Exact Structural Analysis of Multimode Modelica Models: Towards the Generation of Correct Simulation Code

Albert Benveniste, Benoît Caillaud, Mathias Malandain

Exact Structural Analysis of Multimode Modelica Models: Towards the Generation of Correct Simulation Code

Albert Benveniste, Benoît Caillaud, Mathias Malandain

RESEARCH REPORT
N° 9459
February 2022
Project-Team Hycomes

ISSN 0249-6399 ISRN INRIA/RR--9459--FR+ENG

Exact Structural Analysis of Multimode Modelica Models: Towards the Generation of Correct Simulation Code

Albert Benveniste*, Benoît Caillaud*, Mathias Malandain*

Project-Team Hycomes

Research Report n° 9459 — February 2022 — 46 pages

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 report, we illustrate this behavior on several small-sized examples, shedding light on the shortcomings of the approximate structural analysis implemented in current Modelica tools. To address part of these issues, we propose a systematic transformation process for multimode Modelica models, based on the results of an already implemented multimode structural analysis, that guarantees that the output Modelica model is correctly compiled by state-of-the-art Modelica tools.
Still, this transformation is limited to models that do not exhibit impulsive behaviors at mode changes: the remaining issues illustrated by our introductory examples can only be solved by a structural analysis of mode changes, coupled with a specific handling of impulsive variables. We address these points in this report by proposing, first, a structural analysis method able to handle modes and mode changes in a unified framework, and second, a compile-time identification and characterization of impulsive variables. Implementations of both methods, based on efficient symbolic representations and algorithms, are in the works.

Key-words: Modelica, multimode DAE, structural analysis, model transformations

* Inria Rennes, Campus de Beaulieu, 35042 Rennes cedex, France; e-mail: prenom.nom@inria.fr

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Analyse Structurelle Exacte de Modèles Modelica Multimodes: Vers la Génération de Codes de Simulation Corrects

Résumé : Le langage Modelica permet de définir des modèles de systèmes dynamiques possédant plusieurs modes ayant chacun une dynamique spécifiée par un système de DAE différent. Le traitement de tels modèles par les outils Modelica de référence actuels n'est pas satisfaisant, des exceptions survenant à la simulation pour des modèles physiquement corrects.

Dans ce rapport, nous illustrons cette problématique sur plusieurs modèles de petite taille. Nous explicitons les deux raisons-clé de la mauvaise prise en charge de ces modèles par les outils Modelica, qui sont l'utilisation d'une analyse structurelle approchée et le manque d'un traitement spécifique des changements de mode.

En réponse à ces difficultés, nous proposons, d'une part, une technique de réécriture source à source de modèles Modelica qui assure leur simulation correcte par les outils Modelica actuels, et, d'autre part, une extension de l'analyse structurelle aux changements de mode, doublée d'une analyse à la compilation des éventuels comportements impulsifs d'un modèle. La mise en œuvre efficace de ces méthodes est en cours, grâce à l'utilisation de représentation symboliques issues de la vérification formelle.

Mots-clés : analyse structurelle, équations algébro-différentielles (DAE), systèmes multi-mode, modèles à structure variable

Contents

1 Introduction

Since its 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 [13]. 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 [29] and Hydra [15, 22] 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.

In this report, we explain what we think are the two technical reasons for the above problems to occur with the current tools. First, as we shall see, the stuctural analyses performed by these tools at compile time are only approximate when the model structure and/or index varies with the modes. Second, the handling of mode changes is also often incorrect, particularly when impulsive behaviors occur. Indeed, mode change events require a special kind of structural analysis, that is not performed by any existing tool (including the ones performing structural analysis at every mode change, at runtime).

Years ago, we started a project aiming at addressing all the above issues, with a different perspective in mind, that focuses on compile-time, rather than runtime, analyses. In [4, 3, 2], we explain our approach, and we illustrate it on two simple, yet physically meaningful, examples. 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 [8], we present an effective approach to achieve compile-time, mode-dependent, structural analysis of continuous modes *without enumerating them* (as this would not be able to scale up).

The advantages of our approach are twofold: (i) it provides, at compile-time, invaluable information that helps users debug their models, and (ii) efficient code generation is made 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 report, we present our works on multimode structural analysis for both continuous modes and mode changes, thus providing all theoretical foundations for a compilation chain of multimode DAE models.

First, we demonstrate how the results of our multimode structural analysis of continuous modes can be used for transforming a multimode Modelica model into an equivalent model that is guaranteed to yield correct execution on state-of-the-art Modelica tools; we called this process the RIMIS (Reduced-Index Mode-Independent Structure) transformation [7]. The 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. We demonstrate the source-to-source RIMIS transformation and show that the output model is correctly simulated

by both OpenModelica and Dymola. This approach is then formalized and generalized.

Regarding mode changes, we propose a "discrete-time structural analysis" that is able to handle both continuous modes and mode changes, and we illustrate it on two examples exhibiting impulsive behaviors at some mode changes: an idealized clutch and a cup-and-ball game. We also propose a complementary compile-time impulse analysis that allows to identify impulsive variables, still at compile time, in order to prepare for their specific handling during the simulation of the model.

Section 2 reviews some examples of failed simulations with Modelica tools. Three examples are developed and analysed: a water tank, an idealized clutch, and a cup-and-ball game. In Section 3, the reasons for the incorrect handling of such "truly multimode" models are analysed. In Section 4, the RIMIS transformation is introduced, based on the multimode structural analysis of continuous modes introduced in [8]. Our structural analysis of mode changes is presented in Section 5, and the impulse analysis is developed in Section 6.

2 Some examples of failed simulations with Modelica tools

2.1 A Water Tank

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

The corresponding Modelica model, given in Figure 1, uses two complementarity conditions [26] 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 [14] and Dymola 2021 [11]; 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, and a source-to-source model transformation that leads to correct simulations of the Water Tank model with the same Modelica tools is presented in Section 4.

2.2 An ideal clutch

The clutch depicted in Figure 3 is an idealized clutch interconnecting two rotating shafts. It is assumed that this system is closed, meaning in particular that the two shafts are not connected to anything else, whence the corresponding model:

$$\left\{\begin{array}{lll} & & \omega_1' = f_1(\omega_1, \tau_1) \quad (e_1) \\ & & \omega_2' = f_2(\omega_2, \tau_2) \quad (e_2) \\ \texttt{if } \gamma & \texttt{then} & \omega_1 - \omega_2 = 0 \quad (e_3) \\ & \texttt{and} & \tau_1 + \tau_2 = 0 \quad (e_4) \\ \texttt{if not } \gamma & \texttt{then} & \tau_1 = 0 \quad (e_5) \\ & \texttt{and} & \tau_2 = 0 \quad (e_6) \end{array}\right. \tag{1}$$

In model (1), the dynamics of each shaft i is described by the ODE $\omega_i' = f_i(\omega_i, \tau_i)$ for some, yet unspecified, function f_i, where ω_i is the angular velocity of the shaft and τ_i is the torque applied

```
model WaterTank
  constant Real xmax = 1.0; // max water quantity
  constant Real xmin = 0.0; // min water quatity
  Real x(start=(xmin+xmax)/2,fixed=true); // stored water mass
  Real y; // default output flow
  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
  // default output flow law
  /* e0: */ y = defaultOutputFlow(time);
  // input flow law
  /* e1: */ z = inputFlow(time);
  // tank level differential equation
  /* e2: */ der(x) = z + yl - yh - y;
  // 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 9 and 10.

to it. Depending on the value of the input Boolean variable γ, the clutch is either engaged ($\gamma = \text{T}$, the constant "true") or released ($\gamma = \text{F}$, the constant "false"). When the clutch is released, the two shafts rotate freely: no torque is applied to them ($\tau_i = 0$). When the clutch is engaged, it ensures a perfect join between the two shafts, forcing them to have the same angular velocity ($\omega_1 - \omega_2 = 0$) and opposite torques ($\tau_1 + \tau_2 = 0$). When $\gamma = \text{T}$, equations (e_3, e_4) are active and equations (e_5, e_6) are disabled, and vice-versa when $\gamma = \text{F}$. If the clutch is initially released, then, at the instant of contact, the relative speed of the two rotating shafts jumps to zero; as a consequence, an impulse is expected on the torques.

The model yields an ODE system when the clutch is released, and a DAE system of index 1 when the clutch is engaged (see Section 5.1.1).

The clutch in Modelica: Figure 4 details the Modelica model of the Ideal Clutch system. It is a faithful translation in the Modelica language of the two-mode DAE (1), except that the two differential equations have been linearized and the mode-dependent equation $(e_3 - -e_6)$ have been fused together for conveniency. Also, the trajectory of the input guard γ (here called `g`) has been fully specified: it takes the value T between t_1 and t_2 and F elsewhere.

This model is deemed structurally nonsingular by both OpenModelica 1.17.0 and Dymola

```
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.

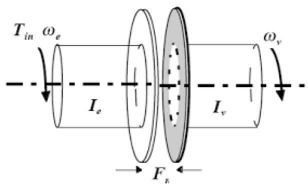

Figure 3: An ideal clutch with two shafts.

```
model ClutchBasic
  parameter Real w01=1;      // initial speed of shaft 1
  parameter Real w02=1.5;    // initial speed of shaft 2
  parameter Real j1=1;       // moment of inertia of shaft 1
  parameter Real j2=2;       // moment of inertia of shaft 2
  parameter Real k1=0.01;    // friction coefficient, shaft 1
  parameter Real k2=0.0125;  // friction coefficient, shaft 2
  parameter Real t1=5;       // time clutch is engaged
  parameter Real t2=7;       // time clutch is disengaged
  Boolean g(start=false);    // clutch mode variable
  Real w1(start = w01, fixed=true); // speed of shaft 1
  Real w2(start = w02, fixed=true); // speed of shaft 2
  Real f1;                   // torque exerted on shaft 1 by shaft 2
  Real f2;                   // torque exerted on shaft 2 by shaft 1
equation
  g = (time >= t1) and (time <= t2);
  j1*der(w1) = -k1*w1 + f1;
  j2*der(w2) = -k2*w2 + f2;
  0 = if g then w1-w2 else f1;
  f1 + f2 = 0;
end ClutchBasic;
```

Figure 4: Modelica code for the Ideal Clutch.

2021. However, none of these tools generates correct simulation code from this model. Indeed, simulations fail precisely at the instant when the clutch switches from the uncoupled mode

(`g=false`) to the coupled one (`g=true`). This is evidenced by a division by zero exception, as shown in Figure 5.

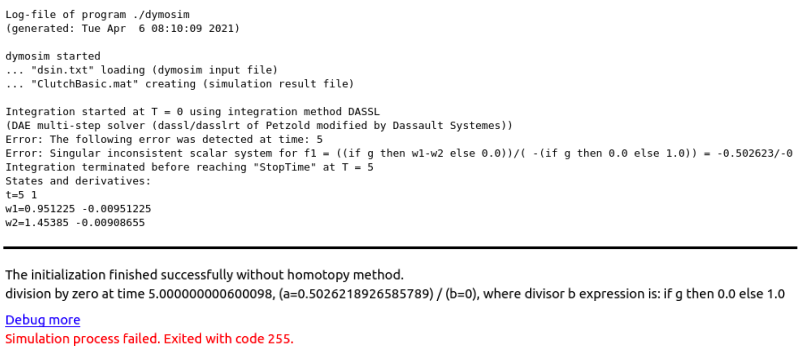

Figure 5: Division by zero exceptions with Dymola 2021 (top) and OpenModelica 1.17.0 (bottom) occuring when simulating the Ideal Clutch Modelica model.

As we shall see in Section 3, the root cause of this exception is that none of these tools performs a multimode structural analysis. Instead, the structure of the model is assumed invariant, and the structural analysis method used on it, albeit correct on single-mode DAE systems, generally fails on multimode systems unless the model structure is independent of the mode. This is the case for the Ideal Clutch model: the differentiation index jumps from 0 to 1 when the shafts are coupled, and the structure is not invariant. Our structural analysis of mode changes is illustrated on this example in Section 5.

2.3 A cup-and-ball game

Figure 6: The Cup-and-Ball game.

We sketch here a multimode extension of the popular example of the pendulum in Cartesian coordinates [23], namely the Cup-and-Ball game illustrated by Figure 6. A ball, modeled by a point mass, is attached to one end of a rope, while the other end of the rope is fixed, to the origin of the plane in the model. The ball is subject to the unilateral constraint set by the rope, but moves freely while the distance between the ball and the origin is less than its length. The system is assumed closed. The model for a 2D-version of this example is:

$$\begin{cases} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 \leq L^2 - (x^2 + y^2) & (\kappa_1) \\ 0 \leq \lambda & (\kappa_2) \\ 0 = \left[L^2 - (x^2 + y^2)\right] \times \lambda & (\kappa_3) \end{cases} \tag{2}$$

where the dependent variables are the position (x, y) of the ball in Cartesian coordinates and the rope tension λ.

The subsystem $(\kappa_1, \kappa_2, \kappa_3)$ expresses that the tension is nonnegative, the distance of the ball from the origin is less than or equal to L, and one cannot have a nonzero tension and a distance less than L at the same time. Constraints κ_1 and κ_2 are unilateral, which is not supported by Modelica and related languages. Therefore, using the technique presented in [19], we redefine the graph of this complementarity condition as a parametric curve, represented by the following three equations:

$$\begin{array}{lcl} s & = & \texttt{if } \gamma \texttt{ then } -\lambda \texttt{ else } L^2-(x^2+y^2) \\ 0 & = & \texttt{if } \gamma \texttt{ then } L^2-(x^2+y^2) \texttt{ else } \lambda \\ \gamma & = & [s \leq 0] \end{array} \tag{3}$$

Similarly to the Clutch model, impulsive behavior is expected on the torques. However, an other possible difficulty is present: subsystem $(\kappa_1, \kappa_2, \kappa_3)$ of (2) leaves the impact law at mode change insufficiently specified; it could be fully elastic, fully inelastic, or in between.

The Cup-and-Ball in Modelica: Figure 7 details the Modelica model of the Cup-and-Ball game. It is a faithful translation of the two-mode DAE (2) using rewriting (3). The point mass, modeling the ball, initially stands at the origin of the plane with zero velocity; the Boolean guard γ, named `gamma` in the model, is thus set to `false`.

```
model CupAndBall
  constant Real g = 9.81;
  constant Real L = 1.0;
  Real x(start=0,fixed=true);
  Real y(start=0,fixed=true);
  Real u(start=0,fixed=true);
  Real v(start=0,fixed=true);
  Real lambda;
  Real s;
  Boolean gamma(start=false,fixed=true);
equation
  der(x) = u;
  der(y) = v;
  der(u) + lambda*x = 0;
  der(v) + lambda*y + g = 0;
  gamma = (s <= 0);
  0 = if gamma then L^2 - (x^2 + y^2)
                else lambda;
  s = if gamma then - lambda
                else L^2 - (x^2 + y^2);
end CupAndBall;
```

Figure 7: Modelica code for the Cup-and-Ball.

As is the case for the Clutch model presented above, this model is deemed structurally nonsingular by both OpenModelica 1.17.0 and Dymola 2021, but the simulation fails at the instant of mode change. Figure 8 depicts the resulting trajectory of variables `y` and `gamma`; it ends when `gamma` switches from `false` to `true`, as the tool is unable to correctly reinitialize the model after the mode change. Replacing condition `s <= 0` with `last(s) <= 0` in order to break the fixpoint equation defining variable `gamma` (see the introduction of Section 6.1) leads to the same simulation results, but with a division by zero error similar to that shown in Figure 5 occurring at the moment of mode change.

The Cup-and-Ball model is a perfect test case for both our structural analysis of mode changes, as shown in Section 5.2, and the associated impulse analysis, presented in Section 6.

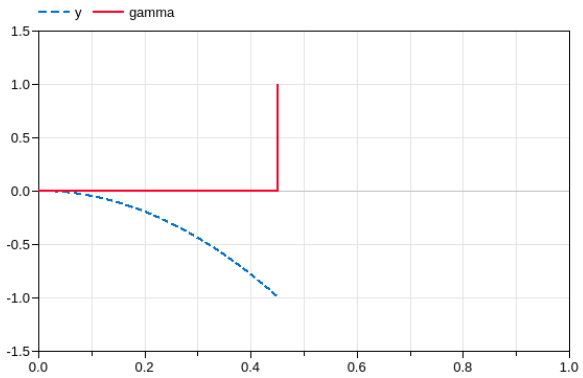

Figure 8: Trajectory of the Cup-and-Ball Modelica model: it stops around $t = 0.452s$, when the rope becomes straight.

.

3 Why do Modelica tools fail on such examples?

It appears from the above examples that fundamental studies are needed to correctly simulate multimode models. Smoothing equations that contain `if − then − else` statements could help solve the above issues by essentially turning multimode models into single-mode models. This, however, requires a delicate and definitely non-modular tuning, as it depends on the different time scales arising in the system.

We believe that, as the tools reputedly support multimode DAE models, they should handle them correctly. Whence the question: why do Modelica tools fail on such examples?

In order to gain insight into the root causes of these limitations, one has to recall that 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* [9] 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 [23], the Dummy Derivatives method [20] or the less known Σ-method [24] can be used for single-mode DAE systems.

3.1 Approximate structural analysis

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 structural analysis* for the generation of simulation code from multimode models.

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 a "single-mode" 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 9. Let us focus on equation `eh2`, which, according to this decomposition, has to be solved for variable `yh`.

When performing the pivoting of this equation, mode dependencies have to be taken into account again. Equation `eh2` reads:

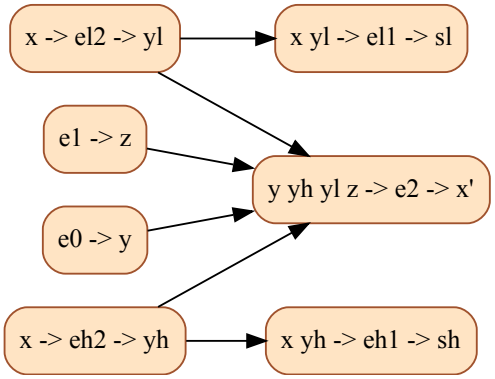

Figure 9: Dependency graph resulting from the approximate structural analysis of the Water Tank model. Vertices are equation blocks of the form $R - E \to W$, where: E is the block of equations; R is a set of variables to read (they are free variables, i.e., parameters of the block of equations); and 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 $E \to W$. Edges express causal dependencies, meaning that a block can be solved only after all its predecessors have been solved.

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

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

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

Unknown `yh` can finally be isolated:

$$\texttt{yh} = -\frac{\texttt{if bh then x} - \texttt{xmax else } 0}{\texttt{if bh then } 0 \texttt{ else } 1} \tag{4}$$

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 (4) 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`.

The compilation of correct simulation code for multimode models with mode-dependent structure and/or index, such as the Water Tank example, would require the use of an exact multimode structural analysis. Note, however, that the approximate structural analysis is used "by necessity" by existing Modelica tools. Indeed, handling a multimode model seems to require enumerating all its modes, which will typically grow exponentially with respect to the number of interconnected subsystems (or the number of equations) in the model. As mode enumeration would prevent any tool for properly scaling up, mainstream tools settled on the use of approximate structural analysis at compile time, with the resulting pitfalls we illustrated on the Water Tank example.

3.2 Failure to properly handle mode changes

The failed simulations of the Clutch and Cup-and-Ball models highlight a complementary shortcoming in current Modelica compilers, related to the handling of mode changes.

Both models (Figures 4 and 7) have a mode-dependent structure, so that the approximate structural analysis used by mainstream Modelica tools may yield simulation code that is incorrect, at least in some modes. However, these models raise two more issues related to mode changes:

- How to check whether a multimode model is well determined at all mode changes, so that sanity diagnosis of mode changes can be returned to the model designer?
- How to predict and handle impulsive behaviors at mode changes, such as the possible impulses on torques in the Clutch model, and rope tension in the Cup-and-Ball model?

3.3 The way forward: issues for consideration

The above remarks lead to considering the following two central issues:

Issue 1 *Exploiting the results of an exact structural analysis for the generation of simulation code at compile time.*

Issue 2 *Designing a structural analysis of mode changes, covering cases in which impulsive behaviors occur.*

In this report, Issue 1 is addressed by introducing a multimode model transformation that ensures that the output Modelica code is correctly handled by existing Modelica compilers. To take aim at Issue 2, we then present a comprehensive approach for the structural analysis of mode changes.

4 Exact multimode structural analysis and the RIMIS transformation

In this section, we address Issue 1; we use the Water Tank model as illustrative support. This example does not exhibit impulsive behaviors at mode changes, making it a perfect fit for the method introduced below.

4.1 Implicit structural analysis

A novel method for the exact structural analysis of a multimode DAE system in all its modes was proposed in [8]. The core idea of this method is the introduction of a "dual" representation of the mode-dependent structure of a multimode system.

As an illustration, instead of describing, for each mode, the set of active equations, this representation handles, for each equation, a propositional formula describing the subset of modes in which this equation is active. The whole structural information of the system is stored in a similar way, so that the structural analysis of the model can then be performed in an "all-modes-at-once" fashion, without enumerating the modes; for this purpose, we rely on the use of BDDs, or Binary Decision Diagrams [6], for concise encoding of, and efficient computations on, the propositional formulas. This structural analysis relies on a multimode adaptation of Pryce's Σ-method [24], with several novel algorithms required for performance; more details can be found in [8].

It is indeed of paramount importance, in order to turn the core idea into an efficient implementation, that the dual data structure is manipulated at all stages of structural analysis. The output is therefore given in a similar "dual" form: while naive algorithms would yield, in each mode, the block structure found by the structural analysis in this specific mode, our multimode structural analysis provides, for each block (respectively, for each causal dependency between blocks), the predicate characterizing the set of modes in which this block is active (resp., in which this dependency holds).

In the worst case, the size of these dual representations, for both the model structure and the output of the structural analysis, can still be proportional to the number of modes (that is, possibly exponential in the number of equations or variables of the model). However, for a wide variety of large sparse systems, a drastic reduction occurs in practice.

The IsamDAE[1] tool implements this exact multimode structural analysis suite. When used to perform the structural analysis of the Water Tank model, it yields the Conditional Dependency Graph (CDG) shown in Figure 10. This graph is to be compared with the one computed by the approximate structural analysis, shown in Figure 9.

Remark that, for this system, the differentiation index 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.

4.2 Reduced Index Mode-Independent Structure (RIMIS)

The CDG of Figure 10 could be directly used to generate simulation code, but at the price of a thorough rewriting of the backend of Modelica tools. We shall see, instead, how the results of exact multimode structural analysis can be used to transform the Modelica model into an equivalent one, that triggers no runtime error when using state-of-the-art Modelica tools based on an approximate structural analysis.

The main idea is to generate Modelica code whose approximate structural analysis yields a CDG that includes the one of Figure 10 as a subgraph. This is made difficult, among others, by the fact that the Modelica language does not permit to enable or disable an equation depending on the mode. To overcome this limitation, the 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. Then, a mode-dependent post-selection (or *multiplexing*) of the computed signals is performed to recover the expected simulation results. Of course, this leads to useless computations at runtime. However, this turns out to be a systematic way to ensure the correct simulation of a wide class of multimode Modelica models.

The method is detailed below, in informal terms, and then illustrated on a simple example. A mathematical definition of the transformation is detailed in Section 4.6. Remark that models with initial equations, `when` or `reinit` statements, discontinuous state variables, or impulsive mode changes are not covered in this section. 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 [21]. Because of a current restriction of the IsamDAE software, mode variables are assumed to be of type Boolean.

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

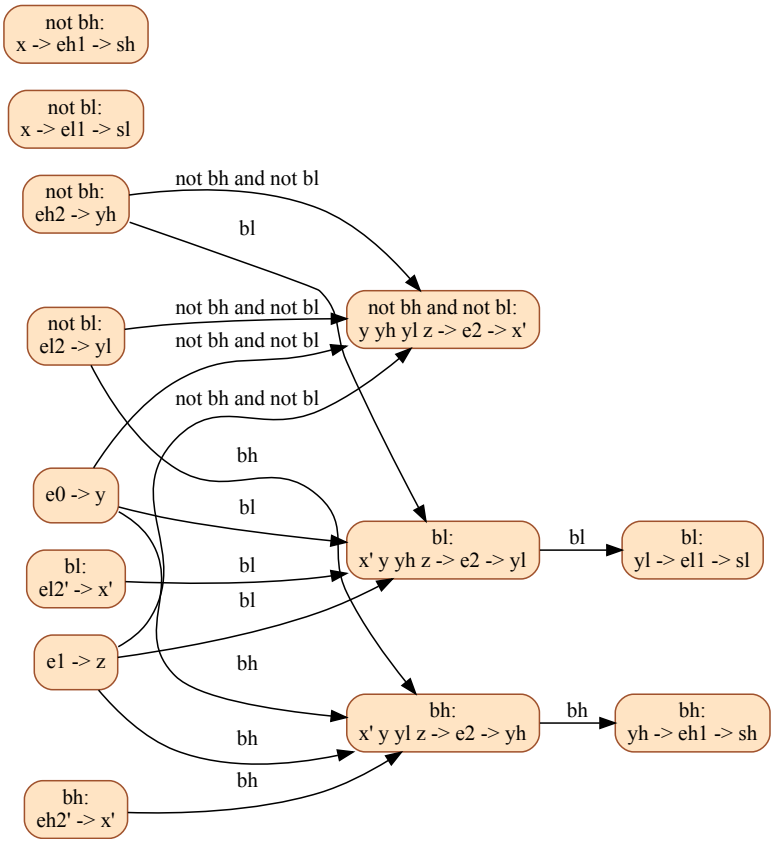

Figure 10: Conditional Dependency Graph (CDG) resulting from the multimode structural analysis of the Water Tank model. Vertices are conditional equation blocks of the form $p : R - E \to W$, where: E is the block of equations; p is a Boolean condition, defining the set of modes in which the block has 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 $p : E \to W$. When p is the proposition `true`, it is omitted, and the notation becomes: $R - E \to W$, or $E \to W$. 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.

4.3 Source-to-source transformation to RIMIS

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 [20] 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 and only 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, up to a variable renaming, 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 [17]. 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 [20].

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.

4.4 A mini-example and its RIMIS

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 11) 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.

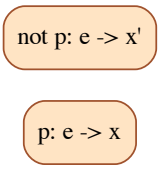

Figure 11: CDG of the Two Equations model.

The approximate structural analysis of both Dymola and OpenModelica determines that the leading variable is `x'` in all modes; however, the second equation is singular in `x'` when `p` is `true`. Unsurprisingly, an exception is raised during simulation, as shown in Figure 12.

```
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 12: Failed simulation of the Two Equations model with Dymola 2021.

Let us apply the transformation one step after the other:

1. The **CDG graph** of the source model is shown in Figure 11.
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 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 `p` : `e` $\rightarrow$ `x`, one replicate equation is generated by replacing variable `x` with its replicate `x_2`, then performing the partial evaluation [17] 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 of the Two Equations model is given in Figure 13. The result of the successful simulation of this model is shown in Figure 14. Remark that the mode switching from `p` = `false` to `p` = `true` is correct, and that the reinitialization statement is never evaluated, as `p` remains `true` forever after time `t` = 1.

```
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 13: Two Equations model in RIMIS.

4.5 The Water Tank and its RIMIS

The RIMIS transformation is illustrated on the Water Tank model (Section 2.1, Figure 1); the resulting model is shown in Figure 15. Simulation results obtained with Dymola 2021 are shown in Figure 16. 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 [14], under the (still unexplained) provision that the Newton solver is used instead of the KINSOL nonlinear solver.

4.6 Formalizing the transformation to RIMIS

The mathematical definition of the RIMIS transformation relies on the partial evaluation of equations. Once variable renaming is also properly defined, the seven-step transformation mentioned in Section 4.3 is formalized. Finally, an optimization aiming at reducing the transformed model is presented.

4.6.1 Partial evaluation

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 [17, 10].

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

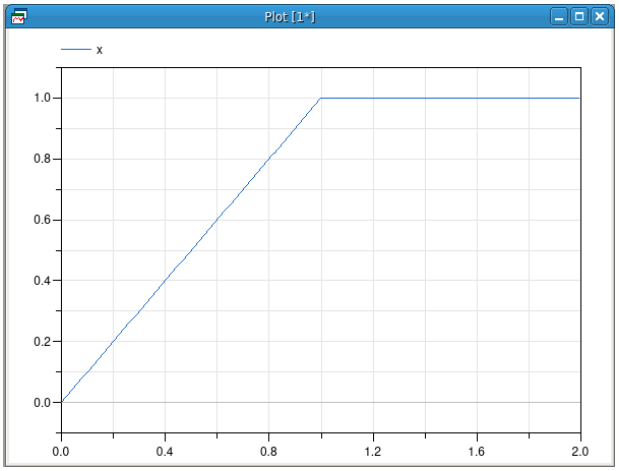

Figure 14: Simulation of the Two Equations model in RIMIS with Dymola 2021.

implies $e = e'$ and $\text{free}(e') \subseteq \text{free}(e)$, where 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:

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

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.

```
model WaterTankRIMIS
  // Constants
  constant Real xmax = 1.0;
  constant Real xmin = 0.0;
  // Variables
  Real x(start=(xmin+xmax)/2,fixed=true);
  Real y;
  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(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 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 e0 -> y
  y = defaultOutputFlow(time);
  // Block e1 -> z
  z = inputFlow(time);
  // Block not bl: el2 -> yl
  yl_2 = 0;
  // Block bh: eh2' -> x'
  x_p_6 = 0;
  // Block bl: x' y yh z -- e2 -> yl
  yl_4 = y + x_p_4 + yh_5 - z;
  // Block not bh & not bl: y yh yl z -- e2 -> x'
  x_p_7 = z + yl_2 - yh_5 - y;
  // Block bh: x' y yl z  -- e2 -> yh
  yh_6 = z + yl_2 - x_p_6 - y;
  // Block bl: yl -- el1 -> sl
  sl_4 = yl_4;
  // Block bh: yh -- eh1 -> sh
  sh_6 = yh_6;
end WaterTankRIMIS;
```

Figure 15: The Water Tank system in RIMIS.

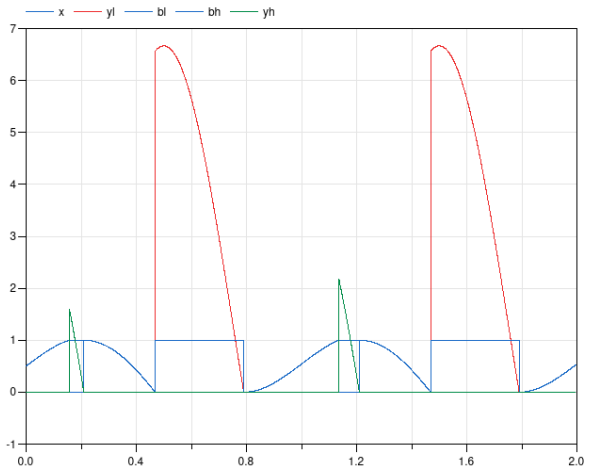

Figure 16: Simulation of the Water Tank system in RIMIS with Dymola 2021.

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

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

4.6.2 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 $\rho_p^n(v)$. The operator ρ is assumed to satisfy the following axioms:

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

Checking the equivalence of two Boolean expressions is, in general, a difficult problem. In this Section, Boolean expressions that appear in conditional statements are restricted to propositional formulas only. Mode equations are restricted to the form $\texttt{v} = (e >= 0)$, where e is an affine expression. Under these assumptions, equivalence checking can be done with BddApron, a logico-numerical abstract domain library [16] combining BDDs (Boolean Decision Diagrams) [6] and polyhedra [28]. Such a use of BddApron is considered, among other program analyses, in Chapter 7 of [27].

4.6.3 Formal definition

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

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

where:

- MD is the set of (Boolean) **m**ode 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 transformation, as this would require a multimode structural analysis of mode changes, described in Section 5 of this report, which is not yet implemented in the IsamDAE software [8]. Because of a current restriction of IsamDAE, mode variables are assumed to be Boolean. Because state variables can only be copied in the REINIT section of the generated code, state variables must be continuous. Nevertheless, variable-dimension state vectors are supported. This means, for instance, that a variable can be algebraic in some modes, and a state variable in some other modes. Impulsive variables are not supported, because of the limitations of state-of-the-art Modelica tools that do not support impulsive mode changes.

In what follows, model M is assumed to be structurally nonsingular in all modes: this is actually guaranteed by our multimode structural analysis [8], which will only compute the CDG in this case. The said CDG then 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 (one may refer to Figure 10, page 15, as an illustration on the Water Tank model):

- $\text{cond}(b)$, 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:

$$\text{leading}(b) = \{v \in \text{Read}(b) \cup \text{Write}(b) | \, \text{leading}_{\text{cond}(b)}(v)\}$$

is the set of leading variables appearing in block b; and:

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

is the set of blocks that define variable $v = (u,k)$ in some mode satisfying the Boolean formula p, either because v itself is written, or because one of its higher order derivative is written.

The resulting RIMIS model can be decomposed in several parts:

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

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 4.3, step 3) consist in the declaration of the following set of real variables:

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

where ρ is a fixed replication operator as defined in Section 4.6.2.

Replicate variable initializations (Section 4.3, step 3) consist in the initialization of all replicate variables $\rho^0_{\text{cond}(b)}(u)$ that are state variables, using the initialization expression $\text{RI}(u)$ given for variable u in the original model M:

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

Replicate equations (Section 4.3, step 5) consist in the differentiation to a given order of the equations of each block of equations:

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

where π is the partial evaluation operator defined in Section 4.6.1, 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 $\rho^0_{\mathrm{cond}(b)}(u)$, any derivative of the form $\mathtt{der}(u)$ by the replicate variable $\rho^1_{\mathrm{cond}(b)}(u)$, and so on for higher order derivatives.

Multiplexing equations (Section 4.3, step 6) serve two purposes: (i) linking written variables and read variables in different blocks, and (ii) defining the original real variables from M:

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

where case_v is defined by induction over the set of blocks $\mathrm{Def}_{\mathtt{true}}(v)$ that define variable v in the modes of interest:

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

A fine illustration of this construction is the multiplexing equation computing variable `x_p` (the first-order derivative of `x`) in the Water Tank RIMIS code, shown Figure 15, page 21. Depending on the values of mode variables `bh` and `bl`, one of the three following replicate variables are used to compute `x_p`: `x_p_6`, `x_p_4`, or `x_p_7`.

Differential equations (Section 4.3, step 5) serve the purpose of defining replicate state variables from the replicate dummy derivatives:

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

Finally, upon the occurrence of a mode change, **reinitialization statements** (Section 4.3, step 7) serve the purpose of copying the state vector from a formerly active replicate state variable to a newly active one:

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

4.6.4 Optimization

Modelica code generated with the procedure described in Section 4.6.3 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 $b \in \mathrm{Blocks}$ reads a variable that is written by exactly one block $b' \in \mathrm{Blocks}$. 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 $q \in \mathrm{Eqs}(b)$ shall be replaced by $\rho^k_{p'}(u)$.

Remark that this optimization has been applied to the Water Tank model in RIMIS (Figure 15). For instance, equation $\mathtt{sh\_5} = \mathtt{x} - \mathtt{xmax}$ 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 $\mathtt{x} = \mathtt{x\_5}$. The same optimization has been applied to variable `z`.

4.7 Computational efficiency

A possible drawback of the approach described above 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 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.

It can also be noted that the illustrative models in this Section 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.

4.8 Take away summary

The IsamDAE tool, based on a dual representation of the mode-dependent structure of a multimode Modelica model, is able to perform the structural analysis of such a model in an efficient way, in an "all-modes-at-once" fashion, by relying on BDD techniques. More information about the underlying algorithms can be found in [8].

We have shown above how the results of this multimode structural analysis makes it possible to perform source-to-source transformations, so that an equivalent Modelica model is produced for which the approximate structural analysis of state-of-the-art Modelica tools yields correct simulation code.

At this point, Issue 2 is still to be tackled, as the structural analysis performed by IsamDAE does not yet handle mode changes and the possible impulsive behaviors that occur during them. The following contributions address this issue by proposing novel algorithms for the structural analysis of mode changes, which are amenable to a future implementation in the IsamDAE tool.

5 Structural analysis of mode changes

In this section, we address Issue 2 by proposing a structural analysis for mode changes; both the Clutch and Cup-and-Ball examples are used for illustrative purposes.

The Clutch example is among the simplest ones exhibiting impulsive behaviors at mode changes: we will see that the torques in model (1) are impulsive at the mode change when the clutch gets engaged. Nevertheless, this model is well determined as such, with no need for any extra information about restart conditions.

In constrast, the Cup-and-Ball model (2) will raise new difficulties. It is expected that the tension is impulsive when the rope gets straight, but we shall see that the results of the analysis change depending on whether or not the impact is assumed fully elastic.

For both examples, the structural analysis of mode changes that is proposed in this Section is able to, either return precise diagnostics about over- or underdeterminations at mode changes, or generate meaningful restart conditions.

5.1 The ideal clutch

Its model was given in (1). We first analyze separately the model for each mode of the clutch. Then, we focus on mode changes and propose a comprehensive analysis.

5.1.1 Separate Analysis of Each Mode

In the released mode, i.e., when $\gamma = \text{F}$ in System (1), the two shafts are independent and one obtains the following two independent ODEs for ω_1 and ω_2:

$$\begin{array}{llll} \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \tau_1 = 0 & (e_5) \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \tau_2 = 0 & (e_6) \end{array} \tag{5}$$

In the engaged mode, however ($\gamma = \text{T}$), the two velocities and torques are algebraically related:

$$\begin{array}{llll} \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \omega_1 - \omega_2 = 0 & (e_3) \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \tau_1 + \tau_2 = 0 & (e_4) \end{array} \tag{6}$$

System (6) is a DAE. Its structural analysis tells that equation (e_3) must be differentiated and added to the model (it is highlighted in red):

$$\begin{array}{llll} \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \omega_1 - \omega_2 = 0 & (e_3) \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \omega_1' - \omega_2' = 0 & (e_3') \\ & & \tau_1 + \tau_2 = 0 & (e_4) \end{array} \tag{7}$$

Although this change of differentiation index is the root cause of the runtime exceptions shown in Figure 5, solving this issue would not be enough for the correct simulation of the model, because of the need of handling mode changes.

As a matter of fact, while the cold initialization of the engaged mode yields 6 dependent variables for only 5 equations, thus leaving one degree of freedom (the common velocity of the two shafts), the mode change $\gamma : \text{F} \to \text{T}$, when the clutch gets engaged, is physically determinate, which makes the point that mode changes cannot be handled as "cold restarts".

Inferring by hand the reset values for rotation velocities when the clutch gets engaged is definitely non-trivial. Furthermore, these values depend on the whole system model, so that the task of determining them becomes complex if external components are added.

It is therefore highly desirable, for this example, to let the compiler infer these reset values from model (1).

5.1.2 Infinitesimal time discretization

If DAE dynamics is approximated in discrete time, then the whole model becomes discrete-time. To avoid the problem of approximation error, our idea is to use an "infinitesimal" time step in the discrete time approximation. This will yield an approximation up to an infinitesimal accuracy.

This can be made rigorous by relying on *nonstandard analysis* [25, 18, 3], which extends the set $\mathbb{R}$ of real numbers to a superset ${}^\star\mathbb{R}$ of *hyperreals* that includes infinite sets of infinitely large numbers and infinitely small numbers.

For the understanding of what follows, it is enough to know the following about nonstandard analysis. There exist *infinitesimals*, defined as hyperreals that are smaller in absolute value than any real number. The arithmetic operations $+$, $\times$, etc., and usual relations, are lifted to ${}^\star\mathbb{R}$.

For every finite hyperreal $x \in {}^\star\mathbb{R}$, there is a unique standard real number $\mathsf{st}(x) \in \mathbb{R}$ such that $\mathsf{st}(x) - x$ is infinitesimal, and $\mathsf{st}(x)$ is called the *standard part* (or *standardization*) of x. Standardizing functions or systems of equations, however, requires some care.

One important issue is derivatives. For $t \mapsto x(t)$ an $\mathbb{R}$-valued (standard) signal ($t \in \mathbb{R}$),

$$\begin{array}{l} x \text{ is differentiable at instant } t{\in}\mathbb{R} \text{ if and only if} \\ \text{there exists } a{\in}\mathbb{R} \text{ such that, for any infinitesimal} \\ \partial \in {}^\star\mathbb{R}, \ \frac{x(t+\partial)-x(t)}{\partial} - a \text{ is infinitesimal; then,} \\ a = x'(t). \end{array} \tag{8}$$

We can then consider the time index set $\mathbb{T} \subseteq {}^\star\mathbb{R}$:

$$\mathbb{T} = 0, \partial, 2\partial, 3\partial, \cdots = \{n\partial \mid n \in {}^\star\mathbb{N}\} \tag{9}$$

where ${}^\star\mathbb{N}$ denotes the set of *hyperintegers*, consisting of all integers augmented with additional infinite numbers called *nonstandard*, and ∂ is an arbitrary, but fixed, infinitesimal.[2] The following features of $\mathbb{T}$ are important: (1) any finite real time $t \in \mathbb{R}$ is infinitesimally close to some element of $\mathbb{T}$ (hence, $\mathbb{T}$ covers $\mathbb{R}$ and can be used to index continuous-time dynamics); and (2) $\mathbb{T}$ is "discrete": every instant $n\partial$ has a predecessor $(n{-}1)\partial$ (except for $n = 0$) and a successor $(n{+}1)\partial$.

Let x be a nonstandard signal indexed by $\mathbb{T}$. The *forward-* and *backward-shifted* signals $x^\bullet$ and ${}^\bullet x$ are defined by:

$$x^\bullet(n\partial) =_{\text{def}} x((n{+}1)\partial) \quad \text{and} \quad {}^\bullet x((n{+}1)\partial) =_{\text{def}} x(n\partial)\,,$$

implying that an initial value for ${}^\bullet x(0)$ must be provided. For $f(X)$ a function of the tuple X of signals, we set $(f(X))^\bullet =_{\text{def}} f(X^\bullet)$ where the forward shift $X \mapsto X^\bullet$ applies pointwise to all the components of the tuple. For example, $f^\bullet(x,y)(t) = f(x^\bullet, y^\bullet) = f(x(t{+}\partial), y(t{+}\partial))$.

Using (8), we represent, up to an infinitesimal, the derivative x' of a signal by its first-order explicit Euler approximation $\frac{1}{\partial}(x^\bullet - x)$. Solutions of multi-mode DAE systems may be non-differentiable or even non-continuous at events of mode change. To give a meaning to x' at any instant, *we* ***define*** *it everywhere as*

$$x' =_{\text{def}} \frac{1}{\partial}(x^\bullet - x)\,. \tag{10}$$

The nonstandard expansion of two-mode system (5,7) is:

$$\left\{\begin{array}{lll} & & \frac{\omega_1^\bullet - \omega_1}{\partial} = f_1(\omega_1, \tau_1) \qquad (e_1^\partial) \\ & & \frac{\omega_2^\bullet - \omega_2}{\partial} = f_2(\omega_2, \tau_2) \qquad (e_2^\partial) \\ \texttt{if } \gamma & \texttt{then} & \omega_1 - \omega_2 = 0 \qquad (e_3) \\ & \texttt{and} & \omega_1^\bullet - \omega_2^\bullet = 0 \qquad (e_3^\bullet) \\ & \texttt{and} & \tau_1 + \tau_2 = 0 \qquad (e_4) \\ \texttt{if not } \gamma & \texttt{then} & \tau_1 = 0 \qquad (e_5) \\ & \texttt{and} & \tau_2 = 0 \qquad (e_6) \end{array}\right. \tag{11}$$

Note that the latent differentiated equation (e_3') of model (7) has been replaced by the forward shifted equation $(e_3^\bullet)$ (both are equivalent from a structural point of view). The state variables are ω_1, ω_2 whereas the leading variables are now τ_1, τ_2, $\omega_1^\bullet$, $\omega_2^\bullet$, in both modes $\gamma = \text{F}$ and $\gamma = \text{T}$. This yields a sort of explicit Euler scheme for model (1), which is exact up to infinitesimals within each mode. The structural analysis is correct in each mode.

[2] It is proved in [3] that the simulation code that is finally generated does not depend on the choice of this infinitesimal time step.

5.1.3 Structural analysis of mode changes

We focus on mode change $\gamma : \text{F} \to \text{T}$, when the clutch gets engaged. At the considered instant, we have ${}^\bullet\gamma = \text{F}$ and $\gamma = \text{T}$. We unfold System (11) at the two successive (previous and current) instants by taking the actual values for the guard at those instants into account:

$$\begin{array}{r} \text{previous} \\ \text{instant} \\ \gamma = \text{F} \end{array} \left\{ \begin{array}{ll} \frac{\omega_1 - {}^\bullet\omega_1}{\partial} = f_1({}^\bullet\omega_1, {}^\bullet\tau_1) & ({}^\bullet e_1^\partial) \\ \frac{\omega_2 - {}^\bullet\omega_2}{\partial} = f_2({}^\bullet\omega_2, {}^\bullet\tau_2) & ({}^\bullet e_2^\partial) \\ {}^\bullet\tau_1 = 0 & \\ {}^\bullet\tau_2 = 0 & \end{array} \right.$$
$$\begin{array}{r} \text{current} \\ \text{instant} \\ \gamma = \text{T} \end{array} \left\{ \begin{array}{ll} \frac{\omega_1^\bullet - \omega_1}{\partial} = f_1(\omega_1, \tau_1) & \\ \frac{\omega_2^\bullet - \omega_2}{\partial} = f_2(\omega_2, \tau_2) & \\ \omega_1 - \omega_2 = 0 & (e_3) \\ \omega_1^\bullet - \omega_2^\bullet = 0 & \\ \tau_1 + \tau_2 = 0 & \end{array} \right. \tag{12}$$

We regard System (12) as an algebraic system of equations with dependent variables ${}^\bullet\tau_i, \omega_i; \tau_i, \omega_i^\bullet$ for $i = 1, 2$, i.e., the leading variables of System (11) at the previous and current instants. System (12) is structurally singular, as it includes the following subsystem[3] which has five equations and only four dependent variables $\omega_1, \omega_2, {}^\bullet\tau_1, {}^\bullet\tau_2$:

$$\left\{ \begin{array}{ll} \frac{\omega_1 - {}^\bullet\omega_1}{\partial} = f_1({}^\bullet\omega_1, {}^\bullet\tau_1) & ({}^\bullet e_1^\partial) \\ \frac{\omega_2 - {}^\bullet\omega_2}{\partial} = f_2({}^\bullet\omega_2, {}^\bullet\tau_2) & ({}^\bullet e_2^\partial) \\ {}^\bullet\tau_1 = 0 & \\ {}^\bullet\tau_2 = 0 & \\ \omega_1 - \omega_2 = 0 & (e_3) \end{array} \right. \tag{13}$$

We resolve this conflict by applying the following principle:

Principle 1 (causality) *What was done at the previous instant cannot be undone at the current instant.*

Applying (1) leads to removing, from subsystem (13), the conflicting equation (e_3). This yields the following nonstandard code for the restart at mode change $\gamma : \text{F} \to \text{T}$:

$$\left\{ \begin{array}{l} \omega_1, \omega_2, {}^\bullet\tau_1, {}^\bullet\tau_2 \text{ set by previous instant} \\ \omega_1^\bullet = \omega_1 + \partial \times f_1(\omega_1, \tau_1) \\ \omega_2^\bullet = \omega_2 + \partial \times f_2(\omega_2, \tau_2) \\ \omega_1^\bullet - \omega_2^\bullet = 0 \\ \tau_1 + \tau_2 = 0 \end{array} \right. \tag{14}$$

The consistency equation $(e_3) : \omega_1 - \omega_2 = 0$ has been removed from System (14), thus modifying the original model. However, this removal occurs only at mode change events $\gamma : \text{F} \to \text{T}$. What we have done amounts to *delaying by one nonstandard instant the satisfaction of some of the constraints in force in the new mode* $\gamma = \text{T}$. Since our time step ∂ is infinitesimal, this takes zero standard time.

[3] Over- and underdetermined subsystems are structurally found by computing the Dulmage-Mendelsohn decomposition of the system [12].

5.1.4 Generating effective code for restart

We wish to use System (14) by identifying current values for the states ω_i with the *left-limits* ω_i^- i.e., the values of the velocities just before the mode change. From these values, we would then compute the restart values for the velocities $\omega_i^+ =_{\text{def}} \omega_i^\bullet$, together with the torques τ_i.

Unfortunately, hyperreals are unknown to computers, hence, System (14) cannot be used as such, but needs to be *standardized,* by "washing out" ∂. Since the time step ∂ is infinitesimal, it is tempting to get rid of of it in (14) by simply setting $\partial = 0$. Unfortunately, doing this leaves us with a structurally singular system, since the two torques are then involved in only one equation.

This problem of structural singularity is in fact due to the existence of impulsive variables. To discover them in a systematic way, we perform an *impulse analysis.*

Impulse analysis: Before engaging the clutch, we must generically assume $\omega_1 - \omega_2 \neq 0$. Since $\omega_1^\bullet - \omega_2^\bullet = 0$ holds, $\frac{(\omega_1^\bullet - \omega_2^\bullet) - (\omega_1 - \omega_2)}{\partial} = f_1(\omega_1, \tau_1) - f_2(\omega_2, \tau_2)$ cannot be finite because, if it was, then the function $\omega_1 - \omega_2$ would be continuous, contradicting the assumption that $\omega_1 - \omega_2 \neq 0$. Hence, the hyperreal $f_1(\omega_1, \tau_1) - f_2(\omega_2, \tau_2)$ is necessarily infinite. However, we assumed continuous functions f_i and finite state (ω_1, ω_2). Thus, one of the torques τ_i must be infinite at mode change, and because of equation $(e_4) : \tau_1 + \tau_2 = 0$, both torques are in fact infinite, i.e., are *impulsive.*

Eliminating impulsive variables: We now assume that the f_i's are linear in the torques, i.e., each f_i has the form

$$f_i(\omega_i, \tau_i) \quad = \quad a_i(\omega_i) + b_i(\omega_i)\tau_i \,, \tag{15}$$

where b_1 and b_2 are the inverse moments of inertia of the rotating masses and a_1 and a_2 are damping factors divided by the corresponding moments of inertia. This yields the following system of equations, to be solved for $\omega_1^\bullet, \omega_2^\bullet, \tau_1, \tau_2$ at the instant when γ switches from F to T:

$$\left\{ \begin{array}{ll} \omega_1^\bullet = \omega_1 + \partial\,(a_1(\omega_1) + b_1(\omega_1)\tau_1) & (e_1^\partial) \\ \omega_2^\bullet = \omega_2 + \partial\,(a_2(\omega_2) + b_2(\omega_2)\tau_2) & (e_2^\partial) \\ \omega_1^\bullet - \omega_2^\bullet = 0 & (e_3^\bullet) \\ \tau_1 + \tau_2 = 0 & (e_4) \end{array} \right. \tag{16}$$

We now eliminate the impulsive variables from System (16), namely, the two torques. Using (e_4) yields $-\tau_2 = \tau_1 =_{\text{def}} \tau$. Premultiplying the system of equations

$$\left\{ \begin{array}{ll} \omega_1^\bullet = \omega_1 + \partial\,(a_1(\omega_1) + b_1(\omega_1)\tau) & (e_1^\partial) \\ \omega_2^\bullet = \omega_2 + \partial\,(a_2(\omega_2) - b_2(\omega_2)\tau) & (e_2^\partial) \end{array} \right.$$

by the row matrix $\left[\begin{array}{cc} b_2(\omega_2) & b_1(\omega_1) \end{array} \right]$ yields

$$\begin{array}{l} b_2(\omega_2)\,\omega_1^\bullet + b_1(\omega_1)\,\omega_2^\bullet = \\ b_2(\omega_2)\,(\omega_1 + \partial\,a_1(\omega_1)) + b_1(\omega_1)\,(\omega_2 + \partial\,a_2(\omega_2))\,. \end{array}$$

Using in addition $(e_3^\bullet)$ and setting $\omega^\bullet =_{\text{def}} \omega_1^\bullet = \omega_2^\bullet$ yields

$$\begin{array}{rcl} \omega^\bullet & = & \dfrac{b_2(\omega_2)\omega_1 + b_1(\omega_1)\omega_2}{b_1(\omega_1) + b_2(\omega_2)} \\ & & +\partial\,\dfrac{a_1(\omega_1)b_2(\omega_2) + a_2(\omega_2)b_1(\omega_1)}{b_1(\omega_1) + b_2(\omega_2)} \end{array} \tag{17}$$

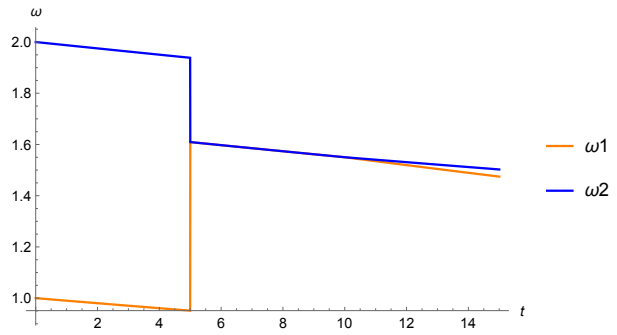

Figure 17: Simulation of the Clutch model with resets. Mode change F → T occurs at $t = 5s$ and mode change T → F occurs at $t = 10s$.

It is now legitimate to set $\partial = 0$ in its right-hand side. This yields, by identifying $\mathsf{st}(\omega_i) = \omega_i^-$ and $\mathsf{st}(\omega_i^\bullet) = \omega_i^+$:

$$\omega_1^+ = \omega_2^+ = \frac{b_2(\omega_2^-)\,\omega_1^- + b_1(\omega_1^-)\,\omega_2^-}{b_1(\omega_1^-) + b_2(\omega_2^-)}, \tag{18}$$

where we recall that $\mathsf{st}(\omega)$ is the standard part of ω, see the beginning of Section 5.1.2. Equation (18) provides us with the reset values for the positions in the engaged mode, which is enough to restart the simulation in this mode.

Figure (17) shows a simulation of the Clutch where the resets are computed following this approach. As expected, the reset value sits between the two values of ω_1^- and ω_2^- when $\gamma : \text{F} \to \text{T}$ (at $t = 5$s), and the transition is continuous at the second reset (at $t = 10$s). An alternative approach for the computation of the reset values, which does not require the elimination of impulsive variables, is developed in [3], see also [1].

5.2 The Cup-and-Ball example

Using (3), the original model (2) is rewritten as

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1)\\ & 0 = y'' + \lambda y + g & (e_2)\\ & \gamma = [s \le 0] & (k_0)\\ \texttt{if } \gamma \texttt{ then} & 0 = L^2 - (x^2+y^2) & (k_1)\\ \texttt{and} & 0 = \lambda + s & (k_2)\\ \texttt{if not } \gamma \texttt{ then} & 0 = \lambda & (k_3)\\ \texttt{and} & 0 = (L^2-(x^2+y^2)) - s & (k_4) \end{array}\right. \tag{19}$$

Two issues have to be addressed by our structural analysis: the expected impulsive behavior of the accelerations at mode changes, and the insufficient specification of the nature (elastic, inelastic or in between) of the impact.

We implicitly add to model (19) the following two equations, for each state variable v:

$$v' = \frac{v^\bullet - v}{\partial} \quad ; \quad v'' = \frac{v^{\bullet 2} - 2v^\bullet + v}{\partial^2}\,, \tag{20}$$

where

$$\begin{array}{rcl} v^{\bullet}(t) & =_{\text{def}} & v(t+\partial)\,, \\ v^{\bullet 2}(t) & =_{\text{def}} & v(t+2\partial) \text{ and, more generally,} \\ v^{\bullet n}(t) & =_{\text{def}} & v(t+n\partial)\,. \end{array}$$

Equation (20) means that the derivatives x', y', x'', y'' are interpreted using the explicit first-order Euler scheme with an *infinitesimal time step* ∂. Note that (20) implies

$$x'' \quad = \quad \frac{x'^{\bullet} - x'}{\partial} \quad . \tag{21}$$

After performing the substitutions given by (20), we observe that the subsystem collecting equations (k_0)–(k_4) is a logico-numerical fixpoint equation, with dependent variables $x^{\bullet 2}, y^{\bullet 2}, \lambda, \gamma$. A possible solution would consist in performing a relaxation, by iteratively updating the numerical variables based on the previous value for the guards, and then re-evaluating the guard based on the updated values of the numerical variables, hoping for a fixpoint to occur. Such fixpoint equation, however, can have zero, one, several, or infinitely many solutions. No characterization exists that could serve as a basis for a (graph-based) structural analysis. *We thus decide to refuse solving such mixed logico-numerical systems.*

As a consequence, we are unable to evaluate guard γ, so that the mode the system is in cannot be determined: model (19) is rejected.

To break the fixpoint equation defining γ, we choose to systematically introduce infinitesimal delays to guards. For the Cup-and-Ball, the predicate $s \leq 0$ then defines the value of the guard *at the next nonstandard instant.*[4] This yields the corrected model (22), where the modification is highlighted in red.

$$\left\{ \begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma^{\bullet} = [s \leq 0]; \gamma(0) = \text{F} & (k_0) \\ \texttt{if } \gamma \texttt{ then} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not } \gamma \texttt{ then} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array} \right. \tag{22}$$

This model is understood in the nonstandard setting, meaning that the derivatives are expanded using (20). The leading variables in all modes are $\lambda, s, x^{\bullet 2}, y^{\bullet 2}$.

[4] The condition triggering the mode change is based on the positions, which remain continuous at mode changes, even though the velocities are discontinuous. As a result, the shifting of this guard by an infinitesimal time step only yields an infinitesimal change in the values of state variables, which will be erased by the standardization process, so that the numerical solution is not impacted by this change in the model.

5.2.1 Structural analysis of mode changes

Due to equation (k_1), the mode $\gamma = \text{T}$ (where the rope is straight) requires index reduction. We thus augment model (22) with the two latent equations shown in red:

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1)\\ & 0 = y'' + \lambda y + g & (e_2)\\ & \gamma^\bullet = [s \le 0]; \gamma(0) = \text{F} & (k_0)\\ \texttt{if } \gamma \texttt{ then} & 0 = L^2 - (x^2+y^2) & (k_1)\\ \texttt{and} & 0 = L^2 - (x^2+y^2)^\bullet & (k_1^\bullet)\\ \texttt{and} & 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2})\\ \texttt{and} & 0 = \lambda + s & (k_2)\\ \texttt{if not } \gamma \texttt{ then} & 0 = \lambda & (k_3)\\ \texttt{and} & 0 = (L^2 - (x^2+y^2)) - s & (k_4) \end{array}\right. \tag{23}$$

Note that, as in System (11), the two latent equations $(k_1^\bullet)$ and $(k_1^{\bullet 2})$ were obtained by *shifting* (k_1) *forward*, which is equivalent to differentiating it for the structural analysis. To perform structural analysis at the considered mode change, we unfold model (23) at the successive instants

$$^{\bullet 2}t \ =_{\text{def}} \ t - 2\partial, \quad ^{\bullet}t \ =_{\text{def}} \ t - \partial, \quad \text{and} \quad t\,,$$

where t denotes the current instant. In the following, equation (e_1) at the instant $t - 2\partial$ (respectively, $t - \partial$) will be denoted by $(^{\bullet 2}e_1)$ (resp., $(^{\bullet}e_1)$).

In this unfolding, the two equations (k_1) and $(k_1^\bullet)$ are in conflict with selected equations from the previous two instants, shown in blue in the following subsystem, whose dependent variables are the leading variables at instants $t - 2\partial$ and $t - \partial$, namely $x, y, ^{\bullet 2}\lambda; x^\bullet, y^\bullet, ^\bullet\lambda$:

$$\left\{\begin{array}{ll} 0 = \frac{x - 2\,^\bullet x + {}^{\bullet 2}x}{\partial^2} + {}^{\bullet 2}\lambda\, {}^{\bullet 2}x & (^{\bullet 2}e_1)\\ 0 = \frac{y - 2\,^\bullet y + {}^{\bullet 2}y}{\partial^2} + {}^{\bullet 2}\lambda\, {}^{\bullet 2}y + g & (^{\bullet 2}e_2)\\ 0 = \frac{x^\bullet - 2x + {}^\bullet x}{\partial^2} + {}^\bullet\lambda\, {}^\bullet x & (^\bullet e_1)\\ 0 = \frac{y^\bullet - 2y + {}^\bullet y}{\partial^2} + {}^\bullet\lambda\, {}^\bullet y + g & (^\bullet e_2)\\ 0 = L^2 - (x^2+y^2) & (k_1)\\ 0 = L^2 - (x^2+y^2)^\bullet & (k_1^\bullet) \end{array}\right.$$

We resolve this conflict by applying causality Principle 1, which leads to erasing, in model (23), equations (k_1) and $(k_1^\bullet)$ at the instant of mode change $^\bullet\gamma{=}\text{F}, \gamma{=}\text{T}$. This yields:

$$\text{at} \begin{bmatrix} ^\bullet\gamma{=}\text{F} \\ \gamma{=}\text{T} \end{bmatrix} : \left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1)\\ 0 = y'' + \lambda y + g & (e_2)\\ 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2})\\ 0 = \lambda + s & (k_2) \end{array}\right. \tag{24}$$

System (24) uniquely determines all the leading variables from the state variables x, y and $x^\bullet, y^\bullet$. In turn, equations (k_1) and $(k_1^\bullet)$, which were erased from this model, are not satisfied. At the next instant, i.e., when $^{\bullet 2}\gamma{=}\text{F}, ^\bullet\gamma{=}\text{T}, \gamma{=}\text{T}$, the same argument is used. We thus erase, in model (23), the only equation (k_1) at the next instant. This yields:

$$\text{at} \begin{bmatrix} ^{\bullet 2}\gamma{=}\text{F} \\ ^\bullet\gamma{=}\text{T} \\ \gamma{=}\text{T} \end{bmatrix} : \left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1)\\ 0 = y'' + \lambda y + g & (e_2)\\ 0 = L^2 - (x^2+y^2)^\bullet & (k_1^\bullet)\\ 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2})\\ 0 = \lambda + s & (k_2) \end{array}\right. \tag{25}$$

Note that $(k_1^\bullet)$ is a consistency equation that is satisfied by the state variables $x^\bullet, y^\bullet$. In turn, equation (k_1), which was erased from this model, is not satisfied. At subsequent instants, equation erasure is no longer needed.

This completes the nonstandard structural analysis of the mode change $\gamma : \text{F}\to\text{T}$, i.e., when the rope gets straight.

5.2.2 Getting effective code for restart

Code generation for restarts consists in standardizing nonstandard systems (24) and (25), in a way similar to Section 5.1.4. We focus on the standardization of the mode change $\gamma : \text{F} \to \text{T}$, i.e., when the rope gets straight. Our task is to standardize systems (24) and (25), by targeting discrete-time dynamics, for the two successive instants composing the restart phase. This will provide us with restart values for positions and velocities.

Due to the expansion of derivatives in equations $(e_1, e_2, e_1^\bullet, e_2^\bullet)$, tensions λ and $\lambda^\bullet$ are both impulsive, hence so are s and $s^\bullet$ by $(k_2, k_2^\bullet)$. We eliminate the impulsive variables by ignoring $(k_2, k_2^\bullet)$, combining (e_1) and (e_2) to eliminate λ, and $(e_1^\bullet)$ and $(e_2^\bullet)$ to eliminate $\lambda^\bullet$. This yields:

$$\text{at} \begin{bmatrix} {}^\bullet\gamma{=}\text{F} \\ \gamma{=}\text{T} \end{bmatrix} : \begin{cases} 0 = y''x + gx - x''y \\ 0 = L^2{-}(x^2{+}y^2)^{\bullet 2} \end{cases} \tag{26}$$

$$\text{at} \begin{bmatrix} {}^{\bullet 2}\gamma{=}\text{F} \\ {}^\bullet\gamma{=}\text{T} \\ \gamma{=}\text{T} \end{bmatrix} : \begin{cases} 0 = y''x + gx - x''y \\ 0 = L^2{-}(x^2{+}y^2)^{\bullet} \\ 0 = L^2{-}(x^2{+}y^2)^{\bullet 2} \end{cases} \tag{27}$$

In System (26), we expand second derivatives using (20), whereas in System (27) we expand them using (21). Consequently, (26) has dependent variables $x^{\bullet 2}, y^{\bullet 2}$, whereas (26) has dependent variables $x'^\bullet, y'^\bullet$. We are now ready to standardize the two systems.

System (26) to define restart positions: We expand second derivatives using (20):

$$\begin{cases} 0 = (y^{\bullet 2} - 2y^\bullet + y)x - (x^{\bullet 2} - 2x^\bullet + x)y + \partial^2 gx \\ 0 = L^2{-}(x^2{+}y^2)^{\bullet 2} \end{cases} \tag{28}$$

Setting $\partial = 0$ in this system yields a structurally regular system. We can then invoke the following result, proved in [3]:

Theorem 1 (standardizing systems of equations) *For $\mathbf{H} : \mathbb{R}^{n+1} \to \mathbb{R}^n$ a $\mathcal{C}^1$ (standard) function, consider the nonstandard system of equations $\mathbf{H}(\partial, X) = 0$ where X is a n-vector of variables. If system $\mathbf{H}(0, X) = 0$ is structurally nonsingular, then setting $\partial = 0$ in system $\mathbf{H}(\partial, X) = 0$ yields the correct standardization of it, meaning that the solution $x_*(\partial)$ of $\mathbf{H}(\partial, X) = 0$ standardizes as the solution x_* of $\mathbf{H}(0, X) = 0$.*

Thus, the correct standardization of System (28) is the following system:

$$\begin{cases} 0 = (y^{\bullet 2} - 2y^\bullet + y)x - (x^{\bullet 2} - 2x^\bullet + x)y \\ 0 = L^2{-}(x^2{+}y^2)^{\bullet 2} \end{cases} \tag{29}$$

In the resulting system, we interpret x and $x^\bullet$ as the left-limit x^- of state variable x in previous mode, and $x^{\bullet 2}$ as the restart value x^+ for the new mode. This yields

$$\begin{cases} 0 = (y^+ - y^-)x^- - (x^+ - x^-)y^- \\ 0 = L^2{-}(x^2{+}y^2)^+ \end{cases} \tag{30}$$

which determines the restart values for positions. The constraint that the rope is straight is satisfied. Furthermore, as $0 = L^2{-}(x^2{+}y^2)^-$ also holds (the rope is straight at the mode change), $x^+ = x^-, y^+ = y^-$ is the unique solution of (30): positions are continuous.

System (26) to define restart velocities: We expand second derivatives using (21):

$$\begin{cases} 0 = (y'^{\bullet} - y')x - (x'^{\bullet} - x')y + \partial.gx \\ 0 = L^2 - (x^2+y^2)^{\bullet} \\ 0 = L^2 - (x^2+y^2)^{\bullet 2} \end{cases} \tag{31}$$

By expanding $x^{\bullet 2} = x^{\bullet} + \partial x'^{\bullet}$, the right-hand side of the last equation rewrites

$$\begin{aligned} L^2-(x^2+y^2)^{\bullet 2} = \quad & L^2-(x^2+y^2)^{\bullet} \\ + \; & 2\partial(x^{\bullet}x'^{\bullet} + y^{\bullet}y'^{\bullet}) \\ + \; & \partial^2\big((x'^{\bullet})^2 + (y'^{\bullet})^2\big) \\ = \quad & 0 \quad \text{(using (31))} \\ + \; & 2\partial(x^{\bullet}x'^{\bullet} + y^{\bullet}y'^{\bullet}) + O(\partial^2) \end{aligned} \tag{32}$$

Using this expansion, setting $\partial = 0$ in (32) yields

$$\begin{cases} 0 = (y'^{\bullet} - y')x - (x'^{\bullet} - x')y \\ 0 = x^{\bullet}x'^{\bullet} + y^{\bullet}y'^{\bullet} \end{cases} \tag{33}$$

where the dependent variables are now $x'^{\bullet}, y'^{\bullet}$—other variables are state variables whose values were set at previous time steps. System (33) is structurally regular, hence, it is the correct standardization of System (31).

To get effective code for restart, we perform, in (32), the following substitutions, where superscripts $^-$ and $^+$ denote left- and right-limits, and continuity of positions is used:

$$x = x^- \;;\; x^{\bullet} = x^+ \quad \text{and} \quad x' = x'^- \;;\; x'^+ = x'^{\bullet} \tag{34}$$

and similarly for y. This finally yields

$$\begin{cases} 0 = (y'^+ - y'^-)x^- - (x'^+ - x'^-)y^- \\ 0 = x^+x'^+ + y^+y'^+ \end{cases} \tag{35}$$

System (35) determines x'^+ and y'^+, which are the velocities for restart. The second equation guarantees that the velocity will be tangent to the constraint. With (30) and (35), we determine the restart conditions for positions and velocities. Invariants from the physics are satisfied.

Our reasoning so far produces a behavior in which the two modes (free motion and straight rope) gently alternate; the system always stays in one mode for some positive period of time before switching to the other mode.

This indeed amounts to assuming that the impact is totally inelastic at mode change, an assumption that was not explicit at all in (22)***.*** So, what happened? In fact, ***the straight rope mode was implicitly assumed to last for at least three nonstandard successive instants, since we allowed ourselves to shift*** (k_1) ***twice.***

5.2.3 Handling transient modes

Let us instead assume elastic impact, represented by the cascade of mode changes $\gamma : \text{F} \to \text{T} \to \text{F}$, reflecting that the straight rope mode is *transient* (it is left immediately after being reached).

Consider again model (22). We regard the instant of the cascade when $\gamma = \text{T}$ occurs as the current instant. We cannot add latent equations by simply shifting (k_1), since these shifted versions are not active in the mode $\gamma = \text{F}$. Set

$$\begin{aligned} S(\text{T}) &= \{(e_1), (e_2), (k_1), (k_2)\} \\ S(\text{F}) &= \{(e_1), (e_2), (k_3), (k_4)\} \end{aligned}$$

Systems $S^{\bullet}(\mathrm{T})$ and $S^{\bullet}(\mathrm{F})$ are obtained by shifting once the equations constituting $S(\mathrm{T})$ and $S(\mathrm{F})$; systems $S^{\bullet k}(\mathrm{T})$ and $S^{\bullet k}(\mathrm{F})$ are defined similarly for all $k \in \mathbb{N}$. Consider the *differentiation array* originally proposed by [9], except that we take into account the trajectory $\mathrm{T}, \mathrm{F}, \mathrm{F}, \ldots$ for guard γ. Using shifting instead of differentiation yields the following *difference array*:

$$\mathcal{A}_n(S) =_{\mathrm{def}} \left[\begin{array}{cccc} S(\mathrm{T}) & S^{\bullet}(\mathrm{F}) & S^{\bullet 2}(\mathrm{F}) & \ldots \quad S^{\bullet n}(\mathrm{F}) \end{array} \right]^T \tag{36}$$

The dependent variables of System $\mathcal{A}_n = 0$ are $x^{\bullet 2}, y^{\bullet 2}, \lambda$, whereas $x^{\bullet(k+2)}, y^{\bullet(k+2)}, \lambda^{\bullet(k)}, k > 0$ must be eliminated. We look for the smallest n such that $\mathcal{A}_n = 0$ is structurally nonsingular in this sense. Unfortunately, although shifting (k_4) twice in System (22) produces one more equation involving the leading variables $x^{\bullet 2}, y^{\bullet 2}$, this equation also involves the new variable $s^{\bullet 2}$, which keeps the augmented system underdetermined; shifting other equations fails as well. Therefore, the structural analysis rejects this model as being underdetermined at transient mode $\gamma = \mathrm{T}$.

The user is then asked to provide one more equation. For example, they could specify an impact law for the velocity y' by providing the equation $(y')^+ = -(1-\alpha)(y')^-$, where $0 \leq \alpha < 1$ is a fixed damping coefficient. This is reinterpreted in the nonstandard domain as $y'^{\bullet} = -(1-\alpha)y'$, yielding the following refined system for use at mode $\gamma{=}\mathrm{T}$ within the cascade $\gamma{:}\mathrm{F}{\to}\mathrm{T}{\to}\mathrm{F}$:

$$\left\{ \begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = y'^{\bullet} + (1-\alpha)y' & (\tau_1) \\ 0 = L^2 - (x^2 + y^2) & (k_1) \\ 0 = \lambda + s & (k_2) \end{array} \right. \tag{37}$$

The modified difference array is now structurally nonsingular. The so modified model is accepted and two-step restart code for the mode change is generated as before.

5.2.4 Declaring transient modes

Through the Cup-and-Ball example, *we demonstrated the need for the following user-given information: is the current mode long or transient?* Long / Transient is an information regarding modes, that cannot be found by an automatic inspection of the model. It must be inferred from understanding the system physics and must be manually specified. The natural way of performing this is to provide a different syntax for specifying long modes on the one hand, and events corresponding to transient modes on the other hand (mode changes separating two successive long modes need not be specified).

The '`if`' and '`when`' statements of the Modelica language are fit candidates for this purpose. We devote the '`if`' statement to long-lasting modes specified by a predicate, while the '`when`' statement, pointing to the event when a predicate switches from F to T, could be further restricted to be a zero-crossing condition, by which a $\mathbb{R}$-valued expression crosses zero from below [5]. Using this feature, the Cup-and-Ball example with elastic impact is specified as follows:

$$\left\{ \begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma = [s^- \leq 0]; \gamma(0) = \mathrm{F} & (k_0) \\ \texttt{when } \gamma \texttt{ then} & y'^+ = -\alpha y'^- & (\tau_1) \\ \texttt{if not } \gamma \texttt{ then} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2+y^2)) - s & (k_4) \end{array} \right.$$

5.3 Multimode structural analysis in general

We consider multimode DAE systems possessing *long modes* (having DAE-based dynamics for a positive duration) alternating with finite cascades of *transient modes* (having a zero duration,

such as the straight rope mode in the Cup-and-Ball model with elastic impact).

We assume that the information regarding the type of a mode (long vs. transient) is known by the compiler—the two different Modelica primitives `if` and `when` should be used to declare long and transient modes, respectively.

In addition, we require that the current mode is defined by the left-limits of some predicates, see the reasoning leading to the corrected model (22) for the Cup-and-Ball.

For such models, the structural analysis proceeds according to the following steps:

1. The multimode model is mapped to its nonstandard expansion by using a first-order explicit Euler expansion for derivatives, with infinitesimal time step ∂, and mapping left-limits to values at the previous instant. In particular, the mode at each nonstandard instant is known at the end of the previous instant.

2. The structural analysis for each specific mode is performed, depending on its long/transient type:

   - If the mode is long, then classical structural analysis applies: by, e.g., using Pryce's Σ-method [24], latent equations are added for the DAE system associated to each long mode;
   - Alternatively, if the mode is transient, a structural analysis of the *difference array* associated to the considered cascade of transient modes is performed.

3. Having done this, given the mode at the current instant:

   - If no mode change occurs, then the (classical) mode-specific structural analysis applies;
   - Otherwise, the conflict that may possibly exist between consistency equations of the current mode and leading equations of the previous mode is analyzed, using the Dulmage-Mendelsohn decomposition; conflicting subsystems are identified and the equations from the current instant that cause conflicts are erased.

Implementing the multimode structural analysis in the above described form would be very inefficient. As announced in Section 4.8, the principles of the approach proposed in Section 4 extend to the structural analysis of mode changes.

5.4 Discussion

The approach developed in Sections 5.1.4 and 5.2.2 is a systematic way to define the solution of a multimode DAE system, that can be generalized to large-scale and/or multi-physics models. The use of implicit "dual" representations, such as the ones used in the IsamDAE tool, is currently being studied for efficiency purposes.

However, this approach still presents several other difficulties regarding its possible mechanization in a tool. We list below the main three developments that are required for its automatization and hint at how we are currently addressing them:

- *Identification of impulsive variables.* We present in Section 6 a calculus for this, which is ready for automatization (this is under development in our IsamDAE tool).

- *Elimination of impulsive variables.* This is easy if impulsive variables enter linearly in the model—this was the case for the Clutch and Cup-and-Ball examples. It is highly costly but still doable if impulsive variables enter polynomially in the model, but cannot be done

practically in all other cases. As a result, the elimination of such variables only seems adapted in practice to a subclass of multimode models in which these variables occur in a linear fashion. Alternative approaches for the handling of impulsive variables are proposed in Section 6.2.2.

- *Clever choice of how to map nonstandard variables to restart conditions.* This was straightforward for the Clutch, but definitely not for the Cup-and-Ball (Section 5.2.2), where expansion (20) for the derivatives was used for resetting positions, whereas expansion (21) was used for resetting velocities. Works are in progress for automating this choice.

6 Impulse analysis

As discussed above, in order to address Issue 2 further, a specific focus is required on the detection of impulsive behaviors. The identification of impulsive variables at mode changes would indeed pave the way toward effective methods for computing restart conditions at a mode change, either by eliminating impulsive variables (when it is actually feasible) or by handling them in a specific way to avoid numerical divergence.

In this section, we propose a calculus by which impulsive variables can be identified at compile time, with a quantitative characterization of their magnitude order in terms of the discretization time step. The approach is developed on the Cup-and-Ball example, then generalized. Possible methods that can be used for computing actual restart conditions, from the knowledge of impulsive variables and their respective magnitude orders, are also illustrated on the same example.

6.1 The Cup-and-Ball

6.1.1 Impulse analysis at mode changes

Here we focus on identifying possible impulsive behaviors at mode change $\gamma : \mathrm{F} \to \mathrm{T}$. This is achieved by analyzing nonstandard systems (24) and (25) defining the values for restart. The intent is that the former will set the restart positions, whereas the latter will set the restart velocities.

Our impulse analysis not only identifies impulsive variables but also quantifies their order of magnitude, thanks to the following notion of impulse order:

Definition 1 (Impulse order and analysis) *Given a nonstandard system of equations E defining the values for restart, say that a dependent variable x has* impulse order $\mathfrak{o}\in\mathbb{R}$ in E, *if the solution of system E is such that $x\partial^{-\mathfrak{o}}$ is provably a finite non-zero (standard) real number.*

Let $[\![x]\!]$ denote the impulse order of x. Say that x is impulsive *if $[\![x]\!] > 0$. By convention $[\![0]\!] = -\infty$.*

The impulse analysis *of a system of equations S is the system of constraints satisfied by the impulse orders of the dependent variables of S.*

Impulse analysis relies on the following generic assumption, which expresses that DAE within long modes must be reinitialized with finite values for the state variables:

Assumption 1 *State variables are not impulsive; that is, for any state variable v, one has $[\![v]\!] \leq 0$.*

As an example, if, in the new mode, a variable x is differentiated up to order n, then its $(n-1)$-th derivative is a state variable and thus subject to Assumption 1. Consequently, its k-th order derivatives for $k = 0, \ldots, n-2$ are continuous at the considered mode change.

We are now ready to successively analyze Systems (24) and (25).

System (24): The state variables are x, y, x', y'. By Assumption 1, we get the following prior information, which expresses that velocities are not impulsive:

$$[\![x'^{\bullet} - x']\!] \leq 0 \quad ; \quad [\![y'^{\bullet} - y']\!] \leq 0 \ . \tag{38}$$

Conditions (38) imply that positions should be continuous. While performing our impulse analysis, we include equation (21) relating second derivatives and first derivatives. System (24) involves equation $(e_1) : x''+\lambda x=0$, which, by using (21), rewrites

$$x'^{\bullet} - x' + \partial\lambda x = 0 \ . \tag{39}$$

By (38), equation (39) implies $[\![\lambda]\!] \leq 1$. Exploiting all equations of System (24) yields the following information

$$[\![\lambda]\!] = [\![s]\!] \leq 1 \ , \tag{40}$$

whereas other dependent variables have impulse order zero. System (25) is handled similarly, with the same conclusion. In Section 6.2, we mechanize the impulse analysis for an arbitrary restart system. Prior to doing this, we now explain how this impulse analysis can be exploited for generating effective code for restart.

6.1.2 Computing restart conditions

Code generation for restarts consists in standardizing nonstandard systems (24) and (25). See Section 5.1.2 for the meaning of "standardization". Standardizing systems of equations requires more care than standardizing numbers, due to impulsive behaviors and singularity issues that result.

We can exploit the impulse analysis using three different methods. The first method is mostly described for didactic purposes, as it requires the symbolic elimination of variables, which can be very costly or even impossible in nonlinear systems. In practice, the second and third methods shall be used.

Eliminating impulsive variables When this is practical, the simplest method from a conceptual point of view is to eliminate impulsive variables from the restart system, as they are of no use for restarting the new mode. On the Cup-and-Ball example, this was detailed in Section 5.2.2.

This is a satisfactory solution when the elimination of impulsive variables is practical. In our example, they entered linearly in the restart system, so that elimination was straightforward. When this is not the case, elimination becomes costly or even impossible. Moreover, generalizing and mechanizing this elimination process appears to be a very difficult task. We thus need to look for alternatives for computing the velocities for restart.

Rescaling impulsive variables Focus again on System (25). Impulse analysis told us that λ, s both have impulse order ≤ 1. We thus rescale them accordingly:

$$\widehat{\lambda} =_{\text{def}} \partial^1\!\times\!\lambda \quad \text{and} \quad \widehat{s} =_{\text{def}} \partial^1\!\times\!s \tag{41}$$

Using this rescaling together with expansion (21), System (25) rewrites

$$\begin{cases} 0 = x'^{\bullet} - x' + \widehat{\lambda}x & (e_1) \\ 0 = y'^{\bullet} - y' + \widehat{\lambda}y + \partial g & (e_2) \\ 0 = L^2 - (x^2+y^2)^{\bullet} & (k_1^{\bullet}) \\ 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ 0 = \widehat{\lambda} + \widehat{s} & (k_2) \end{cases} \tag{42}$$

In System (42), $(k_1^\bullet)$ is a consistency equation satisfied as a result of performing (24) at the previous instant. We can also discard equation (k_2), which only serves to determine the auxiliary variable s. Thus, we are left with the sub-system collecting equations $(e_1), (e_2), (k_1^{\bullet 2})$. We can again expand the right-hand side of $(k_1^{\bullet 2})$ by using (31). In the resulting system, we can safely set $\partial \leftarrow 0$ since it yields the following structurally regular system:

$$\left\{\begin{array}{ll} 0 = x'^{+} - x'^{-} + \widehat{\lambda} x^{-} & (e_1) \\ 0 = y'^{+} - y'^{-} + \widehat{\lambda} y^{-} & (e_2) \\ 0 = 0 = x^{+} x'^{+} + y^{+} y'^{+} & (k_1^{\bullet 2}) \end{array}\right. \tag{43}$$

System (43) determines $x'^{+} = x'^{\bullet}, y'^{+} = x'^{\bullet}$, and the rescaled impulsive tension $\widehat{\lambda}$, as functions of state variables x', y', x, y, which were identified with the left-limits of velocities and positions at previous mode. Note that eliminating the rescaled tension $\widehat{\lambda}$ from System (43) yields System (35).

Rescaling impulsive variables is simpler than eliminating them. This method is also promising in terms of designing and implementing algorithms for its mechanization, as the computation of the impulse orders amounts to finding a minimal solution to a system of linear unilateral constraints. Unfortunately, it does not work in full generality since impulse orders can be infinite, as the following example shows:

$$x = \exp(y/\partial)\,,$$

where y is known to have impulse order zero. Indeed, the impulse order of $(y/\partial)^n$ is n. Since the exponential expands as a power series of infinite support, we deduce that the impulse order of $\exp(y/\partial)$ is the maximum of all impulse orders of $(y/\partial)^n$, hence it is infinite. Thus, impulsive variable x cannot be rescaled.

The last method addresses such cases, at the price of a possibly poor numerical conditioning.

Bruteforce solving of the restart system When none of the above methods apply, it is still possible to solve system (42) with $\partial = \delta$ (a small positive time step) for the original variables λ and s, without rescaling them.

Then, it is proved in [3], see also [2] that *solving these systems for their dependent variables and then discarding the values found for the impulsive variables yields a converging approximation for the states and velocities at restart.* First numerical experiments on toy examples showed no issue as long as the time step δ was kept reasonably high.

Of course, without rescaling, the numerical conditioning is less favorable, so that rescaling is recommended when impulse orders are finite. Work is in progress for the implementation of this method, coupled with the rescaling of impulsive variables when they have finite order.

6.2 General Impulse Analysis

Here, we explain how the reasoning used for the Cup-and-Ball example can be mechanized as a compilation stage following multimode structural analysis. The reader is referred to Section 5.3 for an overview of multimode structural analysis including mode changes.

6.2.1 General Rules of Impulse Analysis

Problem setting

Restart systems of equations, as resulting from the structural analysis at mode changes, are nonstandard systems of equations of the following generic form:

$$\text{expand } X' \text{ as } \tfrac{X^\bullet - X}{\partial} \text{ in } \; 0 = \mathbf{H}(X', X^\bullet, V, X) \tag{44}$$

where V collects the algebraic variables, X collects the state variables, and $\frac{X^\bullet - X}{\partial}$ is the nonstandard semantics of X'. $\mathbf{H}(\cdot)$, seen as a vector function in its arguments, is by itself standard, since the equations of system $0 = \mathbf{H}$ are obtained by shifting or differentiating equations specified by the user. The reason for (44) being nonstandard is indeed twofold:

1. Since $X^\bullet$ is involved, the infinitesimal ∂ occurs in time; and
2. Since X' is involved, the infinitesimal ∂ occurs both in time and *space,* due to the expansion $X' \leftarrow \frac{X^\bullet - X}{\partial}$.

The occurrence of ∂ in time is not an issue: shifted state variables will correspond to restart values for states, whereas non-shifted ones correspond to values prior to the change. In contrast, the occurrence of ∂ in space is the root cause of possible impulsive behaviors. Identifying them is the subject of impulse analysis.

The rules of impulse analysis

We now develop the *impulse analysis* introduced in Definition 1. This analysis is useful as a postprocessing of structural analysis, prior to generating effective code for restarts. Note that Assumption 1 is still enforced in what follows.

Figures 18 and 19 display the rules defining the translation of a system of equations of the form (44) into its impulse analysis, for the restricted class where only rational expressions are involved. Figure (18) describes the syntax of a mini-language specifying such systems of equations.

$$\begin{array}{lcl} e & ::= & 0 \mid c \mid \partial \mid x \mid e^c \mid e + e \mid e \times e \\ E & ::= & e = e \mid E \texttt{ and } E \end{array}$$

Figure 18: Syntax: E is a system of one or several equations $e = e$. An expression e is 0, a nonzero (standard) real constant c, the infinitesimal ∂, a variable x, the monomial e^c, a sum, or a product.

$$\begin{array}{lrcl} \text{(R1)} & [\![0]\!] & = & -\infty \\ \text{(R2)} & [\![c]\!] & = & 0 \\ \text{(R3)} & [\![\partial]\!] & = & -1 \\ \text{(R4)} & [\![e^c]\!] & = & c[\![e]\!] \\ \text{(R5)} & [\![e_1 \times e_2]\!] & = & [\![e_1]\!] + [\![e_2]\!] \\ \text{(R6)} & [\![e_1 + e_2]\!] & \leq & \max\{[\![e_1]\!], [\![e_2]\!]\} \end{array}$$

$$\frac{E \vdash e = e'}{[\![E]\!] \vdash [\![e]\!] = [\![e']\!]} \qquad \text{(R7)}$$

$$\frac{\left.\begin{array}{l} E \vdash x = y + e \quad \text{or} \\ E \vdash 0 = y - x + e \end{array}\right\} \text{ and } \quad E \nvdash y = x - e}{E \vdash E \texttt{ and } y = x - e} \qquad \text{(R8)}$$

Figure 19: Rules: The left column displays the impulse order of the primitive expressions. Rule (R7) indicates that $[\![e]\!] = [\![e']\!]$ is an equation of the impulse analysis $[\![E]\!]$ if $e = e'$ is an equation of E; rule (R8) indicates that, if E involves the equation $x = y + e$ but not the equation $y = x - e$, then we augment E with the latter, i.e., we saturate E with the rule $x = y + e \implies y = x - e$.

The left column of Figure 19 gives the rules for mapping expressions to their corresponding impulse orders.

The reason for the inequality in (R6) is that in the sum $e_1 + e_2$, the dominant terms in the expansion of e_i as a series over ∂ may cancel each other. For an example of this, see equation

(e_2) in System (25): rewriting this equation as $-g = y'' + \lambda y$, we see a case of strict inequality for (R6) since gravity g has order zero, whereas it is equal to the difference of two terms of order one.

We will use Rule (R6) in the following way, thereby reinforcing it. Consider an equation

$$e : z = x + y \ .$$

We can rewrite e in the following equivalent ways: $0 = x + y - z$, $x = z - y$, or $y = z - x$. To each of them we apply the max rule. This yields the following system of constraints, called the *impulse analysis of equation* e:

$$\left\{ \begin{array}{l} [\![z]\!] \leq \max\{[\![x]\!], [\![y]\!]\} \\ [\![0]\!] \leq \max\{[\![x]\!], [\![y]\!], [\![z]\!]\} \\ [\![x]\!] \leq \max\{[\![z]\!], [\![y]\!]\} \\ [\![y]\!] \leq \max\{[\![x]\!], [\![z]\!]\} \end{array} \right. \tag{45}$$

Note that the constraint $[\![0]\!] \leq \ldots$ is vacuously satisfied since $[\![0]\!] = -\infty$. Then, among the three nontrivial inequalities of (45), at least two of them must be saturated. We will use impulse analysis (45) for handling sums of terms. This reinforcement of the max rule is formalized by Rule (R8) of Figure 19, which mechanizes the association, to any equation, of its different rewritings.

Using the rules of Figures 18 and 19 in the numerical expressions, we map any system of equations of the form (44) into a system of constraints over impulse orders.

To cover functions beyond polynomials, we need to extend $\mathbb{R} \cup \{-\infty\}$ with $+\infty$. In this extension, we take the convention that $-\infty + \infty = -\infty$, justified by both Rules (R1,R5) and the equality $0 \times x = 0$ for any nonstandard x. For functions $f(x) = \sum_{k=0}^{\infty} a_k x^k$ that can be represented as absolutely converging power series, we then get

$$[\![f(x)]\!] = [\![\sum_{k=0}^{\infty} a_k x^k]\!] = [\![x]\!].\sup(A) \ , \tag{46}$$

where $A = \{k \mid a_k \neq 0\}$ is the support of the series and $\sup(A)$ is the supremum of set A. In particular, if $[\![x]\!] > 0$ and if the support of the series is infinite, we get $[\![f(x)]\!] = +\infty$.

6.2.2 Computing conditions for restarts

So far, Rules (R1)–(R8) of the impulse analysis apply to any system of nonstandard equations. Here we particularize the impulse analysis to systems of equations of the form (44), where the only reason for ∂ to occur is the expansion of derivatives using the Euler scheme:

$$0 = \mathbf{H}\left(\frac{X^\bullet - X}{\partial}, X^\bullet, V, X\right)$$

The dependent variables are $X^\bullet, V$. It will be convenient to introduce the auxiliary variables

$$U =_{\text{def}} X^\bullet - X \, ,$$

so that the systems we consider take the following form, where $X^\bullet, V, U$ are the dependent variables:

$$\left\{ \begin{array}{rcl} 0 & = & \mathbf{H}\left(\frac{U}{\partial}, X^\bullet, V, X\right) \\ U & = & X^\bullet - X \end{array} \right. \tag{47}$$

The following condition for System (47) can be assumed, based on physical considerations (restart values for an ODE or a DAE cannot be impulsive):

Assumption 2 *Since X is a state, both X (a known value) and $X^\bullet$ must be finite.*

First, the impulse orders $[\![X]\!]$ are all known, from previous nonstandard instants. Next, from Assumption 2 we deduce the inequalities:

$$[\![X^\bullet]\!] \leq 0 \quad \text{and} \quad [\![U]\!] \leq 0\,. \tag{48}$$

The impulse orders $[\![V]\!]$ are a priori unknown. We have, however, more prior information, thanks to the structural analysis. From the structural analysis at the considered mode change, we know which consistency equations of the new mode were conflicting with the dynamics of previous mode. Formally, call $G{=}0$ the subsystem collecting all the equations that were erased while solving this conflict—for the Cup-and-Ball model (23), at the instant of mode change ${}^\bullet\gamma{=}\mathrm{F}, \gamma{=}\mathrm{T}$, G collects the bodies of the two violated consistency constraints (k_1) and $(k_1^\bullet)$.

As a result, $G{=}0$ no longer holds at the considered mode change, and thus, G defines a tuple R of variables (one per entry of G) called *residuals*, by setting

$$R = G\,, \tag{49}$$

which are all finite and nonzero. An example of residual in the Cup-and-Ball is $r = L - (x^2 + y^2)$, which is both finite and nonzero at mode change ${}^\bullet\gamma{=}\mathrm{F}, \gamma{=}\mathrm{T}$. The residuals are found by the structural analysis.

Finally, the system of equations that we need to solve collects all the above items, namely:

$$\left\{\begin{array}{rcl} 0 & = & \mathbf{H}\left(\frac{U}{\partial}, X^\bullet, V, X\right) \\ U & = & X^\bullet - X \\ R & = & G \end{array}\right. \tag{50}$$

with dependent variables $X^\bullet, V, U, R$, and the following prior information on impulse orders is known:

$$[\![\frac{1}{\partial}]\!] = 1\ ;\ [\![X^\bullet]\!] \leq 0\ ;\ [\![U]\!] \leq 0\ ;\ [\![R]\!] = 0\,. \tag{51}$$

System (50) is then mapped to its impulse analysis by using Rules (R1–R8) of Figures 18 and 19.

A suitable constraint solver is then used to solve the resulting set of constraints on impulse orders, by using side information (51). The choice of an appropriate constraint solver remains to be done.

7 Conclusion

Although multimode DAE models can be described in the Modelica language since 2012, the handling of such models by the current Modelica tools is not satisfactory, with mathematically sound models yielding exceptions at runtime.

In this report, we first explained the reasons for this problem. We showed how its root cause is the fact that mainstream Modelica tools implement an approximate structural analysis, in which equations containing `if` − `then` − `else` statements are handled as if they were mode-independent smooth equations. Although this results in a huge reduction of compilation costs, it causes runtime errors for a large, not well-defined class of multimode models with mode-dependent structure.

A related issue, that we showed is equally problematic, is that of the handling of mode changes. We made the point that, for models with mode-dependent index and/or structure, computing

restart states at mode changes would require a new kind of structural analysis. This analysis should be performed at compile time, so that precise diagnostics can be returned to the user if their model is not well-determined (in some modes and/or at some mode changes), and possible impulsive behaviors at mode changes should be predicted and handled in a specific fashion.

[8] proposed an exact multimode structural analysis method, based on an implicit "dual" description of the mode-dependent structure of a model. Adapted data structures and specific algorithms (including, but not limited to, a parametric extension of Pryce's Σ-method) make it possible to perform the whole structural analysis process without enumerating modes; the output of this algorithmic suite provides all the information needed to generate correct and efficient simulation code for the input model.

Unfortunately, the Modelica compilers at the core of mainstream tools such as OpenModelica and Dymola were not designed to handle such mode-dependent data structures. An important rewriting of the backend of such tools would be needed to implement our approach. As an alternative, we proposed a source-to-source rewriting of Modelica models, based on the results of the exact multimode structural analysis and index reduction. The output model is correctly handled by existing Modelica tools and produces correct simulations, at the cost of computational overheads that proved reasonable for all the models we studied so far except rare pathological cases.

To address the second problem, we then proposed a discrete-time structural analysis that handles both continuous modes and mode changes in a unified framework. If mode changes are not well specified in a model, this analysis rejects the model and provides precise diagnostics of the under/over-specification responsible for this rejection; otherwise, it generates systems of equations determining, in a finite number of successive discrete time steps, the conditions for restart. This approach holds for models in which continuous modes alternate with finite cascades of mode change events.

At this stage, an important difficulty remained: the possible occurrence of impulsive behaviors at some mode changes, for some variables. Hence, we proposed a compile-time algorithm for identifying such impulsive behaviors and quantifying them in terms of their magnitude orders. We showed how this calculus can be formalized, and how its results can be used for the numerical handling of impulsive variables at mode changes.

All these results form a robust theoretical core for a complete compilation chain for multimode DAE systems. So far, only the exact structural analysis of long modes has been implemented in the IsamDAE tool, but empirical results seem to show that our "dual" representation of multimode models, along with associated data structures and algorithms, is the right path towards the scaling-up of multimode structural analysis. As a result, similar techniques are being designed for our structural analysis of mode changes, for the purpose of its efficient implementation in IsamDAE.

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

[1] Albert Benveniste, Benoît Caillaud, and Mathias Malandain. "Compile-Time Impulse Analysis in Modelica". In: *MODELICA 2021 - 14th International Modelica Conference*. Linköping, Sweden, Sept. 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03281394.

[2] Albert Benveniste, Benoît Caillaud, and Mathias Malandain. "Handling Multimode Models and Mode Changes in Modelica". In: *Modelica 2021 - 14th International Modelica Conference*. Linköping, Sweden, Sept. 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03281410.

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

[4] Albert Benveniste et al. "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. 2019, pp. 283–310. DOI: 10.1007/978-3-319-91908-9_16.

[5] Timothy Bourke and Marc Pouzet. "Zélus: A Synchronous Language with ODEs". In: *Hybrid Systems: Computation and Control (HSCC)*. ACM. Philadelphia, USA, Apr. 2013, pp. 113–118. DOI: 10.1145/2461328.2461348.

[6] Randall E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (Aug. 1986), pp. 677–691. DOI: 10.1109/tc.1986.1676819.

[7] Benoît Caillaud, Mathias Malandain, and Albert Benveniste. "A Reduced Index Mode-Independent Structure Model Transformation for Multimode Modelica Models". In: *MODELICA 2021 - 14th International Modelica Conference*. Linköping, Sweden, Sept. 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03320499.

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

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

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

[11] Dassault Systèmes AB. *Dymola official webpage*. Accessed: 2021-12-17. URL: https://www.3ds.com/products-services/catia/products/dymola/.

[12] Andrew L. Dulmage and Nathan S. Mendelsohn. "Coverings of Bipartite Graphs". In: *Canadian Journal of Mathematics* 10 (1958), pp. 517–534. DOI: 10.4153/CJM-1958-052-0.

[13] Hilding Elmqvist et al. "State Machines in Modelica". In: *Proc. of the Int. Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Modelica Association. Munich, Germany, Sept. 2012, pp. 37–46. DOI: 10.3384/ecp1207637.

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

[15] George Giorgidze and Henrik Nilsson. "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, 2011, pp. 138–155. ISBN: 978-3-642-24452-0. URL: https://dl.acm.org/doi/10.5555/2044476.2044484.

[16] Bertrand Jeannet. *BddApron*. Accessed: 2021-12-17. Aug. 2012. URL: http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/.

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

[18] Tom Lindstrøm. "An Invitation to Nonstandard Analysis". In: *Nonstandard Analysis and its Applications*. Ed. by N.J. Cutland. Cambridge Univ. Press, 1988, pp. 1–105.

[19] Sven Erik Mattsson, Martin Otter, and Hilding Elmqvist. "Modelica Hybrid Modeling and Efficient Simulation". In: *38th IEEE Conference on Decision and Control*. Ed. by IEEE. 1999, pp. 3502–3507.

[20] Sven Erik Mattsson and Gustaf Söderlind. "Index reduction in Differential-Algebraic Equations using dummy derivatives". In: *Siam J. Sci. Comput.* 14.3 (1993), pp. 677–692.

[21] The Modelica Association. *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.5*. Accessed: 2021-12-17. Feb. 2021. URL: https://specification.modelica.org/maint/3.5/MLS.pdf.

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

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

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

[25] Abraham Robinson. *Nonstandard Analysis*. ISBN 0-691-04490-2. Princeton Landmarks in Mathematics, 1996.

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

[27] Peter Schrammel. "Méthodes logico-numériques pour la vérification des systèmes discrets et hybrides. (Logico-Numerical Verification Methods for Discrete and Hybrid Systems)". Accessed: 2021-12-17. PhD thesis. Grenoble Alpes University, France, 2012. URL: https://tel.archives-ouvertes.fr/tel-00809357.

[28] A. Schrijver. *Theory of linear and integer programming*. Wiley, Apr. 1998.

[29] Dirk Zimmer. "Equation-Based Modeling of Variable-Structure Systems". PhD thesis. ETH Zürich, No. 18924, 2010. URL: http://www.inf.ethz.ch/personal/fcellier/PhD/zimmer_phd.pdf.

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399