The Assumption of Fixed Causality

- Current main-stream non-causal modelling and simulation languages, like Modelica, are designed and implemented assuming causality remains fixed during simulation:
  - Simplifies the language
  - Facilitates efficient implementation
- But this assumption is very limiting for hybrid modelling; even simple systems often cannot be simulated:
  - Breaking pendulum
  - Ideal diodes in various configurations.

This Talk

- Introduction to Functional Hybrid Modelling (FHM):
  - Novel approach to non-causal, hybrid modelling and simulation.
  - Designed and implemented assuming an evolving system:
    - in particular, causality allowed to change
    - even more drastic changes possible.
- Application to modelling with ideal diodes:
  - Half-wave rectifier with in-line inductor
  - Full-wave rectifier

FHM in a Nutshell

- A functional approach to modelling and simulation of (physical) systems.
- Two-level design:
  - equational level for modelling components
  - functional level for spatial and temporal composition of components
- Non-causal modelling: undirected equations.
- First-class models (= systems of equations) at the functional level.
- Equation systems allowed to evolve over time.
Functional?

“Functional” as in Pure Functional Programming:

- **Declarative** programming paradigm
- Programs are pure functions: *no side effects*.
- Not just functions on “numbers”: arguments and results may be arbitrary types, including:
  - functions
  - models = systems of equations
- Both functions and models are thus *first-class entities*.

Different?

Is FHM very different from current non-causal languages like Modelica?

Yes, in some ways, but:

- FHM *implementation techniques* could be used in the implementations of existing non-causal languages to improve their support for systems with evolving structure.
- FHM could be viewed as a *core language*:
  - semantics
  - compilation target

Prototype Hydra Implementation (1)

The current FHM instance is called *Hydra*:

- Embedding in *Haskell*.
- Model *transformed* to form suitable for simulation, then *JIT compiled to native code* by an embedded compiler.
- State-of-the art *numerical solvers from SUNDIALS* suite (from LLNL) used for simulation and event detection.
- Transformation and compilation *repeated* when system structure changes at events.

Prototype Hydra Implementation (2)

```latex
f(\vec{y}, \frac{d\vec{y}}{dt}, t) = 0
SR a
```

- **SR a**
- Embedded Compiler
- LLVM JIT
- Event Handler
- Numerical Solver
- Event Detector
- Simulation Result

Simulation Result

Event
**Example: A Simple Circuit**

![Simple Circuit Diagram]

**Simple Circuit: Causal Model**

\[
\begin{align*}
R_2 i_2 &= u_L \\
L i_2' &= u_C \\
R_1 i_1 &= u_{in} \\
C i_1 &= i_2 \\
\end{align*}
\]

**Simple Circuit: Non-Causal Model (1)**

Non-causal resistor model:
\[
\begin{align*}
\nu_p - \nu_n &= u \\
i_p + i_n &= 0 \\
R i_p &= u
\end{align*}
\]

Non-causal inductor model:
\[
\begin{align*}
\nu_p - \nu_n &= u \\
i_p + i_n &= 0 \\
L i_p' &= u
\end{align*}
\]

Note the commonality: can be factored out as a separate *two pin* component.

**Simple Circuit: Non-Causal Model (2)**

A non-causal model of the entire circuit is created by *instantiating* the component models: copy the equations and rename the variables.

The instantiated components are then *composed* by adding connection equations according to Kirchhoff’s laws, e.g.:
\[
\begin{align*}
\nu_{R_1,n} &= v_{C,p} \\
i_{R_1,n} + i_{C,p} &= 0
\end{align*}
\]
Record describing an electrical connection with fields \( v \) for voltage and \( i \) for current.

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

(Partial) model represented by relation over 5 time-varying entities, i.e. signals.

(Note: Somewhat idealised syntax compared with present implementation.)

Parametrised model represented by function mapping parameters to a model. Note: first class models!

Signal relation application allows modular construction of models from component models.

Inductors and capacitors are modelled similarly:

\[
inductor :: \text{Inductance} \rightarrow SR(Pin, Pin)
\]
\[
inductor l = \text{sigrel}(p, n) \ 	ext{where}
\]
\[
twoPin \diamond (p, n, u)
\]
\[
l \cdot \text{der}(p.i) = u
\]

\[
capacitor :: \text{Capacitance} \rightarrow SR(Pin, Pin)
\]
\[
capacitor c = \text{sigrel}(p, n) \ 	ext{where}
\]
\[
twoPin \diamond (p, n, u)
\]
\[
c \cdot \text{der}(u) = p.i
\]

\[
simpleCircuit :: SR \text{ Current}
\]
\[
simpleCircuit = \text{sigrel} i \ 	ext{where}
\]
\[
\text{resistor}(1000) \diamond (r1p, r1n)
\]
\[
\text{resistor}(2200) \diamond (r2p, r2n)
\]
\[
\text{capacitor}(0.00047) \diamond (cp, cn)
\]
\[
\text{inductor}(0.01) \diamond (lp, ln)
\]
\[
\text{vSourceAC}(12) \diamond (acp, acn)
\]
\[
\text{ground} \diamond gp
\]
\[
\ldots
\]
Simple Circuit in FHM (4)

\[ i = r_1 p \cdot i + r_2 p \cdot i \]

Notes on the Causal Model (1)

- Topology of causal model and modelled system do not agree:

Notes on the Causal Model (2)

- A small change in the modelled system can lead to large changes in the model.

Example: Ideal Diodes (1)

- In non-causal modelling, user need not worry about causality, but the simulator may well exploit structural properties like causality for e.g. efficient simulation.
- Once-off exploitation of any structural properties will preclude significant dynamic structural changes.
The in-line inductor means that an assumption of fixed causality will cause *simulation to fail* with a division by zero when the switch opens.

\[
i_{\text{Diode}} :: \text{SR}(\text{Pin}, \text{Pin})
\]
\[
i_{\text{Diode}} = \text{sigrel}(p, n) \text{ where }
\]
\[
two\text{Pin} \bowtie (p, n, u)
\]
\[
\text{initially: when } p.v - n.v > 0 \implies u = 0
\]
\[
\text{when } p.i < 0 \implies p.i = 0
\]

(Note: again, syntax somewhat idealised compared with present implementation.)
To simulate the full-wave rectifier:

- The diode model has to be extended to allow expressing the voltage over the diodes always pairwise equal:
  \[
  ic_{Diode} :: SR(Pin, Pin, Voltage)
  \]
  \[
  ic_{Diode} = sigrel(p, n, u) \text{ where}
  \]
  \[
  \text{twoPin} \diamond (p, n, u)
  \]
  \[
  \text{initially; when } p.v - n.v > 0 \Rightarrow u = 0
  \]
  \[
  \text{when } p.i < 0 \Rightarrow p.i = 0
  \]

Example: Ideal Diodes (7)

- Redundant, semantically identical equations needs to be eliminated ("constant propagation" suffice in this case).
- End result is a fairly compositional model.
- No separate formalism, such as state charts, for controlling the switching.
- No need to worry about the here \(2^4 = 16\) (and, in general, \(2^n\)) possible modes: each mode computed on demand.
- No domain-specific assumptions built into the language itself.

Conclusions

- Assuming unchanging structural properties like causality severely limits what hybrid models can be simulated.
- Avoiding this restriction allows a number of challenging systems to be modelled and simulated in a straightforward manner.
- Of course not the whole story; many challenging problems remain: e.g., state transfer between structural configurations, chattering . . .