EXPLOITING STRUCTURAL DYNAMISM IN FUNCTIONAL HYBRID MODELLING FOR SIMULATION OF IDEAL DIODES

Henrik Nilsson[1], **George Giorgidze**[1]

[1]School of Computer Science, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB
United Kingdom

nhn@cs.nott.ac.uk(Henrik Nilsson)

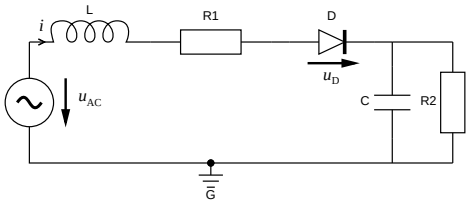
Abstract

Current main-stream non-casual modelling and simulation languages, like Modelica, are designed and implemented on the assumption that model causality remains fixed during simulation. This simplifies the language design and facilitates the generation of efficient simulation code. In particular, simulation code can be generated once and for all, at compile time. However, for hybrid models, the assumption of fixed causality is very limiting: there are many examples, including simple ones, that cannot be simulated. A half-wave rectifier with an ideal diode and an in-line inductor is one such example. Functional Hybrid Modelling is a new approach to non-causal modelling where models are first-class entities and structural dynamism is supported by switching among model configurations. Fixed causality is thus *not* assumed. Continuous simulation remains efficient thanks to just-in-time generation of simulation code at structural changes and the use of a standard, industrial-strength solver. Re-generation of code at each structural change of course incurs an overhead, but this is typically modest. In this paper we demonstrate how Functional Hybrid Modelling makes it possible to simulate electrical circuits with ideal diodes in a straightforward manner. We consider both a half-wave rectifier and a significantly more challenging full-wave rectifier.

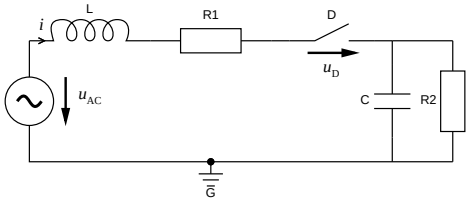
Keywords: Non-causal Modelling and Simulation, Structurally Dynamic Systems, Just-In-Time Compilation

Presenting Author's Biography

Dr. Henrik Nilsson is a Lecturer at the School of Computer Science, University of Nottingham. He holds a PhD in Computer Science from Linköping University, Sweden. Prior to taking up his current post, he held a position as Associate Research Scientist at the Department of Computer Science, Yale University. His research interests include functional programming and programming language design and implementation, specifically domain-specific ones. At present, his research is focused on non-causal modelling languages supporting structurally dynamic systems.



(a) Circuit



(b) Circuit modelled using an ideal diode model

Fig. 1 Half-wave rectifier with in-line inductor.

1 Introduction

The support offered by current, main-stream, non-casual modelling and simulation languages like Modelica [1] for handling structurally dynamic models, or *structural dynamism*, is quite limited [2, 3, 4]. One reason is that this kind of language tends to be designed and implemented on the assumption that the causality of the model does not change during simulation. This assumption simplifies the language design and facilitates the generation of efficient simulation code. In particular, the causality can be analysed and code can be generated once and for all, at compile time, paving the way for using a fast, explicit solver for simulation.

However, from a modelling perspective, the assumption of fixed causality is quite restricting: there are many examples, even of quite simple systems, where a straightforward, non-causal model violates this assumption. One such example is that of a breaking pendulum as discussed in the Modelica tutorial [5, pp. 31–33], where a causal reformulation is suggested as a workaround. Another example, from Cellier's and Kofman's book *Continuous System Simulation* [6, pp. 439-443], is that of a half-wave rectifier with an in-line inductor as illustrated in Fig. 1(a).

Consider modelling this circuit using an ideal model of the diode; i.e., a switch controlled by the polarity of the voltage and current through it, as illustrated in Fig. 1(b). This can be *modelled* easily enough in languages like Modelica. However, any attempt to *simulate* this model assuming fixed causality, as current main-stream non-causal language implementations tend to, will fail as the causalised model will lead to a division by zero when the switch is open: there simply is no one fixed-causality model that is valid both when the switch is open and closed.

One common solution to this problem is to avoid the ideal model and opt for a slightly more realistic *leaky* diode model instead. This works, but often leads to very stiff equations. Thus, if an ideal model would suffice for the purpose at hand, that would many times be preferable [6, p. 443].

Another solution in this particular case, as the structural change actually is not very drastic (the number of equations and variables stays the same), would be to avoid causalising the equations and use an implicit solver for simulation. However, explicit solvers are more efficient, and the applicability of this approach is limited as structural changes in general can be rather more profound (even the simple aforementioned breaking pendulum model is an example of this).

In this paper, we explore an alternative approach to modelling ideal diodes offered by *not* assuming fixed causality. This means that drastic changes to the configuration of a model, such as changing the number equations or variables, become possible.

The setting for our work is Functional Hybrid Modelling (FHM) [7, 8, 9], a novel approach to purely declarative, non-causal modelling where models are first class entities and support is provided for modelling highly structurally dynamic systems. Obviating the need to assume fixed causality and support for highly dynamic systems are both achieved by generating new equations whenever switching causes the model configuration, or *mode*, to change. Generation of equations on demand, as new modes are entered, also means that systems with arbitrarily many modes can be handled. For example, the fact that the number of possible modes of a circuit involving ideal diodes grows exponentially with the number of diodes is not *in itself* a show-stopper in this setting. Of course, there may be other difficulties, such as state transfer and initialisation. We discuss our current approach in section 2.2, but we acknowledge that these problem in general are hard.

Continuous simulation remains efficient in our present FHM implementation, called *Hydra*, thanks to just-in-time (JIT) compilation of the equations to native code at each change of the configuration (using the Low Level Virtual Machine (LLVM) [10]) and the use of a standard, industrial-strength solver (the SUNDIALS suite `http://www.llnl.gov/casc/sundials/`). Re-generation of equations and compilation of them into native code at the point of each structural change of course incurs an overhead, but this overhead is usually modest [9]. Hydra is implemented as a domain-specific language embedded in the functional language Haskell. At present, Hydra is very much work in progress. The latest version of the source code can be obtained via the second author's web page (`http://www.cs.nott.ac.uk/~ggg`).

The flexibility of FHM makes it possible to simulate electrical circuits with ideal diodes in a very straightforward manner. We demonstrate this in the following using Hydra, our present FHM implementation. We consider both the half-wave rectifier from Fig. 1(b) and a significantly more challenging full-wave rectifier.

Other structurally dynamic models that we have sim-

ulated successfully, using a non-causal model formulation, include the breaking pendulum and a switched engine model due to Zimmer [4]. The point of the latter is to replace a detailed engine model by a simpler one some time after the simulation start for efficiency purposes. Neither of these examples can be simulated if the causality is assumed to be fixed.

In a wider perspective, the key ideas put forward in this paper are not tied to FHM, but should be applicable to other equation-based modelling languages offering support for (highly) structurally dynamic systems, such as Sol [4] and, as long as the structural modes are not too many, MOSILAB [11]. This paper can thus be seen as a case study illustrating some benefits of a more flexible support for structural dynamism in future versions of equation-based languages like Modelica in general.

Moreover, many of the ideas could also be applicable in a *causal* setting. The conceptual design of FHM, in particular the first-class treatment of models and the support for structural dynamism, originated in the work on Functional Reactive Programming (FRP) [12, 13, 14]. From a modelling perspective, FRP can be viewed as a causal (or block-oriented) modelling language, distinguished by its support for structural dynamism. While the numerical sophistication of current FRP implementations typically is not adequate for serious simulation work, the conceptual similarity between FHM and FRP suggests that the implementation techniques used for FHM also could be used for supporting structural dynamism in a setting of causal modelling and simulation, or, indeed, as a basis for combined causal and non-causal languages.

The rest of this paper is organised as follows. Section 2 provides some further background on FHM and Hydra. However, it is not intended as an in-depth introduction: for that, the interested reader is referred to earlier papers on FHM [7, 8, 9]. We then consider modelling and simulation of the half-wave rectifier in section 3 and the full-wave rectifier in section 4. Related work is discussed in section 5. Finally, section 6 considers future work and provides some concluding remarks.

2 Functional Hybrid Modelling

At the heart of Functional Hybrid Modelling (FHM) is the notion of a *Signal Relation*. A signal relation is simply an encapsulated (fragment of a) system of Differential Algebraic Equations (DAE): it has an interface, consisting of a number of variables, and the encapsulated equations expresses relations among these variables (or equivalently, constraints on them). As we are concerned with DAEs, the variables will in general stand for time-varying entities, or *signals*. Hence signal relation.

A signal relation is thus in many ways like a *class* in object-oriented, non-causal languages like Modelica. However, unlike an object-oriented modelling language, there is no inheritance. Reuse is instead achieved by defining new signal relations in terms of others, much like a procedure can be defined in terms of other procedures by calling them. Furthermore, signal relations are *first-class entities* in FHM. This means they can be manipulated programmatically (past as arguments to functions, returned as results of functions, etc.), just like any other type of value such as integers or Booleans. This is sometimes referred to as *higher-order modelling* [15] (Here, *higher-order* is used analogously to the use in functional programming, where a function operating on functions is referred to as a higher-order function. Not to be confused with other meanings of higher-order.)

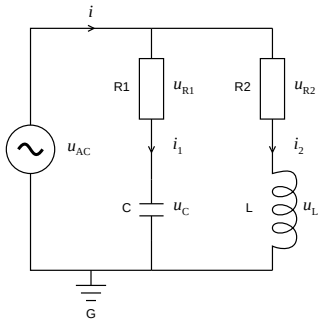


Fig. 2 A simple electrical circuit.

Higher-order modelling, with just a minimum of additional language constructs, lends itself very well to *expressing* highly structurally dynamic systems: all that is needed is the means to allow new model fragments to be computed not only before simulation starts, but also *during* simulation, at events, and to be integrated into the simulated system at those points. This is the approach taken by FHM. Indeed, the ease by which higher-order modelling can express structural dynamics was partly what motivated our research into FHM in the first place [7].

2.1 A Simple Electrical Circuit in Hydra

To illustrate the basic ideas of FHM, we model the electrical circuit in Fig. 2 (adapted from [5]) in Hydra. To avoid introducing too many aspects at once, our initial example has an entirely static structure; we return to structural dynamism in the next subsection. As Hydra is work in progress, and to prevent minor syntactic details from getting in the way of the presentation, we adopt a somewhat idealised Hydra syntax in the following.

We start by modelling a generic electrical component with two pins, capturing the aspects that are common to all such components through two equations:

$$
\begin{aligned}
&twoPin :: SR\ (Pin, Pin, Voltage)\\
&twoPin = \mathbf{sigrel}\ (p, n, u)\ \mathbf{where}\\
&\quad p.v - n.v = u\\
&\quad p.i + n.i\ \ = 0
\end{aligned}
$$

The symbol :: is pronounced “has type” and is used to declare the type of defined entities. SR is the type constructor for signal relations. The signal relation $twoPin$ is thus notionally a relation between three signals: two of type Pin and one of type $Voltage$. However, the type Pin is a record type describing an electrical connection. It has fields v for voltage and i for current. The name

Pin is perhaps a bit misleading since it just represents a pair of physical quantities, *not* a physical "pin component". The signal relation $twoPin$ is thus actually a relation among *five* physical quantities: the potential $p.v$ at the positive pin p, the current $p.i$ into the positive pin p, the potential $n.v$ at the negative pin n, the current $n.i$ into the negative pin n, and the voltage u across the component.

Specific electrical two-pin components can now be defined as extensions of the $twoPin$ model. Unlike object-oriented modelling languages like Modelica, this is not accomplished through a class-based inheritance mechanism. FHM instead takes advantage of the first-class status of signal relations and uses a more direct approach based on *signal relation application*, denoted by $\diamond$. The idea is simple: the equations describing the applied signal relation is simply copied into the context of the application, substituting the expressions the relation was applied to for its formal arguments (renaming as needed to avoid name clashes). Additional equations for describing component-specific behaviour are then added. For example, a resistor, an inductor, and a capacitor can be defined as follows:

$$
\begin{aligned}
&resistor :: Resistance \rightarrow SR\ (Pin, Pin)\\
&resistor\ r = \mathbf{sigrel}\ (p, n)\ \mathbf{where}\\
&\quad twoPin \diamond (p, n, u)\\
&\quad r * p.i = u\\
&inductor :: Inductance \rightarrow SR\ (Pin, Pin)\\
&inductor\ l = \mathbf{sigrel}\ (p, n)\ \mathbf{where}\\
&\quad twoPin \diamond (p, n, u)\\
&\quad l * \mathbf{der}\ p.i = u\\
&capacitor :: Capacitance \rightarrow SR\ (Pin, Pin)\\
&capacitor\ c = \mathbf{sigrel}\ (p, n)\ \mathbf{where}\\
&\quad twoPin \diamond (p, n, u)\\
&\quad c * \mathbf{der}\ u = p.i
\end{aligned}
$$

Note that the above definitions exploit the first-class status of signal relations in another way as well: in order to parametrise the component models on the component values, the components are modelled by *functions* that *return* signal relations when applied to some specific component value. For example, $resistor$ is a function that when applied to a specific resistance will return a signal relation defined by three equations: two that originates from $twoPin$, and one which is Ohm's law instantiated with a specific value of the resistance. Since the parameters (e.g. r of $resistor$) are normal function arguments, *not* variables denoting signals (like u or $p.i$), their values remain unchanged throughout the lifetime of the returned relations. (In Modelica terms, they are **parameter**-variables.) As signal relations are first class entities, signal relations can even be parametrised on other signal relations in the same way.

To assemble these components into the full model, a Modelica-inspired **connect**-notation is used as a convenient abbreviation for connection equations. In FHM, this is just syntactic sugar that is expanded to basic equations: equality constraints for connected potential quantities and a sum-to-zero equation for connected flow quantities. In the following, **connect** is only applied to Pin records, where the voltage field is declared as a potential quantity whereas the current field is declared as a flow quantity.

We assume that a voltage source model $vSourceAC$ and a ground model $ground$ are available in addition to the component models defined above. We are only interested in the total current through the circuit. This is specified in a monitor construct. As there are no model inputs or outputs, the model becomes a *nullary* signal relation:

$$
\begin{aligned}
&simpleCircuit :: SR\ ()\\
&simpleCircuit = \mathbf{sigrel}\ ()\ \mathbf{where}\\
&\quad resistor\ 1000 \quad \diamond (r1p, r1n)\\
&\quad resistor\ 2200 \quad \diamond (r2p, r2n)\\
&\quad capacitor\ 0.00047 \diamond (cp, cn)\\
&\quad inductor\ 0.01 \quad \diamond (lp, ln)\\
&\quad vSourceAC\ 12 \quad \diamond (acp, acn)\\
&\quad ground \quad \diamond gp\\
&\quad \mathbf{connect}\ acp\ r1p\ r2p\\
&\quad \mathbf{connect}\ r1n\ cp\\
&\quad \mathbf{connect}\ r2n\ lp\\
&\quad \mathbf{connect}\ acn\ cn\ ln\ gp\\
&\quad \mathbf{monitor}\ r1p.i + r2p.i\ \mathbf{as}\ \texttt{"i"}
\end{aligned}
$$

There is no need to declare variables like $r1p$, $r1n$: their types are inferred. Note the signal relation expressions like $resistor\ 1000$ to the left of the signal relation application operators $\diamond$.

As an illustration of signal relation application, let us expand $resistor\ 1000 \diamond (r1p, r1n)$ using the definitions of $twoPin$ and $resistor$. The following are the three resulting equations, where $u1$ is a fresh name to avoid the name clash that would have ensued had the original name u been retained throughout:

$$
\begin{aligned}
r1p.n - r1n.v &= u1\\
r1p.i + r1n.i &= 0\\
1000 * r1p.i &= u1
\end{aligned}
$$

2.2 Dynamic Structure

To express structurally dynamic systems, FHM employs a switch construct that allows equations to be brought into and removed from a model as needed. Again, for illustrative purposes, we use an idealised syntax in the following as, at present, only the core switching primitive is implemented in Hydra, and using it directly is a little involved:

$$
\begin{aligned}
&\mathbf{initially}; \mathbf{when}\ condition_1 \Rightarrow\\
&\quad equations_1\\
&\mathbf{when}\ condition_2 \Rightarrow\\
&\quad equations_2
\end{aligned}
$$

The idea here is that only the equations from exactly one branch, what follows a $\Rightarrow$, are active at any one point in time. The equations of a branch are switched in whenever the condition guarding the branch *becomes* true, at which point the equations from the branch that was active previously are switched out. The special keyword **initially** designates the branch that initially

should be active. There can be any number of branches, exactly one of which must be designated as the initially active one, and each branch can be guarded by any number of conditions.

Whenever a switch occurs, continuous simulation stops, a new system of equations is generated, this is turned into simulation code, and continuous simulation can then resume. However, to resume simulation, the new system of equations must be properly initialised. This is a hard problem in general, and Hydra does not attempt to provide any generic, automatic solution. Instead, Hydra provides a couple of simple language constructs, *initialisation* and *reinitialisation equations*, that allow the modeller to explicitly express the modelling intent. While likely overly simplistic, these constructs have sufficed for our purposes thus far. We leave improvements as future work.

The semantics of initialisation and reinitialisation equations is as follows. An initialisation equation is only active at the moment the signal relation containing it is first instantiated. A reinitialisation equations becomes active whenever an already active instance of the signal relation containing it needs to be reactivated after a switch. Reinitialisation equations are typically used to express continuity assumptions. For example, the charge of a capacitor should persist across a switch, a fact that can be expressed as a continuity assumption on the voltage across it. Thus we get the following refined model of a capacitor that behaves as desired at switches and additionally can can be initialised:

$$
\begin{aligned}
&\mathit{iCapacitor} :: \mathit{Voltage} \to \mathit{Capacitance} \\
&\qquad \to \mathit{SR}\ (\mathit{Pin}, \mathit{Pin}) \\
&\mathit{iCapacitor}\ u0\ c = \mathbf{sigrel}\ (p, n)\ \mathbf{where} \\
&\quad \mathbf{init} \quad u = u0 \\
&\quad \mathbf{reinit}\ u = \mathbf{pre}\ u \\
&\quad \mathit{twoPin} \diamond (p, n, u) \\
&\quad c * \mathbf{der}\ u = p.i
\end{aligned}
$$

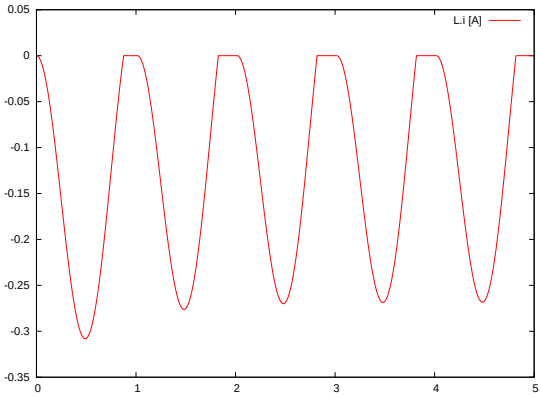
The operator **pre** is only defined in the context of reinitialisation equations. Semantically, it stands for the limit of a signal as time approaches the current point in time from the left. Thus an equation like $u = \mathbf{pre}\ u$ is genuinely a statement of continuity (assuming continuity to the right).

3 Simulation of the Half-Wave Rectifier

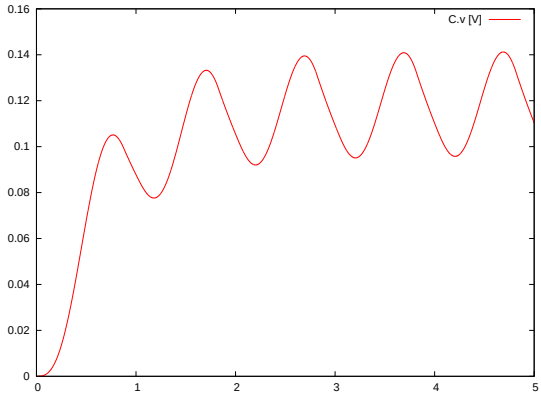
We are now in a position to define an ideal diode model in Hydra. Let us define a variant $\mathit{icDiode}$ that is initially closed:

$$
\begin{aligned}
&\mathit{icDiode} :: \mathit{SR}\ (\mathit{Pin}, \mathit{Pin}) \\
&\mathit{icDiode} = \mathbf{sigrel}\ (p, n)\ \mathbf{where} \\
&\quad \mathit{twoPin} \diamond (p, n, u) \\
&\quad \mathbf{initially}; \mathbf{when}\ p.v - n.v > 0 \Rightarrow \\
&\qquad u = 0 \\
&\quad \mathbf{when}\ p.i < 0 \Rightarrow \\
&\qquad p.i = 0
\end{aligned}
$$

Using the ideal diode model $\mathit{icDiode}$, a model of the circuit in Fig. 1(a) can be constructed as follows:



(a) Current through L



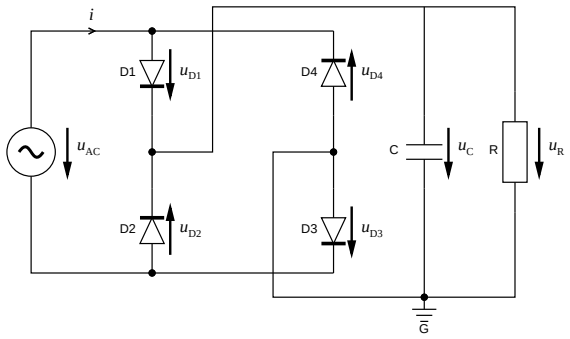
(b) Voltage across C

Fig. 3 Simulation results for the half-wave rectifier with in-line inductor.

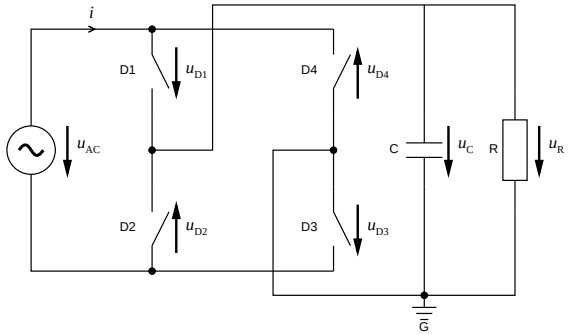
$$
\begin{aligned}
&\mathit{halfWaveRectifier} :: \mathit{SR}\ () \\
&\mathit{halfWaveRectifier} = \mathbf{sigrel}\ ()\ \mathbf{where} \\
&\quad \mathit{iInductor}\ 0.0\ 1.0 \quad \diamond (\mathit{lp}, \mathit{ln}) \\
&\quad \mathit{resistor}\ 1.0 \quad \diamond (\mathit{r1p}, \mathit{r1n}) \\
&\quad \mathit{resistor}\ 1.0 \quad \diamond (\mathit{r2p}, \mathit{r2n}) \\
&\quad \mathit{icDiode} \quad \diamond (\mathit{dp}, \mathit{dn}) \\
&\quad \mathit{iCapacitor}\ 0.0\ 1.0 \diamond (\mathit{cp}, \mathit{cn}) \\
&\quad \mathit{vSourceAC}\ 1.0\ 1.0 \diamond (\mathit{acp}, \mathit{acn}) \\
&\quad \mathit{ground} \quad \diamond \mathit{gp} \\
&\quad \mathbf{connect}\ \mathit{acp}\ \mathit{lp} \\
&\quad \mathbf{connect}\ \mathit{ln}\ \mathit{r1n} \\
&\quad \mathbf{connect}\ \mathit{r1n}\ \mathit{dp} \\
&\quad \mathbf{connect}\ \mathit{dn}\ \mathit{cp}\ \mathit{r2p} \\
&\quad \mathbf{connect}\ \mathit{acn}\ \mathit{cn}\ \mathit{r2n}\ \mathit{gp} \\
&\quad \mathbf{monitor}\ \mathit{lp}.i\ \mathbf{as}\ \texttt{"L.i"} \\
&\quad \mathbf{monitor}\ \mathit{cp}.v\ \mathbf{as}\ \texttt{"C.v"} \\
&\quad \mathbf{monitor}\ \mathit{ci}.v\ \mathbf{as}\ \texttt{"C.i"}
\end{aligned}
$$

This model thus corresponds to Fig. 1(b).

The simulation of this model turned out to be entirely straightforward. Fig. 3 shows the plots of the current through the inductor and the voltage across the capacitor for the first five seconds.



(a) Circuit



(b) Circuit modelled using an ideal diode model

Fig. 4 Full-wave rectifier

4 Simulation of the Full-Wave Rectifier

Simulation of the full-wave rectifier in Fig. 4(a) using ideal diodes as per Fig. 4(b) is much more challenging than simulating the half-wave rectifier. A key difficulty is that the circuit breaks down into two isolated halves when all diodes are open. The lack of a ground reference for the left part means the system becomes *under-determined* and it cannot be simulated.

However, a more detailed analysis reveals that this is as it should be as the model is *incomplete*: for the model to make sense, there is further tacit modelling knowledge that needs to be stated explicitly in the form of additional equations. If the diodes are truly ideal, this means that they are also *identical*, which in turn implies that the voltage drops over them are always going to be pairwise equal, even when they are open. That this must be so can be realised by replacing the diodes with high-impedance resistors in the open mode, each with the *same* resistance R, and then analysing the voltage drops across them as R is taken towards infinity.

The model of an ideal diode therefore needs to be refined to make the voltage drop across it part of its interface. For example, for the initially closed diode model:

$$
\begin{array}{l}
icDiode :: SR\ (Pin, Pin, Voltage) \\
icDiode = \mathbf{sigrel}\ (p, n, u)\ \mathbf{where} \\
\quad twoPin \diamond (p, n, u) \\
\quad \mathbf{initially}; \mathbf{when}\ p.v - n.v > 0 \Rightarrow \\
\quad\quad u = 0 \\
\quad \mathbf{when}\ p.i < 0 \Rightarrow \\
\quad\quad p.i = 0
\end{array}
$$

Once this has been done, the model for the full circuit can be described along the lines we saw in the previous section, except that two extra equations, stating the pairwise equality of the voltages across the diodes, are needed. That is:

$$u_{D1} = u_{D3} \quad (1)$$
$$u_{D2} = u_{D4} \quad (2)$$

However, adding Eq. (1) and Eq. (2) results in additional complications for simulation as the system now seemingly becomes *over-determined* when some diodes are closed. It turns out, though, that the system is only *trivially* over-determined in that the extra equations are mathematically equivalent to other equations in the system. This is easy to see: when a diode is closed, there is an equation provided by the model of the diode itself that states that the voltage across it is 0. If, for example, $D1$ and $D3$ are closed, we have:

$$u_{D1} = 0 \quad (3)$$
$$u_{D3} = 0 \quad (4)$$

But, additionally, u_{D1} and u_{D2} are related by Eq. (1) that is provided by model of the overall circuit.

In this case, a simple symbolic simplification pass suffice to eliminate the redundant equations in the modes where the diodes are pairwise closed. This simplification essentially amounts to constant propagation: using (3) and (4), (1) can be simplified to the trivially satisfied equation $0 = 0$ that then can be eliminated. After this the model can be simulated without further issues. Note that dynamic generation of equations followed by symbolic processing, as provided by FHM, is crucial to this approach to simulating ideal diodes. (At the time of writing, the symbolic simplification has not been fully integrated into the main branch of the Hydra implementation, meaning that some manual tweaking is still necessary.)

Additionally, it should be pointed out that the switch construct provides a bit of "inertia" in that the changes induced by an instance of a switch only concern equations originating from that switch instance: all other equations remain as they were. Thus, even though, in the case of ideal diodes, a circuit with n diodes has up to 2^n distinct structural configurations or modes, it is always entirely clear which mode to move to after a switch; there is no need to search among the up to 2^n possibilities for a consistent successor mode. As to reinitialisation after switches, static continuity assumptions through the mechanism of **reinit** equations sufficed in this case.

As a result, we have obtained a model of an ideal full-wave rectifier that is constructed modularly from individual, reusable components. The proper behaviour emerges from simply assembling the components, with just some minor additional guidance from the modeller in the form of a couple of extra equations. There is no

need for any heavyweight, auxiliary mechanisms, such as an explicit finite state machine, to control how the model moves between structural configurations.

5 Related Work

Sol is a Modelica-like language [16, 4, 17]. It introduces language constructs that enable the description of systems where objects are dynamically created and deleted, thus aiming at supporting modelling of highly structurally dynamic systems. Sol is capable of handling a range of structurally dynamic models, including the ideal half-wave rectifier with an in-line inductor [17]. However, the published work on Sol thus far does not consider any full-wave rectifier. The Sol research emphasis has been on the design of the language itself, along with support for incremental dynamic recausalisation and dynamic handling of structural singularities. An interpreter is used for simulation. The work on Sol is thus complementary to ours: techniques for dynamic compilation would be of interest in the context of Sol to enable it to target high-end simulation tasks; conversely, algorithms for incremental recausalisation is of interest to us to minimise the amount of work needed to regenerate simulation code after structural changes.

MOSILAB is an extension of the Modelica language that supports the description of structural changes using object-oriented statecharts [11]. This enables modelling of structurally dynamic systems. It is a compiled implementation. However, the statechart approach implies that all structural modes must be explicitly specified in advance, meaning that MOSILAB does not support *highly* structurally dynamic systems. Even so, if the number of possible configurations is large, techniques like those we have investigated here might be of interest also in the implementation of MOSILAB.

The Modelling Kernel Language (MKL) [18, 15] is intended to be a core language, with a formally defined semantics, for non-causal modelling languages such as Modelica. MKL takes a functional approach to non-causal modelling, where both functions and non-causal models are first-class entities. This enables higher-order, non-causal modelling. There are thus many similarities between MKL and FHM, leading to a similar style of modelling in both settings. Thus far, the work on MKL has not specifically considered support for structural dynamism. However, if one wanted to do this, the implementation techniques discussed in this paper should be of interest.

6 Future Work and Concluding Remarks

This paper has demonstrated how the structural dynamism available in FHM can be exploited to simulate ideal diodes in a straightforward manner. Both a half-wave rectifier with an in-line inductor and a full-wave rectifier were considered as test cases. To make the latter work, the diode model had to be somewhat refined to make the consequences of the assumption of ideal and thus identical diodes manifest. However, we believe this is natural and reasonable: key modelling knowledge should always be expressed explicitly. Thus, in both cases, the models are essentially completely modular, constructed from individual, reusable components and with a behaviour that emerges from the way they are assembled, with only minor additional guidance from the modeller.

The adopted modelling approach did lead to a structurally dynamic system of equations that in specific modes was formally over-determined. But, as the extra equations essentially were copies of other equations in the model, these redundant equations could be eliminated by symbolic simplification pass, a pass that is useful in any case, for ensuring that the generated code is as efficient as possible, and possibly for dealing with other problems related to over-determined systems of equations.

This said, it should be emphasised that what we have presented is a preliminary investigation. Specifically:

- Our FHM implementation Hydra is currently being developed. While what we have described in this paper has been tested and does work, a lot of work remains to provide a palatable surface syntax for Hydra and to properly integrate the various techniques we have developed into a whole.
- The symbolic elimination of redundant equations is somewhat opaque as exactly what kind of redundant equations can be eliminated is not obvious. Ideally an exact characterisation of what kind of redundant equations are going to be eliminated should be developed. Maybe additional language constructs should be introduced to make it manifest what equations could become redundant.
- Proper, pairwise switching of the diodes in the full-wave rectifier is currently predicated on the quality of the simulation back-end: there is no *guarantee* that the diode pairs actually will switch in synchrony, as they should under ideal circumstances, or that the configuration where all diodes are closed, thus shorting the voltage source, will not be entered. Language constructs that would enable a modeller to explicitly express assumptions of discrete events occurring synchronously could make models more robust. In effect, such constructs would amount to equations expressing *temporal* constraints among switching conditions.

Acknowledgement

This work was supported by EPSRC grant EP/D064554/1. The authors wish to thank the anonymous referees for useful suggestions that helped improve the paper.

7 References

[1] The Modelica Association. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.2*, March 2010.

[2] Pieter J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In Fritz W. Vaadrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control '99*, number 1569 in Lecture Notes in Computer Science, pages 165–177, 1999.

[3] Günther Zauner, Daniel Leitner, and Felix Breitenecker. Modelling structural-dynamics systems in Modelica/Dymola, Modelica/MOSILAB, and AnyLogic. In Peter Fritzson, François Cellier, and Christoph Nytsch-Geusen, editors, *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 24 in Linköping Electronic Conference Proceedings, pages 99–110, Berlin, Germany, 2007. Linköping University Electronic Press.

[4] Dirk Zimmer. Introducing Sol: A general methodology for equation-based modeling of variable-structure systems. In *Proceedings of the 6th International Modelica Conference*, pages 47–56, Bielefeld, Germany, 2008.

[5] The Modelica Association. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial version 1.4*, December 2000.

[6] François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag, 2006.

[7] Henrik Nilsson, John Peterson, and Paul Hudak. Functional hybrid modeling. In *Proceedings of PADL'03: 5th International Workshop on Practical Aspects of Declarative Languages*, volume 2562 of *Lecture Notes in Computer Science*, pages 376–390, New Orleans, Lousiana, USA, January 2003. Springer-Verlag.

[8] Henrik Nilsson, John Peterson, and Paul Hudak. Functional hybrid modeling from an object-oriented perspective. *Simulation News Europe*, 17(2):29–38, September 2007.

[9] George Giorgidze and Henrik Nilsson. Higher-order non-causal modelling and simulation of structurally dynamic systems. In *Proceedings of the 7th International Modelica Conference*, Linköping Electronic Conference Proceedings, pages 208–218, Como, Italy, September 2009. Linköping University Electronic Press.

[10] Chris Lattner. LLVM: An infrastructure for multi-stage optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL, December 2002. See `http://llvm.org`.

[11] Christoph Nytsch-Geusen, Thilo Ernst, André Nordwig, Peter Schwarz, Peter Schneider, Matthias Vetter, Christof Wittwer, Thierry Nouidui, Andreas Holm, Jürgen Leopold, Gerhard Schmidt, Alexander Mattes, and Ulrich Doll. MOSILAB: Development of a Modelica-based generic simulation tool supporting model structural dynamics. In *Proceedings of the 4th International Modelica Conference*, pages 527–535, Hamburg, Germany, 2005.

[12] Conal Elliott and Paul Hudak. Functional reactive animation. In *Proceedings of ICFP'97: International Conference on Functional Programming*, pages 163–173, June 1997.

[13] Zhanyong Wan and Paul Hudak. Functional reactive programming from first principles. In *Proceedings of PLDI'01: Symposium on Programming Language Design and Implementation*, pages 242–252, June 2000.

[14] Henrik Nilsson, Antony Courtney, and John Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (Haskell'02)*, pages 51–64, Pittsburgh, Pennsylvania, USA, October 2002. ACM Press.

[15] David Broman and Peter Fritzson. Higher-order acausal models. In Peter Fritzson, Fran cois Cellier, and David Broman, editors, *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 29 in Linköping Electronic Conference Proceedings, pages 59–69, Paphos, Cyprus, 2008. Linköping University Electronic Press.

[16] Dirk Zimmer. Enhancing Modelica towards variable structure systems. In Peter Fritzson, François Cellier, and Christoph Nytsch-Geusen, editors, *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 24 in Linköping Electronic Conference Proceedings, pages 61–70, Berlin, Germany, 2007. Linköping University Electronic Press.

[17] Dirk Zimmer. *Equation-Based Modeling of Variable-Structure Systems*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 2010.

[18] David Broman. Flow Lambda Calculus for declarative physical connection semantics. Technical Reports in Computer and Information Science 1, Linköping University Electronic Press, 2007.