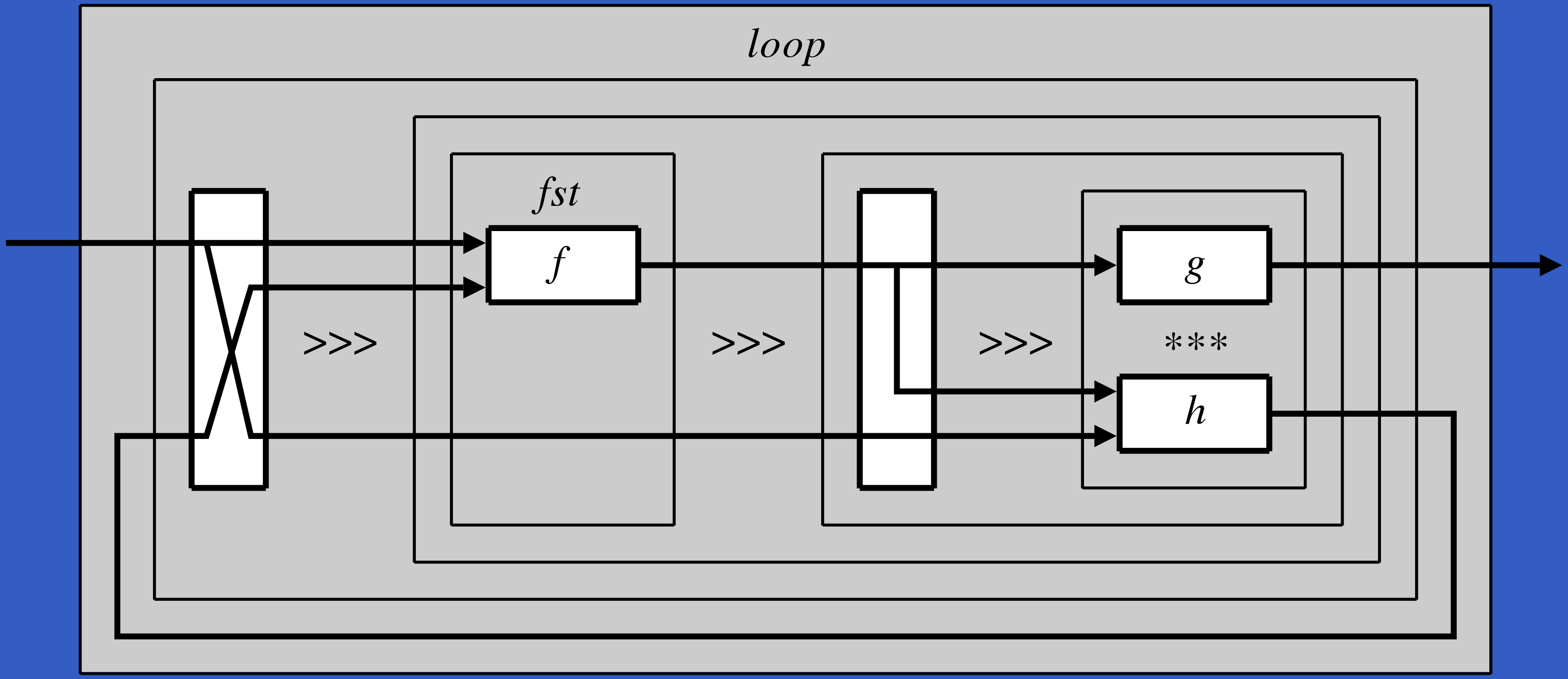
Example: Constructing a network



Example: Constructing a network

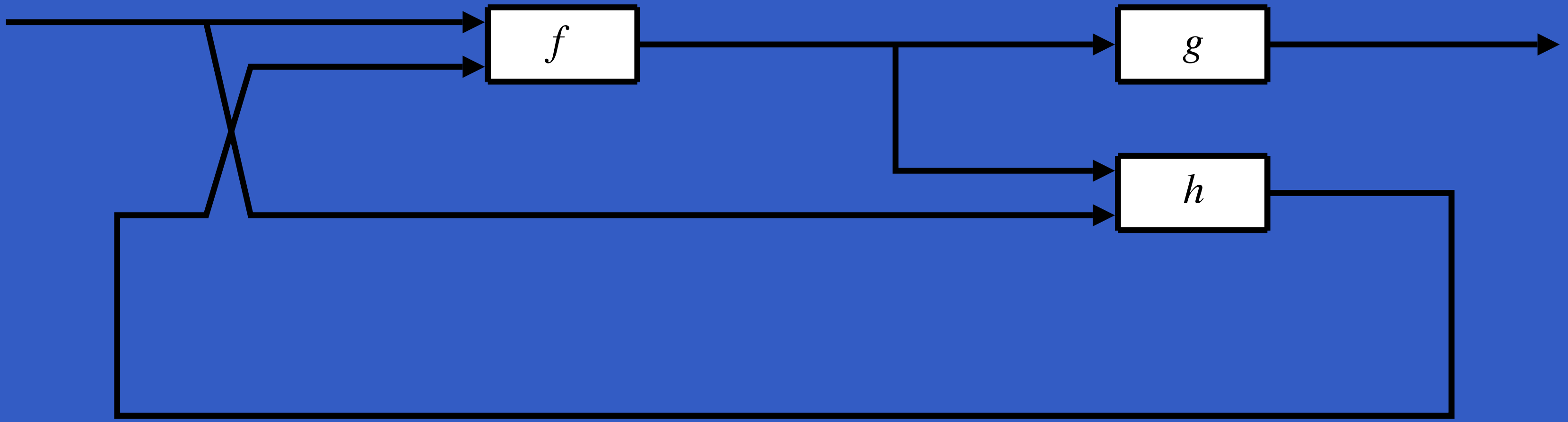


Example: Constructing a network

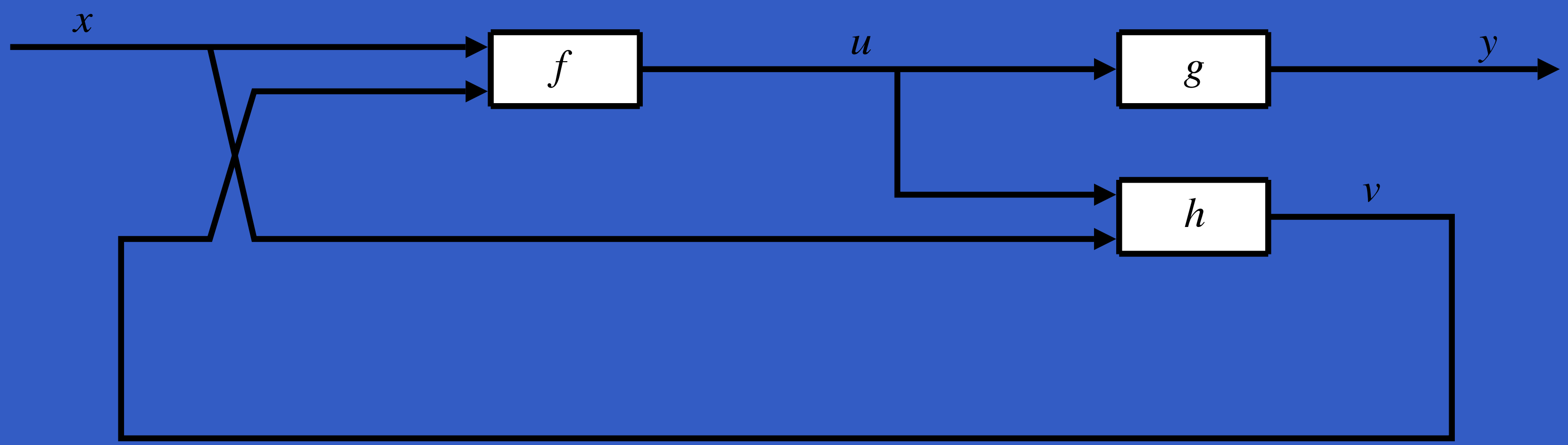


$$\begin{aligned} &loop\ (arr\ (\lambda(x,y) \rightarrow ((x,y),x)) \\ &\qquad \ggg (fst\ f \\ &\qquad\qquad \ggg (arr\ (\lambda(x,y) \rightarrow (x,(x,y))) \ggg (g *\!\!*\!\!* h)))) \end{aligned}$$

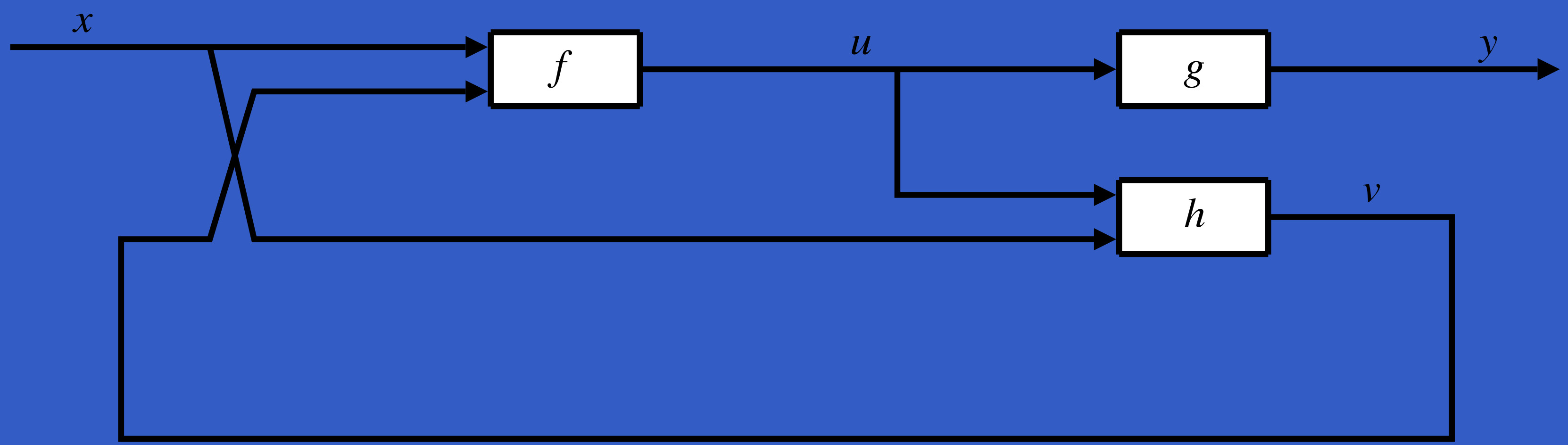
The Arrow notation



The Arrow notation



The Arrow notation



$$
\begin{array}{l}
\textbf{proc } x \rightarrow \textbf{do} \\
\quad \textbf{rec} \\
\qquad u \leftarrow f \prec (x, v) \\
\qquad y \leftarrow g \prec u \\
\qquad v \leftarrow h \prec (u, x) \\
\quad returnA \prec y
\end{array}
$$

Switching

Some switching combinators:

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

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

What makes Yampa different?

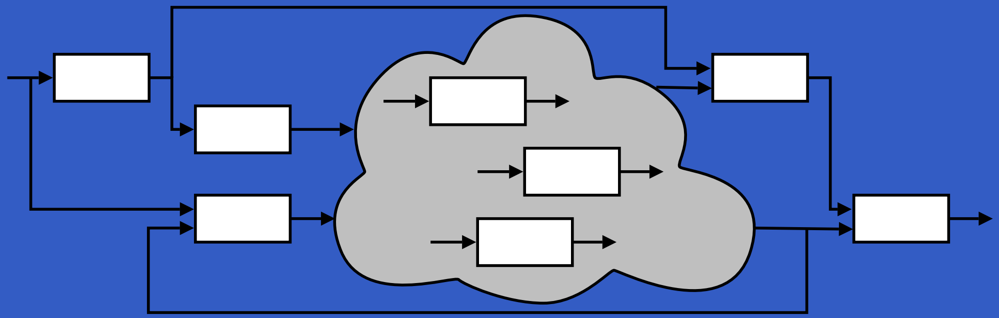
- First class reactive components (signal functions).

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***:



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 `Signal` α:

`SR` α

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

der :: `SR (Real, Real)`

Since a signal carrying pairs is isomorphic to a pair of signals, der can be understood as a binary relation on two signals.

Defining relations

The following tentative construct denotes a signal relation:

sigrel $pattern$ **where** $equations$

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:

sigrel p **where** $\ldots$ :: 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.

Equations: examples

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

$$der \diamond (x, f(x, y))$$

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

$$\textbf{der}(x) = f(x, y)$$

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 `Pin` is assumed to be a record type describing an electrical connection. It has fields v for voltage and i for current.

$twoPin$ :: SR (`Pin`, `Pin`, `Voltage`)

$twoPin =$ **sigrel** (p, n, v) **where**

$$v = p.v - n.v$$
$$p.i + n.i = 0$$

Modeling electrical components (2)

$$
\begin{aligned}
&\mathit{resistor} :: \texttt{Resistance} \rightarrow \texttt{SR}\,(\texttt{Pin}, \texttt{Pin})\\
&\mathit{resistor}(r) = \textbf{sigrel}\ (p, n)\ \textbf{where}\\
&\qquad\qquad\qquad \mathit{twoPin} \diamond (p, n, v)\\
&\qquad\qquad\qquad r \cdot p.i = v\\
&\mathit{inductor} :: \texttt{Inductance} \rightarrow \texttt{SR}\,(\texttt{Pin}, \texttt{Pin})\\
&\mathit{inductor}(l) = \textbf{sigrel}\ (p, n)\ \textbf{where}\\
&\qquad\qquad\qquad \mathit{twoPin} \diamond (p, n, v)\\
&\qquad\qquad\qquad l \cdot \textbf{der}(p.i) = v
\end{aligned}
$$

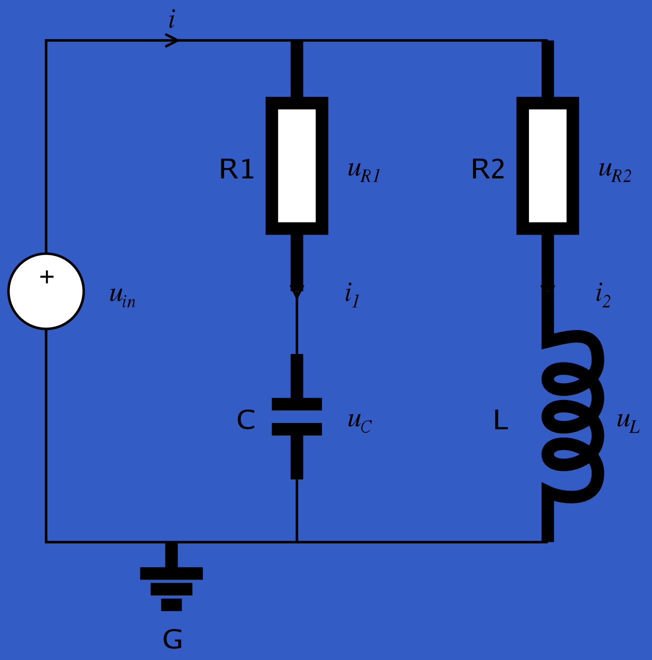
Modeling electrical components (3)

$$
\begin{aligned}
&capacitor :: \texttt{Capacitance} \rightarrow \texttt{SR}\,(\texttt{Pin}, \texttt{Pin})\\
&capacitor(c) = \textbf{sigrel}\ (p, n)\ \textbf{where}\\
&\qquad\qquad\qquad twoPin \diamond (p, n, v)\\
&\qquad\qquad\qquad c \cdot \textbf{der}(v) = p.i
\end{aligned}
$$

Modeling an electrical circuit (1)

$$
\begin{array}{l}
simpleCircuit :: \mathsf{SR}\ \mathtt{Current} \\
simpleCircuit = \textbf{sigrel}\ \mathsf{i}\ \textbf{where} \\
\qquad resistor(1000) \diamond (r1p, r1n) \\
\qquad resistor(2200) \diamond (r2p, r2n) \\
\qquad capacitor(0.00047) \diamond (cp, cn) \\
\qquad inductor(0.01) \diamond (lp, ln) \\
\qquad vSourceAC(12) \diamond (acp, acn) \\
\qquad ground \diamond gp \\
\qquad \ldots
\end{array}
$$

Modeling an electrical circuit (2)



...

connect acp, $r1p$, $r2p$

connect $r1n$, cp

connect $r2n$, lp

connect acn, cn, ln, gp

$i = r1p.i + r2p.i$

Central Research Questions

- Adaptating 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.