Structural Analysis of Multi-Mode DAE Systems

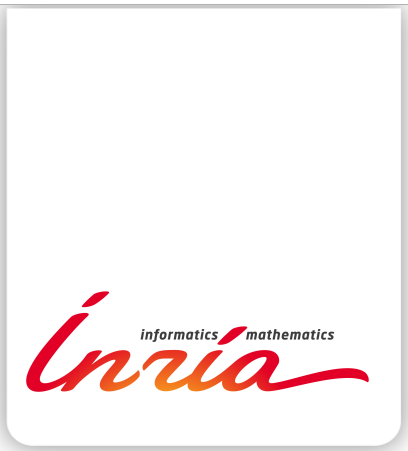
Albert Benveniste, Benoît Caillaud, Marc Pouzet, Hilding Elmqvist, Martin Otter

Structural Analysis of Multi-Mode DAE Systems

Albert Benveniste , Benoît Caillaud , Marc Pouzet
Hilding Elmqvist , Martin Otter

RESEARCH REPORT
N° 8933
July 2016
Project-Teams Hycomes, Parkas

ISSN 0249-6399 ISRN INRIA/RR--8933--FR+ENG



Structural Analysis of Multi-Mode DAE Systems

Albert Benveniste [*], Benoît Caillaud [†], Marc Pouzet [‡]
Hilding Elmqvist [§], Martin Otter [¶]

Project-Teams Hycomes, Parkas

Research Report n° 8933 — July 2016 — 29 pages

This work was supported by the ITEA/Modrio project.

[*] INRIA, Rennes, France. corresp. author: Albert.Benveniste@inria.fr
[†] INRIA, Rennes, France
[‡] Ecole Normale Supérieure (ENS), Paris
[§] Mogram AB, Lund, Sweden
[¶] DLR, Wessling, Germany

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Abstract: Multi-mode DAE systems constitute the mathematical model supporting physical modeling languages such as Modelica. Multi-mode DAE systems are systems of equations of the form

$$\texttt{if } \gamma_i \texttt{ do } f_i(\text{the } x_j \text{ and derivatives of them}) = 0\,, \tag{1}$$

where γ_i is a predicate involving system variables and guarding the DAE $f_i(\dots) = 0$. Single-mode (i.,e., usual) DAE systems face the issue of *differentiation index,* originating from the possible existence of so-called "latent constraints". DAE systems having index 2 are not comfortable to solvers and those having index ≥ 3 are problematic or even out of reach. Unfortunately, complex Modelica models may lead to such cases, even more so when composed with the control software. The *Structural Analysis* of DAE systems, of which the popular Pantelides algorithm is representative, is a symbolic pre-processing preparing the simulation for DAE systems of large index.

Unlike for single-mode DAE systems, however, no theory exists that can support the structural analysis of multi-mode DAE systems. As a practical consequence, Modelica compilers handle some models and refuse other ones, including some models that are sound and useful to consider, from physical and practical points of view. Worse, there is no clear definition of the class of Modelica models that can/cannot be handled. As we said, the reason for this is the lack of rigourous mathematical support for the structural analysis of multi-mode DAE systems. It is generally considered that the structural analysis is mode-dependent. This, however, tells nothing about how mode changes must be handled. Even more so when mode changes occur in cascades.

In this paper we develop a comprehensive mathematical approach to the structural analysis of multi-mode DAE systems. We first observe that a single-mode DAE system and its explicit first order Euler approximation possess identical structural analyses, regardless of the step size. We thus propose to take a step size that is *infinitesimal* in the sense of non-standard analysis, so that the Euler scheme is no longer an approximation but rather a re-interpretation of the original system, in the non-standard domain. Building on top of this, we are able to propose a full mathematical study of the structural analysis of multi-mode DAE systems and we show how simulation code can be deduced from this analysis.

Key-words: Hybrid systems, DAE, index, nonstandard analysis

Analyse Structurelle des systèmes de DAE multi-modes

Résumé : Les systèmes de DAE multi-modes constituent le modèle mathématique sous-jacent à la modélisation de systèmes physiques par des langages tels que Modelica. Ils sont de la forme

$$\texttt{if } \gamma_i \texttt{ do } f_i(\text{les } x_j \text{ et leurs dérivées successives}) = 0\,, \tag{2}$$

où γ_i est un prédicat en les variables du système, servant de garde à l'équation $f_i(\dots) = 0$. On associe aux systèmes de DAE mono-modes (c'est-à-dire les systèmes de DAE usuels) une notion d'*index de différentiation,* relié à l'existence de contraintes dites "cachées", que l'on révèle par differentiation d'un ensemble bien choisi d'équations du système. L'index de différentation est alors le plus grand parmi les nombres minimaux de fois qu'il faut différentier chaque équation pour faire apparaitre toutes les contraintes cachées. Les systèmes de DAE d'index 2 sont délicats à simuler par les solveurs tandis que ceux d'index ≥ 3 sont problématiques, voire impossibles à simuler. Malheureusement, les systèmes de DAE obtenus à partir de modèles Modelica peuvent être d'ordre élevé, d'autant plus lorsqu'ils sont en interaction avec un modèle représentant le logiciel qui les contrôle. L'*analyse structurelle* des systèmes de DAE est un prétraitement symbolique effectuant, entre autres, la réduction d'index, permettant ainsi le travail des solveurs dans des conditions satisfaisantes.

Malheureusement, il n'existe pas de définition mathématique claire de ce qu'est l'analyse structurelle pour les systèmes multi-modes. En conséquence de quoi les compilateurs existants pour Modelica (et les langages de la même classe) traintent convenablement certains modèles mais en refusent d'autres, qui sont pourtant dotés de sens physique. De plus, il n'existe pas de caractérisation claire de la classe des modèles qui sont bien traités. Habituellement, on se contente de parler de l'index du système dans un mode particulier. Mais le problème est que ceci n'aide en rien à comprendre comment doivent les transitons entre modes être négociées, avec leurs actions de réinitialisation.

Dans cet article nous développons une approche mathématique claire pour l'analyse structurelle des systèmes de DAE multi-modes. Notre approche repose sur l'utilisation de l'analyse non-standard, qui nous permet de regarder, pour l'analyse symbolique qu'est l'analyse structurelle, le système dynamique considéré comme étant à temps discret, avec un pas de temps infinitésimal.

Mots-clés : Systèmes hybrides, DAE, index, analyse non standard

CONTENTS

I. INTRODUCTION

A. Background on DAE systems and index

We consider DAE initial value problems in n dependent variables $x_j = x_j(t)$ where t is the time:

$$f_i(\text{the } x_j \text{ and time derivatives of them}) = 0, 1 \leq i \leq n \quad (3)$$

To simplify notations, we omit the possible dependence of f_i on time t; it can be encoded through an additional dependent variable τ such that $\tau'(t) \equiv 1$. Regarding (3) as a system of algebraic equations relating the x_j and derivatives of them seen as independent variables, suppose that the following holds:

$$\text{we can solve (3) for the highest derivatives of the } x_j\text{s in terms of lower ones, i.e.:} \quad (4)$$

$$\begin{cases} x_1^{(d_1)} &= g_1\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \\ x_2^{(d_2)} &= g_2\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \\ &\vdots \\ x_n^{(d_n)} &= g_n\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \end{cases} \quad (5)$$

Then, putting (5) in state-space form makes it an ODE.

Condition (4) may not hold in general, due to so-called "latent equations". The latter are exhibited by differentiating one or more times selected equations from (3). That is, denoting by $f^{(c)} =_{\text{def}} \frac{d^c}{dt^c} f$ the c-th time derivative of a function f of the x_j and derivatives of them, and setting

$$\begin{aligned} F &=_{\text{def}} (f_1, \ldots, f_1^{(c_1)}, \ldots, f_n \ldots, f_n^{(c_n)})^T \\ Z &=_{\text{def}} (x_1, \ldots, x_1^{(d_1-1)}, \ldots, x_n \ldots, x_n^{(d_n-1)})^T \\ X &=_{\text{def}} (x_1^{(d_1)}, \ldots, x_n^{(d_n)})^T \end{aligned}$$

where d_j is the highest differentiation degree of x_j in F, we must check whether the Jacobian $\mathbf{J} = \partial F/\partial X$ has full column rank (f.c.r.) and we are interested by the minimal differentiation degrees c_i making this Jacobian f.c.r. The maximum over i of the c_i is called the *differentiation index* of DAE (3). It is usually denoted by ν_d.

To avoid computing a large number of Jacobians, several graph-based algorithms were proposed [20], [22], [25] to search for the c_i. These algorithms aim at checking the regularity of the above mentioned Jacobian *generically* (the term *almost everywhere* is also used), meaning that the Jacobian $\mathbf{J} = \partial F/\partial X$ has f.c.r. when its non-zero entries vary in a neighborhood but outside an exceptional set of values.

All these algorithms rely on the following approach. Let $\mathcal{G}_F$ be the non-directed bipartite *incidence graph* of F, having a branch (x, f) if and only if the dependent variable x taken from vector (Z, X) occurs in function f taken from system F. Then, the following result holds:[1]

Lemma 1 (see [4] for a proof): The extended DAE system

$$F(X, Z) = 0 \quad (6)$$

[1] The fact that almost everywhere properties of Jacobians are checked using graph-based algorithms seems to belong to the folklore of the DAE subject area. We could not find a comprehensive reference providing all the needed results. So we have decided to write ourselves such a material. It can be found in [4].

satisfies condition (4) generically *if and only if the following assignment problem for* $\mathcal{G}_F$ *has a solution:*

$$\begin{array}{l}\textit{Find a bijective assignment } \psi : X \to F \textit{ such}\\ \textit{that } (x, \psi(x)) \in \mathcal{G}_F \textit{ for every } x \in X.\end{array} \tag{7}$$

The above result relies on the following assumption, which we will refer to in the sequel as a "principle" on which graph-based algorithms rely:

Principle 1: For any pair $(x, f) \in \mathcal{G}_F$*, variable* x *can be* generically *evaluated as a function of other variables involved in* f.

By the Implicit Function Theorem, Principle 1 holds, locally, for smooth algebraic equations.

Having ψ as in (7), we turn the incidence graph $\mathcal{G}_F$ into a directed graph $\vec{\mathcal{G}}_F$ by setting $\psi(x) \to x$ and $y \to \psi(x)$ for every $y \neq x$ such that $(y, \psi(x)) \in \mathcal{G}_F$. The resulting directed graph $\vec{\mathcal{G}}_F$ can have cycles and we call *blocks* the minimal cycles of $\vec{\mathcal{G}}_F$. Blocks are partially ordered by $\vec{\mathcal{G}}_F$, let $\preceq$ be this order and $\prec$ be the associated strict order. For β a block, let $\psi(\beta)$ be the set of maximal blocks β' such that $\beta' \prec \beta$, where "maximal" refers to the order $\preceq$. Knowing the values of the variables involved in $\psi(\beta)$, β defines an algebraic equation system that must be solved for the variables involved in it. As a result, the order $\preceq$ gives rise to a Block Lower Triangular (BLT) decomposition of the Jacobian $\mathbf{J}$.

If it exists, the solution of (7) is generally non unique. We are interested in finding an assignment giving *minimal blocks*. Several methods were proposed for solving this problem efficiently, e.g., [26], [20], [22].

Using (7), the method, for both finding a consistent initial condition and performing an integration step for (3), is by:

1) Searching for the minimal c_i so that (7) has a solution; $\nu_s = \max_i c_i$ is then called the *structural index* of DAE system (3); various algorithms for this task have been proposed, see, e.g., [20], [22], [25].
2) Forming the extended system (6) and computing the Jacobian $\mathbf{J}$ to apply the Implicit Function Theorem. Jacobian $\mathbf{J}$ inherits the block structure found by the solution to the assignment problem.

This two-step procedure is generally referred to as *index reduction.* The above body of tasks is referred to in the literature of physical system modeling as *structural analysis.*

B. Multi-mode DAE systems

Multi-mode DAE systems, also called *Hybrid DAE systems* consist of a collection of *modes* in which a mode-dependent dynamics holds in the form of a DAE system. Modes are themselves characterized by properties expressed in terms of the system variables. When its characteristic properties get violated, the current mode is exited and a transition relation defines the next mode. Successive mode changes can occur instantaneously, thus forming *cascades*.

The dynamics in different modes can have a different structure, thus resulting in different solutions for the assignment problem (7) and even sometimes different structural indexes. Such a situation frequently occurs when the system model collects nominal and failure modes, to capture both functional and degraded modes of operation. The complexity of this class of models further grows if the dynamics in a given mode can itself be multi-mode, in a nested way.

Two kinds of difficulties arise when moving from single-mode to multi-mode DAE systems:

(a) The syntax of multi-indices itself becomes intractable and expressing any symbolic (graph-based) algorithm requires developing new representations.
(b) On the more mathematical side, the whole approach in handling (mono-mode) DAE systems relies on a repeated use of differentiation to reveal latent equations. What does it mean to differentiate equations around the mode changes, since the dynamics is not even continuous there? Clearly, a revision of index theory is needed.

Despite these difficulties, modeling tools succeed in simulating certain classes of multi-mode DAE systems, see, e.g., the successive generations of Modelica/Dymola compilers [12], [19], which are able to handle larger classes for each new generation. In fact, the way these tools handle multi-mode systems is not by extending index theory and associated algorithms to multi-mode systems. Instead, they perform source-to-source transformations of the model so that changes in the structure are not a problem any more for the model resulting from the transformation. While this has the merit of addressing certain classes of multi-mode systems, general systems are not covered and the sub-class of systems that is covered is hard to characterize so that the user may not guess in advance what the compiler will say.

C. Our contribution

In this paper we propose a general index theory and structural analysis, for multi-mode DAE systems.

To simplify the development, we do this under the technical restriction that nested modes are not addressed. From our experience in the introduction of *mode machines* in synchronous languages, handling nested modes makes the semantics and contruction of execution schemes technically more involved — some kind of recursion is required and execution schemes are formally expressed by using so-called SOS rules, where SOS means: "structural operational semantics". No fundamental difficulty occurs, however, when moving from flat to nested guards. Needless to say, this generalization is needed to support practical languages, unless flattening is systematically performed as a first step. We leave this for future work. The bottom line is that we do not fully overcome difficulty (a). In contrast, our approach fully addresses difficulty (b).

With comparison to (3), we consider systems of equations of the form

$$\textbf{if } \gamma_i \textbf{ do } f_i(\text{the } x_j \text{ and derivatives of them}) = 0 \tag{8}$$

That is, (8) is a *guarded equation*, of wich γ_i is a boolean expression called the *guard* and the DAE $f_i(\dots) = 0$ is the *body.* The meaning of (8) is that equation $f_i(\dots) = 0$ must be satisfied if and only if γ_i takes the value "true" and otherwise equation $f_i(\dots) = 0$ is discarded.

The following two points are essential in our approach:

(*c*) The bodies of our guarded equations have the form $f_i{=}0$ where Principle 1 applies to f_i. Of course, Principle 1 does not apply to guarded equations of the form (8) as a whole: in a statement "**if** b **then** $f(x) = 0$" one cannot expect to evaluate the boolean b as a function of x.

(*d*) We need to find a way to handle the mode changes, where differentiation cannot be applied to reveal latent equations.

The following key observation can be formulated:

> Replacing, everywhere in DAE system (3), any derivative $x'(t)$ by its Euler approximation
> $$\frac{x(t+\varepsilon) - x(t)}{\varepsilon} \tag{9}$$
> for some fixed and positive step size ε, and solving (3) for the maximally forward shifted variables $x(t + d_j\varepsilon)$ leaves the assignment problem (7) unchanged.

By this we mean that problem (7) has a solution for the original continuous time system if and only if it has a solution for its first order Euler approximation. The reason for having (9) is that the mapping $(X, Z) \to (\widehat{X}, \widehat{Z})$ obtained by replacing $x_j^{(\ell)}(t)$ by $x_j(t + \ell\varepsilon)$ is lower triangular and invertible, see [4] for a formalization of this. Now, substitution (9) is not satisfactory from the numerical point of view since it only yields an approximation of the original DAE. If, however, we use as a step size an *infinitesimal* ∂ in the sense of non-standard analysis [23], [11], [18], [8], [3], (9) becomes:

> Replace, everywhere in DAE system (3), any derivative $x'(t)$ by its non-standard interpretation
> $$\frac{x(t+\partial) - x(t)}{\partial} \tag{10}$$
> and solve (3) for the maximally forward shifted variables $x(t + d_j\partial)$.

Transformation (10) is no longer an approximation, but rather a re-interpretation of the original DAE. Consequently:

> The original DAE system (3) and its non-standard interpretation using (10) possess identical structural analyses. (11)

The interest of doing this is that we can now circumvent difficulty (*d*) by using the following approach. Replacing the form (8) for a multi-mode DAE system by

$$\textbf{if } \gamma_i \textbf{ do } f_i(\textit{the } x_j \textit{ and shifts of them}) = 0 \tag{12}$$

where the dynamics has non-standard discrete time index

$$\mathbb{T} =_{\text{def}} \{k\partial \mid k \in {}^{\star}\mathbb{Z}\} \tag{13}$$

and then performing the structural analysis of the resulting dAE system yields a conservative extension of the notion of structural analysis, for both DAE and dAE systems. Elements of time index set $\mathbb{T}$ are called *instants* and are generically denoted by t. In (13), $\partial > 0$ is *infinitesimal* (smaller than any positive real number) and ${}^{\star}\mathbb{Z}$ is the set of *non-standard* positive or negative integers, so that any real number $s \in \mathbb{R}$ has an element $t(s) \in \mathbb{T}$ such that $|s - t(s)|$ is infinitesimal.

The paper is organized as follows. Section II is devoted to the discussion of small illustrative examples. In Section II-A we parallel the structural analyses of continuous time and discrete time single-mode systems and we revisit in particular the popular pendulum example. In Section II-B we extend our analysis to multi-mode systems and study in particular in detail the cup-and-ball game, sort of a two-mode extension of the pendulum. Our whole approach is illustrated on this example. In addition, we discuss the interest of using complementarity conditions and non-smooth systems solvers [1]. The rest of the paper is devoted to the formalization of our approach. In Section III we develop the structural analysis of multi-mode discrete time dAE systems, by closely following the background from synchronous languages. The two tasks of mapping to non-standard domain and back-standardization are then developed in Section IV. A mathematical justification of our approach is developed in Section V by using non-standard analysis.

II. Informal discussion of toy examples

In this section we smoothly introduce our approach. We begin with simple (single mode) DAE systems and reformulate them in our framework. Then, we analyse multi-mode examples. The formalization of all of this is deferred to the next section.

A. Single-mode systems

1) The simplest DAE system: Our first example is the following DAE system, where all variables are scalar:

$$\left\{\begin{array}{lcl} x' &=& f(x,w) \\ 0 &=& g(x) \end{array}\right. \tag{14}$$

Variable x is a state and w is an algebraic variable. System (14) does not satisfy condition (4) since one cannot solve (14) for the pair (x', w) when x' is interpreted as a free variable, not as the derivative of x — say that x' is seen as a *dummy variable*.

However, differentiating the algebraic equation $0 = g(x)$ yields the following system, in which a new *latent* equation (e_3) is revealed:

$$\left\{\begin{array}{lcll} x' &=& f(x,w) & (e_1) \\ 0 &=& g(x) & (e_2) \\ 0 &=& g'_x(x)x' \;=\; g'_x(x)f(x,w) & (e_3) \end{array}\right. \tag{15}$$

Since system (14) is time invariant, equation (e_3) is implied by (e_1, e_2). However, e_3 involves no additional variable but provides an additional equation to the triple x, x', w. Thus, systems (14) and (15) are equivalent. The interest of the latter is best explained by considering the discrete time counterpart of (14):

$$\left\{\begin{array}{lcl} x^{\bullet} &=& f(x,w) \\ 0 &=& g(x) \end{array}\right. \tag{16}$$

where $x^\bullet$ denotes the forward shifted version of x, so that

$$x^\bullet_n = x_{n+1} \text{ holds for every } n. \tag{17}$$

Let us ask ourselves the following question:

Can we deterministically simulate (16), instant by instant, by only solving algebraic equation systems? (18)

At each instant we must evaluate the pair $(w, x^\bullet)$, knowing x which was evaluated at the previous instant. The problem is that we only have at our disposal the equation $x^\bullet = f(x, w)$, which is insufficient to uniquely determine the pair $(w, x^\bullet)$.

Shifting in (16) the algebraic equation forward yields the following system, in which the new latent equation (e_3) is revealed, which involves no additional variable:

$$\left\{\begin{array}{lcll} x^\bullet & = & f(x,w) & (e_1)\\ 0 & = & g(x) & (e_2)\\ 0 & = & g(x^\bullet) = g(f(x,w)) & (e_3)\end{array}\right. \tag{19}$$

Now, the execution scheme shown in Listing 1 can be applied when simulating one instant of the discrete time dynamical system (19).

Given x such that $g(x) = 0$:
1) Use (e_3) to evaluate w;
2) Use (e_1) to evaluate $x^\bullet$, which satisfies $g(x^\bullet) = 0$;

Move to the next instant.

Listing 1. Execution scheme for (19).

Note that an algebraic equation solver is needed to perform step 1). Rather than performing the atomic computations 1) and 2) in sequence, we could rather use the equation solver in solving (e_1, e_3) jointly. In solving (e_1, e_3) for the variables $w, x^\bullet$, the dynamic link (17) relating x and $x^\bullet$ is ignored. Please, note that we implicitly invoked Principle 1.

The following question arises: why not shifting the algebraic equation $0 = g(x)$ twice, where $x^{\bullet 2} =_{\text{def}} (x^\bullet)^\bullet$:

$$\left\{\begin{array}{lcll} x^\bullet & = & f(x,w) & (e_1)\\ 0 & = & g(x) & (e_2)\\ 0 & = & g(x^\bullet) & (e_3)\\ 0 & = & g(x^{\bullet 2}) & (e_4)\end{array}\right.$$

Well, equation (e_4) is useless since it involves the extra variable $x^{\bullet 2}$ and tells nothing more about the pair $(x^\bullet, w)$. In contrast, if $x^{\bullet 2}$ was sitting on the left hand side of (e_1) instead of $x^\bullet$, then equation (e_4) would become a useful latent equation, for use in evaluating the leading variables $(x^{\bullet 2}, w)$.

Now, what similar can we say regarding the continuous time DAE system (15)? For a given x satisfying (e_2), we regard (e_1, e_3) as a system of equations in the variables w, x'. Doing so amounts to considering (15) as an ODE system in which w is a static state feedback (a function of x).

Let us relate the above reasoning to the notion of *differentiation index* of DAE systems [24] (or *difference index* of dAE systems), which we briefly recall now on our example. We begin with the original DAE system (14), which we rewrite $F(v, x) = 0$ where

$$v =_{\text{def}} \left[\begin{array}{c} x' \\ w \end{array}\right] \quad \text{and} \quad F(v,x) =_{\text{def}} \left[\begin{array}{c} x' - f(x,w) \\ g(x) \end{array}\right]$$

We wish to solve for v the equation $F(v, x) = 0$, in which x is given satisfying $g(x) = 0$. Unfortunately, the Jacobian

$$\mathcal{J} = \frac{\partial F}{\partial v} = \left[\begin{array}{cc} 1 & -f'_w(x,w) \\ 0 & 0 \end{array}\right]$$

is *structurally* singular, meaning that it is singular whatever the particular function f is. So there is no way that the equation $F(v, x) = 0$ can define v as a function of x. The solution consists in extending F to a larger array involving the new *dummy variable* $\xi =_{\text{def}} \frac{d}{dt} v = (x'', w')$:

$$\begin{array}{rcl} F_2(\xi, v, x) & =_{\text{def}} & \left[\begin{array}{c} F(v,x) \\ \frac{d}{dt} F(v,x) \end{array}\right] \\ & = & \left[\begin{array}{c} x' - f(x,w) \\ g(x) \\ x'' - f'_x(x,w)x' - f'_w(x,w)w' \\ g'(x)x' \end{array}\right] \end{array}$$

Array F_2 is then used to evaluate v as a function of x by

$$\text{solving for } v \text{ the equation: } \exists \xi . F_2(\xi, v, x) = 0. \tag{20}$$

Eliminating ξ from the equation $F_2(\xi, v, x) = 0$ is simple: discard the third row in F_2 since the latter involves the dummy variables x'', w'. The corresponding Jacobian is now

$$\mathcal{J}_{\text{reduced}} = \frac{\partial}{\partial v} \left[\begin{array}{c} x' - f(x,w) \\ g(x) \\ g'(x)x' \end{array}\right] = \left[\begin{array}{cc} 1 & -f'_w(x,w) \\ 0 & 0 \\ 1 & 0 \end{array}\right]$$

which is full column rank (f.c.r.) provided that $f'_w(x, w) \neq 0$, a generic situation. Having $\mathcal{J}_{\text{reduced}}$ f.c.r. ensures that (20) has, locally, a unique solution for v. Considering the larger array F_3 obtained by adding $\frac{d^2}{dt^2} F$ is totally useless since two more equations are added involving two fresh dummy variables $x^{(3)}, w''$.

Computing the various Jacobians associated to the successive arrays is costly. In contrast, searching for sufficient conditions ensuring that (20) has a unique solution for v, *generically* (outside of exceptional values for the non-zero entries of the associated Jacobian) amounts to solving the assignment problem (7), which can be efficiently solved, e.g., by performing the *structural analysis* using the Pantelides algorithm [20] or one of its alternatives, e.g. [22].

The other remark is that the DAE system (14) and its dAE counterpart (16) are equivalent regarding the assignment problem (7), in that the latent equations needed to reduce the index of each system match together.

2) The pendulum and its dAE version: This popular example will allow us to introduce our use of non-standard analysis.

$$\left\{\begin{array}{ll} x'' - \lambda x = 0 & (e_1)\\ y'' - \lambda y + g = 0 & (e_2)\\ x^2 + y^2 - L^2 = 0 & (e_3)\end{array}\right. \tag{21}$$

System (21) is a DAE system having x, y, x', y' (the positions and velocities) as state variables and λ (the tension) as algebraic variable, whereas g (the gravity) and L (the length of the pendulum) are constants. We now develop the approach that was outlined in (10).

Moving to non-standard form: We now regard system (21) as a dynamical system on the non-standard discrete time index set $\mathbb{T} =_{\text{def}} \{k\partial \mid k \in {}^{\star}\mathbb{Z}\}$ introduced in (10), with the following formulas for the first and second order derivatives:

$$x' = \frac{x^{\bullet} - x}{\partial} \quad \text{and} \quad x'' = \frac{x^{\bullet 2} - 2x^{\bullet} + x}{\partial^2} \tag{22}$$

That is, we use explicit Euler schemes with *infinitesimal* time step $\partial > 0$ and $x^{\bullet}$ is the forward shifted version of x by the infinitesimal amount ∂. In the sequel we shall freely use expansion (22) to express derivatives. By using (22), (21) becomes a dAE system (in discrete time). This system has the form:

$$\begin{cases} f_1(x^{\bullet 2}, x^{\bullet}, x, \lambda) = 0 & (e_1) \\ f_2(y^{\bullet 2}, y^{\bullet}, y, \lambda) = 0 & (e_2) \\ x^2 + y^2 - L^2 = 0 & (e_3) \end{cases} \tag{23}$$

To let the assignment problem (7) have a solution for this system, we must add the shifted versions $e_3^{\bullet}$ and $e_3^{\bullet 2}$ of e_3:

$$\begin{cases} f_1(x^{\bullet 2}, x^{\bullet}, x, \lambda) = 0 & (e_1) \\ f_2(y^{\bullet 2}, y^{\bullet}, y, \lambda) = 0 & (e_2) \\ x^2 + y^2 - L^2 = 0 & (e_3) \\ (x^2 + y^2)^{\bullet} - L^2 = 0 & (e_3^{\bullet}) \\ (x^2 + y^2)^{\bullet 2} - L^2 = 0 & (e_3^{\bullet 2}) \end{cases} \tag{24}$$

The execution scheme of (24) is given in Listing 2.

1) Given $x, y, x^{\bullet}, y^{\bullet}$ satisfying $(e_3, e_3^{\bullet})$:
2) Use $(e_1, e_2, e_3^{\bullet 2})$ to evaluate $\lambda, x^{\bullet 2}, y^{\bullet 2}$;
3) Move to the next instant.

Listing 2. Execution scheme for (24).

By construction, step 1) will hold at the next instant as a consequence of executing the current instant. Hence, this execution scheme applies repeatedly.

Back to the standard world: This is fine but not effective, since infinitesimal time shifts cannot be applied in practice. To overcome this, we *back-standardize* (24). We first replace (e_1, e_2) by their counterparts in (21) — so far this was easy. Then we must get rid of $(e_3^{\bullet}, e_3^{\bullet 2})$, which is not immediate. To this end, we use non-standard Taylor expansions by simply reverting formulas (22) — note that these expansions are exact, not approximate. Equation $(e_3^{\bullet})$ expands as

$$\begin{aligned} 0 \quad = \quad & (x^2 + y^2 - L^2) \\ & +2\partial(xx' + yy') \end{aligned} \tag{25}$$

Similarly, equation $(e_3^{\bullet 2})$ expands as

$$\begin{aligned} 0 \quad = \quad & (x^2 + y^2 - L^2) \\ & +2\partial\ (xx' + yy') \\ & +2\partial^2(xx'' + yy'' + x'^2 + y'^2) \end{aligned} \tag{26}$$

Now, given that (e_3) holds, (25) reduces to

$$0 \quad = \quad xx' + yy' \tag{27}$$

and, since both (e_3) and (27) hold, (26) reduces to

$$0 \quad = \quad xx'' + yy'' + x'^2 + y'^2 \tag{28}$$

Finally, by standardizing (24) we recover the familiar latent equations of the continuous time pendulum:

$$\begin{cases} x'' - \lambda x = 0 & (e_1) \\ y'' - \lambda y + g = 0 & (e_2) \\ x^2 + y^2 - L^2 = 0 & (e_3) \\ xx' + yy' = 0 & (e_3') \\ xx'' + yy'' + x'^2 + y'^2 = 0 & (e_3'') \end{cases} \tag{29}$$

Comment 1: So far we used the whole triple $(e_3, e_3^{\bullet}, e_3^{\bullet 2})$ in our argument. A more thorough analysis reveals that, indeed, only $(e_3^{\bullet 2})$ is sufficient. Consider again (26). Since we are developing a structural analysis, we want conditions for this equality to hold that are *almost everywhere* valid for non-zero values of the infinitesimal step ∂. Therefore, thanks to Taylor expansion (26), $(e_3^{\bullet 2})$ alone implies the three equations (e_3, e_3', e_3''). □

The above procedure may seem like an overshoot to recover the usual result but the nice point about it is that it generalizes to multi-mode DAE systems as we shall now see.

B. Multi-mode systems

1) Cup-and-ball game: Figure 1 shows a picture of this game. We study a simplified version of the game. First, we skip the modeling of the objective of the game, namely throwing the ball on the top of the bar. Second, we only model the phase of the game during which the ball falls until the string gets straigth. Third, it is assumed that a completely inelastic impact occurs (the coefficient of restitution is zero). As a result, when the length constraint is reached, the ball remains on the constraint $x^2 + y^2 = L^2$.

Figure 1. The cup-and-ball game.

The equations are the following:

$$\begin{cases} & & x'' = \lambda x & (e_1) \\ & & y'' = \lambda y - g & (e_2) \\ & & \gamma \ = [\, x^2 + y^2 \geq L^2 \,] & (\gamma) \\ \texttt{if} & \gamma & \texttt{do} \quad x^2 + y^2 = L^2 & (e_3) \\ \texttt{if} & \texttt{not}\ \gamma & \texttt{do} \quad \lambda = 0 & (e_4) \end{cases} \tag{30}$$

Equations (e_1, e_2) and the body of (e_3) are the equations of the pendulum. When the string is not tight, the body of equation (e_4) expresses that the tension is zero. Equations (e_3) and (e_4) are *guarded* by the condition γ and its negation.

Moving to the non-standard form: Using model (30) with the non-standard expansion (22) for the derivatives makes it a two-modes dAE system (operating in discrete time). What is a structural analysis for such a multi-mode dAE system? The right answer is by searching for an execution scheme such as

in Figures 1 and 2. For a structural analysis, the precise form of the equations does not matter. Only the presence/absence of a variable in an equation matters, very much like we did in abstracting the pendulum as model (23). Repeating this for the system (22,30) yields:

$$\left\{\begin{array}{llll} & & e_1(x^{\bullet 2}, x^{\bullet}, x, \lambda) & (e_1) \\ & & e_2(y^{\bullet 2}, y^{\bullet}, y, \lambda) & (e_2) \\ & & \gamma(x,y) & (\gamma) \\ \texttt{if} & \gamma & \texttt{do} \quad [\![e_3]\!](x,y) & (e_3) \\ \texttt{if} & \texttt{not}\ \gamma & \texttt{do} \quad [\![e_4]\!](\lambda) & (e_4) \end{array}\right. \tag{31}$$

Some comments follow:

- $e_1(x^{\bullet 2}, x^{\bullet}, x, \lambda)$ is the convenient abstraction for $f_1(x^{\bullet 2}, x^{\bullet}, x, \lambda) = 0$ used in model (23);
- we only need to know which variables are involved in the evaluation of the guard γ;
- $[\![e_3]\!]$ denotes the body of equation (e_3) and $[\![e_3]\!](x,y)$ indicates that this body involves the two variables x, y.

Principles supporting the structural analysis: Our structural analysis relies on the following principles:

Principle 2 (numerical equations): Principle 1 applies to all blocks of n numerical equations involving n dependent variables, and only to them.

Referring to (31), this means that Principle 1 applies to $(e_1, e_2, [\![e_3]\!], [\![e_4]\!])$, but not to (γ) and not to (e_3, e_4) as whole equations having γ, x, y and γ, λ as variables. We thus need two more principles to deal with the latter equations:

Principle 3 (predicates): A predicate can only be evaluated when all its involved variables have been evaluated. In this case, the predicate outputs any of the two values F, T.

Since numerical values are abstract, none of the two boolean values can be rejected as a candidate output. Thus the two outcomes F and T must be hypothesized and their consequences explored.

Principle 4 (guarded equations): A guarded equation can only be evaluated when its guard has been evaluated.

Unlike Principle 2 for numerical equations, Principles 3 and 4 impose certain causality constraints on how to handle guards. Let us try to derive an execution scheme for the abstract system (31) by complying with Principles 2, 3, and 4. This is shown in Listing 3. As explained in the caption, the execution scheme fails to evaluate all variables and solve all equations.

Considering the reason for the failure of step 2(a)ii, the only possible countermeasure consists in shifting forward $[\![e_3]\!]$ twice — since time step ∂ is infinitesimal, this does not introduce any real-time delay; it only affects causality and scheduling. Having done this, (31) becomes

$$\left\{\begin{array}{llll} & & e_1(x^{\bullet 2}, x^{\bullet}, x, \lambda) & (e_1) \\ & & e_2(y^{\bullet 2}, y^{\bullet}, y, \lambda) & (e_2) \\ & & \gamma(x,y) & (\gamma) \\ \texttt{if} & \gamma & \texttt{do} \quad [\![e_3]\!](x^{\bullet 2}, y^{\bullet 2}) & (e_3^{\bullet 2}) \\ \texttt{if} & \texttt{not}\ \gamma & \texttt{do} \quad [\![e_4]\!](\lambda) & (e_4) \end{array}\right. \tag{32}$$

1) Assuming consistent values for $x, y, x^{\bullet}, y^{\bullet}$:
2) Case:
   a) $\gamma = \text{T}$:
      i) regard (e_4) as dead code;
      ii) *failure to solve* $[\![e_3]\!]$;
   b) $\gamma = \text{F}$:
      i) regard (e_3) as dead code;
      ii) solve $[\![e_4]\!]$ for λ;
      iii) following Principle 2, solve for $x^{\bullet 2}, y^{\bullet 2}$ the block of equations $\{e_1, e_2\}$;
3) Move to the next instant.

Listing 3. Attempt to derive an execution scheme for (31). To comply with Principles 3 and 4, the execution scheme orders atomic actions for each case $\gamma = \text{F}/\text{T}$. A *failure point*, shown in italics, is discovered while performing this trial. Since values for the variables x, y have already been set when reaching step 2(a)ii, we fail at solving a system consisting of one equation and no dependent variable (violation of Principle 2).

The resulting execution scheme is shown on Listing 4.

1) Assuming consistent values for $x, y, x^{\bullet}, y^{\bullet}$:
2) Case:
   a) $\gamma = \text{T}$:
      i) regard (e_4) as dead code;
      ii) following Principle 2, solve for $\lambda, x^{\bullet 2}, y^{\bullet 2}$ the block of equations $\{e_1, e_2, [\![e_3]\!]\}$;
   b) $\gamma = \text{F}$:
      i) regard (e_3) as dead code;
      ii) solve $[\![e_4]\!]$ for λ;
      iii) solve for $x^{\bullet 2}, y^{\bullet 2}$ the block $\{e_1, e_2\}$;
3) Move to the next instant.

Listing 4. A successful execution scheme for (32).

The execution scheme is now successful. Note that any tuple of future values for the tuple of states $x, y, x^{\bullet}, y^{\bullet}$ is consistent for the next instant. Hence, step 1 is not a problem. Note that step 2(a)ii is now successful.

Back to the standard world: So far the execution scheme of Figure 4 is expressed in terms of non-standard discrete time and involves the infinitesimal parameter ∂. This is not effective. Deriving from this non-standard execution scheme an effective one operating in continuous time $\mathbb{R}$ (in fact, an extension of $\mathbb{R}$ to account for reset actions) is performed by *standardizing* the execution scheme of Figure 4 as explained next.

Outside instants of mode change: For this case the system stays within one of its two modes and the standardization is obtained by

- Translating equations (e_1) and (e_2) of (32) backward to the same equations from the original system (30), and
- By reusing the argument (25–28) based on Taylor expansions and taking Comment 1 into account, we standardize $(e_3^{\bullet 2})$ as the following triple of equations (e_3, e_3', e_3''), thus obtaining:

$$\left\{\begin{array}{lll} & x'' = \lambda x & (e_1)\\ & y'' = \lambda y - g & (e_2)\\ & \gamma \;\; = [\, x^2 + y^2 \geq L^2 \,] & (\gamma)\\ \texttt{if } \gamma \texttt{ do} & x^2 + y^2 = L^2 & (e_3)\\ \texttt{and} & xx' + yy' = 0 & (e_3')\\ \texttt{and} & xx'' + yy'' + x'^2 + y'^2 = 0 & (e_3'')\\ \texttt{if not } \gamma \texttt{ do} & \lambda = 0 & (e_4) \end{array}\right. \tag{33}$$

At instants of mode change: Let τ be a non-standard instant such that $\gamma(x_\tau^\bullet, y_\tau^\bullet) \neq \gamma(x_\tau, y_\tau)$, meaning that τ is an instant of mode change. Let $t \in \mathbb{R}$ be the standard part of τ, i.e., the unique real such that $t - \tau$ is infinitesimal (see Section V-A for a formal definition). We then create the ordered *super-dense* instants

$$(t,0) < (t,1) < (t,2)$$

where $(t,0) \equiv t$ and $(t,2) < t'$ for any $t' \in \mathbb{R}$ such that $t' > t$. The super-dense instant $(t,2)$ will hold the reset value for the positions and velocities when entering the new mode.

The evaluation of instants $(t,1), (t,2)$ is performed based on the discrete time dynamics specified by the non-standard form of (32), that is, using the discrete time non-standard interpretation (22) for the derivatives:

$$\left\{\begin{array}{lll} & \frac{1}{\partial^2}(x^{\bullet 2} - 2x^\bullet + x) = \lambda x & (e_1)\\ & \frac{1}{\partial^2}(y^{\bullet 2} - 2y^\bullet + y) = \lambda y - g & (e_2)\\ & \gamma \;\; = [\, x^2 + y^2 \geq L^2 \,] & (\gamma)\\ \texttt{if } \gamma \texttt{ do} & (x^2 + y^2)^{\bullet 2} = L^2 & (e_3^{\bullet 2})\\ \texttt{if not } \gamma \texttt{ do} & \lambda = 0 & (e_4) \end{array}\right. \tag{34}$$

When seen as a non-standard system, (34) allows, at non-standard instant τ of mode change, to evaluate $(x^{\bullet 2}, y^{\bullet 2})$, i.e., the values of the positions (x, y) two non-standard instants ahead, when we start the new mode. We standardize (34) by viewing the pair $(x^{\bullet 2}, y^{\bullet 2})$ as the value of the positions at super-dense instant $(t,2)$, where the reset value for the positions is given when entering the new mode.

However, we also need to know the value of velocities at that same instant, so that positions and velocities together form a consistent set of states. To get velocities, we position ourselves at the next non-standard instant $\tau^\bullet = \tau + \partial$ and we reexpress the second derivatives in terms of first ones. In addition, we must replace the shifted equation $(e_3^{\bullet 2})$ by the shifting of the equation (e_3') taken from equation (33):

$$\left\{\begin{array}{lll} & \frac{1}{\partial}(x'^\bullet - x') = \lambda x & (e_1)\\ & \frac{1}{\partial}(y'^\bullet - y') = \lambda y - g & (e_2)\\ & \gamma \;\; = [\, x^2 + y^2 \geq L^2 \,] & (\gamma)\\ \texttt{if } \gamma \texttt{ do} & (xx' + yy')^\bullet = 0 & (e_3'^\bullet)\\ \texttt{if not } \gamma \texttt{ do} & \lambda = 0 & (e_4) \end{array}\right. \tag{35}$$

When seen as a non-standard system, (35) allows, at non-standard instant $\tau + \partial$, to evaluate $(x'^\bullet, y'^\bullet)$, i.e., consistent values for the velocities (x', y') two instants ahead, when we start the new mode. Again, we standardize (35) by considering that it delivers the value, at super-dense instant $(t,2)$, of the reset values for the velocities.

So, at a first glance, it seems that, with (34,35) interpreted as delivering reset values at $(t,2)$, we have everything we need to initialize the new mode.

At this point, however, we still have a problem. Let us discuss it for the event of entering the mode $\gamma = \text{T}$. Focusing on (34), we see that the block $(e_1, e_2, e_3^{\bullet 2})$ is used to define the triple of leading variables $(x^{\bullet 2}, y^{\bullet 2}, \lambda)$. However the first two equations involve expressions that are standard when expressed in terms of x'', y''. However we need to reexpress x'', y'' interms of positions, which involves ∂^2 through $x'' = \frac{1}{\partial^2}(x^{\bullet 2} - 2x^\bullet + x)$. So, this system of equations is still not in standard form, let's make this explicit

$$\left\{\begin{array}{rcll} x^{\bullet 2} - 2x^\bullet + x & = & \partial^2 \lambda x & (e_1)\\ y^{\bullet 2} - 2y^\bullet + y & = & \partial^2(\lambda y - g) & (e_2)\\ (x^2 + y^2)^{\bullet 2} & = & L^2 & (e_3^{\bullet 2}) \end{array}\right. \tag{36}$$

Simply zeroing ∂ in these equations would leave us with a singular system of three equations and two dependent variables, since λ would disappear. Thus, we must get rid of ∂ in a different way. Only λ is directly affected by ∂. We thus eliminate it and get the following system

$$\left\{\begin{array}{rcll} y(x^{\bullet 2} - 2x^\bullet + x) & = & x(y^{\bullet 2} - 2y^\bullet + y) - \partial^2 gx & (e_{12})\\ (x^2 + y^2)^{\bullet 2} & = & L^2 & (e_3^{\bullet 2}) \end{array}\right.$$

We can now zero the infinitesimal in these equations to get

$$\left\{\begin{array}{rcll} y(x^{\bullet 2} - 2x^\bullet + x) & = & x(y^{\bullet 2} - 2y^\bullet + y) & (e_{12})\\ (x^2 + y^2)^{\bullet 2} & = & L^2 & (e_3^{\bullet 2}) \end{array}\right.$$

This is a standard set of equations that, at instant $(t,0)$ of the event of mode change, fixes $(x^{\bullet 2}, y^{\bullet 2})$, i.e., the values of the positions (x, y) two instants ahead in super-dense time, i.e., at super-dense instant $(t,2)$ when the new mode starts. We also infer that λ has an impulse at this instant, but we skip evaluating it.

How do we recover the velocities at the instant when the new mode start? Move to $(t,1)$ and reformulate (36) as follows

$$\left\{\begin{array}{rcll} x'^\bullet - x' & = & \partial \lambda x & (e_1)\\ y'^\bullet - y' & = & \partial(\lambda y - g) & (e_2)\\ (xx' + yy')^\bullet & = & 0 & (e_3^{\bullet 2}) \end{array}\right. \tag{37}$$

We again eliminate λ from (37), set ∂ to zero, which leaves us with the system

$$\left\{\begin{array}{rcll} y(x'^\bullet - x') & = & x(y'^\bullet - y') & (e_{12})\\ (xx' + yy')^\bullet & = & 0 & (e_3^{\bullet 2}) \end{array}\right. \tag{38}$$

At this point we are done: at $(t,1)$ we know $(x^\bullet, y^\bullet)$ since we just computed $(x^{\bullet 2}, y^{\bullet 2})$ at time $(t,0)$! So we end up with a system of two equations and two variables, namely $(x'^\bullet, y'^\bullet)$.

Thus we have evaluated in a standard way the positions and velocities at the entrance of the new mode. Again, we get an impulse on λ. Note that velocities are discontinuous here. To summarize, if t is the first instant of hitting the mode $\gamma = \text{T}$ we have to solve the following systems of algebraic equations:

$$\begin{array}{l} \text{at } (t,0): \left\{\begin{array}{rcl} y(x^{\bullet 2} - 2x^\bullet + x) & = & x(y^{\bullet 2} - 2y^\bullet + y)\\ (x^2 + y^2)^{\bullet 2} & = & L^2 \end{array}\right.\\ \text{to be solved for } x^{\bullet 2}, y^{\bullet 2}\\ \text{at } (t,1): \left\{\begin{array}{rcl} y(x'^\bullet - x') & = & x(y'^\bullet - y')\\ (xx' + yy')^\bullet & = & 0 \end{array}\right.\\ \text{to be solved for } x'^\bullet, y'^\bullet \end{array} \tag{39}$$

We insist that (39) is standard, since the discrete time shift $x \to x^\bullet \to x^{\bullet 2}$ refers to the discrete shift in super-dense time:

$$x_t = x_{(t,0)} \to x_t^\bullet = x_{(t,1)} \to x_t^{\bullet 2} = x_{(t,2)}\,,$$

and similarly for x' and other variables.

A similar (and indeed easier) reasoning provides the code for the instant of hitting the mode $\gamma = \text{F}$, since we directly infer that $\lambda = 0$ in this case. The automaton describing the execution scheme is shown on Figure 2.

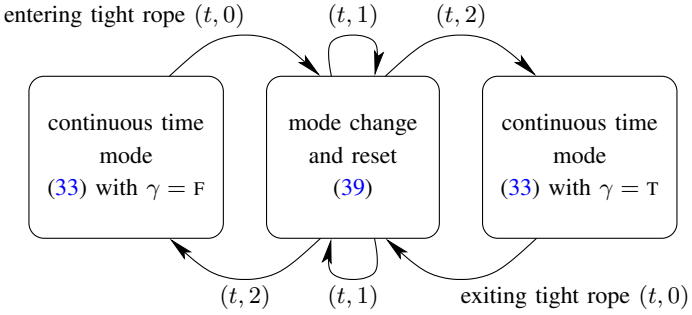


Figure 2. Automaton controlling the execution of the cup-and-ball system.

Comment 2: Exiting the two continuous time modes is by monitoring the associated zero-crossings separating the region $x^2 + y^2 < L^2$ from its complement $x^2 + y^2 \geq L^2$. □

Comment 3: Our approach above has limited applicability since it relies on variable elimination using computer algebraic techniques. This works well when the variables for elimination occur linearly in the system. This is doable but costly for polynomial systems, by using Gröbner bases.

The following alternating procedure can be considered. Let us explain it for the system (36). Expand the impulsive variable λ as a polynomial in ∂^{-1}, whose coefficients $\lambda_0, \lambda_1, \lambda_2$ are new dependent variables:

$$\lambda \;=\; \lambda_0 + \lambda_1\partial^{-1} + \lambda_2\partial^{-2} \tag{40}$$

Other (non-impulsive) variables are also interpreted as a formal power series in ∂^{-1}, albeit with zero coefficients except for the constant term of the serie.

Collecting separately the coefficients of ∂^0, ∂^1, and ∂^2, system (36) expands a collection of three systems of equations:

$$\text{for } \partial^0 \;:\; \left\{\begin{array}{rcll} x^{\bullet 2} - 2x^\bullet + x &=& \lambda_2 x & (e_1)\\ y^{\bullet 2} - 2y^\bullet + y &=& \lambda_2 y & (e_2)\\ (x^2+y^2)^{\bullet 2} &=& L^2 & (e_3^{\bullet 2}) \end{array}\right. \tag{41}$$

$$\text{for } \partial^1 \;:\; 0 \;=\; \lambda_1 \tag{42}$$

$$\text{for } \partial^2 \;:\; 0 \;=\; \lambda_0 \tag{43}$$

(We can safely zero the infinitesimal constant $\partial^2 g$ in (41).)

This technique replaces the use of elimination and is much simpler. It works for systems in which the numerical expressions are all polynomial in their variables. We guess it covers multi-body mechanical systems and electric circuits.

As a straightforward and practical approach, we can apply (34,35) with a small (but non infinitesimal) time step ∂, for two successive steps. This works in all cases. How well this performs and how it should be implemented in practice remains open. □

2) Simple clutch: This is an example borrowed from [19] where Dirac impulses can occur (since a clutch might be closed if the relative velocity is not zero). This generalized clutch model is not supported by the existing Modelica tools at the date of this writing. We consider a clutch relating two rotating shafts. The clutch is either engaged and then ensures perfect join between the two shafts or is released, in which case no link exists between the shafts. The continuous time model is the following — we abstract away the particular form of the first two equations:

$$\left\{\begin{array}{lll} & \omega_1' = f_1(\omega_1,\tau_1) & (e_1)\\ & \omega_2' = f_2(\omega_2,\tau_2) & (e_2)\\ \texttt{if } \gamma \texttt{ do} & \omega_1 = \omega_2 & (e_3)\\ \texttt{and} & \tau_1 + \tau_2 = 0 & (e_4)\\ \texttt{if not } \gamma \texttt{ do} & \tau_1 = 0 & (e_5)\\ \texttt{and} & \tau_2 = 0 & (e_6) \end{array}\right. \tag{44}$$

In (44) the boolean γ is an input that indicates the status of the clutch: released/engaged is encoded by F/T; ω_1 and ω_2 are the speeds of the shafts and τ_1 and τ_2 are the torques. The move of γ from T to F captures the onset of a press. The move of γ from F to T corresponds to the onset of a engagement.

Let us follow the same approach as before. We first consider (44) in combination with the non-standard interpretation (22) for the derivatives. The leading variables are $\omega_i^\bullet, \tau_i, i = 1,2$. We immediately see that the guarded equation (e_3) violates Principle 4: guard γ must be evaluated prior to enforcing the body of the equation. However, the body of (e_3) involves only non-leading variables, which, by definition, were already evaluated in the previous instants. Thus no successful execution scheme can exist for (44) in combination with the non-standard interpretation (22) for the derivatives.

The same counter-measure as before is applied to unlock this fixpoint: we replace guarded equation (e_3) by its non-standard shifted version $(e_3^\bullet)$:

$$\left\{\begin{array}{lll} & \omega_1' = f_1(\omega_1,\tau_1) & (e_1)\\ & \omega_2' = f_2(\omega_2,\tau_2) & (e_2)\\ \texttt{if } \gamma \texttt{ do} & \omega_1^\bullet = \omega_2^\bullet & (e_3^\bullet)\\ \texttt{and} & \tau_1 + \tau_2 = 0 & (e_4)\\ \texttt{if not } \gamma \texttt{ do} & \tau_1 = 0 & (e_5)\\ \texttt{and} & \tau_2 = 0 & (e_6) \end{array}\right. \tag{45}$$

The resulting successful execution scheme is shown on Listing 5. The removed equation (e_3) is indeed identical to $(e_3^\bullet)$ taken at the previous instant.

1) Assuming consistent values for ω_1, ω_2:
2) Case:
   a) $\gamma = \text{T}$:
      i) regard (e_5, e_6) as dead code;
      ii) using Principle 2, solve for $\tau_1, \tau_2, \omega_1^\bullet, \omega_2^\bullet$ the block of equations $\{e_1, e_2, [\![e_3^\bullet]\!], e_4\}$;
   b) $\gamma = \text{F}$:
      i) regard $(e_3, e_3^\bullet, e_4)$ as dead code;
      ii) solve $[\![e_5]\!]$ for τ_1;
      iii) solve $[\![e_6]\!]$ for τ_2;
      iv) solve e_1 for $\omega_1^\bullet$;
      v) solve e_2 for $\omega_2^\bullet$;
3) Move to the next instant.

Listing 5. A successful execution scheme for (45).

To derive the final code, it remains to standardize (45). We follow the same approach as for the cup-and-ball example.

Outside instants of mode change:
For this case the system stays within one of its two modes and the standardization is obtained by

- Translating equations (e_1) and (e_2) of (45) backward to the same equations from the original system (44), and
- By reusing the argument (25–28) based on Taylor expansions and taking Comment 1 into account, we standardize $(e_3^\bullet)$ as the following pair of equations (e_3, e_3'), thus obtaining:

$$\left\{\begin{array}{lll} & & \omega_1' = f_1(\omega_1, \tau_1) \quad (e_1) \\ & & \omega_2' = f_2(\omega_2, \tau_2) \quad (e_2) \\ \texttt{if } \gamma & \texttt{do} & \omega_1 = \omega_2 \quad (e_3) \\ & \texttt{and} & \omega_1' = \omega_2' \quad (e_3') \\ & \texttt{and} & \tau_1 + \tau_2 = 0 \quad (e_4) \\ \texttt{if not } \gamma & \texttt{do} & \tau_1 = 0 \quad (e_5) \\ & \texttt{and} & \tau_2 = 0 \quad (e_6) \end{array}\right. \tag{46}$$

At instants of mode change: Let τ be a non-standard instant such that $\gamma^\bullet \neq \gamma$, meaning that τ is an instant of mode change. Let $t \in \mathbb{R}$ be the standard part of τ, i.e., the unique real such that $t - \tau$ is infinitesimal. We then create the super-dense instants $(t, 0) < (t, 1)$ where $(t, 0) = t$ and $(t, 1) < t'$ for any $t' \in \mathbb{R}$ such that $t' > t$. The evaluation of instant $(t, 1)$ is performed based on the discrete time dynamics specified by (22,45), i.e., by applying as such one step of the execution scheme of Figure 5. This will provide us with the values for the rotation speeds (ω_1, ω_2) at the next super-dense instant $(t, 1)$.

Again, derivatives are expanded using (22), so we must get rid of the infinitesimal parameter ∂. This is performed by using one of the techniques used fpor the cup-and-ball example and the torques experience an impulse at the onset of the engagement of the clutch — we omit details since we did not give the precise form for the two functions f_1 and f_2. The intuition is, however, clear: when γ just moved from F to T (onset of an engagement of the clutch), equation $\omega_1 = \omega_2$ is still not active, since (45) does not contain (e_3), and the values for ω_1 and ω_2 were set at the previous instant by the two equations (e_1, e_2). At this first time, however, $\omega_1^\bullet = \omega_2^\bullet$ is enforced as a consequence of step 2(a)ii. The move $(\omega_1, \omega_2) \to (\omega_1^\bullet, \omega_2^\bullet)$ is non-infinitesimal but occurs in zero physical time (from instant $(t, 0)$ to instant $(t, 1)$): it corresponds to an *impulse* for the torques, which matches the physics. The opposite mode change T → F does not yield any impulse. The final code is obtained by using the same technique as for the previous example.

A variation of Figure 2 can be drawn for this case too.

3) Unilateral constraint: As a third example, we consider the following variation of our example (14):

$$\left\{\begin{array}{lcl} x' & = & f(x, u) \\ 0 & \leq & g(x) \end{array}\right. \tag{47}$$

which we rewrite using guarded equations:

$$\left\{\begin{array}{lll} & & x' = f(x, u) \quad (e_1) \\ & & \gamma = [0 \geq g(x)] \quad (\gamma) \\ \texttt{if } \gamma & \texttt{do} & g(x) = 0 \quad (e_2) \end{array}\right.$$

As before, we interpret the derivatives in non-standard analysis. We already know we will have a problem with the pair of equations (γ, e_2). We overcome this by shifting (e_2) forward:

$$\left\{\begin{array}{lll} & & x' = f(x, u) \quad (e_1) \\ & & \gamma = [0 \geq g(x)] \quad (\gamma) \\ \texttt{if } \gamma & \texttt{do} & g(x^\bullet) = 0 \quad (e_2^\bullet) \end{array}\right. \tag{48}$$

An attempt of an execution scheme for (48) is shown on Listing 6.

1) Assuming a consistent value for x:
2) Case:
   a) $\gamma = \text{T}$:
      i) using Principle 2, solve $[\![e_2^\bullet]\!]$ for $x^\bullet$;
      ii) using Principle 2, solve e_1 for u;
   b) $\gamma = \text{F}$:
      i) regard (e_2) as dead code;
      ii) *failure to solve* (e_1) *for the pair* $(u, x^\bullet)$;
3) Move to the next instant.

Listing 6. Attempt of an execution scheme for (48).

Shifting equations does not solve the problem arising in this execution scheme. Here, in the mode $\gamma = \text{F}$ (meaning that the unilateral constraint is not saturated), u becomes an input and the model has to express this in one way or another. One way could consist in adding, to (47), a specification of how u is computed as a static feedback from x, e.g., $h(x, u) = 0$. The resulting model would get a successful execution scheme and get standardized as for the previous examples:

$$\left\{\begin{array}{lll} & & x' = f(x, u) \quad (e_1) \\ & & \gamma = [0 \geq g(x)] \quad (\gamma) \\ \texttt{if } \gamma & \texttt{do} & g(x^\bullet) = 0 \quad (e_2^\bullet) \\ \texttt{if not } \gamma & \texttt{do} & h(x, u) = 0 \quad (e_3) \end{array}\right. \tag{49}$$

The so added "stub" is only a way to express that a value for u is expected and that this value can be delivered after knowing the state x.

C. Using Complementarity Conditions

So far Principle 2 was extensively used in deriving execution schemes. It applies to blocks of n numerical (smooth) equations involving n dependent variables. In this section we explain how to extend the class of systems amenable of Principle 2, and why it is of interest to do so.

Consider again example (49) and rewrite it first as follows:

$$\left\{\begin{array}{lcll} 0 & = & x' - f(x,u) & (e_1)\\ 0 & \leq & g(x^\bullet) & (e_{21})\\ 0 & \leq & |h(x,u)| & (e_{22})\\ 0 & = & g(x^\bullet)\,|h(x,u)| & (e_{23}) \end{array}\right. \tag{50}$$

The block (e_{21}, e_{22}, e_{23}) is called a *complementarity condition* [1] and is written as the following single expression

$$0 \leq g(x^\bullet) \;\perp\; |h(x,u)| \geq 0$$

Using this notation, (50) rewrites

$$\left\{\begin{array}{ll} 0 = x' - f(x,u) & (e_1)\\ 0 \leq g(x^\bullet) \;\perp\; |h(x,u)| \geq 0 & (e_2) \end{array}\right. \tag{51}$$

System (e_1, e_2) is seen as a system with two equations and two dependent variables $(u, x^\bullet)$. So-called *non-smooth systems solvers* exist that, under suitable conditions, ensure the existence and uniqueness of its solutions [1]. So, we can regard (51) as a block of equations amenable of Principle 2, which yields the much simpler execution scheme:

1) solve (51) for $(u, x^\bullet)$;
2) move to the next instant

(52)

We still need to standardize (51), which is performed as follows. First, we use the non-standard definition for the derivative:

$$g'(x) \approx \frac{g(y) - g(x)}{y - x} \text{ for any } y \text{ such that } y \approx x\,, \tag{53}$$

where $y \approx x$ means that $y - x$ is infinitesimal — see Section V-A for a primer on non-standard analysis. Using (53) we derive the following Taylor expansion:

$$g(x^\bullet) \;\approx\; g(x) + \partial g'(x) x' \tag{54}$$

Since ∂ is infinitesimal, $g(x^\bullet) > 0$ and $g(x^\bullet) \not\approx 0$ both hold if and only if

- either $g(x) > 0$ and $g(x) \not\approx 0$,
- or $g(x) \approx 0$ and $g'(x)x' > 0$ and $g'(x)x' \not\approx 0$;

and $g(x^\bullet) \approx 0$ holds if and only if both $g(x) \approx 0$, and $g'(x)x' \approx 0$. The standardization of (51) is then given by:

$$\left\{\begin{array}{ll} 0 = x' - f(x,u) & (e_1)\\ 0 \leq \left[\begin{array}{c} g(x) \\ g'(x)x' \end{array}\right] \perp |h(x,u)| \geq 0 & (e_2) \end{array}\right. \tag{55}$$

where $\mathbb{R}^2$ is equipped with the lexicographic order.

Finally, we reconsider the cup-and-ball example (30), which we rewrite as follows using complementarity conditions in non-standard time:

$$\left\{\begin{array}{ll} 0 = x'' - \lambda x & (e_1)\\ 0 = y'' - \lambda y + g & (e_2)\\ 0 \leq (x^2 + y^2 - L^2)^{\bullet 2} \;\perp\; \lambda \geq 0 & (e_3) \end{array}\right. \tag{56}$$

This is a complementarity condition with three equations and three dependent variables $(\lambda, x^{\bullet 2}, y^{\bullet 2})$, hence Principle 2 applies, which again yields a trivial execution scheme. The standardization of (56) uses Taylor expansions (26) for (e_3). Performing as for the previous example yields the non-smooth system

$$\left\{\begin{array}{ll} 0 = x'' - \lambda x & (e_1)\\ 0 = y'' - \lambda y + g & (e_2)\\ 0 \leq \left[\begin{array}{c} x^2 + y^2 - L^2 \\ xx' + yy' \\ xx'' + yy'' + x'^2 + y'^2 \end{array}\right] \perp \lambda \geq 0 & (e_3) \end{array}\right. \tag{57}$$

in which $\mathbb{R}^3$ is equipped with the lexicographic order. Note that our two-steps approach, in which we first identify the correct complementarity condition in discrete non-standard time, and then proceed to the standardization, implicitly performs the counterpart of index reduction.

D. Summarizing discussion

Based on the first set of examples, we have revisited the classical (single-mode) DAE systems by casting their structural analysis into the construction of execution schemes for their non-standard discrete time interpretation.

This opened the way to handle multi-mode DAE systems (mDAE). Our informal approach consists in the following two steps

1) Getting a successful execution scheme for the non-standard discrete time interpretation of the considered mDAE.
2) Standardizing the so obtained execution scheme:
   - at mode changes and to perform resets, by mapping the non-standard time to the super-dense time;
   - outside mode changes, by using non-standard Taylor expansions to re-express non-standard forward shifts in terms of standard derivatives.

Step 1) may fail due to either an excess or a lack of equations with regard to the available pool of dependent variables. Various counter-measures have been proposed to bring the execution scheme to success. Some, but not all of them relate to the known process of searching for latent equations and performing index reduction.

Step 2) points to different treatments depending on whether the system is at a mode change or not. This is entirely new and cannot be derived by intuitive extrapolation of the single-mode case. In the following sections we formalize all of this.

Our approach provides answers to the handling of impulses at mode changes. How to handle impulses at mode changes is well established for mechanical systems [21]. Our approach is general. Still, some further thinking is required to properly interpret the nature of the impulse (Dirac, dipole, etc.) in the standard domain.

III. The Constructive Semantics of mDAE systems

In this section we focus on discrete time systems and Step 1 of the two-step procedure of Section II-D.

A. Multi-Mode dAE (mdAE) systems: syntax

For defining multi-mode (discrete time) dAE systems we wish a formalism offering the following features:

- *Numerical variables* and dAE involving them;
  - example: $f(x^\bullet, x, y) = 0$, where f is smooth and $x^\bullet$ denotes the forward shifted version of x, so that
$$x^\bullet_n = x_{n+1} \text{ for every } n \tag{58}$$
- *Predicates* over numerical variables, giving rise to *propositional formulas*, formed on top of predicates;
  - example: "$x{\geq}2$" is a predicate returning `T` or `F` depending on the value of the numerical variable x, and "b_1 `or not` b_2" is a propositional formula in the predicates b_1 and b_2;
- *Invariants*, which are propositional formulas that must hold true for any execution of the system;
  - example: "`inv` : `not`$(b$ `and` $b')$" to express that b and b' are mutually exclusive; this way we can represent "if-then-else" equations by using simple "if" guards only and we can also specify finite state automata through propositional formulas involving predicates and their shifted versions.
- *Guarded equations*, which are dAE guarded by a propositional formula;
  - example: "`if` $z{\geq}0$ `do` $f(x^\bullet, x, y) = 0$": when z is active (see below) and ≥ 0, then the guard $z{\geq}0$ is true and the dAE applies. Otherwise the dAE is seen as dead code and discarded.
- A variable x may be involved in some but not all modes. We say that x is *dead* (reminiscent of "dead code"), denoted by the special value
$$x = \perp,$$
if x is not involved in the considered mode. When $x = \perp$, we take the convention that any guard involving x (e.g.: $x{\geq}0$) also takes the special value $\perp$. Say that x is *active* if $x \neq \perp$ holds.

We thus consider the **mdAE** mini-language whose syntax is displayed on Table I.

A variable $x : \mathit{var}$ is, recursively, either a basic variable $x \in \Xi$ or a forward shifted version $x^\bullet$ of a variable x. $X : \mathit{vars}$ is a tuple of basic variables. $S : \mathit{system}$ is a nested system of (scalar) guarded equations "e : `if` γ `do` $f(X){=}0$", or invariants $i : \mathit{inv}$, with associated declaration of variables. A propositional formula (PF) γ is a predicate $\varphi(X) : \mathit{pred}$ in the variables from some numerical tuple X, the conjunction of PF, the disjunction of PF, or the negation of a PF, all extended to account for the dead value $\perp$ as follows. Set $\perp \prec \mathtt{F} \prec \mathtt{T}$ and define `and` and `or` to be the infimum and supremum for that order; with reference to γ, `not` γ exchanges the values $\mathtt{F}, \mathtt{T}$ and otherwise keeps the rest unchanged. An invariant is a propositional formula. The invariant must hold true throughout the entire execution of S. By convention, predicate $\varphi(X)$ takes the value $\perp$ (dead) if and only if at least one of the variables constituting X is dead, the value `T` if and only if none of the variables constituting X is dead and the predicate holds true, and the value `F` otherwise. We assume that dAE $f(X) = 0$ is amenable of Principle 1.

We assume an underlying set of *names* Ξ, a subset of names $\Phi \subset \Xi$ denoting *predicates* over real variables, and a subset of names $F \subset \Xi$ denoting *scalar functions* over real variables.

var	::=	Ξ | *var*$^\bullet$
vars	::=	*var* | *var*, *vars*
prop	::=	`T` | `F` | *var* | `not` *prop* | *prop* `and` *prop* | *prop* `or` *prop*
num	::=	`num`
pred	::=	`pred` "predicate on *vars*"
inv	::=	`inv` *prop*
eqn	::=	`if` *prop* `do` "equation on *vars*"
term	::=	*num* | *pred* | *inv* | *eqn*
def	::=	*var* : *term*
system	::=	*def* | *def* ; *system*

Well formed programs satisfy the following conditions. Variables occurring in a *prop* are all *pred* and variables occurring in a *pred* are all *num*. Finally, variables x_i occurring in an *eqn* of the form "`if` *prop* `do` $f(x_1, \ldots, x_n){=}0$" are all *num*.

Table I
Syntax of the **mdAE** language. Keywords are in red.

Numerical variables, propositional formulas, invariants, equations, and systems, can be shifted forward, see Table II. With reference to Table I, variables from Ξ can be *num, pred, inv, eqn.* Observe that, in guarded equations, the guard is *not* shifted, reflecting the fact that, for both unlocking logico-numerical fixpoints and searching for latent equations, only bodies are shifted, not guards. On the other hand, invariants, being propositional formulas, are shifted.

$(\xi : t)^\bullet$	$\equiv$	$(\xi^\bullet, t)$, where
$(x, X)^\bullet$	$\equiv$	$x^\bullet, X^\bullet$
$\varphi(X)^\bullet$	$\equiv$	$\varphi(X^\bullet)$
$(P_1$ `and/or` $P_2)^\bullet$	$\equiv$	$P_1^\bullet$ `and/or` $P_2^\bullet$
$($`not` $P)^\bullet$	$\equiv$	`not` $P^\bullet$
$($`if` γ `do` $f(X){=}0)^\bullet$	$\equiv$	`if` P `do` $f(X^\bullet){=}0$
$(S_1; S_2)^\bullet$	$\equiv$	$S_1^\bullet; S_2^\bullet$

Table II
Shifting forward. With reference to Table I, the well-formedness conditions are in force, and X is defined as *vars*, $\varphi(X)$ is defined as *pred*, P is defined as *prop*, and γ is a guard. Observe that the guard is *not* shifted.

A *run* of S is any finite or infinite sequence of type consistent assignments of a value to all variables of S satisfying its constitutive equations. The *denotational semantics* of a **mdAE** system S is the set of all its runs. The denotational semantics does not say how the runs should be computed. We develop in Section III a *constructive semantics* for S, which consists of a constructive algorithm for computing the runs.

B. Principles of the Constructive Semantics

The notion of constructive semantics was introduced by the community of synchronous languages [6], [2], [5], where it

served to establish compilation schemes on a formal basis. As a background we first recall its principles by insisting on the notion of "atomic action", which turns out to become a non-trivial issue for multi-mode dAE systems.

A *constructive semantics* for a discrete time dynamical system consists of:

- A specification of the set of *atomic actions*, which are all effective, non-interruptible computation services provided by some underlying library; applying an atomic action is often referred to as performing a *micro-step;*
- A specification of all possible schedulings of the set of micro-steps constituting a given *reaction*, by which discrete time progresses, from the current discrete instant k to the next instant $k+1$.

This notion was originally proposed for *synchronous languages* [6], [2], [5], which are used to specify, with concurrency, discrete time transition systems. Take the simplest case of a single-clocked transition system collecting equations of the form

$$x \quad = \quad exp(\text{other } y \text{ and delayed versions of them})$$

where x is a flow variable and *exp* is an expression involving other flow variables y and delayed versions of them: $pre(y), pre(pre(y))$, etc. To such a transition system S we associate a *directed* bipartite graph $\mathcal{G}_S$ such that $(x, exp) \in \mathcal{G}_S$, where *exp* is the expression defining x, and $(exp, y) \in \mathcal{G}_S$ where y is a variable involved in *exp* with no delay. System S is *correct* if and only if $\mathcal{G}_S$ has no cycle. In this case, $\mathcal{G}_S$ defines a partial order relating flow variables and every scheduling that is a linear extension of this partial order is a correct execution scheme for each reaction of S, where the atomic actions consist in evaluating the different expressions involved in S. Generalizations of this simple situation consist:

- For Lustre [15], in allowing for several clocks (or several modes) in S, meaning that some flow variables may be defined only if some predicates hold;
- For Signal [14], in allowing, in addition, constraints relating different clock variables;
- For Esterel [9], in having an imperative language with parallel constructs instead of a transition systems defined by equations.

Nevertheless, for all synchronous languages, atomic actions are restricted to either evaluating expressions or forwarding the control in specific ways. For dAE systems, however, the class of atomic actions is much richer and requires a thorough inspection.

1) Eligible atomic actions for dAE systems: Eligible atomic actions for dAE systems consist of the following:

(γ) ***Evaluating a predicate*** in some set of numerical variables (e.g., "$z > 0$ and $2 < y < 3$"); this returns one of the values T, F, or $\perp$ (the latter if one of the numerical variables is dead);

(ι) ***Evaluating an invariant*** (a propositional formula involving predicates); failure to statisfy the invariant results in pruning the micro-steps not meeting this invariant. In doing so, we take the safe approximation that, once it is known to be active, a predicate $\varphi(X)$ can take both values T and F. Since the operations "`and`" and "`or`" on guards are monotonous, doing so guarantees that the remaining micro-steps will still meet this invariant.

(β) ***Solving a block of equations*** for its set Y of dependent variables. In doing so, the variables belonging to Y are considered dummy ($x^\bullet$ is not seen as the shifted version of x, but as a fresh variable) and it is required that the solution for Y exists and is unique, in a structural sense.

The class (β) of atomic actions requires some explanation and we now review interesting classes of blocks that are candidate atomic actions. The aim is that this class conforms Principle 2 introduced in Section II-B.

Blocks of linear equations: Blocks consisting of n linear equations in n dependent variables for which the assignment problem (7) has a solution, are structurally regular, and thus define eligible atomic actions.

The Implicit Function Theorem: Blocks consisting of n smooth equations in n dependent variables for which the assignment problem (7) has a solution, have their Jacobian structurally regular. Hence, existence and uniqueness of solutions is guaranteed, locally and structurally, by the implicit function theorem. Such blocks define eligible atomic actions provided that the search for a solution can be performed locally.

Complementarity Conditions: Another class of interest consists of the *complementarity conditions* [1]. A simple example is

$$\left\{ \begin{array}{l} h(v,i) = 0 \\ 0 \leq v \perp i \geq 0 \end{array} \right. \tag{59}$$

where

$$0 \leq v \perp i \geq 0 \quad \text{expands as} \quad \left\{ \begin{array}{l} 0 \leq v \\ 0 \leq i \\ 0 = iv \end{array} \right.$$

and $h(v,i)$ is a smooth function. An instance of complementarity condition is a closed electrical circuit involving a perfect diode, with v and i being the potential and the current. Complementarity conditions were informally discussed in Section II-C. Complementarity conditions generically possess a unique solution, locally, and solvers exist for such *non-smooth* systems [1]. Therefore, blocks of the form (59) are candidate atomic actions.

The following warning should be formulated at this point, regarding (59). This system has two equations and two unknowns. When seen as a block, it can be used to determine both v and i. However, one cannot say that each individual equation can be used to determine one variable as a function of the other. This is clearly wrong for the second equation "$0 \leq v \perp i \geq 0$" taken in isolation. Thus, when using a complementarity condition, we must combine it with the external link between its variables — here: $h(v,i)=0$ — and consider the two as an atomic block. This must be taken into account in the algorithms of Section III-C.

Unilateral constraints: Another interesting class is that of algebraic systems subject to unilateral constraints:

$$\left\{\begin{array}{lcl} 0 & = & h(x,u) \\ 0 & \leq & g(x) \end{array}\right. \tag{60}$$

When the constraint is saturated, then the first equation determines u, whereas u can be taken as an input otherwise. Reusing a technique of *dummy equations* proposed by Mattsson, Otter, and Elmqvist [12], we can complement (60) with a dummy scalar equation $c(x,u)=0$ which is active only when the unilateral constraint is non-saturated, and provides a way to select input u in this mode. The whole can be re-expressed in terms of complementarity conditions as follows:

$$\left\{\begin{array}{l} 0 = h(x,u) \\ 0 \leq g(x) \perp |c(x,u)| \geq 0 \end{array}\right. \tag{61}$$

Logico-numerical fixpoint equations are not eligible atomic actions: Some systems involve equations whose guard depends on numerical variables whose evaluation is under the scope of that same guard, thus forming a fixpoint. Here is a simple example:

$$\left\{\begin{array}{l} \texttt{if } \gamma \texttt{ do } g(x)=0 \\ \texttt{if not } \gamma \texttt{ do } h(x)=0 \\ \gamma = [x > 5] \end{array}\right.$$

The point here is that general logico-numerical fixpoint equations may not guarantee local existence and uniqueness of their solution, even if the constraints sitting in the body are regular. They may be several modes in which a solution exists, and there may be modes in which no solution exists. Thus, at the moment we do not see how to include such blocks as atomic actions. Maybe this is doable for specific sub-classes.

Having defined atomic actions we are now ready to introduce the *Scott variables,* which are the essential tool in formalizing the execution of a reaction as a properly scheduled sequence of micro-steps.

2) Scott variables: At a given point in the execution of a reaction, some expressions and equations have been evaluated and other remain to be evaluated. This status of the execution is captured by interpreting the names of set $\boldsymbol{\Xi}$ as *Scott variables*.[2] With reference to Table I, Scott variables encompass both the system variables and guards, the predicates, the equations, and the invariants. Tuples of Scott variables identify with subsets $\Xi \subseteq \boldsymbol{\Xi}$. Equip the set $\boldsymbol{\Xi}$ of Scott variables with the following flat partially ordered domain $(D, \leq)$ of values:

$$D = \{\star, \perp, \mathrm{T}, \mathrm{F}\} \quad \text{with} \quad \star < \perp, \mathrm{T}, \mathrm{F} \tag{62}$$

D is an extension, with the value $\star$, of the domain $\{\perp, \mathrm{T}, \mathrm{F}\}$ already mentioned for system variables. In (62):

- $\star$ is a special value *undefined* needed by the constructive semantics ("*not evaluated yet*" would be a better term). In the course of the execution of a given reaction, Scott variable $\xi \in \boldsymbol{\Xi}$ either remains to be evaluated ($\xi = \star$) or has already been evaluated (ξ takes its values in $\{\perp, \mathrm{T}, \mathrm{F}\}$). This holds for any ξ, representing a system variable or an equation, etc.
- $\perp$ is the value *dead*; $\xi = \perp$ means that ξ (a system variable or an equation, predicate, etc.) is not involved in the considered reaction; typically, the system is currently in a mode where this entity is not involved, hence this entity is seen as dead code. We insist that $\perp$ (dead) is different from $\star$ (not evaluated yet, also called "undefined").
- T, F are the *true* and *false* values of the Boolean domain. Since we want to support a symbolic analysis of multi-mode dAE systems, we are not interested in actual values taken by numerical variables. So we abstract the entire domain $\mathbb{R}$ as the single Boolean value T and the actual domain for $\xi \in num$ is $\{\star, \perp, \mathrm{T}\}$. The same holds for a Scott variable $\xi \in pred \uplus eqn$ and $\xi = \mathrm{T}$ means that the referred entity has been evaluated.

The Scott domain (62) allows expressing, as equations, scheduling constraints regarding the successive evaluation of the different variables at a given instant. For example, the equation

$$(y = \star) \vee \neg(x = \star) \quad = \quad \mathrm{T}$$

states that y cannot be evaluated unless x has been evaluated.

3) Abstract syntax: Table III shows the syntax of the abstract language used for the constructive semantics.

var	::=	$\boldsymbol{\Xi}$ | *var*$^\bullet$
vars	::=	*var* | *var*, *vars*
prop	::=	T | F | *var* | not *prop* | *prop* and *prop*
num	::=	num
pred	::=	pred (*vars*)
inv	::=	inv *prop*
eqn	::=	if *prop* do (*vars*)
term	::=	*num* | *pred* | *inv* | *eqn*
def	::=	*var* : *term*
system	::=	*def* | *def* ; *system*

Well formed programs satisfy the following conditions. Variables occurring in a *prop* are all *pred* and variables occurring in a *pred* or a *eqn* are all *num*.

Table III
Syntax of the abstract **mdAE** language, used for the constructive semantics. Keywords are in red. Compare with Table I.

The differences between Tables I and III concern the definitions of *pred* and *eqn*. The statements "predicate on *vars*" and "equation on *vars*" of Table I were abstracted as the corresponding lists "(*vars*)" of the variables involved in the considered predicate or equation. The rest of the syntax is not modified by the abstraction. The mapping of any well formed **mdAE** program to its abstract counterpart follows. This abstraction keeps track of the control of the original program (propositions and invariants) while abstracting predicates and equations to their bipartite graph encoding the association of *num* variables to *pred* or *eqn*.

The variables of an abstract **mdAE** program are Scott variables whose domain is D, defined in (62). For a definition

[2] Although we are not explicitly using Scott topology here, it is the basis for classical developments on constructive semantics. Hence this name for the variables of this semantics.

def of the form

$$\xi : \texttt{pred}(\Xi) \quad \text{or} \quad \xi : \texttt{when}\ \mathit{prop}\ \texttt{do}\ (\Xi)$$

where ξ is a *var* and Ξ is a finite set of *num*, we say that Ξ collects the *arguments* of ξ and we write this

$$\xi(\Xi).$$

C. The Constructive Semantics

We are now ready to specify all legal execution schemes for a system S.

1) Preliminaries: Here we introduce some needed material.

Status: Call *status* a valuation of all Scott variables

$$\sigma : \boldsymbol{\Xi} \to D$$

where $\sigma(\xi) = \star$ means that Scott variable ξ has not been evaluated yet. Let $\Sigma = \boldsymbol{\Xi} \to D$ denote the set of all statuses. Set Σ inherit the order defined in (62) by

$$\begin{array}{l}\sigma' \geq \sigma \text{ iff } \sigma'(\xi) \geq \sigma(\xi) \text{ holds} \\ \text{for every Scott variable } \xi.\end{array} \tag{63}$$

Say that status σ is *coherent*, written

$$\mathit{coherent}(\sigma), \tag{64}$$

if the following condition holds, which relates a Scott variable $\xi(\Xi)$ to its arguments:

$$\exists \xi' \in \Xi \text{ such that } \sigma(\xi') = \star \quad \Rightarrow \quad \sigma(\xi) = \star \tag{65}$$

$$\left.\begin{array}{r}\sigma(\xi) \neq \star \\ \exists \xi' \in \Xi \text{ such that } \sigma(\xi') = \bot\end{array}\right\} \quad \Rightarrow \quad \sigma(\xi) = \bot \tag{66}$$

(65) expresses that ξ cannot be defined unless all its arguments are defined and (66) means that, if ξ is defined, then it is dead as soon as at least one of its arguments is dead.

With reference to a given status σ, mark a considered tuple $\Xi \subseteq \boldsymbol{\Xi}$ of Scott variables by the symbol "!":

$$\sigma : !\Xi \qquad (\text{“}\Xi \text{ is } \mathit{enabled} \text{ in } \sigma\text{”}) \tag{67}$$

to indicate that all variables of Ξ can be selected for changing their value, from $\star$ to some defined value from the set $\{\bot, \mathrm{T}, \mathrm{F}\}$, through the application of some atomic action.

Satisfaction: Let $\gamma : \mathit{prop}$ be a propositional formula in some tuple of predicates. Write

$$\sigma \models \gamma \qquad (\text{“}\sigma \textit{ satisfies } \gamma\text{”}) \tag{68}$$

to indicate that, either $\sigma(\gamma) = \star$, or $\sigma(\gamma) = \mathrm{T}$. For an equation e : "$\texttt{if}\ \gamma\ \texttt{do}\ (X)$", write

$$\sigma \models e \qquad (\text{“}\sigma \textit{ satisfies } e\text{”}) \tag{69}$$

$$\text{if} \quad \left\{\begin{array}{rl}\text{either} & \sigma(e) = \star \\ \text{or} & \sigma(\gamma) = \mathrm{F} \\ \text{or} & \sigma(\gamma) = \mathrm{T} \text{ and } X = \mathrm{T}\end{array}\right.$$

to indicate that, either $\sigma(e) = \star$, or $\sigma(\gamma) = \mathrm{F}$, or $\sigma(\gamma) = \mathrm{T}$ and then $X = (\mathrm{T}, \ldots, \mathrm{T})$. It is intended that the actual numerical values of tuples X statisfy some concrete equation $f(X) = 0$ but this cannot be captured by our Scott domain D, which is abstract. For S a system, write

$$\sigma \models S \qquad (\text{“}\sigma \textit{ satisfies } S\text{”}) \tag{70}$$

to indicate that σ satisfies every invariant and equation of it.

Our definition of satisfaction deserves some comment. A better term for $\models$ should be "does not violate". In fact, if the propositional formula or equation is undefined, then $\models$ holds. Said differently, $\models$ says that nothing wrong has been done yet. This way of defining satisfaction is adequate for the constructive semantics.

Degree: In single-mode DAE (resp. dAE) systems, the differentiation (resp. difference) degree of a variable x is the largest integer $k \geq 0$ such that the derivative $x^{(k)}$ (resp. shift $x^{\bullet k}$) is involved in the considered system. In multi-mode dAE systems, however, the degree of a variable can vary from one instant to the next. For σ a status and ξ a numerical variable or a predicate, define the *degree of* ξ *in* σ, denoted by

$$d^o_\sigma(x),$$

as being the maximal shift of x in the equations e that are active in σ, i.e., $e = \mathrm{T}$. Observe that the degree is monotonic in σ: $d^o_{\sigma'}(x) \geq d^o_\sigma(x)$ holds whenever $\sigma' \geq \sigma$, see (63).

Shifted system: As we have seen from the examples, making a dAE system executable may require adding so-called latent equations — e.g., for the purpose of index reduction. Thus the system S we consider has the form

$$S = \widehat{S}\ ;\ \widetilde{S} \tag{71}$$

where $\widehat{S}$ is the original system and $\widetilde{S}$ collects the added latent equations.

Leading, useful, and dummy variables: In single-mode systems, leading variables are are those having the form $x^{\bullet k}$ where k is the degree of x for the original system $\widehat{S}$. Since the notion of degree is dynamic (status dependent) for multi-mode systems, we need a different definition, not related to the notion of degree. Indeed, for single-mode systems, leading variables must be evaluated at each new instant, whereas non-leading variables inherit their value from the previous instants (except at initialization). The *leading variables* in status σ are defined in Definition 1 below, by extending to a definition the latter property of leading variables in single-mode systems. If $x^{\bullet k}$ is a leading variable in status σ, then all variables of the form $x^{\bullet j}$ for $j \leq k$ are called *useful* in σ. Variables that are not useful in σ are called *dummy* in σ.

Definition 1 (constructive semantics): The constructive semantics *of a system S is the set of all maximal*[3] *increasing sequences of statuses, called* runs*,*

$$\sigma_0 < \sigma_1 < \cdots < \sigma_k < \sigma_{k+1} < \cdots < \sigma_K \tag{72}$$

of variable length $K < \infty$, where σ_0 is the initial status of the reaction, and, for each $0 \leq k < K$:

1) *Status σ_k is coherent, see (64), and satisfies S.*

[3] "Maximal" refers to the prefix order on finite sequences.

2) *Next status σ_{k+1} is obtained from current status σ_k by changing the values of a subset Ξ_k of variables, from $\star$ to some value $\neq \star$, by performing some* micro-step *(atomic action).*

For all statuses $\sigma_0, \ldots, \sigma_K$ of the run, the leading variables *are the variables ξ that are involved in the original system $\widehat{S}$ and such that $\sigma_0(\xi) = \star$. The considered run is* successful *if no useful variable has the value $\star$ in the final status σ_K.* □

If the constructive semantics succeeds, the system can proceed to executing its next reaction. Dummy variables can be eliminated via existential quantification. For our constructive semantics, evaluating them is thus optional and done only if necessary to evaluate all useful variables.

2) Algorithms: We are now ready to describe the symbolic algorithms computing the constructive semantics of S.

With reference to Table III, for S a system, we denote by $\mathbb{E}_S$ the set of all equations of type *eqn*, and by $\mathbb{X}_S$ the set of all numerical variables of type *num*. For $e \in \mathbb{E}_S$, $\gamma(e)$ denotes the proposition guarding e and $X(e)$ denotes the set of all numerical variables involved in the body of e. Consider the *block function*

$$\beta : \Sigma \times \wp(\mathbb{E}_S) \to \wp(\mathbb{X}_S)$$

which assigns,

- to every status $\sigma \in \Sigma$ and every subset $E \subseteq \mathbb{E}_S$ of equations that are active in σ but not evaluated yet,
- the tuple $\beta(\sigma, E) \subseteq \mathbb{X}_S$ of variables x that are involved in at least one equation of E, and yet undefined in σ.

$$\forall \sigma \in \Sigma, \forall E \subseteq \mathbb{E}_S : \forall e \in E \Rightarrow \left\{ \begin{array}{l} \sigma(\gamma(e)) = \mathrm{T} \\ \sigma(e) = \star \end{array} \right.$$
$$\beta(\sigma, E) =_{\mathrm{def}} \{x \in X(E) \mid \sigma(x) = \star\} \qquad (73)$$

where $X(E) =_{\mathrm{def}} \bigcup_{e \in E} X(e)$. E is called *enabled*, written

$$\sigma : !E \qquad (74)$$

if solving E for the variables belonging to $\beta(\sigma, E)$ belongs to the class of atomic actions — see Section III-B1. E is then called a *block*.

We are now ready to describe the algorithm computing the constructive semantics. With reference to forthcoming algorithms, its overall organization is as follows:

Algorithm 1 (Constructive Semantics of dAEs): For multi-mode dAE systems, the constructive semantics consists in:

1) Performing initialization using Algorithm 2;
2) Applying one of the following micro-steps, with the following set of priorities:

$$\begin{array}{ll} & \text{Algorithm 3 for performing a } \textit{tick} \\ < & \text{Algorithm 4 for enforcing invariants} \\ < & \text{Algorithm 5 for the evaluation of blocks} \\ < & \text{Algorithm 6 for the evaluation of predicates} \end{array} \qquad (75)$$

until termination — which occurs in finitely many micro-steps, see Theorem 1 below. □

The referred algorithms are described next.

Algorithm 2 (initialization): An initialization is a coherent status σ_0 assigning a (possibly undefined) value to all variables in a way that satisfies S:

$$\mathit{coherent}(\sigma_0) \quad \text{and} \quad \sigma_0 \models S \qquad (76)$$

Mark as enabled all predicates of S involving variables that are defined:

$$\begin{array}{l} \sigma_0 : !\Phi_0, \text{ where} \\ \Phi_0 =_{\mathrm{def}} \{p(X) : \mathit{pred} \mid \forall x \in X \Rightarrow \sigma_0(x) \neq \star\} \end{array} \qquad (77)$$

whereas no equation is marked enabled. By convention trivial predicates having constant value T are marked enabled. □

For the initial instant of system S, condition $\sigma_0 \models S$ in (76) is a system of equations that needs to be solved — it is called the *consistency* condition. For a subsequent instant, a status satisfying (76) is provided by the previous instant.

With reference to Definition 1 and formula (72), in the following, if $\sigma = \sigma_k$ denotes the current status while computing the constructive semantics, $\sigma^\circ = \sigma_{k+1}$ denotes the next status and ${}^\circ\sigma = \sigma_{k-1}$ denotes the previous status.

Algorithm 3 (tick): This algorithm applies if the current status σ satisfies the following conditions:

1) σ meets the invariants: $\sigma(\iota) \neq \mathrm{F}$ for every invariant ι;
2) $\sigma(\xi) \neq \star$ for every Scott variable ξ of S that either belongs to $\widehat{S}$ (see (71)) or is useful in S at the status σ.

We then set $\sigma_K =_{\mathrm{def}} \sigma$ and perform a *tick,* which is a special update $\sigma_K \to \sigma_0^\bullet$, such that, for every x:

$$\sigma_0^\bullet(x) = \left\{ \begin{array}{rl} \textbf{if } \sigma_K(x^\bullet) \in \{\bot, \mathrm{F}, \mathrm{T}\} \textbf{ then} & \sigma_K(x^\bullet) \\ \textbf{else} & \star \end{array} \right. \qquad (78)$$

The so defined status $\sigma_0^\bullet$ is the initial status for the next instant. It satisfies (76) by construction. □

Algorithm 4 (invariants): If $\sigma(\iota) = \mathrm{F}$ for some invariant ι, then, σ violates S and no successor of it can change this. We thus backtrack to the previous status ${}^\circ\sigma$ and terminate the run by setting $\sigma_K = {}^\circ\sigma$, see (72) for the definion of K. Otherwise, σ is validated and the constructive semantics can further proceed. □

Algorithm 5 (blocks): Select *non-deterministically* a maximal subset of enabled blocks that are pairwise *independent*: $\beta(\sigma, E) \cap \beta(\sigma, E') = \emptyset$. Solve these blocks, which updates σ to σ° by assigning a value T to more numerical variables. Update the block function β accordingly. The choice between different enabled blocks gives priority to the blocks with smallest maximal-shifting-degree. □

Algorithm 6 (predicates): Let Φ be the subset of predicates $\varphi(X)$ that are enabled in σ, i.e., $\sigma : !\Phi$. Move to the status σ° such that, for every $\varphi \in \Phi$, $\sigma^\circ(\varphi) \in \{\bot, \mathrm{T}, \mathrm{F}\}$ is *non-deterministically* selected so that σ° is coherent in the sense of (64) and satisfies:

$$\forall \xi \in X : \sigma(\xi) = \mathrm{T} \quad \Longrightarrow \quad \sigma^\circ(\varphi) \in \{\mathrm{T}, \mathrm{F}\} \qquad (79)$$

A defined value for some more propositions follows. Equations e whose guard got the value $\bot$ or F are dead by getting the

value $\sigma^\circ(e) = \bot$. Consequently, numerical variables that are only involved in equations that were found dead, are set dead too. Update the block function β by discarding equations that were found dead and mark the blocks that are enabled. □

The non-determinism in (79) accounts for the fact that, since the constructive semantics is a symbolic evaluation, the actual numerical value of the arguments $\xi \in X$ of predicate φ is not known.

Lemma 2: A run is successful in the sense of Definition 1 if and only if it terminates through Algorithm 3. □

Definition 2 (success): An instant is called successful *if some of its runs terminate successfully. A system is called* executable *if all its reachable reactions are successful.* □

The values assigned to all system variables at the end of a successful reaction extend by one instant the solution of the dAE, so these values do not depend on the nondeterministic selection of the blocks to be solved. Hence:

Theorem 1: Algorithm 1 terminates in finitely many micro-steps, and is confluent with respect to nondeterministic choices performed in its sub-Algorithm 5. □

IV. Multi-Mode DAE systems (mDAE)

A. Syntax and approach

We consider the **mDAE** language defined in Table IV, compare with the **mdAE** language of Table I. The only difference between **mDAE** and **mdAE** lies in the first and second lines of the syntax. Here, instead of the forward shift, we have two operators, namely

- the derivative x'; and
- the right-limit x^+, to specify resets at mode changes.

The non-standard interpretation of these operators clarifies their meaning:

$$x' = \tfrac{1}{\partial}(x^\bullet - x) \quad ; \quad x^+ = x^\bullet \tag{80}$$

We assume an underlying set of *names* Ξ, a subset of names $\Phi \subset \Xi$ denoting *predicates* over real variables, and a subset of names $F \subset \Xi$ denoting *scalar functions* over real variables.

Δ	::=	**evt** | **con**
var	::=	Ξ | var' | var^+
vars	::=	*var* | *var*, *vars*
prop	::=	`T` | `F` | *var* | `not` *prop* | *prop* `and` *prop* | *prop* `or` *prop*
num	::=	`num`
pred	::=	`pred` "predicate on *vars*"
inv	::=	`inv` *prop*
eqn	::=	`if` (*prop*, Δ) `do` "equation on *vars*"
term	::=	*num* | *pred* | *inv* | *eqn*
def	::=	*var* : *term*
system	::=	*def* | *def* ; *system*

Well formed programs satisfy the same conditions as in Table I. In addition, expressions such as $(x^+)'$ are forbidden and derivatives (respectively right limits) are forbidden in the body of *eqn* if the duration type Δ is **evt** (respectively **con**).

Table IV
Syntax of the **mDAE** language. Keywords are in red. Compare with Table I.

In addition, we have a *duration type* Δ taking the value **evt** (*event*) or **con** (*continuous*, indicating a mode of positive duration where the continuous time dynamics applies). Guards of *eqn* are typed propositional formulas. The well formedness conditions for an *eqn* in Table IV indicate which one among the derivative or the right limit, is allowed depending on the type of the guard.

Comment 4: Typed guards are important. They allow distinguishing events in which right limit is typically used e.g., for resetting variables,[4] from continuous modes involving a continuous dynamics with ODE/DAEs. The simple mechanism of duration type performs exactly the kind of distinction we need for our analysis. □

Comment 5: Events arise for two reasons. First, guards of equations *eqn* may be typed zero duration (Δ is **evt**). Second, the boundary instants of a guard having positive duration (Δ is **con**) are events of mode change. □

Comment 6: In practical formalisms (e.g., Modelica, but also the underlying syntax associated to solvers such as DASSL) offer mode change mechanisms based on *zero-crossings*. For instance, the Modelica statement "`when x <= 0 do reinit(x, 1)`" interprets the `when` as the transition event, from false to true, of the mentioned predicate — the reset acts on zero-crossings. This system possesses two modes: the instants of zero-crossing when reset gets performed, and the complement, where the underlying ODE/DAE dynamics applies. For such formalisms, syntax based typing analysis is needed to infer the duration type of a guard. Developing such typing techniques is left for future work. □

Our approach for the structural analysis of **mDAE** is illustrated on Figure 3. This approach was followed in Section II-B1 for the cup-and-ball example.

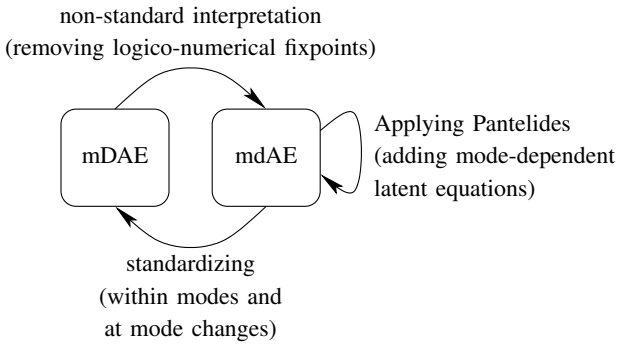


Figure 3. Our approach for the structural analysis of multi-mode DAE systems.

We develop this in the forthcoming subsections. Prior to this we provide in Table V the syntax of the abstract form of the **mDAE** language, for use in our compilation scheme.

[4] For example, a simple coding of the bouncing ball involves a reset of the horizontal and vertical velocities x and y by equations of the form $x^+ = x$ and $y^+ = -0.9y$.

We assume an underlying set of *names* Ξ, a subset of names $\Phi \subset \Xi$ denoting *predicates* over real variables, and a subset of names $F \subset \Xi$ denoting *scalar functions* over real variables.

$$
\begin{array}{rcl}
\Delta & ::= & \mathbf{evt} \mid \mathbf{con} \\
\mathit{var} & ::= & \Xi \mid \mathit{var}' \mid \mathit{var}^+ \\
\mathit{vars} & ::= & \mathit{var} \mid \mathit{var}, \mathit{vars} \\
\mathit{prop} & ::= & \mathtt{T} \mid \mathtt{F} \mid \mathit{var} \mid \mathtt{not}\ \mathit{prop} \mid \\
 & & \mathit{prop}\ \mathtt{and}\ \mathit{prop} \mid \mathit{prop}\ \mathtt{or}\ \mathit{prop} \\
\mathit{num} & ::= & \mathtt{num} \\
\mathit{pred} & ::= & \mathtt{pred}\ (\mathit{vars}) \\
\mathit{inv} & ::= & \mathtt{inv}\ \mathit{prop} \\
\mathit{eqn} & ::= & \mathtt{if}\ (\mathit{prop}, \Delta)\ \mathtt{do}\ (\mathit{vars}) \\
\mathit{term} & ::= & \mathit{num} \mid \mathit{pred} \mid \mathit{inv} \mid \mathit{eqn} \\
\mathit{def} & ::= & \mathit{var} : \mathit{term} \\
\mathit{system} & ::= & \mathit{def} \mid \mathit{def}\ ;\ \mathit{system}
\end{array}
$$

Well formed programs satisfy the same conditions as in Table IV.

Table V
Syntax of the abstract **mDAE** language, used in our compilation scheme.

B. Structural Analysis of *mDAE*

Our first task consists in mapping the considered **mDAE** system to its non-standard counterpart. In principle, this is straightforward: simply apply (80). The mapping of event and reset actions is not so obvious, however. Consider again the cup-and-ball example and its first attempt of a model, namely (30). We have seen that a brute force application of (80) produced a deadlocked system, in which a logico-numerical fixpoint occured — recall such fixpoint equations do not belong to our library of legitimate atomic actions. Inspecting (30) suggests that the mistake was to use $x^2+y^2=L^2$ as the body of reset equation (e_3). Since both x and y are states, we should rather reset their right-limits x^+ and y^+, which would result in shifting by 1 the body of (e_3). This would not be enough, however, as we have seen. Shifting twice was indeed necessary. So, unless we put a serious burden on the shoulders of the programmer (by offering not only x^+ but also something like x^{+k} for $k>0$, a rather mysterious statement), we should not expect the original system specification to directly provide the appropriate translation to the non-standard domain.

We therefore need to consider that discovering the correct shifting of the body of reset equations (if any), is part of the duty of our structural analysis. Having obtained a suitable translation to the non-standard domain, the task of revealing possible latent equations remains. All of this is explained next.

Our structural analysis of **mDAE** works by applying Algorithm 1 to the considered **mDAE** system while translating on-the-fly its equations to their suitable non-standard counterpart and searching for possible mode-dependent latent equations. Referring to the approach illustrated in Figure 3, the two steps 1) mapping to non-standard domain and 2) applying multi-mode Pantelides are collapsed together and performed on-the-fly.

*Algorithm 7 (structural analysis of **mDAE**):* Loop over the following, until termination:

> Apply Algorithm 1 while mapping on-the-fly the **mDAE** system to the non-standard domain using (80); then:
>
> - Case of *successful termination*: terminate Algorithm 7 successfully;
> - Case of *failed termination*: apply the following algorithms with the following priority
>
> $$\begin{array}{rl} & \text{Algorithm 8 for unlocking fixpoints} \\ < & \text{Algorithm 9 for revealing latent equations} \end{array} \tag{81}$$
>
> until termination. □

The two algorithms are given next.

Algorithm 8 (unlocking fixpoints): This algorithm applies if Algorithm 1 gets blocked with some equation e being non evaluated while having all of its variables X already evaluated:

$$\begin{array}{ll} \text{pre-condition :} & \left.\begin{array}{l} X = \mathtt{T} \\ e = \star \end{array}\right\} \text{in } e : \mathtt{when}\ P(X)\ \mathtt{do}\ (X) \\ & \text{and Algorithm 1 is blocked;} \\ \text{post-condition :} & \text{substitute } e \leftarrow e^\bullet \end{array}$$

where $e^\bullet$ is defined in Table II. □

Algorithm 9 (revealing latent equations): This algorithm applies if Algorithm 1 gets blocked with some variables not yet evaluated but no block being enabled at current status σ:

$$\begin{array}{l} \exists E \subseteq \mathbb{E}_S \text{ such that } \beta(\sigma, E) \neq \emptyset \\ \text{and Algorithm 1 is blocked.} \end{array} \tag{82}$$

If (82) holds, search for an E satisfying (82) and an equation e of the considered system such that

$$e \notin E \quad \text{but} \quad \beta(\sigma, E \cup \{e^\bullet\}) = \beta(\sigma, E)\,. \tag{83}$$

If such an equation e exists, add $e^\bullet$ to the considered system at status σ. □

Algorithm 9 is the multi-mode counterpart of a variation of the Pantelides algorithm called the *Dummy Derivatives* method [25]. It may be of interest to investigate, as an alternative, the use of the Σ-method by Pryce et al. [22]. Algorithm 8, however, is new. Due to the escalation in shifts performed by Algorithm 8, Algorithm 7 may not terminate in finitely many steps. Success is not guaranteed, due to the condition (83).

C. Back-Standardization

Assume that the structural analysis succeeds for the system in consideration. To get effective executable code, we must leave the non-standard domain and return to a standard one. Our standard semantic domain is that of dynamical systems defined over *super-dense time*

$$\mathbf{T} \quad =_{\text{def}} \quad \mathbb{R}_+ \times \mathbb{N} \tag{84}$$

where $\mathbb{N} = \{0, 1, \dots\}$, equipped with the *lexicographic order*:

$$(t,n) < (t',n') \quad \text{iff} \quad \left\{ \begin{array}{ll} & t < t' \\ \text{or} & t = t' \text{ and } n < n' \end{array} \right.$$

Elements of $\mathbf{T}$ are generically denoted by the symbol $\mathbf{t}$. In using $\mathbf{T}$ the intent is to capture events in which several computations are performed in a suitable order but in "zero time", indexed by $(t,0), (t,1), \dots, (t,n)$ for some finite n

depending on t. Between events, the dynamical system has continuous dynamics indexed by $\mathbb{R}_+$ as usual. To capture this, we restrict the time domain of a given behavior to have the form

$$\{\mathbf{t} = (t, n) \in \mathbf{T} \mid n \leq h(t)\}$$

where $h : \mathbb{R}_+ \to \mathbb{N}$ is a total function called the *height function*. A single-mode DAE system is characterized by having $h \equiv 0$ for all its behaviors, expressing that its time domain is simply $\mathbb{R}_+$. For the cup-and-ball example of Section II-B1, we have $h(t) = 0$ if t sits inside a mode and $h(t) = 2$ if t is a mode change.

Back-standardization targets the language **hDAE** having the syntax shown on Table VI, where $h(t^-)$ and $h(t^+)$ denote the left- and right-lim sup of h at t:

$$\begin{aligned} h(t^-) &=_{\text{def}} \limsup_{s \nearrow t} h(s) \\ h(t^+) &=_{\text{def}} \limsup_{s \searrow t} h(s)\,. \end{aligned}$$

The control structure of the generated code is shown on Figure 4. We now explain how back-standardization is performed, depending on which of the two modes (continuous mode or event handling) the system is.

Our starting point is an executable **mDAE** system, i.e., having a successful constructive semantics. The three operators

We assume an underlying set of *names* Ξ, a subset of names $\Phi \subset \Xi$ denoting *predicates* over real variables, and a subset of names $F \subset \Xi$ denoting *scalar functions* over real variables.

Δ	::=	**evt** | **con**
var	::=	Ξ | *var*′ | *var*$^+$ | *var*$^\bullet$
vars	::=	*var* | *var*, *vars*
prop	::=	T | F | *var* | **not** *prop* | *prop* **and** *prop* | *prop* **or** *prop*
num	::=	**num**
pred	::=	**pred** "predicate on *vars*"
inv	::=	**inv** *prop*
eqn	::=	**if** (*prop*, Δ) **do** "equation on *vars*"
term	::=	*num* | *pred* | *inv* | *eqn*
def	::=	*var* : *term*
system	::=	*def* | *def* ; *system*

Well formed programs satisfy the same conditions as in Table IV. In addition, for $\mathbf{t} \in \mathbf{T}$ and x : *var*:

- $x'_{\mathbf{t}}$ is defined iff $\mathbf{t}{=}(t, .)$ and $h(t^-){=}h(t){=}h(t^+){=}0$;
- $x^+_{\mathbf{t}}$ is defined iff $\mathbf{t}{=}(t, .)$ and $h(t^+){=}0$;
- $x^\bullet_{\mathbf{t}}$ is defined iff $\mathbf{t}{=}(t, n)$ and $h(t){>}0$ and, for $n{<}h(t)$, $x^\bullet_{\mathbf{t}} =_{\text{def}} x(t, n{+}1)$.

Table VI
Syntax of the **hDAE** language. Keywords are in red.

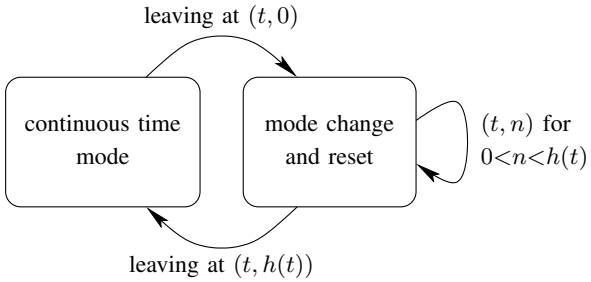


Figure 4. Control structure of an **hDAE** system.

$x', x^+, x^\bullet$ are redundant:

$$x', x^+ \text{ can be expressed in terms of } x^\bullet \text{ by using the formulas } x' = \tfrac{1}{\partial}(x^\bullet - x) \text{ and } x^+ = x^\bullet; \tag{85}$$

$$f(X) \text{ a numerical expression involving the tuple } X \text{ of variables, } f^\bullet(X) =_{\text{def}} f(X^\bullet) \text{ is expanded using the Taylor formula } f^\bullet(X) \approx f(X) + \partial\langle f'(X), X'\rangle, \text{ where } \langle ., .\rangle \text{ denotes the scalar product;} \tag{86}$$

$$\text{each numerical equation } \textstyle\sum_k \partial^k f_k(X_k) = 0, \text{ where the sum ranges over a finite set of indices } k, \text{ is mapped to the system of standard equations } \forall k : f_k(X_k) = 0. \tag{87}$$

In the next sections we develop different standardization schemes for continuous modes and event handling. Then, we need to decide, based on the constructive semantics, in which case we are at a given instant: we call this "event detection".

1) Within continuous modes: Suppose we know the considered instant $\mathbf{t}$ sits within a continuous mode. From the well-formedness conditions for **hDAE** systems in Table VI, $x^\bullet_{\mathbf{t}}$ is not defined, and thus we must get rid of it by using rules (86,87). The resulting code no longer involves shifts nor the infinitesimal ∂: it is our targeted standard code, which we can submit to our solver.

2) Event handling: Suppose we know the considered instant $\mathbf{t}$ is an event. From the well-formedness conditions for **hDAE** systems in Table VI, $x'_{\mathbf{t}}$ and $x^+_{\mathbf{t}}$ are not defined, and thus we must get rid of them by using rule (85). At this point we are left with a purely discrete time system involving shifts, which maps to super-dense time as indicated in the well-formedness conditions for **hDAE** systems in Table VI. We are not done yet, however, since the infinitesimal ∂ remains and we have no rule to remove it. Doing so is a post-processing that is beyond the scope of our structural analysis.

We now informally sketch three methods that can be used to this end. To simplify our exposure, we restrict ourselves to the case in which only 1^{st}-order derivatives are involved in the system.

Elimination: Whenever feasible, eliminate, from a ready block of equations, the variables or expressions that are multiplied by $1/\partial$ or beyond. This leaves us with a smaller ready block in which we can safely enforce $\partial := 0$ (since

no variable will get erased as a result of doing so). Variables multiplied by $1/\partial$ or beyond are impulsive and are not system states in the considered instant. Their value is therefore not needed to perform the reset.

Polynomial expansions: If all numerical expressions involved in the original system are polynomial, then applying rule (85) replaces each numerical variable by a finite formal power series in the variable ∂. The coefficients of these formal power series are then handled as fresh variables on which elimination can be performed.

Numerical evaluation: In all cases, using a standard small positive value for ∂, we can always simulate for the requested number of discrete steps, the discrete time system that we obtained after applying rule (85). This provides us with reset values for the next continuous mode. We can then, either restart the simulation with the reset values from the instant t^+ where the event was detected, or continue the simulation after the instant $t + n\partial$, assuming that n is the number of discrete steps to be performed. This technique remains to be analysed and experimented.

D. Example: three colliding balls

The model: We consider another variation around the pendulum. It consists of three colliding balls moving in a vertical plane, see Figure 5. To keep the model size within reasonable bounds, we take the simplifying assumption that collisions only occur when the colliding bars are vertical — this simplifies the expression of the conservation laws. The

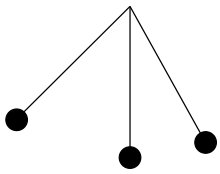


Figure 5. Three colliding balls

masses are equal and collision is fully elastic. The bars have equal lengths and are fixed at the same point in the plane. Although of zero diameter, the three balls are considered linearly ordered: left-most, mid, and right-most, indexed by the subscripts $_{l,m,r}$ respectively. The equations in Cartesian coordinates are found in Table VII-left. The modeling approach is straightforward and quite explicit in describing the collisions. To simplify, we have ruled out simultaneous double collisions (they can occur arbitrarily closely, however). This is specified using the invariant: $\texttt{inv} : \gamma_{lm} \wedge \gamma_{mr} = \text{F}$. Handling this special case would require defining the special dynamics in that mode.

Non-standard translation: The mapping to non-standard form is shown on Table VII-right. In obtaining the latter, we have applied Algorithm 8 for unlocking fixpoints, thus shifting forward two times the constraint $x^2 + y^2 = L^2$ (indicated in red). Then we have applied Algorithm 9 for revealing latent equations, with the result indicated in blue. The resulting non-standard system is shown on Table VII-right. It has four modes according to the values taken by the pair of guards $(\gamma_{lm}, \gamma_{mr})$. The 12 leading variables are the same in all modes, namely $x^{\bullet 2}_{l/m/r}, y^{\bullet 2}_{l/m/r}, v^{\bullet}_{l/m/r}, \lambda_{l/m/r}$, and active equations are 12 as well in all modes. We leave the verification of this as an exercise to the reader, as well as the construction of a successful execution scheme.

Cascades of collisions: Observe that, when $x_{lm} = x_{mr} = v_{mr} = 0$ and $v_{lm} > 0$ or the symmetric case, this non-standard system experiences a cascade of two successive events. At this point we have the following difficulty. We know that the two events can occur in a cascade. But, since we do not allow us to reason about numerical values when deriving our execution schemes, we cannot rule out the situation in which the two events alternate in an infinite cascade — the two balls shaking infinitely fast against each other. Our physical understanding tells us this cannot happen but proving it as part of the structural analysis is beyond our reasoning capability. So, being unable to prove the finiteness of cascades of events, the structural analysis will complain. One counter-measure consists in using *invariants* to manually enter the information that cascades are bounded in length by 2. Indeed, proving this may be performed by some verification tool for hybrid systems. In the sequel we assume that the structural analysis knows that cascades are bounded in length by 2.

Back-standardization: It remains to back-standardize this multi-mode dAE system. Outside collision events the back-standardization yields the known model (29) for the pendulum, with all latent equations made explicit. So, the interesting case is the handling of the events.

We begin with the case of a *single collision* involving the left and mid balls only. As for the cup-and-ball example, since double shifts $\ldots^{\bullet 2}$ are involved in the non-standard dAE system of Table VII, at an instant $t \in \mathbb{R}$ where the l/m-collision occurs, we need to consider *three* super-dense instants $(t,0) \equiv t, (t,1), (t,2)$, and the values of the positions and velocities at $(t,2)$ serve as reset values for the next mode. The l/m-collision model at $(t,0)$ is:

$$
\begin{aligned}
B_l : &\quad \begin{cases} v_l - x_l' = 0 \\ v_l^{\bullet} - (x_l')^{\bullet} = 0 \\ y_l'' - \lambda_l y_l + g = 0 \\ (x_l^2 + y_l^2)^{\bullet 2} - L^2 = 0 \end{cases} \\
B_m : &\quad \begin{cases} v_m - x_m' = 0 \\ v_m^{\bullet} - (x_m')^{\bullet} = 0 \\ y_m'' - \lambda_m y_m + g = 0 \\ (x_m^2 + y_m^2)^{\bullet 2} - L^2 = 0 \end{cases} \\
C_{lm} : &\quad v_l^{\bullet} = v_m \ \texttt{and} \ v_m^{\bullet} = v_l
\end{aligned}
\tag{88}
$$

Observe that C_{lm} already provides us with the reset value for the velocities, as expected.

The leading variables of (88) are $x^{\bullet 2}_{l/m}, y^{\bullet 2}_{l/m}, v^{\bullet}_{l/m}, \lambda_{l/m}$ and we have a block of 8 equations since the first equations from B_l and B_m are just consistency constraints. Expand derivatives

$$B_l: \left\{\begin{array}{r} v_l - x_l' = 0 \\ y_l'' - \lambda_l y_l + g = 0 \\ x_l^2 + y_l^2 - L^2 = 0 \end{array}\right.$$

$$B_m: \left\{\begin{array}{r} v_m - x_m' = 0 \\ y_m'' - \lambda_m y_m + g = 0 \\ x_m^2 + y_m^2 - L^2 = 0 \end{array}\right.$$

$$B_r: \left\{\begin{array}{r} v_r - x_r' = 0 \\ y_r'' - \lambda_r y_r + g = 0 \\ x_r^2 + y_r^2 - L^2 = 0 \end{array}\right.$$

$$C_{lm}: \left\{\begin{array}{l} x_{lm} = x_l - x_m \\ \gamma_{lm} = [x_{lm} \geq 0] \ \texttt{and} \ [v_{lm} > 0] \\ \texttt{if} \quad (\gamma_{lm}, \textbf{evt}) \ \texttt{do} \ v_l^+ = v_m \ \texttt{and} \ v_m^+ = v_l \\ \texttt{if} \ (\texttt{not} \ \gamma_{lm}, \textbf{con}) \ \texttt{do} \ v_l' - \lambda_l x_l = 0 \end{array}\right.$$

$$C_{mr}: \left\{\begin{array}{l} x_{mr} = x_m - x_r \\ \gamma_{mr} = [x_{mr} \geq 0] \ \texttt{and} \ [v_{mr} > 0] \\ \texttt{if} \quad (\gamma_{mr}, \textbf{evt}) \ \texttt{do} \ v_m^+ = v_r \ \texttt{and} \ v_r^+ = v_m \\ \texttt{if} \ (\texttt{not} \ \gamma_{mr}, \textbf{con}) \ \texttt{do} \ v_r' - \lambda_r x_r = 0 \end{array}\right.$$

$$C_{lmr}: \quad \texttt{if} \ (\texttt{not} \ \gamma_{lm}, \textbf{con}) \ \texttt{and} \ (\texttt{not} \ \gamma_{mr}, \textbf{con}) \ \texttt{do} \ v_m' - \lambda_m x_m = 0$$

$$\texttt{inv}: \quad \gamma_{lm} \wedge \gamma_{mr} = \text{F}$$

$$B_l: \left\{\begin{array}{r} v_l - x_l' = 0 \\ v_l^\bullet - (x_l')^\bullet = 0 \\ y_l'' - \lambda_l y_l + g = 0 \\ (x_l^2 + y_l^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$B_m: \left\{\begin{array}{r} v_m - x_m' = 0 \\ v_m^\bullet - (x_m')^\bullet = 0 \\ y_m'' - \lambda_m y_m + g = 0 \\ (x_m^2 + y_m^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$B_r: \left\{\begin{array}{r} v_r - x_r' = 0 \\ v_r^\bullet - (x_r')^\bullet = 0 \\ y_r'' - \lambda_r y_r + g = 0 \\ (x_r^2 + y_r^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$C_{lm}: \left\{\begin{array}{l} x_{lm} = x_l - x_m \\ \gamma_{lm} = [x_{lm} \geq 0] \ \texttt{and} \ [v_{lm} > 0] \\ \texttt{if} \quad (\gamma_{lm}, \textbf{evt}) \ \texttt{do} \ v_l^\bullet = v_m \ \texttt{and} \ v_m^\bullet = v_l \\ \texttt{if} \ (\texttt{not} \ \gamma_{lm}, \textbf{con}) \ \texttt{do} \ v_l' - \lambda_l x_l = 0 \end{array}\right.$$

$$C_{mr}: \left\{\begin{array}{l} x_{mr} = x_m - x_r \\ \gamma_{mr} = [x_{mr} \geq 0] \ \texttt{and} \ [v_{mr} > 0] \\ \texttt{if} \quad (\gamma_{mr}, \textbf{evt}) \ \texttt{do} \ v_m^\bullet = v_r \ \texttt{and} \ v_r^\bullet = v_m \\ \texttt{if} \ (\texttt{not} \ \gamma_{mr}, \textbf{con}) \ \texttt{do} \ v_r' - \lambda_r x_r = 0 \end{array}\right.$$

$$C_{lmr}: \quad \texttt{if} \ (\texttt{not} \ \gamma_{lm}, \textbf{con}) \ \texttt{and} \ (\texttt{not} \ \gamma_{mr}, \textbf{con}) \ \texttt{do} \ v_m' - \lambda_m x_m = 0$$

$$\texttt{inv}: \quad \gamma_{lm} \wedge \gamma_{mr} = \text{F}$$

Table VII

Three colliding balls l, m, r. **Left**: the system modeled using the **mDAE** language of Table IV. B_l, B_m, B_r are the models of the individual balls, i.e., copies of the model of the pendulum. C_{lm} are the equations describing the unilateral constraints between the l and m balls. The two guarded equations expand in our basic formalism **mDAE** the Modelica equation with reset: $v_{lm} = x_{lm}'$; `when` $[x_{lm} \geq 0]$ `and` $[v_{lm} > 0]$ `reset` $v_l^+ = v_m$ `and` $v_m^+ = v_l$. **Right**: the non-standard translation, where derivatives are interpreted using (22). We show in red the result of applying Algorithm 8 for unlocking fixpoints and in blue the result of Algorithm 9 for revealing latent equations. The resulting non-standard system has a successful execution scheme.

in terms of shifts using (22):

$$B_l: \left\{\begin{array}{r} \partial v_l^\bullet - (x_l^{\bullet 2} - x_l^\bullet) = 0 \\ y_l^{\bullet 2} - 2y_l^\bullet + y_l - \partial^2(\lambda_l y_l - g) = 0 \\ (x_l^2 + y_l^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$B_m: \left\{\begin{array}{r} \partial v_m^\bullet - (x_m^{\bullet 2} - x_m^\bullet) = 0 \\ y_m^{\bullet 2} - 2y_m^\bullet + y_m - \partial^2(\lambda_m y_m - g) = 0 \\ (x_m^2 + y_m^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$C_{lm}: \quad v_l^\bullet = v_m \ \texttt{and} \ v_m^\bullet = v_l$$

These equations involve the infinitesimal parameter ∂, which we must get rid of when back-standardizing. As a first action, we set $\partial = 0$ in the first equation of $B_{l/m}$ — we can safely do this since it causes no singularity. We thus infer that positions $x_{l/m}$ experience no discontinuity: $x_{l/m}^{\bullet 2} = x_{l/m}^\bullet$. We cannot do the same with the second equation of $B_{l/m}$ since a singularity would result. We thus expand the two tensions $\lambda_{l/m}$ as formal power series $\lambda = \lambda_0 + \lambda_1 \partial^{-1} + \lambda_2 \partial^{-2}$ and we consider the system of equations resulting from setting $\partial = 0$:

$$B_l: \left\{\begin{array}{r} (x_l^{\bullet 2} - x_l^\bullet) = 0 \\ y_l^{\bullet 2} - 2y_l^\bullet + y_l - \lambda_{l,2} y_l = 0 \\ (x_l^2 + y_l^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$B_m: \left\{\begin{array}{r} (x_m^{\bullet 2} - x_m^\bullet) = 0 \\ y_m^{\bullet 2} - 2y_m^\bullet + y_m - \lambda_{m,2} y_m = 0 \\ (x_m^2 + y_m^2)^{\bullet 2} - L^2 = 0 \end{array}\right.$$

$$C_{lm}: \quad v_l^\bullet = v_m \ \texttt{and} \ v_m^\bullet = v_l$$

We solve this model for the leading variables $x_{l/m}^{\bullet 2}$, $y_{l/m}^{\bullet 2}$, $v_{l/m}^\bullet$, and $\lambda_{(l,2)/(m,2)}$. From this model we infer that positions $x_{l/m}$ experience no discontinuity: $x_{l/m}^{\bullet 2} = x_{l/m}^\bullet$, hence neither does vertical position $y_{l/m}$ due to the length constraint. Consequently the tensions are zero.

Discussion: This example is interesting as its structural analysis requires both Algorithm 8 for unlocking fixpoints and Algorithm 9 for revealing latent equations.

Cascades of two immediate collision events are easily handled. We just perform these sequences of computations in sequence. The consideration of collisions occurring at arbitrary places would not change much: the equations on y-coordinates would get affected and the expression on the conservation laws would be slightly more complex. Modeling simultaneous collision would require an extra effort and cause one more mode to become reachable.

Clearly, the style of modeling we followed is low level and unelegant, since it specifies the effect of collisions in a somehow "imperative" way. In addition the model of collisions is global. Thus, adding one more ball affects the whole set of interaction constraints. It would be much preferable to adopt a more equational modeling style, alike the use of complementarity conditions. We also guess this would improve modeling modularity. The reader is referred to [1], Section 1.4 for such a development.

V. Mathematical justification of our approach

In this section we provide the mathematical arguments justifying the back-standardization procedure of Section IV-C. In doing so we face the following difficulty. Our aim is to relate

what our execution sheme computes to the actual solutions of the multi-mode DAE system. We want something like: "every execution of our scheme yields a solution of the multi-mode DAE system". This requires having a definition of what a solution is, for a multi-mode DAE system. Unfortunately this does not exist. While solutions of (single-mode) DAE systems are well defined (although not obviously so), what a solution should do at the events is not — at least we are not aware of any such definition except through hand waving. In some sense, our execution scheme at the events is one proposal for a definition of a solution at the events. Our objective in this section is therefore more modest: make sure that our scheme computes the right dynamics in each mode of positive duration.

Since doing so requires a non-trivial use of non-standard analysis, we provide as background a short introduction to it.

A. A primer on non-standard analysis

Non-standard analysis was proposed by Abraham Robinson in the 1960s to allow the explicit manipulation of "infinitesimals" in analysis [23], [11]. Robinson's approach is axiomatic; he proposes adding three new axioms to the basic Zermelo-Fraenkel (ZFC) framework. An alternative presentation was later proposed by Lindstrøm [18]. Its interest is that it does not require any fancy axiomatic material but only makes use of the axiom of choice — actually a weaker form of it. The proposed construction bears some resemblance to the construction of $\mathbb{R}$ as the set of equivalence classes of Cauchy sequences in $\mathbb{Q}$ modulo the equivalence relation $(u_n) \approx (v_n)$ iff $\lim_{n\to\infty}(u_n - v_n) = 0$. The important thing for us is that non-standard analysis allows the use of the non-standard discretization of continuous dynamics "as if" it was operational. Iwasaki et al. [16] first proposed using non-standard analysis to discuss the nature of time in hybrid systems. Bliudze and Krob [8], [7] have also used non-standard analysis as a mathematical support for defining a system theory for hybrid systems. In their study of mathematical foundations of hybrid systems modelers of the ODE class, Benveniste et al. [3] used extensively the non-standard semantics of hybrid systems. The following presentation is borrowed verbatim from [3].

1) Intuitive introduction: We begin with an intuitive introduction to the construction of the non-standard reals. The goal is to augment $\mathbb{R}\cup\{\pm\infty\}$ by adding, to each x in the set, a set of elements that are "infinitesimally close" to it. We will call the resulting set ${}^\star\mathbb{R}$. Another requirement is that all operations and relations defined on $\mathbb{R}$ should extend to ${}^\star\mathbb{R}$.

A first idea is to represent such additional numbers as convergent sequences of reals. For example, elements infinitesimally close to the real number zero are the sequences $u_n = 1/n$, $v_n = 1/\sqrt{n}$ and $w_n = 1/n^2$. Observe that the above three sequences can be ordered: $v_n > u_n > w_n > 0$ where 0 denotes the constant zero sequence. Of course, infinitely large elements (close to $+\infty$) can also be considered, e.g., sequences $x_u = n$, $y_n = \sqrt{n}$, and $z_n = n^2$.

Unfortunately, this way of defining ${}^\star\mathbb{R}$ does not yield a total order since two sequences converging to zero cannot always be compared: if u_n and u'_n are two such sequences, the three sets $\{n \mid u_n > u'_n\}$, $\{n \mid u_n = u'_n\}$, and $\{n \mid u_n < u'_n\}$ may even all be infinite. The beautiful idea of Lindstrøm is to enforce that *exactly one of the above sets is important and the other two can be neglected*. This is achieved by fixing once and for all a finitely additive positive measure μ over the set $\mathbb{N}$ of integers with the following properties:[5]

1) $\mu : 2^{\mathbb{N}} \to \{0, 1\}$;
2) $\mu(X) = 0$ whenever X is finite;
3) $\mu(\mathbb{N}) = 1$.

Now, once μ is fixed, one can compare any two sequences: for the above case, exactly one of the three sets must have μ-measure 1 and the others must have μ-measure 0. Thus, say that $u>u', u=u'$, or $u<u'$, if $\mu(\{n \mid u_n>u'_n\})=1$, $\mu(\{n \mid u_n=u'_n\})=1$, or $\mu(\{n \mid u_n<u'_n\})=1$, respectively. Indeed, the same trick works for many other relations and operations on non-standard real numbers, as we shall see. We now proceed with a more formal presentation.

2) Non-standard domains: For I an arbitrary set, a *filter* $\mathcal{F}$ over I is a family of subsets of I such that:

1) the empty set does not belong to $\mathcal{F}$,
2) $P, Q \in \mathcal{F}$ implies $P \cap Q \in \mathcal{F}$, and
3) $P \in \mathcal{F}$ and $P \subset Q \subseteq I$ implies $Q \in \mathcal{F}$.

Consequently, $\mathcal{F}$ cannot contain both a set P and its complement P^c. A filter that contains one of the two for any subset $P \subseteq I$ is called an *ultra-filter*. At this point we recall Zorn's lemma, known to be equivalent to the axiom of choice:

Lemma 3 (Zorn lemma): Any partially ordered set $(X, \leq)$ such that any chain in X possesses an upper bound has a maximal element.

A filter $\mathcal{F}$ over I is an ultra-filter if and only if it is maximal with respect to set inclusion. By Zorn lemma, any filter $\mathcal{F}$ over I can be extended to an ultra-filter over I. Now, if I is infinite, the family of sets $\mathcal{F} = \{P \subseteq I \mid P^c \text{ is finite}\}$ is a *free* filter, meaning it contains no finite set. It can thus be extended to a free ultra-filter over I:

Lemma 4: Any infinite set has a free ultra-filter.

Every free ultra-filter $\mathcal{F}$ over I uniquely defines, by setting $\mu(P) = 1$ if $P \in \mathcal{F}$ and otherwise 0, a finitely additive measure[6] $\mu : 2^I \mapsto \{0, 1\}$, which satisfies

$$\mu(I) = 1 \text{ and, if } P \text{ is finite, then } \mu(P) = 0.$$

Now, fix an infinite set I and a finitely additive measure μ over I as above. Let $\mathbb{X}$ be a set and consider the Cartesian product $\mathbb{X}^I = (x_i)_{i\in I}$. Define $(x_i) \approx (x'_i)$ if and only if $\mu\{i \in I \mid x_i \neq x'_i\} = 0$. Relation $\approx$ is an equivalence relation whose equivalence classes are denoted by $[x_i]$ and we define

$$ {}^\star\mathbb{X} = \mathbb{X}^I / \approx \tag{89}$$

[5]The existence of such a measure is non trivial and is explained later.

[6]Observe that, as a consequence, μ cannot be sigma-additive (in contrast to probability measures or Radon measures) in that it is *not* true that $\mu(\bigcup_n A_n) = \sum_n \mu(A_n)$ holds for an infinite denumerable sequence A_n of pairwise disjoint subsets of $\mathbb{N}$.

$\mathbb{X}$ is naturally embedded into ${}^\star\mathbb{X}$ by mapping every $x \in \mathbb{X}$ to the constant tuple such that $x_i = x$ for every $i \in I$, we denote it by

$$[x]\,.$$

Any algebraic structure over $\mathbb{X}$ (group, ring, field) carries over to ${}^\star\mathbb{X}$ by almost point-wise extension. In particular, if $[x_i] \neq 0$, meaning that $\mu\{i \mid x_i = 0\} = 0$ we can define its inverse $[x_i]^{-1}$ by taking $y_i = x_i^{-1}$ if $x_i \neq 0$ and $y_i = 0$ otherwise. This construction yields $\mu\{i \mid y_i x_i = 1\} = 1$, whence $[y_i][x_i] = 1$ in ${}^\star\mathbb{X}$. The existence of an inverse for any non-zero element of a ring is indeed stated by the formula: $\forall x\,(x = 0 \vee \exists y\,(xy = 1))$. More generally:

Lemma 5 (Transfer Principle): Every first order formula is true over ${}^\star\mathbb{X}$ if and only if it is true over $\mathbb{X}$.

3) Non-standard reals and integers: The above general construction can simply be applied to $\mathbb{X} = \mathbb{R}$ and $I = \mathbb{N}$. The result is denoted ${}^\star\mathbb{R}$; it is a field according to the transfer principle. By the same principle, ${}^\star\mathbb{R}$ is totally ordered by $[u_n] \leq [v_n]$ iff $\mu\{n \mid v_n > u_n\} = 0$.

Call *infinitesimal* any non-standard real number whose absolute value is smaller than any positive real number: $[x_n] \leq [\varepsilon]$ and $[x_n] \geq [-\varepsilon]$ for any $\varepsilon \in \mathbb{R}, \varepsilon > 0$.

Lemma 6: For any finite $[x_n] \in {}^\star\mathbb{R}$, there exists a unique standard real number $x \in \mathbb{R}$, such that $[x] - [x_n]$ is infinitesimal. We call x the standard part *of $[x_n]$ and denote it by $st([x_n])$.*

Proof: To prove this, let $x = \sup\{u \in \mathbb{R} \mid [u] \leq [x_n]\}$. Since $[x_n]$ is finite, x exists and we only need to show that $[x_n] - x$ is infinitesimal. If not, then there exists $y \in \mathbb{R}, y > 0$ such that either $[x] < [x_n] - [y]$ or $[x] > [x_n] + [y]$, which both contradict the definition of x. The uniqueness of x is clear, thus we can define $st([x_n]) = x$. ∎

Infinite non-standard reals have no standard part in $\mathbb{R}$. It is also of interest to apply the general construction (89) to $\mathbb{X} = I = \mathbb{N}$, which results in the set ${}^\star\mathbb{N}$ of *non-standard natural numbers*. The non-standard set ${}^\star\mathbb{N}$ differs from $\mathbb{N}$ by the addition of *infinite natural numbers,* which are equivalence classes of sequences of integers whose essential limit is $+\infty$.

4) Derivatives, Integrals, and ODE: Any sequence (g_n) of functions $g_n : \mathbb{R} \mapsto \mathbb{R}$ point-wise defines a function $[g_n] : {}^\star\mathbb{R} \mapsto {}^\star\mathbb{R}$ by setting

$$[g_n]([x_n]) \quad =_{\text{def}} \quad [g_n(x_n)]\,. \tag{90}$$

A function ${}^\star\mathbb{R} \to {}^\star\mathbb{R}$ so obtained is called *internal*. Properties of and operations on ordinary functions extend point-wise to internal functions of ${}^\star\mathbb{R} \to {}^\star\mathbb{R}$. The *non-standard version* of $g : \mathbb{R} \to \mathbb{R}$ is the internal function ${}^\star g = [g, g, g, \dots]$. The same notions apply to sets. An internal set $A = [A_n]$ is called *hyperfinite* if $\mu\{n \mid A_n \text{ finite}\} = 1$; the *cardinal* $|A|$ of A is defined as $[|A_n|]$, where $|A_n|$ denotes the cardinal of the finite set A_n in the usual sense.

Let $g : \mathbb{R} \to \mathbb{R}$ be a standard differentiable function with ${}^\star g = [g, g, g, \dots]$ its associated internal function. Then, for $x \in \mathbb{R}$, we have

$$g'(x) \quad = \quad st\left(\frac{{}^\star g(y) - {}^\star g([x])}{y - [x]}\right) \tag{91}$$

for every $y \in {}^\star\mathbb{R}$ such that $y \approx [x]$ — the proof of this is similar to that of Lemma 6.

Now, consider an infinite number $K \in {}^\star\mathbb{N}$ and the set

$$T = \left\{0, \frac{1}{K}, \frac{2}{K}, \frac{3}{K}, \dots \frac{K-1}{K}, 1\right\} \tag{92}$$

By definition, if $K = [K_n]$, then $T = [T_n]$ with

$$T_n = \left\{0, \frac{1}{K_n}, \frac{2}{K_n}, \frac{3}{K_n}, \dots \frac{K_n - 1}{K_n}, 1\right\}$$

hence $|T| = [|T_n|] = [K_n + 1] = K + 1$. Now, consider an internal function $g = [g_n]$ and a hyperfinite set $A = [A_n]$. The *sum* of g over A can be defined:

$$\sum_{a \in A} g(a) =_{\text{def}} \left[\sum_{a \in A_n} g_n(a)\right]$$

If T is as above, and $f : \mathbb{R} \to \mathbb{R}$ is a standard function, we obtain

$$\sum_{t \in T} \frac{1}{|T|}\, {}^\star f(t) = \left[\sum_{t \in T_n} \frac{1}{|T_n|} f(t_n)\right]. \tag{93}$$

Now, f continuous implies the convergence of the Riemann sums: $\sum_{t \in T_n} \frac{1}{|T_n|} f(t_n) \to \int_0^1 f(t)dt$. Hence,

$$\int_0^1 f(t)dt \quad = \quad st\left(\sum_{t \in T} \frac{1}{|T|}\, {}^\star f(t)\right). \tag{94}$$

Under the same assumptions, for any $t \in [0, 1]$,

$$\int_0^t f(u)du \quad = \quad st\left(\sum_{u \in T, u \leq t} \frac{1}{|T|}\, {}^\star f(t)\right). \tag{95}$$

Now, consider the following ODE:

$$x' = f(x, t), \quad x(0) = x_0\,. \tag{96}$$

Assume (96) possesses a solution $[0, 1] \ni t \mapsto x(t)$ such that the function $t \mapsto f(x(t), t)$ is continuous. Rewriting (96) in its equivalent integral form $x(t) = x_0 + \int_0^t f(x(u), u)du$ and using (95) yields

$$x(t) = st\left(x_0 + \sum_{u \in T, u \leq t} \frac{1}{|T|}\, {}^\star f(x(u), u)\right). \tag{97}$$

The substitution in (97) of $\partial = 1/|T|$, which is positive and infinitesimal, yields $T = \{t_n = n\partial \mid n = 0, \dots, |T|\}$. The expression in parentheses on the right hand side of (97) is the piecewise-constant right-continuous function ${}^\star x(t), t \in [0, 1]$ such that, for $k = 1, \dots, K$:

$$\begin{array}{rcl} {}^\star x(t_k) & = & {}^\star x(t_{k-1}) + \partial\,.\,{}^\star f({}^\star x(t_{k-1}), t_{k-1}) \\ {}^\star x(t_0) & = & x_0 \end{array} \tag{98}$$

By (97), the solutions x, of ODE (96), and ${}^\star x$, as computed by algorithm (98), are related by $x = st({}^\star x)$. Formula (98) can be seen as a *non-standard semantics* for ODE (96). This semantics seems to depend on the choice of infinitesimal step parameter ∂. Property (97), though, expresses the idea that all these non-standard semantics are equivalent from the standard viewpoint regardless of the choice made for ∂. This fact is referred to as the *standardization principle*.

The same line of arguments applies for systems of ODE possessing a unique continuous solution, i.e., x and f take their values in $\mathbb{R}^m$ in (96). Invoking the implicit function theorem, one can also address DAEs

$$F(x, x', t) = 0 \tag{99}$$

provided that the Jacobian $\frac{\partial}{\partial v}F(x, v, t)$ is invertible in a neighborhood of a solution in some open subset of times. The kind of index reduction procedure we use precisely returns such a form of DAE system. The following lemma holds for this class of DAE systems:

Lemma 7: Let ${}^\star x$ be the piecewise interpolation of the following non-standard scheme: solve for ${}^\star x(t_{k+1})$

$$F\left({}^\star x_{t_k}, \frac{{}^\star x_{t_{k+1}} - {}^\star x_{t_k}}{\partial}, t_k\right) = 0\,. \tag{100}$$

Then, $x =_{\text{def}} st({}^\star x)$ is solution of DAE (99).

In the next section we freely use the same symbol to represent a standard real number x and its non-standard embedding $[x] \in {}^\star\mathbb{R}$, thus writing expressions such as $x + \zeta$ for x standard and ζ non-standard.

B. Main theorem

For S a multi-mode DAE system specified using the **mDAE** language of Table IV, we will now define what its *solution* is. To simplify the exposure and without loss of generality, we assume that S involves at most first order derivatives.

Notations and terminology:

- Let $X_S \subset \boldsymbol{\Xi}$ and $X_S^{\mathsf{der}} \subset \boldsymbol{\Xi}$ denote the largest sets of *num* variables such that every $x \in X_S$ and every y' where $y \in X_S^{\mathsf{der}}$, are dependent variables of S. When S reduces to a single equation e we write X_e and X_e^{der}. For example, for the equation $e : f(x, x', u) = 0$ having dependent variables x, x', u, we have $X_e = \{x, u\}$ and $X_e^{\mathsf{der}} = \{x\}$. With reference to this example, say that $[\![e]\!]$ is *smooth* if $(x, v, u) \to f(x, v, u)$ is sufficiently differentiable.
- Let E_S denote the set of variables of type *eqn* of S, i.e., its set of guarded equations. For $e \in E_S$, let $(\gamma(e), \Delta(e))$ and $[\![e]\!]$ denote the *guard* and the *body* of e, respectively and let X_e be the set of dependent variables of $[\![e]\!]$. The set $\{\gamma(e) \mid e \in E_S\}$ is denoted by Γ_S.
- Let P_S denote the set of all predicates *pred* of system S and let $\mathbb{B} =_{\text{def}} \{\mathrm{F}, \mathrm{T}\}$ denote the Boolean domain. Say that $p \in P_S$ where $p : \mathbb{R}^d \to \mathbb{B}$, is *smooth* if $p^{-1}(\mathrm{T})$ is a closed subset of $\mathbb{R}^d$. An example of a smooth predicate is $p(x, y) = (x + y \geq a)$, where a is a real number.
- System S is called *smooth* if all its predicates are smooth and all its equations possess a smooth body.
- For $\Phi \in \mathbb{R}^{X_S}$ and $x \in X_S$, Φ_x denotes the xth coordinate of Φ, and, for $\Gamma \in \mathbb{B}^{E_S}$ and $e \in E_S$, Γ_e denotes the eth coordinate of Γ.
- Recall that $\mathbb{T}$ is the non-standard time set defined in (13) and $\mathbf{T}$ is the super-dense time set defined in (84). □

In the following definition, the *slanted texts in blue* are informal comments.

Definition 3 (solution of S): Let S be a smooth **mDAE** system. A *solution* of system S is a pair of functions

$$\begin{array}{rcl} (\mathbf{t}, \lambda) & \to & (\Phi(\mathbf{t}, \lambda), \Gamma(\mathbf{t}, \lambda)), \text{ of type} \\ \mathbf{T} \times \Lambda & \to & \left(\mathbb{R}^{X_S} \times \mathbb{B}^{E_S}\right) \cup \{\epsilon\} \end{array}$$

where ϵ is the *undefined* value and Λ is some nonempty open set of $\mathbb{R}^\rho$, satisfying the following conditions:

1) For every $\lambda \in \Lambda$, there exists a height function $t \to h(t, \lambda)$ from $\mathbb{R}_+$ to $\mathbb{N}$ such that $(\Phi(\mathbf{t}, \lambda), \Gamma(\mathbf{t}, \lambda)) = \epsilon$ if and only if $\mathbf{t} = (t, k)$ and $k > h(t, \lambda)$. Furthermore, $h(t, \lambda) = 0$ except for t belonging to some increasing sequence $T(\lambda) = \{t_k(\lambda) \mid k \in \mathbb{Z}\}$ of instants of $\mathbb{R}_+$ satisfying $\lim_{k \to \pm\infty} t_k(\lambda) = \pm\infty$. Write t_k and $h(t_k)$ instead of $t_k(\lambda)$ and $h(t_k(\lambda), \lambda)$ when no confusion can result.

   (t_k) is the sequence of events of the considered run, collecting the zero-duration modes and the mode changes.

2) Regarding the modes: the function $t \to \Gamma((t, 0), \lambda)$ is constant over every open interval (t_k, t_{k+1}) and, denoting by $B_k \in \mathbb{B}^{E_S}$ the corresponding value, we have

$$\Gamma((t_k, h(t_k)), \lambda) \;=\; B_k \;=\; \Gamma((t_{k+1}, 0), \lambda)$$

   The last and first super-dense instants before and after the current mode belong to this mode too.

3) Regarding the numerical variables:

   a) For every $\lambda \in \Lambda$ and every interval (t_k, t_{k+1}), the map $t \to \Phi(t, \lambda)$ is a diffeomorphism from (t_k, t_{k+1}) into $\mathbb{R}^{X_S}$ and, for every $e \in E_S$ such that $\Gamma_e(t, \lambda) = \mathrm{T}$ holds for $t \in (t_k, t_{k+1})$, then

$$[\![e]\!]\left(\Phi_{X_e}(t, \lambda), \tfrac{d}{dt}\Phi_{X_e^{\mathsf{der}}}(t, \lambda)\right) = 0 \tag{101}$$

   holds for every $t \in (t_k, t_{k+1})$.

   All of the DAEs that are active in the current mode are satisfied by the numerical part of the solution.

   b) For every $\lambda \in \Lambda$ and every $t \in T(\lambda)$

$$\begin{array}{rcl} \Phi((t, 0), \lambda) & = & \Phi((t^-, 0), \lambda) \\ \Phi((t, h(t)), \lambda) & = & \Phi((t^+, 0), \lambda) \end{array}$$

   where t^- and t^+ are the left- and right-limits at t.

   At any event, the first values of the numerical variables are their left-limits at the exited mode of positive duration and the last values of the numerical variables give the reset values of these variables for the entered mode of positive duration.

4) If $\mathbf{t} \to (X_S(\mathbf{t}), \Gamma_S(\mathbf{t}))$ satisfies the above conditions 1–3, then $(X_S(\mathbf{t}), \Gamma_S(\mathbf{t})) = (\Phi(\mathbf{t}, \lambda), \Gamma(\mathbf{t}, \lambda))$ holds for some $\lambda \in \Lambda$.

λ parameterizes the degrees of freedom.

Say that S is *solvable* if it possesses a solution. □

Note that Definition 3 says nothing about the values taken by the system trajectories at the successive super-dense instants that unfold an event. Condition 3b only constrains the first and last values of this finite sequence. We adopt this seemingly weak definition because: 1) the intermediate values may involve impulses (as we have seen from several examples), and 2) the examples showed that there is no obvious definition of what the intermediate steps of computation should be, besides our proposed execution scheme.

*Theorem 2: Let S be solvable. Assume that its structural analysis (Algorithm 7) succeeds and let ${}^\star S$ be the non-standard **mdAE** system returned by it. Then, back-standardizing any solution of ${}^\star S$ yields a solution of S following Definition 3.*

Theorem 2 justifies our approach.

So far our assumptions require that the dynamics in each mode of positive duration is smooth. This rules out regarding non-smooth systems (involving complementarity conditions) as being single-mode. It would be of interest to see if the above theorem extends if the class of atomic actions comprises complementarity conditions.

Proof: Let $\{{}^\star\Phi(\tau) \mid \tau \in \mathbb{T}\}$ be the behavior of numerical variables generated by executing the non-standard system ${}^\star S$. Since Definition 3 tells nothing about what happens at the events, we only need to consider the dynamics occurring in the interior of each mode of positive duration, that is, the current non-standard instant $\tau \in \mathbb{T}$ is such that, for some finite non-infinitesimal $\varepsilon > 0$ and every $\sigma \in U(\tau, \varepsilon)$, where

$$U(\tau, \varepsilon) \quad =_{\text{def}} \quad \{\sigma \in \mathbb{T} \mid |\sigma - \tau| < \varepsilon\},$$

the following holds:

$$\text{for every equation } e \text{ whose guard has zero-duration type, i.e., } e : \texttt{when } (\gamma(e), \mathbf{evt}) \texttt{ do } [\![e]\!], \text{ then } \gamma(e, \sigma) = \text{F}. \tag{102}$$

So, for any active equation e at σ, the guard of e has positive duration type. Let

$$e : \texttt{when } (\gamma(e), \mathbf{con}) \texttt{ do } [\![e]\!]$$

be an active equation, i.e., such that $\gamma(e, \sigma) = \text{T}$ holds for every $\sigma \in U(\tau, \varepsilon)$. By construction of ${}^\star S$, we have:

$$\begin{array}{c} \forall \sigma \in U(\tau, \varepsilon) \\ \Downarrow \\ [\![e]\!]\left({}^\star\Phi_{X_e}(\sigma), ({}^\star\Phi_{X_e^{\text{der}}})'(\sigma)\right) = 0\,. \end{array} \tag{103}$$

The theorem then follows from Lemma 7. ■

VI. Conclusion

In this paper we developed a comprehensive mathematical approach to the structural analysis of multi-mode DAE systems of the form

$$\texttt{when } \gamma_i \texttt{ do } f_i(\text{the } x_j \text{ and derivatives of them}) = 0\,. \tag{104}$$

We first observed that a single-mode DAE system and its explicit first order Euler approximation possess identical structural analyses, regardless of the step size. We thus proposed to take a step size that is *infinitesimal in the sense of non-standard analysis,* so that the Euler scheme is no longer an approximation but rather a re-interpretation of the original system, in the non-standard domain. Building on top of this, we were able to propose a full mathematical study of the structural analysis of multi-mode DAE systems.

Our first contribution is the structural analysis of multi-mode dAE systems, i.e., discrete time systems of equations of the form

$$\texttt{when } \gamma_i \texttt{ do } f_i(\text{the } x_j \text{ and forward shifts of them}) = 0\,. \tag{105}$$

Our approach borrowed ideas from the *constructive semantics* of synchronous languages, sort of a structural analysis of synchronous programs. Two key concepts emerged, namely: the *atomic actions*, which are deferred to some external solver, and their *scheduling*. Our constructive semantics for multi-mode dAE systems characterizes the systems that can/cannot be handled, specifies the correct execution schemes, and indicates how latent constraints should be found.

The next question is how should a multi-mode DAE system of the form (104) be mapped to the non-standard domain, thus giving raise to a multi-mode dAE system of the form (105). A mapping to the non-standard domain by simply using the expansion $x'_t = \frac{1}{\partial}(x_{t+\partial} - x_t)$, where ∂ is an infinitesimal time step, can result in a multi-mode dAE system exhibiting logico-numerical fixpoint equations, which do not belong to our class of eligible atomic to be deferred to solvers. Our second contribution is the analysis of how this mapping should be performed while avoiding the above kind of logico-numerical fixpoint.

Having done this mapping, we are left with a multi-mode dAE system of the form (105), to which our structural analysis applies. If the latter is successful, our last duty is to generate executable *standard* code — i.e., not in the non-standard domain. As our standard domain, we target hybrid systems in *super-dense* time, in which sequences of micro-steps to be performed for resets at mode changes can be captured. Our third contribution is the definition of how this "back-standardization" should be performed.

Rebuting the objections raised in [17] page 801, our results once more demonstrate the effectiveness and usefulness of non-standard analysis as a semantic domain for hybrid systems.

This structural analysis is implemented in the SUNDAE PoC tool developed by Benoît Caillaud, which takes, as input, systems of the form (104), and returns the coordination code controlling the numerical solvers.

We believe that our approach provides the ultimate answer to the compilation of multi-mode DAE systems. Much remains to be done, however. We must show that efficient code for large systems can be obtained, at least as good as the heuristics performed by the current compilers for the models they accept. Since our approach is compatible with non-smooth systems solvers, we need to understand when using them is better than keeping guarded equations with smooth dynamics. Last but not least, the execution schemes we provide for multi-mode dAE systems require *local* equation solvers operating on a per-block basis. In contrast, our final code uses a global solver where all blocks are merged — as it is currently done by DAE based modelers. It is thus tempting to investigate the use of *Quantized State Solvers* (QSS), in which system states are quantized and time progresses locally for each variable, until the current quantized value must be left [10], [13]. QSS solvers implement discrete event approximations of continuous time systems. Being event-based rather than time-based, QSS methods are much closer to the kind of code we would like to target.

REFERENCES

[1] Acary Vincent and Brogliato Bernard, *Numerical Methods for Nonsmooth Dynamical Systems*, ser. Lecture Notes in Applied and Computational Mechanics. Springer-Verlag, 2008, vol. 35.

[2] A. Benveniste, B. Caillaud, and P. L. Guernic, "Compositionality in dataflow synchronous languages: Specification and distributed code generation," *Inf. Comput.*, vol. 163, no. 1, pp. 125–171, 2000.

[3] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet, "Nonstandard semantics of hybrid systems modelers," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 877–910, 2012.

[4] ——, "On the index of multi-mode DAE Systems (also called Hybrid DAE Systems)," , Research Report RR-8630, Nov. 2014. [Online]. Available: https://hal.inria.fr/hal-01084069

[5] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[6] G. Berry, "Constructive semantics of Esterel: From theory to practice (abstract)," in *AMAST '96: Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology*. London, UK: Springer-Verlag, 1996, p. 225.

[7] S. Bliudze, "Un cadre formel pour l'étude des systèmes industriels complexes: un exemple basé sur l'infrastructure de l'UMTS," Ph.D. dissertation, Ecole Polytechnique, 2006.

[8] S. Bliudze and D. Krob, "Modelling of complex systems: Systems as dataflow machines," *Fundam. Inform.*, vol. 91, no. 2, pp. 251–274, 2009.

[9] F. Boussinot and R. de Simone, "The esterel language," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1293–1304, September 1991.

[10] F. Cellier and E. Kofman, *Continuous System Simulation*. Springer-Verlag, 2006.

[11] N. Cutland, *Nonstandard analysis and its applications*. Cambridge Univ. Press, 1988.

[12] H. Elmqvist, S.-E. Mattsson, and M. Otter, "Modelica extensions for multi-mode DAE systems," in *Proc. of the 10th Int. Modelica Conference*, H. Tummescheit and K.-E. Arzén, Eds. Lund, Sweden: Modelica Association, Sep. 2014.

[13] J. Fernández and E. Kofman, "A Stand-Alone Quantized State System Solver for Continuous System Simulation." *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 90, no. 7, pp. 782–799, 2014. [Online]. Available: files/qss_solver_v4.pdf

[14] P. L. Guernic, T. Gautier, M. L. Borgne, and C. L. Maire, "Programming real-time applications with SIGNAL," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1321–1336, September 1991.

[15] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous dataflow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, September 1991.

[16] Y. Iwasaki, A. Farquhar, V. Saraswat, D. Bobrow, and V. Gupta, "Modeling time in hybrid systems: How fast is "instantaneous"?" in *IJCAI*, 1995, pp. 1773–1781.

[17] E. A. Lee, "Constructive models of discrete and continuous physical phenomena," *IEEE Access*, vol. 2, pp. 797–821, 2014. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2014.2345759

[18] T. Lindstrøm, "An invitation to nonstandard analysis," in *Nonstandard Analysis and its Applications*, N. Cutland, Ed. Cambridge Univ. Press, 1988, pp. 1–105.

[19] S.-E. Mattsson, M. Otter, and H. Elmqvist, "Multi-Mode DAE Systems with Varying Index," in *Proc. of the 11th Int. Modelica Conference*, H. Elmqvist and P. Fritzson, Eds. Versailles, France: Modelica Association, Sep. 2015.

[20] C. Pantelides, "The consistent initialization of differential-algebraic systems," *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 2, pp. 213–231, 1988.

[21] F. Pfeiffer and C. Glocker, *Multibody Dynamics with Unilateral Contacts*. Wiley, 2008.

[22] J. D. Pryce, "A simple structural analysis method for DAEs," *BIT*, vol. 41, no. 2, pp. 364–394, 2001.

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

[24] Stephen L. Campbell and C. William Gear, "The index of general nonlinear DAEs," *Numer. Math.*, vol. 72, pp. 173–196, 1995.

[25] Sven Erik Mattsson and Gustaf Söderlind, "Index reduction in Differential-Algebraic Equations using dummy derivatives," *Siam J. Sci. Comput.*, vol. 14, no. 3, pp. 677–692, 1993.

[26] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM Journal on Computing*, 1972.

APPENDIX

A. Summary of Algorithms for the structural analysis

1) Multi-Mode dAE systems:

Algorithm 1 (Constructive Semantics of dAEs): For multi-mode dAE systems, the constructive semantics consists in:

1) Performing initialization using Algorithm 2;
2) Applying one of the following micro-steps, with the following set of priorities:

$$\begin{array}{ll} & \text{Algorithm 3 for performing a } \textit{tick} \\ < & \text{Algorithm 4 for enforcing invariants} \\ < & \text{Algorithm 5 for the evaluation of blocks} \\ < & \text{Algorithm 6 for the evaluation of predicates} \end{array} \tag{106}$$

until termination — which occurs in finitely many micro-steps, see Theorem 1 below. □

Algorithm 2 (initialization): An initialization is a coherent status σ_0 assigning a (possibly undefined) value to all variables in a way that satisfies S:

$$\textit{coherent}(\sigma_0) \quad \text{and} \quad \sigma_0 \models S \tag{107}$$

Mark as enabled all predicates of S involving variables that are defined:

$$\begin{array}{l} \sigma_0 : !\Phi_0, \text{ where} \\ \Phi_0 =_{\text{def}} \{p(X) : \textit{pred} \mid \forall x \in X \Rightarrow \sigma_0(x) \neq \star\} \end{array} \tag{108}$$

whereas no equation is marked enabled. By convention trivial predicates having constant value T are marked enabled. □

Algorithm 3 (tick): This algorithm applies if the current status σ satisfies the following conditions:

1) σ meets the invariants: $\sigma(\iota) \neq \text{F}$ for every invariant ι;
2) $\sigma(\xi) \neq \star$ for every Scott variable ξ of S that either belongs to $\widehat{S}$ (see (71)) or is useful in S at the status σ.

We then set $\sigma_K =_{\text{def}} \sigma$ and perform a *tick,* which is a special update $\sigma_K \to \sigma_0^\bullet$, such that, for every x:

$$\sigma_0^\bullet(x) = \left\{ \begin{array}{rl} \textbf{if } \sigma_K(x^\bullet) \in \{\bot, \text{F}, \text{T}\} \textbf{ then} & \sigma_K(x^\bullet) \\ \textbf{else} & \star \end{array} \right. \tag{109}$$

The status $\sigma_0^\bullet$ is the initial status for the next instant. □

Algorithm 4 (invariants): If $\sigma(\iota) = \text{F}$ for some invariant ι, then, σ violates S and no successor of it can change this. We thus backtrack to the previous status ${}^\circ\sigma$ and terminate the run by setting $\sigma_K = {}^\circ\sigma$. Otherwise, σ is validated and the constructive semantics can further proceed. □

Algorithm 5 (blocks): Select *non-deterministically* a maximal subset of enabled blocks that are pairwise *independent*: $\beta(\sigma, E) \cap \beta(\sigma, E') = \emptyset$. Solve these blocks, which updates σ to σ° by assigning a value T to more numerical variables. Update the block function β accordingly. The choice between different enabled blocks gives priority to the blocks with smallest maximal-shifting-degree. □

Algorithm 6 (predicates): Let Φ be the subset of predicates $\varphi(X)$ that are enabled in σ, i.e., $\sigma : !\Phi$. Move to the status σ° such that, for every $\varphi \in \Phi$, $\sigma^\circ(\varphi) \in \{\bot, \text{T}, \text{F}\}$ is *non-deterministically* selected so that σ° is coherent in the sense of (64) and satisfies:

$$\forall \xi \in X : \sigma(\xi) = \text{T} \quad \Longrightarrow \quad \sigma^\circ(\varphi) \in \{\text{T}, \text{F}\} \tag{110}$$

A defined value for some more propositions follows. Equations e whose guard got the value $\bot$ or F are dead by getting the value $\sigma^\circ(e) = \bot$. Consequently, numerical variables that are only involved in equations that were found dead, are set dead too. Update the block function β by discarding equations that were found dead and mark the blocks that are enabled. □

2) Multi-Mode DAE systems:

*Algorithm 7 (structural analysis of **mDAE**):* Loop over the following, until termination:

Apply Algorithm 1 while mapping on-the-fly the **mDAE** system to the non-standard domain using (80); then:

- Case of *successful termination*: terminate Algorithm 7 successfully;
- Case of *failed termination*: apply the following algorithms with the following priority

$$\begin{array}{ll} & \text{Algorithm 8 for unlocking fixpoints} \\ < & \text{Algorithm 9 for revealing latent equations} \end{array} \tag{111}$$

until termination. □

The two algorithms are given next.

Algorithm 8 (unlocking fixpoints): This algorithm applies if Algorithm 1 gets blocked with some equation e being non evaluated while having all of its variables X already evaluated:

$$\begin{array}{ll} \text{pre-condition :} & \left.\begin{array}{l} X = \text{T} \\ e = \star \end{array}\right\} \text{in } e : \texttt{when } P(X) \texttt{ do } (X) \\ & \text{and Algorithm 1 is blocked;} \\ \text{post-condition :} & \text{substitute } e \leftarrow e^\bullet \end{array}$$

where $e^\bullet$ is defined in Table II. □

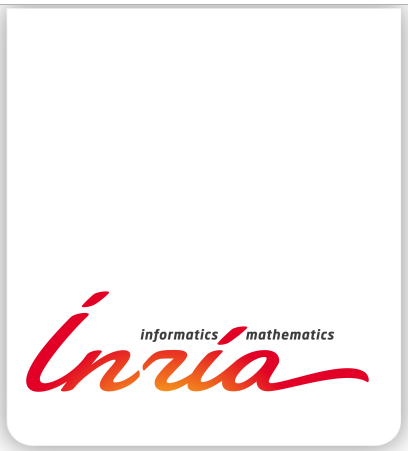
Algorithm 9 (revealing latent equations): This algorithm applies if Algorithm 1 gets blocked with some variables not yet evaluated but no block being enabled at the current status σ:

$$\begin{array}{l} \exists E \subseteq \mathbb{E}_S \text{ such that } \beta(\sigma, E) \neq \emptyset \\ \text{and Algorithm 1 is blocked.} \end{array} \tag{112}$$

If (112) holds, search for an E satisfying (112) and an equation e of the considered system such that

$$e \notin E \quad \text{but} \quad \beta(\sigma, E \cup \{e^\bullet\}) = \beta(\sigma, E)\,. \tag{113}$$

If such an equation e exists, add $e^\bullet$ to the considered system at status σ. □



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