A Reduced Index Mode-Independent Structure Model Transformation for Multimode Modelica Models

Benoît Caillaud[1] Mathias Malandain[1] Albert Benveniste[1]

[1]Inria Centre de Rennes Bretagne Atlantique, University of Rennes 1, France,
{benoit.caillaud,mathias.malandain,albert.benveniste}@inria.fr

Abstract

Since its 3.3 release, Modelica offers the possibility to specify models of dynamical systems with multiple modes having different DAE-based dynamics. However, the handling of such models by the current Modelica tools is not satisfactory, with mathematically sound models yielding exceptions at runtime. In this article, we propose a systematic way of rewriting a multimode Modelica model, based on the results of an already implemented multimode structural analysis. The rewritten Modelica model is guaranteed to be correctly compiled by state-of-the-art Modelica tools. Simulation results are presented on a simple, yet meaningful, physical system whose original Modelica model is not correctly handled by state-of-the-art Modelica tools.

Keywords: Modelica, multimode DAE, structural analysis, model transformations

1 Introduction

Since version 3.3, the Modelica language offers the possibility of specifying *multimode dynamics,* by describing state machines with different DAE dynamics in each different state (Elmqvist et al. 2012). This feature enables describing large complex cyber-physical systems with different behaviors in different modes.

While being undoubtedly valuable, multimode modeling has been the source of serious difficulties for non-expert users of the current generation of Modelica tools. Indeed, while many large-scale Modelica models are properly handled, some physically meaningful models do not result in correct simulations with most Modelica tools. It is actually not difficult to construct such problematic models, thus, chances are significant to produce such bad cases in large models. Quite often, end users have to ask Modelica experts, or even tool developers themselves, to tweak their models in order to make them work as expected. This situation hinders a wider spreading of Modelica tools among a larger class of users, such as Simulink-trained engineers.

New language constructs have been proposed in the past to address the limited capability of the Modelica language to handle multimode models. The Sol (Zimmer 2010) and the Hydra (Giorgidze and Nilsson 2011; Nilsson and Giorgidze 2010) languages have been designed with the capability to enable and disable equations, depending on the current mode of the system. For both languages, structural analysis is performed at runtime, when the system switches to a new mode.

Some years ago, we started a project aiming at addressing all the above issues, with a different perspective in mind, that consists in privileging compile-time, rather than runtime, analyses. In (Benveniste, Caillaud, Elmqvist, et al. 2019; Benveniste, Caillaud, and Malandain 2020) we explain our approach, and we illustrate it on two simple, yet physically meaningful, examples in (Benveniste, Caillaud, and Malandain 2021). One key feature of this approach is structural analysis: it is important that this task is performed for each mode and each mode change at compile time, in order to avoid unexpected behaviour at runtime. In (Caillaud, Malandain, and Thibault 2020), we present an effective approach to achieve compile-time, mode-dependent, structural analysis *without enumerating the modes* (as this would not be able to scale up). The advantages we see in our approach are twofold: (i) it provides, at compile-time, invaluable information that helps users debug their models, and (ii) efficient code generation is possible since the automatic differentiation of latent equations can be done at compile-time and blocks of equations can be compiled into functions that can be passed directly to numerical solvers, without any further processing.

In this article, we demonstrate how the results of this multimode structural analysis can be used for transforming a multimode Modelica model into its RIMIS (Reduced Index Mode-Independent Structure) form, which is guaranteed to yield correct execution on state-of-the-art Modelica tools. This method is illustrated on a water tank model for which current Modelica tools fail to execute; in this model, the differentiation index depends on the mode, which is a problem for these tools. In particular, we explain how existing structural analysis methods fail to yield correct execution code for this model, then demonstrate the generation of a target code under RIMIS form, resulting in a correct simulation of the model. Our approach is then formalized for its broad application to problematic multimode models.

```
model WaterTank
  Real t(start=0,fixed=true); // time (to define input flow)
  constant Real xmax = 1.0; // max water quantity
  constant Real xmin = 0.0; // min water quatity
  constant Real y0 = 6.667; // default output flow
  constant Real rho = 0.8; // input flow parameter
  Real x(start=0.5,fixed=true); // stored water mass
  Real yh; // output flow correction, when tank is full
  Real yl; // output flow correction, when tank is empty
  Real z; // input flow
  Real sh; // parameter of the full-tank CC
  Real sl; // parameter of the empty-tank CC
  Boolean bh(start=false,fixed=true); // mode full-tank
  Boolean bl(start=false,fixed=true); // mode empty-tank
  // bh and bl satisfy assertion not (bh and bl)
equation
  // input flow law
  /* et: */ der(t)=1;
  /* e1: */ z = rho*y0*(1+
            Modelica.Math.cos(2*Modelica.Constants.pi*t));
  // tank level differential equation
  /* e2: */ der(x) = z + yl - yh - y0;
  // Complementarity condition 0 <= xmax - x # yh >= 0
  bh = (sh >= 0);
  /* eh1: */ sh = if bh then yh else x - xmax;
  /* eh2: */ 0 = if bh then x - xmax else yh;
  // complementarity condition 0 <= x - xmin # yl >= 0
  bl = (sl >= 0);
  /* el1: */ sl = if bl then yl else xmin - x;
  /* el2: */ 0 = if bl then xmin - x else yl;
end WaterTank;
```

Figure 1. Modelica model of the Water Tank system. Comments of the form `/* id: */` define equation labels appearing in the dependency graphs in Figures 3 and 4.

2 The Water Tank system and failed simulations with Modelica tools

The Water Tank system is a simple model of a closed tank with a variable water inflow `z` and a default outflow `y0`, where water is considered incompressible. When the tank is full, a positive flow correction `yh` is added to the outflow, as the tank cannot store more water; conversely, when the tank is empty, a negative flow correction `yl` is added to the outflow.

The corresponding Modelica model, given in Figure 1, uses two complementarity conditions (Van Der Schaft and Schumacher 1998) for the flow corrections. The first one, encoded by the multimode equations `eh1` and `eh2`, depends on the Boolean variable `bh`, which is true if and only if variable `sh` is nonnegative. The combined effect of these two equations is that `xmax` − `x` and `yh` are always nonnegative, and that at least one of those is equal to 0 at any time. Equations `el1` and `el2` encode the second complementarity condition in a similar way.

This model fails to simulate properly with both OpenModelica 1.17.0 (Fritzson et al. 2020) and Dymola 2021 (Dassault Systèmes AB 2020); Figure 2 shows the output of Dymola 2021. The root cause is that state-of-the-art Modelica tools perform an approximate structural analysis, disregarding the fact that the structure of the system is mode-dependent. A more detailed explanation is provided in Section 3.1.

```
Log-file of program ./dymosim
(generated: Wed Apr  7 16:56:34 2021)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "SimpleTankComplementarity.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dassault Systemes))
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0

On the final iteration for restart conditions we get:
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0

ERROR: Finding consistent restart conditions failed at time:  0.1576506697732894

Integration terminated before reaching "StopTime" at T = 0.158
```

Figure 2. Simulation of the Water Tank system with Dymola 2021, failing with a division by zero exception.

3 Structural analysis: from single- to multi-mode

DAE-based languages and tools rely on *structural analysis* as a required preprocessing step of a DAE system, needed for the generation of simulation code. This analysis turns the original system into a *reduced index* (Campbell and Gear 1995) system, amenable to numerical solvers, by differentiating one or several times all or part of the equations.

Well-understood methods such as the renowned Pantelides algorithm (Pantelides 1988), the dummy derivatives method (Mattsson and Soderlind 1993) or the less known Σ-method (Pryce 2001) can be used for single-mode DAE systems; however, the structural analysis of multimode DAE systems is still in its infancy, and even state-of-the-art Modelica tools have to rely, at least in part, on an approximate 'single-mode' structural analysis for the generation of simulation code from multimode models.

We show how the use of such single-mode methods can lead to the runtime errors observed on the Water Tank model shown above. We then introduce the exact multimode structural analysis performed by the IsamDAE tool, which will be used for the transformation of multimode models at the core of this article.

3.1 Approximate structural analysis

Structural analysis of a DAE system only relies on the knowledge of which numerical variables appear in which equations. As such, an approximate structural analysis of a multimode DAE system can be performed by abstracting away all mode dependencies inside the equations; for instance, an equation `x = if cond then y else z` will be regarded by the approximate structural analysis as an equation involving variables `x`, `y` and `z`.

Such an analysis of the Water Tank model shown in Figure 1 results in the decomposition shown in Figure 3. In this decomposition, equation `eh2` has to be solved for the variable `yh`.

When performing the pivoting of this equation, mode

DOI
10.3384/ecp21181519

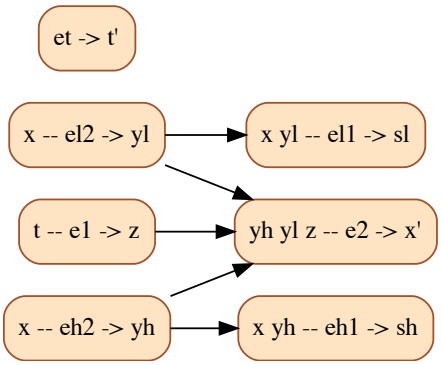

Figure 3. Dependency graph resulting from the approximate structural analysis of the Water Tank model. Vertices are equation blocks of the form $R - E \rightarrow W$, where: E is the block of equations; R is a set of variables to read (they are free variables, i.e., parameters of the block of equations); and W is a set of variables to write (they are the unknowns of the block of equations). When R is empty, the shorthand notation is $E \rightarrow W$. Edges express causal dependencies, meaning that a block can be solved only after all its predecessors have been solved.

dependencies have to be taken into account again. Equation `eh2` reads:

$$0 = \texttt{if bh then x} - \texttt{xmax else yh}$$

which can be rewritten as an equation of the form $0 = \texttt{a yh} + \texttt{b}$ where `a` and `b` are mode-dependent:

$$\begin{aligned} 0 =& (\texttt{if bh then 0 else 1}) \times \texttt{yh} \\ &+ (\texttt{if bh then x} - \texttt{xmax else 0}) \end{aligned}$$

Unknown `yh` can finally be isolated:

$$\texttt{yh} = -\frac{\texttt{if bh then x} - \texttt{xmax else 0}}{\texttt{if bh then 0 else 1}} \qquad (1)$$

This technique may be used for the generation of simulation code, but in this case, a problem is bound to occur when Boolean variable `bh` is `true`. As a matter of fact, equation (1) is exactly the equation responsible for the division by zero exception shown in Figure 2, which occurs at the initial time, when `bh` is `true`.

3.2 Exact multimode structural analysis

The IsamDAE[1] tool (Caillaud, Malandain, and Thibault 2020) has been used to perform a multimode structural analysis of the model, resulting in the Conditional Dependency Graph (CDG) shown in Figure 4.

Remark that the differentiation index of the system is mode-dependent. For instance, equation `el2` is used differentiated, to compute the derivative of `x`, when `bl` is `true`, while it is kept undifferentiated, to compute `yl`, when `bl` is `false`. Also notice that equation `eh2` is no longer used to compute `yh` in all modes, but only when `bh` is `false`, thus preventing the runtime error explained above.

[1] `https://team.inria.fr/hycomes/software/isamdae/`

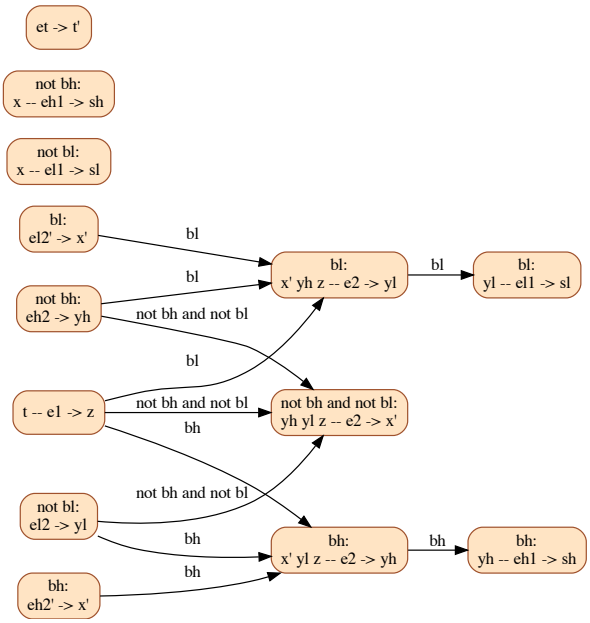

Figure 4. Conditional Dependency Graph resulting from the multimode structural analysis of the Water Tank model. Vertices are conditional equation blocks of the form $p : R - E \rightarrow W$, where: E is the block of equations; p is a Boolean condition, defining the set of modes in which the block has to be solved; R is a set of variables to read, or free variables, i.e., parameters of the block of equations; and W is a set of variables to write, meaning that they are the unknowns of the block of equations. When R is empty, the shorthand notation is $p : E \rightarrow W$. When p is the proposition `true`, it is omitted, and the notation becomes: $R - E \rightarrow W$, or $E \rightarrow W$. Edges express causal dependencies, meaning that a block can be solved only after all its predecessors have been solved. They are labeled by Boolean conditions, characterizing the modes in which the dependency applies.

We shall see next how the CDG (Figure 4) can be used to transform the model into an equivalent one, that triggers no runtime error when using Modelica tools based on an approximate structural analysis.

4 A Reduced Index Mode-Independent Structure (RIMIS) form

Using multimode structural analysis to transform a multimode Modelica model into a reduced-index model, that simulates correctly with state-of-the-art Modelica tools, is made difficult by the fact that the Modelica language does not permit to enable or disable an equation depending on the mode. Based on this limitation, the basic principle of our model transformation is to evaluate all equation blocks of the CDG in a mode-independent fashion, irrespectively of the mode in which the system is. Of course, this leads to useless computations during simulation. However, this

turns out to be a systematic way to ensure a correct simulation of multimode Modelica models.

The method proposed in this paper is detailed below, in informal terms, then illustrated on a simple example. A mathematical definition of the transformation is detailed in Section 6. Remark that models with initial equations, `when` or `reinit` statements are not covered in this paper. Also note that models with non-scalar variables or class instances of any kind are not considered here. It is assumed that the models have been flattened according to the procedure described in Chapter 5 of the Modelica Language Specification (The Modelica Association 2021). Because of a current restriction of the IsamDAE software, mode variables are assumed to be of type Boolean.

4.1 The RIMIS form transformation

The method decomposes in the following seven steps:

1. **Conditional Dependency Graph:** The CDG of the source model is computed by the multimode structural analysis method. This graph defines a block-triangular decomposition of the reduced-index system, for each mode of the system. It will be used throughout the transformation.

2. **Source Variables:** Variable declarations are copied unchanged, with the exception of real variables, whose initialization parts are removed.

3. **Replicate and Dummy Derivative Variables:** For each block of the CDG, replicates of written variables (unknowns) are declared. Whenever an unknown appears differentiated, a dummy derivative variable (Mattsson and Soderlind 1993) is declared. Initialization statements for state variables are copied from the source model. As an optional optimization, non-leading replicate variables can be factored among a disjunction of modes, in order to decrease the number of variables in the resulting model.

4. **Mode Equations:** Equations defining mode variables are copied unchanged. For the sake of simplicity, these equations are assumed to be of the form $\mathtt{b} = (expr >= 0)$, where *expr* is a real expression.

5. **Replicate and Dummy Equations:** Equations are replaced with replicates, according to the following principle:

   For each block in the CDG, equations appearing in this block are replicated, substituting (i) every written variable (unknown of the block) by the replicate declared in step 3, and (ii) every read variable (parameter of the block) by the corresponding replicate, if it is a leading variable. Both mode variables and read state variables are left unchanged.

   As a result, the single-mode structural analysis of the resulting equation system yields a block-triangular decomposition that contains all the blocks of the CDG obtained by the multimode structural analysis of the original model.

   For each equation in the fresh model, the propositional formula conditioning the block in which this equation appears can be taken into account: a partial evaluation of the equation is performed (Jones, Gomard, and Sestoft 1993). This has the effect of simplifying the equation, by eliminating some of the conditionals (`if` ... `then` ... `else` ... operators).

   Note that the resulting equations may still be multimode: in general, not all conditionals can be eliminated by partial evaluation. However, the fact that the structure of the resulting equations is independent of the mode is still guaranteed: the multimode structural analysis ensures that each equation block has the same structure (in particular, the same read and written variables) in all the modes in which it is defined, even if one or several of its equations contain conditional statements.

   First-order differential equations are also added in accordance to the dummy derivatives method.

6. **Multiplexing Equations:** In order to retrieve the values of the source model variables from the replicates in the fresh model, mutiplexing equations have to be added. These are multimode equations, containing conditional operators, but these equations contain no dynamics: each multiplexing equation focuses on a source model variable that corresponds to several replicates in the transformed model, specifying which of the latter currently holds the value of the former.

7. **Reinitializations:** Reinitialization statements finally have to be inserted, in order to reset replicate variables that are state variables to a correct value upon the occurrence of a mode switching. Therefore, these statements are triggered by mode changes.

4.2 Transformation of a simple model

We illustrate the method on a simplistic, yet relevant, two equations model:

```
model TwoEquations
  Real x(start=0,fixed=true);
  Boolean p(start=false,fixed=true);
equation
  p = (x >= 1);
  1 = if p then x else der(x);
end TwoEquations;
```

This model has one real equation, one Boolean equation, and no particular physical meaning. However, it captures in a nutshell the difficulty raised with the Water Tank system. As a matter of fact, the CDG (Figure 5) resulting from the multimode structural analysis distinguishes between two cases:

DOI 10.3384/ecp21181519

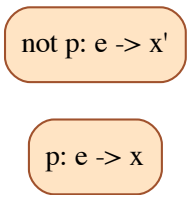

Figure 5. CDG of the Two Equations model.

```
Log-file of program ./dymosim
(generated: Wed May  5 08:49:35 2021)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "OneEquation.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dassault Systemes))
Error: The following error was detected at time: 1.000000000000786
Error: Singular inconsistent scalar system for der(x) = ((if p then x else 0.0)-1)/
                                    ( -(if p then 0.0 else 1.0)) = 7.86482e-13/-0
Integration terminated before reaching "StopTime" at T = 1
```

Figure 6. Failed simulation of the Two Equations model with Dymola 2021.

- when p is `true`, x is a leading variable, meaning that it is the unknown that needs to be solved;
- when p is `false`, the leading variable is $\mathtt{x}'$, the first-order time derivative of x, while x itself is a state variable.

The approximate structural analysis of both Dymola and OpenModelica determines that the leading variable is $\mathtt{x}'$ in all modes; however, the real equation is singular in $\mathtt{x}'$ when p is `true`. Unsurprisingly, an exception is raised during simulation, as shown in Figure 6.

Let us apply the transformation one step after the other:

1. The **CDG graph** of the source model is shown in Figure 5.
2. **Declarations** of variables x and p are copied.

   ```
   Real x;
   Boolean p(start=false,fixed=true);
   ```

   Remark that the declaration of x has been stripped of its initialization part.
3. **Replicate variables** are created according to the two blocks of the CDG. Two leading replicate variables x_2 (holding the value of x if p holds) and x_p_3 (holding the value of $\mathtt{x}'$ if `not p` holds), and one state replicate variable x_3 that is meaningful only if `not p` holds, are declared.

   ```
   Real x_2;
   Real x_p_3;
   Real x_3(start=0,fixed=true);
   ```

   Note that the initialization of variable x in the source model is copied here, to initialize the replicate state variable x_3.
4. One **mode equation** is copied from the source model.

   ```
   p = (x >= 1);
   ```
5. **Replicate equations** are generated from the CDG, which has two blocks of one equation each.

   From the block $\mathtt{p} : \mathtt{e} \rightarrow \mathtt{x}$, one replicate equation is generated by replacing variable x with its replicate x_2, then performing the partial evaluation (Jones, Gomard, and Sestoft 1993) under the assumption that the Boolean condition p holds.

   ```
   // Block e_2 -> x_2
   /* e_2 : */ 1 = x_2;
   ```

   From the second block $\mathtt{not\ p} : \mathtt{e} \rightarrow \mathtt{x}'$, one replicate equation is generated in a similar way.

   ```
   // Block e_3 -> x_p_3
   /* e_3 : */ 1 = x_p_3;
   ```

   A differential equation is also generated, linking replicate variable x_3 with its dummy derivative x_p_3.

   ```
   der(x_3) = x_p_3;
   ```
6. One **multiplexing equation** is generated, to be solved for variable x.

   ```
   x = if p then x_2 else x_3;
   ```
7. Finally, the only case in which a state variable has to be **reinitialized** is when entering the mode `not p`. The value of replicate variable x_3 is then set to be the left limit of x.

   ```
   when not p then
     reinit(x_3,pre(x));
   end when;
   ```

The complete RIMIS form of the Two Equations model is given in Figure 7. The result of the successful simulation of this model is shown in Figure 8. Remark that the mode switching from $\mathtt{p} = \mathtt{false}$ to $\mathtt{p} = \mathtt{true}$ is correct, and that the reinitialization statement is never evaluated, as p remains `true` forever after time $\mathtt{t} = 1$.

5 Successful simulations of the Water Tank system in RIMIS form

The RIMIS transformation is illustrated on the Water Tank model (Figure 1); the resulting model is shown in Figure 9. Simulation results obtained with Dymola 2021 are shown in Figure 10. It can be seen that the simulation is successful, with a correct behavior of the Water Tank system, while the simulation of the original model failed (Figure 2). A correct simulation has also been obtained with OpenModelica 1.17.0 (Fritzson et al. 2020), under the provision that the Newton solver is used instead of the KINSOL nonlinear solver.

```
model TwoEquations_rimis
  // Source variables
  Real x;
  Boolean p(start=false,fixed=true);
  // Replicate variables
  Real x_2;
  Real x_p_3;
  Real x_3(start=0,fixed=true);
equation
  // Mode equations
  p = (x >= 1);
  // Differential equations
  der(x_3) = x_p_3;
  // Multiplexing
  x = if p then x_2 else x_3;
  // Block e_3 -> x_p_3
  /* e_3 : */ 1 = x_p_3;
  // Block e_2 -> x_2
  /* e_2 : */ 1 = x_2;
  // Replicate reinitializations
  when not p then
    reinit(x_3,pre(x));
  end when;
end TwoEquations_rimis;
```

Figure 7. Two Equations model in RIMIS form.

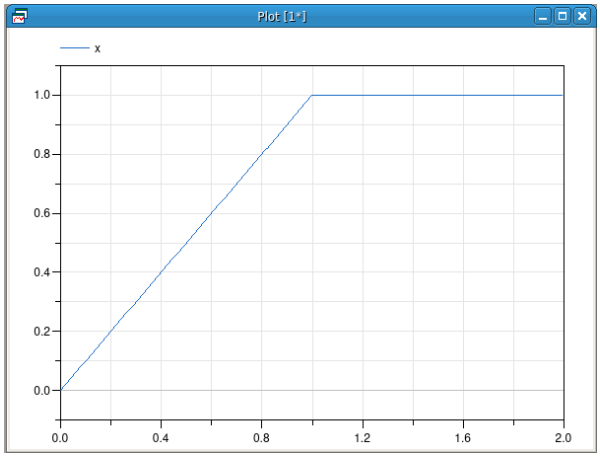

Figure 8. Simulation of the Two Equations model in RIMIS form with Dymola 2021.

6 Formalizing the RIMIS form transformation

The mathematical definition of the RIMIS form transformation relies on the partial evaluation of equations. Once variable renaming is also properly defined, the seven-step transformation mentioned in Section 4.1 is formalized. Finally, an optimization aiming at reducing the transformed model is presented.

6.1 Partial evaluation of expressions and equations

Partial evaluation is an umbrella name for a set of program transformation techniques that aim at specializing a program by taking into account prior knowledge on its input data, possibly improving its performances (Jones, Gomard, and Sestoft 1993; Danvy, Glück, and Thiemann 1996).

In the context of the Modelica language, consider a Boolean expression q, and a real expression e. The partial evaluation of expression e, assuming q, is an expression $e' = \pi_q(e)$, such that q implies $e = e'$ and $\text{free}(e') \subseteq \text{free}(e)$, where $\text{free}(.)$ is the set of free variables appearing in an expression.

To define the partial evaluation operator π, and for the sake of clarity, we only consider the subset of the Modelica expression language defined by the following grammar, where p is a Modelica Boolean expression:

$$
\begin{array}{rcll}
e & ::= & c & \text{where } c \text{ is a constant} \\
 & | & e \text{ op } e & \text{where op} \in \{+,-,\star,\ldots\} \\
 & | & v & \text{where } v \text{ is an identifier} \\
 & | & v(e,\ldots e) & \\
 & | & \texttt{if } p \texttt{ then } e \texttt{ else } e &
\end{array}
$$

Given a Boolean expression q and a real expression e, the partial evaluation of e, assuming q, is defined by induction on the structure of e:

$$
\left\{
\begin{array}{lcl}
\pi_q(c) & \equiv & c \\
\pi_q(e_1 \text{ op } e_2) & \equiv & \pi_q(e_1) \text{ op } \pi_q(e_2) \\
\pi_q(v) & \equiv & v \\
\pi_q(v(e_1,\ldots e_n)) & \equiv & v(\pi_q(e_1),\ldots \pi_q(e_n)) \\
\pi_q(\texttt{if } p \texttt{ then } e_T \texttt{ else } e_F) & \equiv & \text{cond}_q(p,e_T,e_F)
\end{array}
\right.
$$

where

$$
\begin{array}{l}
\text{cond}_q(p,e_T,e_F) \equiv \\
\left|
\begin{array}{ll}
\pi_{q \texttt{ and } p}(e_T) & \text{if } q \texttt{ and not } p \\
 & \text{is unsatisfiable, else} \\
\pi_{q \texttt{ and not } p}(e_F) & \text{if } q \texttt{ and } p \\
 & \text{is unsatisfiable, else} \\
\texttt{if } r & \text{where } r \text{ is such that:} \\
\texttt{then } \pi_{q \texttt{ and } p}(e_T) & p \texttt{ and } q \text{ implies } r, \text{ and} \\
\texttt{else } \pi_{q \texttt{ and not } p}(e_F) & r \text{ implies } p \texttt{ or not } q
\end{array}
\right.
\end{array}
$$

In the above definition, condition r is not unique: whenever possible, it should be chosen such that it is more concise than p.

The extension of the partial evaluation operator to equations is straightforward:

$$\pi_q(e_{LHS} = e_{RHS}) \equiv \pi_q(e_{LHS}) = \pi_q(e_{RHS}) \ .$$

6.2 Variable renaming

Before moving to the formal definition of the RIMIS transformation, variable renaming must be defined, in order to declare replicate variables and transform equations into their replicates.

Given a Boolean expression p, an identifier v, and a differentiation order $n \geq 0$, the replicate of the n-th order derivative of v, under condition p, is the identifier $\rho_p^n(v)$.

DOI 10.3384/ecp21181519

```
model WaterTankRIMIS
  // Constants
  constant Real xmax = 1.0;
  constant Real xmin = 0.0;
  constant Real y0 = 6.667;
  constant Real rho = 0.8;
  // Variables
  Real t(start=0,fixed=true);
  Real x(start=0.5,fixed=true);
  Real yh;
  Real yl;
  Real z;
  Real sh;
  Real sl;
  Boolean bh(start=false,fixed=true);
  Boolean bl(start=false,fixed=true);
  // Dummy derivatives
  Real t_p;
  Real x_p;
  // Replicated algebraic variables
  Real sh_5; // sh if not bh
  Real sh_6; // sh if bh
  Real sl_2; // sl if not bl
  Real sl_4; // sl if bl
  Real x_p_4; // x' if bl
  Real x_p_7; // x' if not bh and not bl
  Real x_p_6; // x' if bh
  Real yh_5; // yh if not bh
  Real yh_6; // yh if bh
  Real yl_2; // yl if not bl
  Real yl_4; // yl if bl
equation
  // Boolean equations
  bh = (sh >= 0);
  bl = (sl >= 0);
  // Differential equations
  der(t) = t_p;
  der(x) = x_p;
  // Multiplexing equations
  yh = if bh then yh_6 else yh_5;
  yl = if bl then yl_4 else yl_2;
  sh = if bh then sh_6 else sh_5;
  sl = if bl then sl_4 else sl_2;
  x_p = if bh then x_p_6 else
          if bl then x_p_4 else x_p_7;
  // Block et -> t'
  t_p = 1;
  // Block not bh: x -- eh1 -> sh
  sh_5 = x - xmax;
  // Block not bl: x -- el1 -> sl
  sl_2 = xmin - x;
  // Block bl: el2' -> x'
  x_p_4 = 0;
  // Block not bh: eh2 -> yh
  yh_5 = 0;
  // Block x -- e1 -> z
  z = rho*y0*(1+
      Modelica.Math.cos(2*Modelica.Constants.pi*t));
  // Block not bl: el2 -> yl
  yl_2 = 0;
  // Block bh: eh2' -> x'
  x_p_6 = 0;
  // Block bl: x' yh z -- e2 -> yl
  yl_4 = y0 + x_p_4 + yh_5 - z;
  // Block not bh & not bl: yh yl z -- e2 -> x'
  x_p_7 = z + yl_2 - yh_5 - y0;
  // Block bh: x' yl z  -- e2 -> yh
  yh_6 = z + yl_2 - x_p_6 - y0;
  // Block bl: yl -- el1 -> sl
  sl_4 = yl_4;
  // Block bh: yh -- eh1 -> sh
  sh_6 = yh_6;
end WaterTankRIMIS;
```

Figure 9. The Water Tank system in RIMIS form.

The operator ρ is assumed to satisfy the following axioms:

$$\begin{array}{lll} (\textbf{Identity}) & \rho^0_{\texttt{true}}(u) = u & \\ (\textbf{Injectivity}) & \rho^n_p(u) = \rho^m_q(v) \text{ implies} & u = v \text{ and} \\ & & p \iff q \text{ and} \\ & & n = m \end{array}$$

Checking the equivalence of two Boolean expressions is, in general, a difficult problem. In this article, Boolean expressions that appear in conditional statements are restricted to propositional formulas only. Mode equations are restricted to the form $\mathrm{v} = (e >= 0)$, where e is an affine expression. Under these assumptions, equivalence checking can be done with BDDAPRON, a logico-numerical abstract domain library (Jeannet 2012) combining BDDs (Boolean Decision Diagrams) (Bryant 1986) and polyhedra (Schrijver 1998). Such a use of BDDAPRON is considered, among other program analyses, in Chapter 7 of (Schrammel 2012).

6.3 Formal definition of the RIMIS form transformation

Consider a Modelica model M that can be decomposed in the following parts:

$$M \equiv \text{MD} \uplus \text{RD} \uplus \text{RI} \uplus \text{ME} \uplus \text{RE}$$

where:

- MD is the set of **m**ode (Boolean) variable **d**eclarations and initializations;
- RD is the set of **r**eal variable **d**eclarations, *stripped of their initializations*;
- RI is the set of **r**eal variable **i**nitializations;
- ME is the set of **m**ode variable **e**quations;
- RE is the set of **r**eal **e**quations.

Remark that models with `when` and `reinit` statements are not covered by the RIMIS form transformation, as this

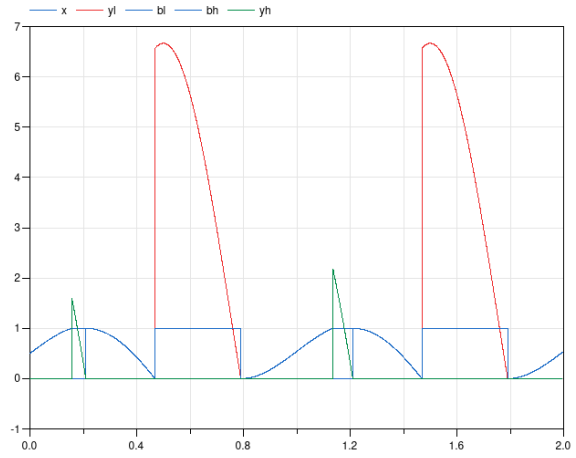

Figure 10. Simulation of the Water Tank system in RIMIS form with Dymola 2021.

would require a multimode structural analysis of mode changes (Benveniste, Caillaud, and Malandain 2020), that is not yet implemented in the IsamDAE software (Caillaud, Malandain, and Thibault 2020). Because of a current restriction of IsamDAE, mode variables are assumed to be Boolean.

Model M is assumed to be structurally nonsingular in all modes. Its CDG computed by the multimode structural analysis (Caillaud, Malandain, and Thibault 2020) consists in a set of blocks of equations and a set of directed edges between blocks; let Blocks and Edges denote the corresponding sets. A block $b \in \text{Blocks}$ consists of four parts:

- $\text{cond}(b)$, a Boolean expression;
- $\text{Eqs}(b)$, a set of equations, possibly differentiated;
- $\text{Read}(b)$, a set of read variables (parameters of the block of equations);
- $\text{Write}(b)$, a set of written variables (unknowns of the block of equations).

Elements of $\text{Eqs}(b)$ are pairs of the form $(0 = e, k)$, where e is an expression and $k \geq 0$ is a differentiation order. Elements of $\text{Read}(b)$ and $\text{Write}(b)$ are pairs of the form (u,k), where u is an identifier and $k \geq 0$ is a differentiation order. An edge $g \in \text{Edges}$ consists of three parts:

- $\text{cond}(g)$, a Boolean expression;
- $\text{from}(g)$, $\text{to}(g) \in \text{Blocks}$, two blocks.

The meaning of an edge g is that whenever $\text{cond}(g)$ holds, block $\text{from}(g)$ has to be solved before block $\text{to}(g)$. By construction, $\text{cond}(g)$ implies both $\text{cond}(\text{from}(g))$ and $\text{cond}(\text{to}(g))$.

In addition, the multimode structural analysis computes several functions and predicates on (differentiated) variables $v = (u,k)$:

- $\text{leading}_p(v)$ decides whether variable u is a leading variable in some mode satisfying the Boolean formula p;
- $\text{algebraic}_p(v)$ decides whether u is an algebraic variable in some mode satisfying p;
- $\text{state}_p(v)$ decides whether u is a state variable in some mode satisfying p.

For the sake of clarity, the following notations are introduced: $\text{leading}(b) = \{v \in \text{Read}(b) \cup \text{Write}(b) \mid \text{leading}_{\text{cond}(b)}(v)\}$ is the set of leading variables appearing in block b; $\text{Def}_p(v)$ is the set of blocks that define variable v in some mode satisfying the Boolean formula p, either because v itself is written, or because a higher order derivative of it is written:

$$\begin{aligned}\text{Def}_p(u,k) \quad = \quad & \{b \in \text{Blocks} \mid p \wedge \text{cond}(b) \text{ is satisfiable,} \\ & \text{and } \exists k' \geq k,\ (u,k') \in \text{Write}(b)\}\end{aligned}$$

The resulting RIMIS form model can be decomposed in several parts:

$$\begin{aligned}\text{RIMIS} \quad \equiv \quad & \text{MD} \uplus \text{RD} \uplus \text{DECL} \uplus \text{INIT} \uplus \\ & \text{ME} \uplus \text{REPL} \uplus \text{MULTI} \uplus \text{DIFF} \uplus \text{REINIT}\end{aligned}$$

where:

- MD is the set of **m**ode (Boolean) variable **d**eclarations and initializations, taken from M;
- RD is the set of **r**eal variable **d**eclarations, taken from M;
- DECL is the set of replicate variable **decl**arations, defined below;
- INIT is the set of replicate variable **init**ializations, defined below;
- ME is the set of **m**ode variable **e**quations, taken from M;
- REPL is the set of **repl**icate equations, defined below;
- MULTI is the set of **multi**plexing equations, defined below;
- DIFF is the set of **diff**erential equations, defined below;
- REINIT is the set of **reinit**ialization equations, defined below.

Replicate variable declarations (Section 4.1, step 3) consist in the declaration of the following set of real variables:

$$\begin{aligned}\text{DECL} \quad \equiv \quad & \bigcup_{b \in \text{Blocks}, (u,k) \in \text{Read}(b) \cup \text{Write}(b)} \\ & \left\{ \rho^i_{\text{cond}(b)}(u) \mid 0 \leq i \leq k \right\} \quad .\end{aligned}$$

DOI 10.3384/ecp21181519

Replicate variable initializations (Section 4.1, step 3) consist in the initialization of all replicate variables $\rho^0_{\text{cond}(b)}(u)$ that are state variables, with the initialization expression for u in M ($\text{RI}(u)$):

$$\text{INIT} \equiv \left\{(\rho^0_p(u), \text{RI}(u)) | \rho^0_p(u) \in \text{DECL and state}_p(u,0)\right\}$$

where ρ is a fixed replication operator as defined in Section 6.2.

Replicate equations (Section 4.1, step 5) consist in the differentiation to a given order of the equations of each block of equations:

$$\begin{aligned}\text{REPL} \quad \equiv \quad & \bigcup_{b\in\text{Blocks}} \\ & \left\{\sigma_b(\pi_{\text{cond}(b)}(\delta_k(q))) \mid (q,k) \in \text{Eqs}(b)\right\}\end{aligned}$$

where π is the partial evaluation operator defined in Section 6.1, equation $\delta_k(q)$ is the k-th order differentiation of equation q, and σ_b is the substitution operator such that $\sigma_b(q)$ substitutes any variable u in equation q with the replicate variable $\rho^0_{\text{cond}(b)}(u)$, any derivative of the form $\texttt{der}(u)$ by the replicate variable $\rho^1_{\text{cond}(b)}(u)$, and so on for higher order derivatives.

Multiplexing equations (Section 4.1, step 6) serve two purposes: (i) linking written variables and read variables in different blocks, and (ii) defining the original real variables from M:

$$\begin{aligned}\text{MULTI} \quad = \quad & \bigcup_{b\in\text{Blocks}, v=(u,k)\in\text{Read}(b)} \\ & \quad \{\rho^k_{\text{cond}(b)}(u) = \text{case}_v(\text{Def}_{\text{cond}(b)}(v))\} \cup \\ & \bigcup_{u\in\text{RD}} \{u = \text{case}_{u,0}(\text{Def}_{\texttt{true}}(u,0))\}\end{aligned}$$

where case_v is defined by induction over the set of blocks $\text{Def}_{\texttt{true}}(v)$ that define variable v in some mode:

$$\begin{aligned}\text{case}_{(u,k)}(\{b\}) \quad &= \quad \rho^k_{\text{cond}(b)}(u) \\ \text{case}_{v=(u,k)}(b \uplus B) \quad &= \quad \texttt{if } \text{cond}(b) \\ & \qquad \texttt{then } \rho^k_{\text{cond}(b)}(u) \\ & \qquad \texttt{else } \text{case}_v(B)\end{aligned}$$

Differential equations (Section 4.1, step 5) serve the purpose of defining replicate state variables from the replicate dummy derivatives:

$$\begin{aligned}\text{DIFF} \quad = \quad & \bigcup_{b\in\text{Blocks}, (u,k)\in\text{Write}(b)} \\ & \left\{\texttt{der}(\rho^i_{\text{cond}(b)}(u)) = \rho^{i+1}_{\text{cond}(b)}(u)\right\}_{0\le i\le k-1}\end{aligned}$$

Finally, upon the occurrence of a mode change, **reinitialization statements** (Section 4.1, step 7) serve the purpose of copying the state vector from a formerly active replicate state variable to a newly active one:

$$\begin{aligned}\text{REINIT} \quad = \quad & \bigcup_{b\in\text{Blocks}, (u,1)\in\text{Write}(b)} \\ & \{\texttt{when } \text{cond}(b) \texttt{ then} \\ & \quad \texttt{reinit}(\rho^0_{\text{cond}(b)}(u), \texttt{pre}(u)); \\ & \quad \texttt{endwhen}\}\end{aligned}$$

6.4 Optimization

Modelica code generated with the procedure described in Section 6.3 may contain multiplexing equations and reinitialization statements that can be eliminated thanks to the optimization described below.

It may happen that a multiplexing equation is of the form $\rho^k_p(u) = \rho^k_{p'}(u)$. This typically happens when a block $b \in$ Blocks reads a variable that is written by exactly one block $b' \in$ Blocks. In this case, no multiplexing equation needs to be generated, and replicate variable $\rho^k_p(u)$ does not need to be declared. Instead, every occurrence of $\rho^k_p(u)$ in equations $q \in \text{Eqs}(b)$ shall be replaced by $\rho^k_{p'}(u)$.

Remark that this optimization has been applied to the Water Tank model in RIMIS form (Figure 9). For instance, equation $\texttt{sh\_5} = \texttt{x} - \texttt{xmax}$ refers directly to variable `x` instead of variable `x_5`, sparing both the declaration of the replicate variable `x_5` and the generation of the multiplexing equation $\texttt{x} = \texttt{x\_5}$. The same optimization has been applied to variable `z`.

7 Conclusion

We presented a method for transforming multimode Modelica models that yield simulation errors with state-of-the-art Modelica tools (such as Dymola 2021 and OpenModelica 1.17.0) into Reduced Index Mode-Independent Structure (RIMIS) models that simulate correctly with the same tools.

This model transformation relies on the multimode structural analysis as performed by the IsamDAE tool (Caillaud, Malandain, and Thibault 2020). The output of this structural analysis, which is a Conditional Dependency Graph (CDG) describing all possible equation blocks in all modes and their dependencies, is used to replicate equations and real variables as needed. This is performed in such a way that the approximate structural analysis implemented in most Modelica tools will create the same equation blocks. Dummy derivatives (Mattsson and Soderlind 1993) are also used so that the resulting model is of index 0.

The 7-step RIMIS transformation was detailed on a very simple multimode model, then applied to the Modelica model of a water tank system; we showed that, while both source models cause division by zero errors at runtime, their RIMIS forms simulate correctly with both Dymola 2021 and OpenModelica 1.17.0, yielding the expected behaviors for their variables. This process was formalized, paving the way towards its automation for the handling of a wider class of multimode models by state-of-the-art Modelica tools.

A possible drawback of this approach is that the size of the RIMIS model may *a priori* be exponential in the size of the source model, as both equations and real variables could be replicated once for every mode of the system. However, experiments on a number of parametric models with the IsamDAE tool show that the number of blocks in the CDG of such models tend to be linear in their size,

except for rare pathological cases. As such, the size of the RIMIS form of a multimode Modelica model will, in a vast majority of cases, be linear in the size of the original model, thus making our approach tractable even for large models.

As a concluding remark, it can be noted that the illustrative models in this article are only made of linear equations, so that the evaluation of all equation blocks, both active and inactive, at every time step is not an issue. For nonlinear blocks, not only could this approach be computationally expensive, but it might fail altogether, as such blocks might be singular outside of a given subset of the modes.

A simple fix, that was not detailed above, consists in transforming the equations from such blocks into conditional equations, so that they become trivial equations outside of the set of modes in which they have to be considered. The matching between equations and variables that is computed during the multimode structural analysis can be used for this task, as it basically tells 'which variable has to be solved using which equation'; a nonlinear equation could then be replaced with the simple assignment of a default value to its matched real variable in the modes in which the equation block is inactive. This additional transformation would still preserve the structure of the model, in the sense that the approximate structural analysis would still result in solving the same blocks for the same real variables.

Acknowledgements

This work was supported by the FUI ModeliScale DOS0066450/00 French national collaborative project, the Glose Inria-Safran Tech bilateral collaboration, and the Inria IPL ModeliScale large scale initiative (`https://team.inria.fr/modeliscale/`).

References

Benveniste, Albert, Benoit Caillaud, Hilding Elmqvist, et al. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard J. Woeginger. Vol. 10000. Lecture Notes in Computer Science. Springer, pp. 283–310. ISBN: 978-3-319-91907-2. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2020). "The mathematical foundations of physical systems modeling languages". In: *Annual Reviews in Control* 50, pp. 72–118. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2020.08.001.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2021-09). "Handling Multimode Models and Mode Changes in Modelica". In: *Proceedings of the 14th International Modelica Conference*. Linköping University Electronic Press.

Bryant, Randal E. (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35, pp. 677–691.

Caillaud, Benoit, Mathias Malandain, and Joan Thibault (2020-04). "Implicit Structural Analysis of Multimode DAE Systems". In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Campbell, Stephen L. and C. William Gear (1995). "The index of general nonlinear DAEs". In: *Numer. Math.* 72, pp. 173–196.

Danvy, Olivier, Robert Glück, and Peter Thiemann, eds. (1996). *Partial Evaluation, International Seminar, Dagstuhl Castle, Germany, February 12-16, 1996, Selected Papers*. Vol. 1110. Lecture Notes in Computer Science. Springer. ISBN: 3-540-61580-6. DOI: 10.1007/3-540-61580-6.

Dassault Systèmes AB (2020). *Dymola official webpage*. Accessed: 2021-06-28. URL: https://www.3ds.com/products-services/catia/products/dymola/.

Elmqvist, Hilding et al. (2012-09). "State Machines in Modelica". In: *Proc. of the Int. Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Modelica Association. Munich, Germany, pp. 37–46.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Giorgidze, George and Henrik Nilsson (2011). "Embedding a Functional Hybrid Modelling Language in Haskell". In: *Implementation and Application of Functional Languages*. Ed. by Sven-Bodo Scholz and Olaf Chitil. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 138–155. ISBN: 978-3-642-24452-0.

Jeannet, Bertrand (2012-08). *BddApron*. URL: http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/.

Jones, Neil D., Carsten K. Gomard, and Peter Sestoft (1993). *Partial evaluation and automatic program generation*. Prentice Hall international series in computer science. Prentice Hall. ISBN: 978-0-13-020249-9.

Mattsson, Sven Erik and Gustaf Soderlind (1993). "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives". In: *SIAM Journal on Scientific Computing* 14.3, pp. 677–692. DOI: 10.1137/0914043.

Nilsson, Henrik and George Giorgidze (2010). "Exploiting structural dynamism in Functional Hybrid Modelling for simulation of ideal diodes". In: *Czech Technical University Publishing House*.

Pantelides, Constantinos C. (1988). "The consistent initialization of differential-algebraic systems". In: *SIAM J. Sci. Stat. Comput.* 9.2, pp. 213–231.

Pryce, John D. (2001). "A simple structural analysis method for DAEs". In: *BIT* 41.2, pp. 364–394.

Schrammel, Peter (2012). "Méthodes logico-numériques pour la vérification des systèmes discrets et hybrides. (Logico-Numerical Verification Methods for Discrete and Hybrid Systems)". PhD thesis. Grenoble Alpes University, France. URL: https://tel.archives-ouvertes.fr/tel-00809357.

Schrijver, A. (1998-04). *Theory of linear and integer programming*. Wiley.

The Modelica Association (2021-02). *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.5*. URL: https://www.modelica.org.

Van Der Schaft, A. J. and J. M. Schumacher (1998). "Complementarity modeling of hybrid systems". In: *IEEE Transactions on Automatic Control* 43.4, pp. 483–490. DOI: 10.1109/9.664151.

Zimmer, Dirk (2010). "Equation-Based Modeling of Variable-Structure Systems". PhD thesis. ETH Zürich, No. 18924.