A Reduced Index Mode-Independent Structure Model Transformation for Multimode Modelica Models
Benoît Caillaud, Mathias Malandain, Albert Benveniste

To cite this version:
A Reduced Index Mode-Independent Structure Model Transformation for Multimode Modelica Models

Benoît Caillaud, Mathias Malandain, Albert Benveniste

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

Several 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 Functional Hybrid Modeling (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 (index-reduction and block-triangular form decomposition) is performed at run-time, when the system switches to a new mode.

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 et al., 2019, 2020) we explain our approach, and we illustrate it on two simple, yet physically meaningful, examples in (Benveniste et al., 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 et al., 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
where water is considered incompressible. When the tank with a variable water inflow
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 correction simulation of the model. Our approach is then formalized for its broad application to problematic multimode models.

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 \( y_0 \), where water is considered incompressible. When the tank is full, a positive flow correction \( y_h \) is added to the outflow. As such, an approximate structural analysis of 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 Structural analysis: from single- to multi-mode

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 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 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 (Mattson and Söderlind, 1993) or the less known \( \Sigma \)-method (Pryce, 2001) can be used for single-mode DAE systems; however, the structural analysis of multimode

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.

```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 quantity
  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,true); // mode full-tank
  Boolean bl(start=false,true); // mode empty-tank
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.
a multimode DAE system can be performed by abstracting away all mode dependencies inside the equations; for instance, an equation \( x = \text{if cond then } y \text{ else } z \) 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 \( \text{eh2} \) has to be solved for the variable \( y_h \).

When performing the pivoting of this equation, mode dependencies have to be taken into account again. Equation \( \text{eh2} \) reads:

\[
0 = \text{if } bh \text{ then } x - x_{\text{max}} \text{ else } y_h
\]

which can be rewritten as an equation of the form \( 0 = a y_h + b \) where \( a \) and \( b \) are mode-dependent:

\[
0 = (\text{if } bh \text{ then } 0 \text{ else } 1) \times y_h + (\text{if } bh \text{ then } x - x_{\text{max}} \text{ else } 0)
\]

Unknown \( y_h \) can finally be isolated:

\[
y_h = \begin{cases} 
\text{if } bh \text{ then } x - x_{\text{max}} \text{ else } 0 & \text{if } bh \text{ then } 0 \text{ else } 1
\end{cases} \tag{1}
\]

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 \( \text{true} \). 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 \( \text{true} \).

### 3.2 Exact multimode structural analysis

The IsamDAE\(^1\) tool (Caillaud et al., 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 \( e12 \) is used differentiated, to compute the derivative of \( x \), when \( b1 \) is \( \text{true} \), while it is kept undifferentiated, to compute \( y_1 \), when \( b1 \) is \( \text{false} \). Also notice that equation \( \text{eh2} \) is no longer used to compute \( y_h \) in all modes, but only when \( bh \) is \( \text{false} \), thus preventing the runtime error explained above.

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.

\(^1\)https://team.inria.fr/hycomes/software/isamdae/
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 (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 Söderlind, 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 \( \text{b} = (\text{expr} \geq 0) \), where \( \text{expr} \) 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 et al., 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, multiplexing 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:
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:

- 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 \( x' \), 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 \( x' \) in all modes; however, the real equation is singular in \( x' \) 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.

```model TwoEquations
  // Source variables
  Real x(start=0,fixed=true);
  Boolean p(start=false,fixed=true);
  // Replicate ... x_2;
  // Replicate reinitializations
  when not p then
    reinit(x_3, pre(x));
  end when;
end TwoEquations;
```

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 \( x' \) if \( \text{not} \ p \) holds), and one state replicate variable \( x_3 \) that is meaningful only if \( \text{not} \ p \) 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 \( p : e \rightarrow x \), one replicate equation is generated by replacing variable \( x \) with its replicate \( x_2 \), then performing the partial evaluation (Jones et al., 1993) under the assumption that the Boolean condition \( p \) holds.

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

From the second block not \( p : e \rightarrow x' \), 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 \( p = \text{false} \) to \( p = \text{true} \) is correct, and that the reinitialization statement is never evaluated, as \( p \) remains true forever after time \( t = 1 \).
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 et al., 1993; Danvy et al., 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 \( \pi \), 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:

\[
\begin{align*}
e & ::= c \\
& \quad | e \ op \ e \\
& \quad | \nu(e, \ldots e) \\
& \quad | \text{if } p \ \text{then } e \ \text{else } e
\end{align*}
\]

where \( c \) is a constant

\( \op \in \{+,-,*,\ldots\} \)

\( \nu \) is an identifier

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 \( e \):

\[
\begin{align*}
\pi_q(c) & \equiv c \\
\pi_q(e_1 \ op \ e_2) & \equiv \pi_q(e_1) \ op \ \pi_q(e_2) \\
\pi_q(e) & \equiv e \\
\pi_q(v(e_1, \ldots e_n)) & \equiv v(\pi_q(e_1), \ldots, \pi_q(e_n)) \\
\pi_q(\text{if } p \ \text{then } e_T \ \text{else } e_F) & \equiv \text{cond}_q(p, e_T, e_F)
\end{align*}
\]

where

\[
\begin{align*}
\text{cond}_q(p, e_T, e_F) & \equiv \begin{cases} \\
\pi_q \ and \ p(e_T) & \text{if } q \ \text{and } p \ \text{is unsatisfiable, else} \\
\pi_q \ and \ \lnot p(e_T) & \text{if } q \ \text{and } p \ \text{is unsatisfiable, else} \\
\pi_q \ and \ p(e_F) & \text{where } r \ \text{is such that} \\
\pi_q \ and \ \lnot p(e_F) & \text{if } r \ \text{implies } p \ \text{and } q, \ \text{and} \\
\text{else } \pi_q \ and \ \lnot p(e_F) & \text{else } \pi_q \ and \ \lnot p(e_F)
\end{cases}
\end{align*}
\]

In the above definition, condition \( r \) is not unique: whenever possible, it should be chosen such that it is more concise than \( p \).
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=0.0,fixed=true);
  Real x(start=0.5,fixed=true);
  Real yh;
  Real sl;
  Real sh;
  Boolean bl(start=false,fixed=true);
  Boolean bh(start=false,fixed=true);
end WaterTankRIMIS;

The extension of the partial evaluation operator to equations is straightforward:
\[ \pi_q(e_{LHS} = e_{RHS}) = \pi_q(e_{LHS}) = \pi_q(e_{RHS}) \]

6.2 Variable renaming

Before moving to the formal definition of the RIMIS transformation, variable renaming must be defined, in order to declare replicable 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 \( \rho^n_p(v) \).

The operator \( \rho \) is assumed to satisfy the following axioms:

- **Identity** \( \rho^0_{true}(u) = u \)
- **Injectivity** \( \rho^n_p(u) = \rho^n_q(v) \) implies \( u = v \) and \( p \leftrightarrow q \) and \( n = m \)

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 \( v = (e >= 0) \), where \( e \) is an affine expression. Under these assumptions, equivalence checking can be done with BDDAPRON, a logico-numerical abstract domain library (Jeanet, 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:

\[ M = MD \uplus RD \uplus RI \uplus ME \uplus RE \]

where:

- MD is the set of mode (Boolean) variable declarations and initializations;
where would require a multimode structural analysis of mode with Dymola 2021.

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

- RD is the set of real variable declarations, stripped of their initializations;
- RI is the set of real variable initializations;
- ME is the set of mode variable equations;
- RE is the set of real equations.

Remark that models with when and reinit statements are not covered by the RIMIS form transformation, as this would require a multimode structural analysis of mode changes (Benveniste et al., 2020), that is not yet implemented in the IsamDAE software (Caillaud et al., 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 et al., 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 $b \in$ 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 $g \in$ Edges consists of three parts:

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

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

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:

$$\text{Def}_p(u, k) = \{ b \in \text{Blocks} \mid p \land \text{cond}(b) \text{ is satisfiable, and } \exists k' \geq k, (u, k') \in \text{Write}(b)\}$$

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

$$\text{RIMIS} \equiv M \cup \text{RD} \cup \text{DECL} \cup \text{INIT} \cup \text{REPL} \cup \text{MULTI} \cup \text{DIFF} \cup \text{REINIT}$$

where:

- $M$ is the set of mode (Boolean) variable declarations and initializations, taken from $M$;
- RD is the set of real variable declarations, taken from $M$;
- DECL is the set of replicate variable declarations, defined below;
- INIT is the set of replicate variable initializations, defined below;
- ME is the set of mode variable equations, taken from $M$;
- REPL is the set of replicate equations, defined below;
- MULTI is the set of multiplexing equations, defined below;
- DIFF is the set of differential equations, defined below;
- REINIT is the set of reinitialization equations, defined below.
Replicate variable declarations (Section 4.1, step 3) consist in the declaration of the following set of real variables:

$$\text{DECL} \equiv \bigcup_{b \in \text{Blocks}(u,k) \in \text{Read}(b)} \left\{ \rho_{\text{cond}(b)}^i(u) \mid 0 \leq i \leq k \right\}.$$

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

$$\text{INIT} \equiv \left\{ (\rho_{\text{cond}(b)}^0(u),\text{RI}(u)) \mid \rho_{\text{cond}(b)}^0(u) \in \text{DECL} \text{ and state}(u,0) \right\}.$$

where $\rho$ 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:

$$\text{REPL} \equiv \bigcup_{b \in \text{Blocks}} \left\{ \sigma_{\text{cond}(b)}(\delta_{\text{cond}(b)}(q))) \mid (q,k) \in \text{Eqs}(b) \right\}$$

where $\pi$ is the partial evaluation operator defined in Section 6.1, equation $\delta_{\text{cond}}(q)$ is the $k$-th order differentiation of equation $q$, and $\sigma_{\text{cond}}$ is the substitution operator such that $\sigma_{\text{cond}}(q)$ substitutes any variable $u$ in equation $q$ with the replicate variable $\rho_{\text{cond}(b)}^0(u)$, any derivative of the form $\text{der}(u)$ by the replicate variable $\rho_{\text{cond}(b)}^1(u)$, 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 $\text{M}$:

$$\text{MULTI} = \bigcup_{b \in \text{Blocks},v=(a,k) \in \text{Read}(b)} \left\{ \rho_{\text{cond}(b)}^k(u) = \text{case}_v(\text{Def}_{\text{cond}(b)}(v)) \right\} \cup \bigcup_{u \in \text{RD}} \{ u = \text{case}_a(\text{Def}_{\text{true}}(u,0)) \}$$

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

- case$_v(\emptyset) = 0$
- case$_v((a,k) \cup B) = \begin{cases} \rho_{\text{cond}(b)}^k(u) & \text{if cond}(b) \\ \max(\text{case}_v(B)) & \text{else} \end{cases}$

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

$$\text{DIFF} = \bigcup_{b \in \text{Blocks},(a,k) \in \text{Write}(b)} \{ \text{der}(\rho_{\text{cond}(b)}^i(u)) = \rho_{\text{cond}(b)}^{i+1}(u) \mid 0 \leq i \leq k - 1 \}$$

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:

$$\text{REINIT} = \bigcup_{b \in \text{Blocks},(a,1) \in \text{Write}(b)} \{ \text{when} \text{cond}(b) \text{then} \text{reinit}(\rho_{\text{cond}(b)}^0(u),\text{pre}(u)); \text{endwhen} \}$$

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 $\rho_{\text{cond}(b)}^0(u) = \rho_{\text{cond}(b)}^k(u)$. This typically happens when a block $b \in \text{Blocks}$ reads a variable that is written by exactly one block $b' \in \text{Blocks}$. In this case, no multiplexing equation needs to be generated, and replicate variable $\rho_{\text{cond}(b)}^0(u)$ does not need to be declared. Instead, every occurrence of $\rho_{\text{cond}(b)}^0(u)$ in equations $q \in \text{Eqs}(b)$ shall be replaced by $\rho_{\text{cond}(b)}^k(u)$.

Remark that this optimization has been applied to the Water Tank model in RIMIS form (Figure 9). For instance, equation $\text{sh}_5 = x - \text{xxax}$ 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 $x = x_5$. 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 et al., 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 Söderlind, 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 tends 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 D05066450/00 French national collaborative project, the Gloso Inria-Safran Tech bilateral collaboration, and the Inria IPL ModeliScale large scale initiative (https://team.inria.fr/modeliscale/).

References


