D.4.1.1 - Semantics of Multi-Mode DAE Systems

WP 4.1 - Theory, semantics and standardization of multi-mode DAE systems

WP 4 - Systems with multiple operating modes

MODRIO (11004)

Version M42 (3.0)
Date 25/04/2016

Authors	Albert Benveniste	INRIA, Rennes, France
	Timothy Bourke	INRIA, Paris, France
	Benoît Caillaud	INRIA, Rennes, France
	Marc Pouzet	Ecole Normale Supérieure (ENS), Paris, France

Summary

Multi-mode DAE systems are the underlying mathematical model supporting physical modeling languages such as Modelica. Multi-mode DAE systems are systems of equations of the form "**when** γ_i **do** $f_i($ the x_j and derivatives of them $) = 0$", where γ_i is a predicate involving system variables and guarding the DAE $f_i(\dots) = 0$. 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. *Structural Analysis* of DAE systems, of which the popular Pantelides algorithm is representative, is used as a pre-processing of simulation for DAE systems of large index, arising in complex Modelica models.

However, unlike for single-mode (also called *pure*) DAE systems, no theory exists that can support the structural analysis of multi-mode DAE systems. As a 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.

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. The structural analysis includes the evaluation of the structural index, which in turn calls for discovering the "latent constraints", i.e., the hidden constraints that are not explicitly stated but result from the dynamic nature of the model. For pure DAE systems, latent constraints are revealed by differentiating selected equations. This by itself forbids a naive extension to multi-mode DAE systems, since discontinuities typically occur at mode changes, so differentiation cannot apply nearby.

In this paper we develop a comprehensive mathematical approach to the structural analysis of multi-mode DAE systems. We first observe that a pure 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. Defining the structural analysis of a multi-mode DAE system as being the structural analysis of its non-standard interpretation yields a conservative extension of the structural analysis of both pure DAE systems and (discrete time) *difference Algebraic Equation* (dAE) systems.

As the central contribution of this paper, a comprehensive structural analysis for multi-mode dAE systems is developed, i.e., systems of equations of the form "$\mathbf{when}$ γ_i **do** $f_i($ the x_j and forward shifts of them $) = 0$". We build on top of ideas borrowed from the *constructive semantics* of synchronous languages, sort of a structural analysis of synchronous programs. Two key concepts emerge, namely: the *atomic actions*, which are deferred to some external solver, and their *scheduling*. Our constructive semantics for multi-mode dAE systems 1) allows us to precisely characterize the systems that can/cannot be handled, 2) performs the reduction of the structural index, and 3) specifies the correct execution schemes. Efficient algorithms implementing this are yet to be found.

We conclude by drawing some remarks about the integration of numerical solvers in our approach. Details on this is deferred to future work.

Foreword

The present deliverable relates the current status of our work on a new structural analysis method for multi-mode DAE systems, capable of handling both continuous evolutions and (more importantly) mode changes with a greater generality than what can be found in current implementations of the Modelica language. We expect to continue this work, with the objective of developing a new compilation technique for the Modelica language, offering an improved handling of mode changes, with a rigorous mathematical semantics.

The Modrio project has been a key enabler for this work. In particular, we would like to thank Hilding Elmqvist, Sven Erik Mattsson (Dassault Systèmes AB), and Martin Otter (DLR) for the very fruitful discussions we had with them during several Modrio meetings. Their deep knowledge of the Modelica language and their guidance have been of a great help for the emergence of the theoretical results presented in the deliverable.

Chapter 1

Introduction

1.1 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:

$$f_i(\text{the } x_j \text{ and time derivatives of them}) = 0, 1 \leq i \leq n \tag{1.1}$$

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 (1.1) as a system of algebraic constraints relating the x_j and derivatives of them seen as independent variables, suppose that the following holds:

we can solve (1.1) for the highest derivatives of the x_js in terms of lower ones, i.e.: (1.2)

$$\left\{\begin{array}{rcl} x_1^{(d_1)} & = & g_1\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \\ x_2^{(d_2)} & = & g_2\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \\ & \vdots & \\ x_n^{(d_n)} & = & g_n\big(x_j^{(\ell_j)}, 0 \leq \ell_j < d_j\big) \end{array}\right. \tag{1.3}$$

Then, putting (1.3) in state-space form makes it an ODE.

Condition (1.2) may not hold in general, due to so-called “latent constraints”. The latter are exhibited by differentiating one or more times selected constraints from (1.1). That is, denoting by $f^{(c)} =_{\text{def}} \frac{d^c}{dt^c} f$ the c-th derivative of a function f of the x_j and derivatives of them, and setting

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

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 (1.1). It is usually denoted by ν_d.

To avoid computing a large number of Jacobians, several graph-based algorithms were proposed [19, 20, 23] to search for the c_i. All these algorithms rely on the following principle: 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 extended DAE system

$$F(X, Z) = 0 \tag{1.4}$$

satisfies condition (1.2) *almost everywhere*[1] if and only if the following assignment problem for $\mathcal{G}_F$ has a solution:

$$\text{Find a bijective assignment } \psi : X \to F \text{ such that } (x, \psi(x)) \in \mathcal{G}_F \text{ for every } x \in X. \tag{1.5}$$

Having ψ, we turn the incidence graph $\mathcal{G}_F$ into a directed graph 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 interpretation is that x is evaluated as an output of equation $\psi(x)$, whereas other variables involved in this equation are evaluated using other equations. This relies on the following principle:

Principle 1 *Any variable $x \in X$ can be evaluated by using any equation f involving it (i.e., such that $(x, f) \in \mathcal{G}_F$).*

The rationale for Principle 1 is that the considered equations are assumed to be smooth in their involved variables so that the Implicit Function Theorem generically applies and yields a unique solution, locally.

The resulting directed graph $\vec{\mathcal{G}}_F$ can have circuits and we call *blocks* the minimal circuits of $\vec{\mathcal{G}}_F$. If it exists, the solution of (1.5) 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., [19, 20].

Using (1.5), the method, for both finding a consistent initial condition and performing an integration step for (1.1), is by:

1. Searching for the minimal c_i so that (1.5) has a solution; $\nu_s = \max_i c_i$ is then called the *structural index* of DAE system (1.1); various algorithms for this task have been proposed, see, e.g., [19, 20, 23].

2. Forming the extended system (1.4) 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.* Assignment problem (1.5) is often reformulated as the following Block Lower Triangular decomposition problem:

[1] The term "almost everywhere" means 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.

Problem 1 (BLT Decomposition) *Given a DAE system consisting of a finite set of equations $F = \{f_1, \ldots, f_n\}$ over the variables $X = \{x_1, \ldots, x_n\}$, with incidence matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$, find a permutation of the equations $\sigma : \{1 \ldots n\} \to \{1 \ldots n\}$ and of the variables $\tau : \{1 \ldots n\} \to \{1 \ldots n\}$, such that the matrix $A' = (a'_{i,j})$, with $a'_{i,j} = a_{\sigma(i),\tau(j)}$ is a block lower triangular matrix with full rank blocks of minimal size.* □

The block structure that is found by performing the BLT decomposition yields a corresponding block structure for the Jacobian $\mathbf{J}$.

Structural Analysis: This body of tasks is referred to in the literature of physical system modeling as *structural analysis.* As the formulation of Problem 1 already suggests, the development and the specification of the algorithms solving the assignment problem (1.5) involve tedious manipulations of complex multi-indices that must be handled with care. Reference [20] is a typical example — this comment should not be taken as a criticism of [20], which proposes a most interesting approach to the structural analysis of DAE systems.

1.2 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 (1.5) 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 difficulty 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 constraints. 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 [13, 18], 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.

1.3 Our contribution

In this paper we propose a first step toward 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. Thus, we do not fully overcome difficulty (a). In contrast, our approach fully addresses difficulty (b). With reference to (1.1), we consider systems of equations the form

$$\textbf{when } \gamma_i \textbf{ do } f_i(\text{the } x_j \text{ and derivatives of them}) = 0 \tag{1.6}$$

That is, DAE f_i is guarded by a predicate γ_i, which is a boolean expression involving the system variables and their derivatives[2]. 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, this does not apply to guarded equations of the form (1.6) as a whole.

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

The following key observation can be formulated:

Replacing, everywhere in DAE system (1.1), any derivative $x'(t)$ by its Euler approximation

$$\frac{x(t+\varepsilon)-x(t)}{\varepsilon} \tag{1.7}$$

for some fixed and positive step size ε, and solving (1.1) for the maximally forward shifted variables $x(t+d_j\varepsilon)$ leaves the assignment problem (1.5) unchanged.

[2]This construction should be understood as an incomplete Modelica *0 = if ... then ... else ...* statement, where the *else* part is missing. It should not be confused with the *when ... do ...* construction of the Modelica language, where the condtion after the *when* defines an event, and the statements appearing after the *do* are procedural.

By this we mean that problem (1.5) 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 (1.7) 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 [5] for a formalization of this. Now, substitution (1.7) 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 [21, 12, 17, 8, 4], (1.7) becomes:

Replace, everywhere in DAE system (1.1), any derivative $x'(t)$ by its non-standard interpretation

$$\frac{x(t+\partial) - x(t)}{\partial} \tag{1.8}$$

and solve (1.1) for the maximally forward shifted variables $x(t + d_j\partial)$.

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

The original DAE system (1.1) and its non-standard interpretation using (1.8) possess identical structural analyses. (1.9)

The interest of doing this is that we can now circumvent difficulty (d) by using the following principle:

Principle 2 *Replacing the form* (1.6) *for a multi-mode DAE system by*

$$\textbf{when } \gamma_i \textbf{ do } f_i(\text{the } x_j \text{ and shifts of them}) = 0 \tag{1.10}$$

where the dynamics has non-standard discrete time index

$$\mathbb{T} =_{\text{def}} \{n \times \partial \mid n \in {}^\star\mathbb{Z}\} \tag{1.11}$$

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 (1.11), $\partial > 0$ 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 $\tau \in \mathbb{R}$ has an element $t(\tau) \in \mathbb{T}$ such that $|\tau - t(\tau)|$ is infinitesimal.

Chapter 2

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.

2.1 Single-mode systems

2.1.1 The simplest DAE system

Our first example is the following DAE system:

$$\left\{\begin{array}{lcl} x' & = & f(x,u) \\ 0 & = & g(x) \end{array}\right. \tag{2.1}$$

Differentiating the static constraint yields the following system, in which a new *latent* constraint (e_3) is revealed:

$$\left\{\begin{array}{lclcll} x' & = & f(x,u) & & & (e_1) \\ 0 & = & g(x) & & & (e_2) \\ 0 & = & g'_x(x)x' & = & g'_x(x)f(x,u) & (e_3) \end{array}\right. \tag{2.2}$$

Since system (2.1) is time invariant, equation (e_3) is implied by (e_1, e_2) and involves no additional variable. Thus, systems (2.1) and (2.2) are equivalent. The interest of the latter is best explained by considering the discrete time counterpart of (2.1):

$$\left\{\begin{array}{lcl} x^\bullet & = & f(x,u) \\ 0 & = & g(x) \end{array}\right. \tag{2.3}$$

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

$$x^\bullet_n = x_{n+1} \text{ holds for every } n. \tag{2.4}$$

Let us ask ourselves the following question:

Can we deterministically simulate (2.3) instant per instant, by relying only on the service offered by a static constraint solver? (2.5)

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

Shifting in (2.3) the static constraint forward yields the following system, in which the new latent constraint (e_3) is revealed, which involves no additional variable:

$$\left\{\begin{array}{lllll} x^\bullet & = & f(x,u) & & (e_1) \\ 0 & = & g(x) & & (e_2) \\ 0 & = & g(x^\bullet) & = \; g(f(x,u)) & (e_3) \end{array}\right. \tag{2.6}$$

Now, the execution scheme shown in Figure 2.1 can be applied when simulating one instant of the discrete time dynamical system (2.6). Note that a static constraint solver is

```
Given $x$ such that $g(x) = 0$:
  1. Use $(e_3)$ to evaluate $u$;
  2. Use $(e_1)$ to evaluate $x^\bullet$, which satisfies
     $g(x^\bullet) = 0$;
Move to the next instant.
```

Figure 2.1: Execution scheme for (2.6).

needed to perform step 1). Rather than performing the atomic computations 1) and 2) in sequence, we could rather use the constraint solver in solving (e_1, e_3) jointly. In solving (e_1, e_3) for the variables $u, x^\bullet$, the dynamic link (2.4) relating x and $x^\bullet$ is ignored. Please, note that we implicitly invoked Principle 1.

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

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

Well, equation (e_4) is useless since it involves the extra variable $x^{\bullet 2}$ and tells nothing more about the pair $(x^\bullet, u)$. 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 constraint, for use in evaluating the leading variables $(x^{\bullet 2}, u)$.

Now, what similar can we say regarding the continuous time DAE system (2.2)? For a given x satisfying (e_2), we regard (e_1, e_3) as a system of equations in the variables u, x'. Doing so amounts to considering (2.2) as an ODE system in which u is a static state feedback (a function of x).

Let us relate the above reasoning to the notion of *differentiation index* of DAE systems [22] (or *difference index* of dAE systems), which we briefly recall now on our example. We begin with the original DAE system (2.1), which we rewrite $F(v, x) = 0$ where

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

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

$$\mathcal{J} = \frac{\partial F}{\partial v} = \begin{bmatrix} 1 & -f'_u(x,u) \\ 0 & 0 \end{bmatrix}$$

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* $w =_{\text{def}} \frac{d}{dt}v = (x'', u')$:

$$\begin{array}{rcl} F_2(w,v,x) & =_{\text{def}} & \begin{bmatrix} F(v,x) \\ \frac{d}{dt}F(v,x) \end{bmatrix} \\ & = & \begin{bmatrix} x' - f(x,u) \\ g(x) \\ x'' - f'_x(x,u)x' - f'_u(x,u)u' \\ g'(x)x' \end{bmatrix} \end{array}$$

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

$$\text{solving for } v \text{ the equation: } \exists w . F_2(w,v,x) = 0. \tag{2.7}$$

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

$$\mathcal{J}_{\text{reduced}} = \frac{\partial}{\partial v} \begin{bmatrix} x' - f(x,u) \\ g(x) \\ g'(x)x' \end{bmatrix} = \begin{bmatrix} 1 & -f'_u(x,u) \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

which is full column rank (f.c.r.) provided that $f'_u(x,u) \neq 0$, a generic situation. Having $\mathcal{J}_{\text{reduced}}$ f.c.r. ensures that (2.7) 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 $x^{(3)}, u''$.

Computing the various Jacobians associated to the successive arrays is costly. In contrast, searching for sufficient conditions ensuring that (2.7) 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 (1.5), which can be efficiently solved, e.g., by performing the *structural analysis* using the Pantelides algorithm [19] and its improved versions, e.g. [20].

The other remark is that the DAE system (2.1) and its dAE counterpart (2.3) are equivalent regarding the assignment problem (1.5), in that the latent equations needed to reduce the index of each system match together.

2.1.2 The pendulum and its dAE version

This popular example will allow us to introduce our use of non-standard analysis.

$$\left\{ \begin{array}{rl} x'' - Tx = 0 & (e_1) \\ y'' - Ty + g = 0 & (e_2) \\ x^2 + y^2 - L^2 = 0 & (e_3) \end{array} \right. \tag{2.8}$$

System (2.8) is a DAE system having x, y, x', y' as state variables and T as algebraic variable, whereas g (the gravity) and L (the length of the pendulum) are constants.

Moving to non-standard form Consider the non-standard interpretation of the first and second order derivatives:

$$x' = \frac{x^{\bullet} - x}{\partial} \quad \text{and} \quad x'' = \frac{x^{\bullet 2} - 2x^{\bullet} + x}{\partial^2} \tag{2.9}$$

In (2.9), 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 (2.9) to express derivatives. By using (2.9), (2.8) becomes a dAE system (in discrete time). This system has the form:

$$\left\{ \begin{array}{rcl} f_1(x^{\bullet 2}, x^{\bullet}, x, T) = 0 & & (e_1) \\ f_2(y^{\bullet 2}, y^{\bullet}, y, T) = 0 & & (e_2) \\ x^2 + y^2 - L^2 = 0 & & (e_3) \end{array} \right. \tag{2.10}$$

To let the assignment problem (1.5) have a solution for this system, we must add the shifted versions $e_3^{\bullet}$ and $e_3^{\bullet 2}$ of e_3:

$$\left\{ \begin{array}{rcl} f_1(x^{\bullet 2}, x^{\bullet}, x, T) = 0 & & (e_1) \\ f_2(y^{\bullet 2}, y^{\bullet}, y, T) = 0 & & (e_2) \\ x^2 + y^2 - L^2 = 0 & & (e_3) \\ (x^2 + y^2)^{\bullet} - L^2 = 0 & & (e_3^{\bullet}) \\ (x^2 + y^2)^{\bullet 2} - L^2 = 0 & & (e_3^{\bullet 2}) \end{array} \right. \tag{2.11}$$

The execution scheme of (2.11) is given in Figure 2.2. By construction, step 1) will hold at

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 $T, x^{\bullet 2}, y^{\bullet 2}$;
3. Move to the next instant.

Figure 2.2: Execution scheme for (2.11).

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 *standardize* (2.11). We first replace (e_1, e_2) by their counterparts in (2.8) — 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. Equation $(e_3^{\bullet})$ expands as:

$$\begin{array}{rcl} 0 & = & (x^2 + y^2 - L^2) \\ & & +2\partial \times (xx' + yy') \end{array} \tag{2.12}$$

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

$$\begin{array}{rcl} 0 & = & (x^2+y^2-L^2) \\ & & +2\partial\ \times(xx'+yy') \\ & & +2\partial^2\times(xx''+yy''+x'^2+y'^2) \end{array} \tag{2.13}$$

Now, given that (e_3) holds, (2.12) reduces to:

$$0 \;=\; xx'+yy' \tag{2.14}$$

and, since both (e_3) and (2.14) hold, (2.13) reduces to:

$$0 \;=\; xx''+yy''+x'^2+y'^2 \tag{2.15}$$

Finally, by standardizing (2.11) we recover the familiar latent equations of the continuous time pendulum:

$$\left\{\begin{array}{rl} x''-Tx=0 & (e_1) \\ y''-Ty+g=0 & (e_2) \\ x^2+y^2-L^2=0 & (e_3) \\ xx'+yy'=0 & (e_3') \\ xx''+yy''+x'^2+y'^2=0 & (e_3'') \end{array}\right. \tag{2.16}$$

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 (2.13). 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 (2.13),* $(e_3^{\bullet 2})$ *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.

2.2 Multi-mode systems

2.2.1 Cup-and-ball game

Figure 2.3 shows a picture of this game, which is sort of a two-mode pendulum.
Its equations are the following:

$$\left\{\begin{array}{llll} & & x''=Tx & (e_1) \\ & & y''=Ty-g & (e_2) \\ & & \gamma\ =[\,x^2+y^2\geq L^2\,] & (\gamma) \\ \texttt{when} & \gamma & \texttt{do}\ \ x^2+y^2=L^2 & (e_3) \\ \texttt{when} & \texttt{not}\ \gamma & \texttt{do}\ \ T=0 & (e_4) \end{array}\right. \tag{2.17}$$

Figure 2.3: The cup-and-ball game.

Some comments are in order. Equations (e_1, e_2) and the body of (e_3) are the equations of the pendulum. When the rope 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, respectively.

Moving to the non-standard form Using model (2.17) with the non-standard expansion (2.9) 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 2.1 and 2.2. For a structural analysis, the concrete 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 (2.10). Repeating this for the system (2.9,2.17) yields:

$$\left\{\begin{array}{llllll} & & & e_1(x^{\bullet 2}, x^{\bullet}, x, T) & & (e_1) \\ & & & e_2(y^{\bullet 2}, y^{\bullet}, y, T) & & (e_2) \\ & & & \gamma(x, y) & & (\gamma) \\ \texttt{when} & & \gamma & \texttt{do} & [e_3](x, y) & (e_3) \\ \texttt{when} & \texttt{not} & \gamma & \texttt{do} & [e_4](T) & (e_4) \end{array}\right. \tag{2.18}$$

Some comments follow:

- $e_1(x^{\bullet 2}, x^{\bullet}, x, T)$ is the convenient abstraction for $f_1(x^{\bullet 2}, x^{\bullet}, x, T) = 0$ used in model (2.10);
- 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 3 (numerical equations) *Principle 1 applies to all blocks of n numerical equations involving n dependent variables, and only to them.*

Referring to (2.18), 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, T as variables. We thus need two more principles to deal with the latter cases:

Principle 4 (predicate) *A predicate can only be evaluated when all its involved variables have been evaluated. In this case, the predicate non-deterministically outputs one of the two values* F, T*.*

The reason for the second statement is that, since numerical values are abstract, none of the two boolean values can be rejected as a candidate output.

Principle 5 (guarded equations) *A guarded equation can only be evaluated when its guard has been evaluated.*

Unlike Principle 3 for numerical equations, Principles 4 and 5 impose certain causality constraints on how to handle guards. Let us try to derive an execution scheme for the abstract system (2.18) by complying with Principles 3, 4, and 5. This is shown in Figure 2.4. As explained in the figure caption, the execution scheme fails to evaluate all variables and solve all equations.

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

Figure 2.4: Attempt to derive an execution scheme for (2.18). To comply with Principles 4 and 5, the execution scheme orders atomic actions for each case $\gamma =$ F/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 3).

Considering the reason for the failure of step 2(a)ii, the only possible countermeasure consists in shifting forward $[e_3]$ twice. Thus, (2.18) becomes

$$\left\{\begin{array}{llllll} & & & e_1(x^{\bullet 2}, x^{\bullet}, x, T) & (e_1) \\ & & & e_2(y^{\bullet 2}, y^{\bullet}, y, T) & (e_2) \\ & & & \gamma(x, y) & (\gamma) \\ \texttt{when} & \gamma & \texttt{do} & [e_3](x^{\bullet 2}, y^{\bullet 2}) & (e_3^{\bullet 2}) \\ \texttt{when} & \texttt{not}\,\gamma & \texttt{do} & [e_4](T) & (e_4) \end{array}\right. \tag{2.19}$$

The resulting execution scheme is shown on Figure 2.5. The execution scheme is now

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 3, solve for $T, x^{\bullet 2}, y^{\bullet 2}$ 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 T;
      iii. solve for $x^{\bullet 2}, y^{\bullet 2}$ the block $\{e_1, e_2\}$;
3. Move to the next instant.

Figure 2.5: A successful execution scheme for (2.19).

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 2.5 is expressed in terms of non-standard discrete time. This is not effective. Deriving from this non-standard execution scheme an effective one operating in the continuous time of $\mathbb{R}$ is performed by *standardizing* the execution scheme of Figure 2.5 as follows:

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

- Translating equations (e_1) and (e_2) of (2.19) backward to the same equations in the concrete system (2.17), and
- By reusing the argument (2.12–2.15) based on Taylor expansions and taking Comment 1 into account, we standardize $(e_3^{\bullet 2})$:

$$\left\{\begin{array}{rll} & x'' = Tx & (e_1) \\ & y'' = Ty - g & (e_2) \\ & \gamma \ = [\, x^2 + y^2 \geq L^2 \,] & (\gamma) \\ \texttt{when}\ \gamma\ \texttt{do} & x^2 + y^2 = L^2 & (e_3) \\ \texttt{and} & xx' + yy' = 0 & (e_3') \\ \texttt{and} & xx'' + yy'' + x'^2 + y'^2 = 0 & (e_3'') \\ \texttt{when not}\ \gamma\ \texttt{do} & T = 0 & (e_4) \end{array}\right. \tag{2.20}$$

At instants of mode change: Let τ be a non-standard instant such that $\gamma(x^\bullet, y^\bullet) \neq \gamma(x, y)$, 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 ordered *super-dense* instants

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

where $(t, 0) = t$ and $(t, 2) < t'$ for any $t' > t$. The evaluation of instants $(t, 1), (t, 2)$ is performed based on the discrete time dynamics specified by the concrete form of (2.19), that is:

$$\left\{\begin{array}{llllll} & & & x'' = Tx & (e_1) \\ & & & y'' = Ty - g & (e_2) \\ & & & \gamma \ = [\, x^2 + y^2 \geq L^2 \,] & (\gamma) \\ \texttt{when} & & \gamma & \texttt{do} & (x^2 + y^2)^{\bullet 2} = L^2 & (e_3^{\bullet 2}) \\ \texttt{when} & \texttt{not} & \gamma & \texttt{do} & T = 0 & (e_4) \end{array}\right. \tag{2.21}$$

where x'', y'' are given by the discrete time non-standard interpretation (2.9) for the derivatives.

At this point, however, we still have a problem. We know that the block $(e_1, e_2, e_3^{\bullet 2})$ is used to define the triple of leading variables $(x^{\bullet 2}, y^{\bullet 2}, T)$. However the first two equations involves expressions that are standard in terms of x'', y'', which, when moving to discrete time, 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

$$\left\{\begin{array}{rcll} x^{\bullet 2} - 2x^\bullet + x & = & \partial^2 Tx & (e_1) \\ y^{\bullet 2} - 2y^\bullet + y & = & \partial^2 (Ty - g) & (e_2) \\ (x^2 + y^2)^{\bullet 2} & = & L^2 & (e_3^{\bullet 2}) \end{array}\right. \tag{2.22}$$

Only T is directly affected by ∂. We 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 infonitesimal 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, fix the pair (x, y) two steps ahead, when the new mode starts. We also infer that T has an impulse at this instant, but we skip this.

How do we recover the velocities at the instant when the new mode start? Move to $(t, 1)$ and reformulate (2.22) as follows

$$\left\{\begin{array}{rcll} x'^{\bullet} - x' &=& \partial T x & (e_1) \\ y'^{\bullet} - y' &=& \partial(Ty - g) & (e_2) \\ (xx' + yy')^{\bullet} &=& 0 & (e_3^{\bullet 2}) \end{array}\right. \tag{2.23}$$

We again eliminate T from (2.23), zet ∂ to zero, which leaves us with the system

$$\left\{\begin{array}{rcll} y(x'^{\bullet} - x') &=& x(y'^{\bullet} - y') & (e_{12}) \\ (xx' + yy')^{\bullet} &=& 0 & (e_3^{\bullet 2}) \end{array}\right. \tag{2.24}$$

At this point we are done: at $(t, 1)$ $(x^{\bullet}, y^{\bullet})$ because 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 T. Note that velocities are discontinuous here. To summarize:

$$\begin{array}{l} \text{at } (t,0) : \left\{\begin{array}{rcl} y(x^{\bullet 2} - 2x^{\bullet} + x) &=& x(y^{\bullet 2} - 2y^{\bullet} + y) \\ (x^2 + y^2)^{\bullet 2} &=& L^2 \end{array}\right. \\ \text{at } (t,1) : \left\{\begin{array}{rcl} y(x'^{\bullet} - x') &=& x(y'^{\bullet} - y') \\ (xx' + yy')^{\bullet} &=& 0 \end{array}\right. \end{array} \tag{2.25}$$

The automaton describing the execution scheme is shown on Figure 2.6.

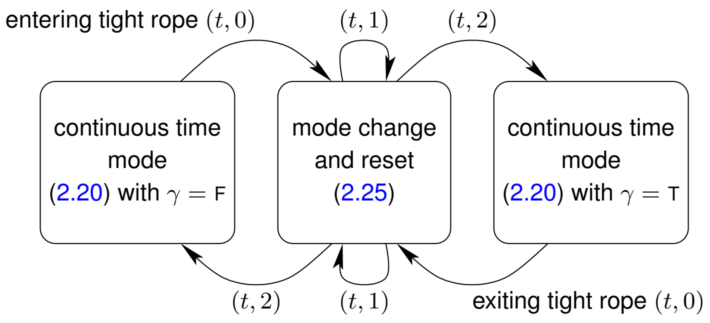
entering tight rope $(t,0)$
$(t,1)$
$(t,2)$
continuous time mode (2.20) with $\gamma =$ F
mode change and reset (2.25)
continuous time mode (2.20) with $\gamma =$ T
$(t,2)$
$(t,1)$
exiting tight rope $(t,0)$

Figure 2.6: Automaton controlling the execution of the cup-and-ball system.

Comment 2

- *It is important to notice that the (intuitive) code (2.20) is only valid inside continuous time modes. Mode changes must be handled by using discrete time code (2.21).*
- *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 $x^2 + y^2 \geq L^2$.*

2.2.2 Simple clutch

This is an example borrowed from [18]. It is not supported by the existing Modelica tools at the date of this writing. We consider a simplified clutch relating two rotating shafts. The clutch is either released and then ensures perfect join between the two shafts or is pressed, 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:

$$\left\{ \begin{array}{rlll} & & u_1' = f_1(u_1, c_1) & (e_1) \\ & & u_2' = f_2(u_2, c_2) & (e_2) \\ \texttt{when } \gamma & \texttt{do} & u_1 = u_2 & (e_3) \\ & \texttt{and} & c_1 + c_2 = 0 & (e_4) \\ \texttt{when not } \gamma & \texttt{do} & c_1 = 0 & (e_5) \\ & \texttt{and} & c_2 = 0 & (e_6) \end{array} \right. \tag{2.26}$$

In (2.26) the boolean γ is an input that indicates the status of the clutch: pressed/released is encoded by $\texttt{F}/\texttt{T}$; u_1 and u_2 are the speeds of the shafts and c_1 and c_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 release.

Let us follow the same approach as before. We first consider (2.26) in combination with the non-standard interpretation (2.9) for the derivatives. An attempt to derive an execution scheme for it is shown on Figure 2.7.

1. *Assuming consistent values for* u_1, u_2:
2. Case:
   (a) $\gamma = \texttt{T}$:
      i. regard (e_5, e_6) as dead code;
      ii. *failure to solve for* $c_1, c_2, u_1^\bullet, u_2^\bullet$ *the block of equations* $\{e_1, e_2, e_4\}$;
   (b) $\gamma = \texttt{F}$:
      i. regard (e_3, e_4) as dead code;
      ii. solve $[e_5]$ for c_1;
      iii. solve $[e_6]$ for c_2;
      iv. solve e_1 for $u_1^\bullet$;
      v. solve e_2 for $u_2^\bullet$;
3. Move to the next instant.

Figure 2.7: Attempt to derive an execution scheme for (2.29).

The reason for the failure of step 2(a)ii is clear:

$$\text{step 2(a)ii gets stuck by having a block of three equations but four dependent variables.} \tag{2.27}$$

The reason for the failure of initialization step 1 is less obvious. At an instant where $\gamma = \mathrm{T}$ but ${}^\bullet\gamma = \mathrm{F}$ (recall that ${}^\bullet\gamma$ is the backward shifting of γ),

$$\text{step 1 gets stuck by having a block of three active equations } ({}^\bullet e_1, {}^\bullet e_2, e_3) \text{ with only two dependent variables } u_1, u_2. \tag{2.28}$$

To overcome (2.27) we once more perform difference index reduction by invoking Comment 1, that is, we replace (e_3) by $(e_3^\bullet)$:

$$\left\{\begin{array}{rll}
 & & u_1' = f_1(u_1, c_1) \quad (e_1) \\
 & & u_2' = f_2(u_2, c_2) \quad (e_2) \\
\texttt{when } \gamma & \texttt{do} & u_1^\bullet = u_2^\bullet \quad (e_3^\bullet) \\
 & \texttt{and} & c_1 + c_2 = 0 \quad (e_4) \\
\texttt{when not } \gamma & \texttt{do} & c_1 = 0 \quad (e_5) \\
 & \texttt{and} & c_2 = 0 \quad (e_6)
\end{array}\right. \tag{2.29}$$

Observe that, while doing so, we solved the problem (2.28) as well. The resulting successful execution scheme is shown on Figure 2.8. Observe that the removed equation (e_3) is indeed identical to $(e_3^\bullet)$ taken at the previous instant.

To derive the final code, it remains to standardize (2.29). We follow the same approach as before.

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 $t' > t$. The evaluation of instant $(t, 1)$ is performed based on the discrete time dynamics specified by (2.9,2.29), i.e., by applying as such one step of the execution scheme of Figure 2.8.

When γ just moved from F to T (onset of a release of the clutch), constraint $u_1 = u_2$ is still not active, since (2.29) does not contain (e_3), and the values for u_1 and u_2 were set at the previous instant by the two equations (e_1, e_2). At this first time, however, $u_1^\bullet = u_2^\bullet$ is enforced as a consequence of step 2(a)ii. The move $(u_1, u_2) \rightarrow (u_1^\bullet, u_2^\bullet)$ is non-infinitesimal but occurs in zero physical time (from instant $(t, 0)$ to instant $(t, 1)$): it corresponds to an *impulse*, which matches the physics. The opposite mode change $\mathrm{T} \rightarrow \mathrm{F}$ does not yield any impulse. The final code is obtained by using the same technique as for the previous example. □

Outside instants of mode change: By using Taylor expansions as in previous exam-

1. Assuming consistent values for u_1, u_2:
2. Case:
   (a) $\gamma =$ T:
      i. regard (e_5, e_6) as dead code;
      ii. using Principle 3, solve for $c_1, c_2, u_1^\bullet, u_2^\bullet$ the block of equations $\{e_1, e_2, [e_3^\bullet], e_4\}$;
   (b) $\gamma =$ F:
      i. regard $(e_3, e_3^\bullet, e_4)$ as dead code;
      ii. solve $[e_5]$ for c_1;
      iii. solve $[e_6]$ for c_2;
      iv. solve e_1 for $u_1^\bullet$;
      v. solve e_2 for $u_2^\bullet$;
3. Move to the next instant.

Figure 2.8: A successful execution scheme for (2.29).

ples, the standardized dynamics is the following, interpreted in the usual (standard) sense:

$$\left\{\begin{array}{lllr}
 & & u_1' = f_1(u_1, c_1) & (e_1) \\
 & & u_2' = f_2(u_2, c_2) & (e_2) \\
\texttt{when } \gamma & \texttt{do} & u_1 = u_2 & (e_3) \\
 & \texttt{and} & u_1' = u_2' & (e_3') \\
 & \texttt{and} & c_1 + c_2 = 0 & (e_4) \\
\texttt{when not } \gamma & \texttt{do} & c_1 = 0 & (e_5) \\
 & \texttt{and} & c_2 = 0 & (e_6)
\end{array}\right. \tag{2.30}$$

Formally, (e_3) should not be there. Indeed (e_3') is sufficient to guarantee (e_3) if initialized from a state satisfying $u_1 = u_2$. We, however, know that constraint (e_3) should be reintroduced to help solvers sustaining it. □

A variation of Figure 2.6 can be drawn for this case too.

2.2.3 Unilateral constraint

As a third example, we consider the following variation of our example (2.1):

$$\left\{\begin{array}{lcl}
x' & = & f(x, u) \\
0 & \leq & g(x)
\end{array}\right. \tag{2.31}$$

which we rewrite using guarded equations:

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

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:

$$\left\{\begin{array}{rrll} & & x' = f(x,u) & (e_1) \\ & & \gamma = [0 \geq g(x)] & (\gamma) \\ \texttt{when}\ \gamma & \texttt{do} & g(x^\bullet)=0 & (e_2^\bullet) \end{array}\right. \tag{2.32}$$

An attempt of an execution scheme for (2.32) is shown on Figure 2.9.

1. Assuming a consistent value for x:
2. Case:
   (a) $\gamma = \mathsf{T}$:
      i. using Principle 3, solve $[e_2^\bullet]$ for $x^\bullet$;
      ii. using Principle 3, solve e_1 for u;
   (b) $\gamma = \mathsf{F}$:
      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.

Figure 2.9: Attempt of an execution scheme for (2.32).

Shifting equations does not solve the problem arising in this execution scheme. Here, in the mode $\gamma = \mathsf{F}$ (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 (2.31), 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:

$$\left\{\begin{array}{rrll} & & x' = f(x,u) & (e_1) \\ & & \gamma = [0 \geq g(x)] & (\gamma) \\ \texttt{when}\ \gamma & \texttt{do} & g(x^\bullet)=0 & (e_2^\bullet) \\ \texttt{when not}\ \gamma & \texttt{do} & h(x,u)=0 & (e_3) \end{array}\right. \tag{2.33}$$

This counter-measure is indeed already used for some cases in Dymola. The so addded “stub” is used for compilation only and does not give raise to effective code at run time. It is only a way to express that a value for u is expected.

2.3 Using Complementarity Conditions

So far Principle 3 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 3, and why it is of interest to do so.

Consider again example (2.33) and rewrite it first as follows:

$$\left\{\begin{array}{lcll} 0 & = & x' - f(x,u) & (e_1) \\ 0 & \leq & g(x^\bullet) & (e_{21}) \\ 0 & \leq & |h(x,u)| & (e_{22}) \\ 0 & = & g(x^\bullet) \times |h(x,u)| & (e_{23}) \end{array}\right. \tag{2.34}$$

For reasons that will be clear later we slightly modify (2.34) by shifting x forward in $h(x,u)$:

$$\left\{\begin{array}{lcll} 0 & = & x' - f(x,u) & (e_1) \\ 0 & \leq & g(x^\bullet) & (e_{21}) \\ 0 & \leq & |h(x^\bullet,u)| & (e_{22}) \\ 0 & = & g(x^\bullet) \times |h(x^\bullet,u)| & (e_{23}) \end{array}\right. \tag{2.35}$$

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

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

Using this notation, (2.35) rewrites

$$\left\{\begin{array}{ll} 0 = x' - f(x,u) & (e_1) \\ 0 \leq g(x^\bullet) \perp |h(x^\bullet,u)| \geq 0 & (e_2) \end{array}\right. \tag{2.36}$$

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 (2.36) as a block of equations amenable of Principle 3, which yields the much simpler execution scheme:

1. solve (2.36) for $(u, x^\bullet)$;
2. move to the next instant

(2.37)

We still need to standardize (2.36), which is performed by applying Taylor expansions:

$$\begin{array}{rclcl} g(x^\bullet) & = & g(x) & + & \partial \times g'(x)x' \\ h(x^\bullet,u) & = & h(x,u) & + & \partial \times h'_x(x,u)x' \end{array} \tag{2.38}$$

Since ∂ is infinitesimal, $g(x^\bullet) > 0$ holds if and only if

- either $g(x) > 0$,
- or $g(x) = 0$ and $g'(x)x' > 0$

and $g(x^\bullet) = 0$ holds if and only if both $g(x) = 0$, and $g'(x)x' = 0$. Similarly, $|h(x^\bullet,u)| > 0$ holds if and only if

- either $|h(x,u)| > 0$,
- or $|h(x,u)| = 0$ and $|h'_x(x,u)x'| > 0$

and $|h(x^\bullet,u)| = 0$ holds if and only if both $|h(x,u)| = 0$, and $|h'_x(x,u)x'| = 0$. The standardization of (2.36) is then given by:

$$\left\{\begin{array}{ll} 0 = x' - f(x,u) & (e_1) \\ 0 \leq \left[\begin{array}{c} g(x) \\ g'(x)x' \end{array}\right] \perp \left[\begin{array}{c} |h(x,u)| \\ |h'_x(x,u)x'| \end{array}\right] \geq 0 & (e_2) \end{array}\right. \tag{2.39}$$

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

Finally, we reconsider the cup-and-ball example (2.17), which we rewrite as follows using complementarity conditions in non-standard time:

$$\left\{\begin{array}{ll} 0 = x'' - Tx & (e_1) \\ 0 = y'' - Ty + g & (e_2) \\ 0 \leq (x^2 + y^2 - L^2)^{\bullet 2} \;\perp\; T \geq 0 & (e_3) \end{array}\right. \tag{2.40}$$

This is a complementarity condition with three equations and three dependent variables $(T, x^{\bullet 2}, y^{\bullet 2})$, hence Principle 3 applies, which again yields a trivial execution scheme. The standardization of (2.40) uses Taylor expansions (2.13) for (e_3). Performing as for the previous example yields the non-smooth system

$$\left\{\begin{array}{ll} 0 = x'' - Tx & (e_1) \\ 0 = y'' - Ty + g & (e_2) \\ 0 \leq \left[\begin{array}{c} x^2 + y^2 - L^2 \\ xx' + yy' \\ xx'' + yy'' + x'^2 + y'^2 \end{array}\right] \perp\; T \geq 0 & (e_3) \end{array}\right. \tag{2.41}$$

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.

2.4 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 section we formalize all of this.

Chapter 3

Multi-mode dAE (mdAE) systems

3.1 Introduction

In this section we focus on Step 1 of the two-step procedure of Section 2.4. For defining multi-mode dAE systems we wish to consider 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
$$x_n^{\bullet} = x_{n+1} \text{ for every } n \tag{3.1}$$
- *Predicates* over numerical variables, giving rise to *propositional formulas*, formed on top of predicates;
  - example: "$x{>}2$" is a predicate returning T or F depending on the value of x, and "b or not b'" is a propositional formula in the predicates b and b';
- *Invariants*, which are propositional formulas that must hold true for any execution of the system;
  - example: "$\texttt{inv} : \texttt{not}(b \texttt{ and } b')$" 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: "$\texttt{when } z > 0 \texttt{ do } f(x^{\bullet}, x, y) = 0$": when z is active (see below) and > 0, then the guard $z > 0$ 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 $x = \bot$, we take the convention that any guard involving x (e.g.: $x > 0$) also takes the special value $\bot$. Say that x is *active* if $x \neq \bot$ holds.

3.2 Syntax

We thus consider the **mdAE** mini-language whose syntax is displayed on Table 3.1.

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	::=	`when` *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 defined as *pred* and variables occurring in a *pred* are all defined as *num*. Finally, variables x_i occurring in an *eqn* of the form "`when` *prop* `do` $f(x_1, \ldots, x_n)=0$" are all defined as *num*.

Table 3.1: Syntax of the **mdAE** language. Keywords are in red.

A variable x : *var* is, recursively, either a basic variable $x \in \Xi$ or a forward shifted version $x^\bullet$ of a variable x. X : *Vars* is, recursively, a tuple of basic variables. S : *system* is a nested system of (scalar) guarded equations "e : `when` γ `do` $f(X)=0$", or invariants i : *inv*, with associated declaration of variables. A propositional formula (PF) γ is a predicate $\varphi(X)$: *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 $\bot$ as follows. Set $\bot \prec$ `F` $\prec$ `T` and define `and` and `or` to be the infimum and supremum for that order; with reference to γ, `not` γ exchanges the values `F`, `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 $\bot$ (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.

Numerical variables, propositional formulas, invariants, equations, and systems, can be shifted forward, see Table 3.2. With reference to Table 3.1, variables from Ξ can be *num, pred, inv, eqn.* Observe that, in guarded equations, the guard is *not* shifted, reflect-

$$\begin{array}{rcl}
(\xi : t)^{\bullet} & \equiv & (\xi^{\bullet}, t), \text{ where} \\
(x, X)^{\bullet} & \equiv & x^{\bullet}, X^{\bullet} \\
\varphi(X)^{\bullet} & \equiv & \varphi(X^{\bullet}) \\
(P_1 \texttt{ and/or } P_2)^{\bullet} & \equiv & P_1^{\bullet} \texttt{ and/or } P_2^{\bullet} \\
(\texttt{not } P)^{\bullet} & \equiv & \texttt{not } P^{\bullet} \\
(\texttt{when } \gamma \texttt{ do } f(X){=}0)^{\bullet} & \equiv & \texttt{when } \gamma \texttt{ do } f(X^{\bullet}){=}0 \\
(S_1; S_2)^{\bullet} & \equiv & S_1^{\bullet}; S_2^{\bullet}
\end{array}$$

Table 3.2: Shifting forward. With reference to Table 3.1, 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.

ing the fact that, in searching for latent equations, only bodies are shifted, not guards. On the other hand, invariants, being propositional formulas, are 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 3.3 a *constructive semantics* for S, which consists of a constructive algorithm for computing the runs.

3.3 Constructive Semantics

The notion of constructive semantics was introduced by the community of synchronous languages [7, 2, 6], 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.

3.3.1 Principles

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* [7, 2, 6], 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 [16], 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 [15], 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.

3.3.2 Eligible atomic actions

They are 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 $\mathtt{T}, \mathtt{F}$, 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 $\mathtt{T}$ and $\mathtt{F}$. 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 3 introduced in Section 2.2.

Blocks of linear equations Blocks consisting of n linear equations in n dependent variables are structurally regular, and thus define eligible atomic actions.

The Implicit Function Theorem Blocks consisting of n smooth equations in n dependent variables are amenable of the Implicit Function Theorem. Hence, existence and uniqueness of solutions is guaranteed, locally and structurally. 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:

$$\left\{ \begin{array}{c} h(v,i) = 0 \\ 0 \leq v \perp i \geq 0 \end{array} \right. \tag{3.2}$$

where:

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

and $h(v,i)$ is a smooth function. An instance of complementarity condition is a closed electrical circuit involving a perfect diod, with v and i being the potential and the current. Complementarity conditions were informally discussed in Section 2.3. Complementarity conditions generically possess a unique solution, locally, and solvers exist for such *non-smooth* systems [1]. Therefore, blocks of the form (3.2) are candidate atomic actions.

The following warning should be formulated at this point, regarding (3.2). 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 3.3.4.

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

$$\left\{ \begin{array}{ccc} 0 & = & h(x,u) \\ 0 & \leq & g(x) \end{array} \right. \tag{3.3}$$

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 [13], we can complement (3.3) 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:

$$\left\{\begin{array}{l} 0 = h(x,u) \\ 0 \leq g(x) \perp c^2(x,u) \geq 0 \end{array}\right. \tag{3.4}$$

What about logico-numerical fixpoint equations? 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:

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

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.

3.3.3 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 Ξ as *Scott variables.*[1] With reference to Table 3.1, Scott variables encompass both the system variables and guards, the predicates, the equations, and the invariants. Tuples of Scott variables identify with subsets $\Xi \subseteq \Xi$. Equip the set Ξ of Scott variables with the following flat partially ordered domain $(D, \leq)$ of values:

$$D = \{\star, \bot, \mathsf{T}, \mathsf{F}\} \quad \text{with} \quad \star \ < \ \bot, \mathsf{T}, \mathsf{F} \tag{3.5}$$

D is an extension, with the value $\star$, of the domain $\{\bot, \mathsf{T}, \mathsf{F}\}$ already mentioned for system variables. In (3.5):

- $\star$ is a special status *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 $\xi \in \Xi$ either remains to be evaluated ($\xi = \star$) or has already been evaluated (ξ takes its values in $\{\bot, \mathsf{T}, \mathsf{F}\}$). This holds for any ξ, representing a system variable or an equation, etc.

[1] 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.

$\perp$ is the status *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. We insist that $\perp$ (dead) is different from $\star$ (not evaluated yet, also called "undefined").

$\mathtt{T}, \mathtt{F}$ 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 $\mathtt{T}$ and the actual domain for $\xi \in$ *num* is $\{\star, \perp, \mathtt{T}\}$. The same holds for a Scott variable $\xi \in$ *pred* $\uplus$ *eqn* and $\xi = \mathtt{T}$ means that the referred entity has been evaluated.

A Scott variable ξ can have other Scott variables as arguments. This is written

$$\xi(\Xi) \tag{3.6}$$

where $\Xi \subseteq \boldsymbol{\Xi}$ is the tuple of arguments. Here are some examples, with reference to Table 3.1:

- $\varphi(X)$ is a predicate $\varphi \in \Phi$ in the numerical variables X, and, for p : *prop*, $p(X)$ is a propositional formula whose constitutive predicates involve the variables of X;
- $f(X)$ is a scalar function of the numerical variables X;
- $e(\gamma, X)$ is an equation guarded by γ whose body involves the numerical variables collected in tuple X.

3.3.4 Semantics

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

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. Say that status σ is *coherent*, written

$$coherent(\sigma), \tag{3.7}$$

if the following condition holds, which relates a Scott variable $\xi(\Xi)$ to its arguments, see (3.6):

$$\exists \xi' \in \Xi \text{ such that } \sigma(\xi') = \star \quad \Rightarrow \quad \sigma(\xi) = \star \tag{3.8}$$

$$\left.\begin{array}{r} \sigma(\xi) \neq \star \\ \exists \xi' \in \Xi \text{ such that } \sigma(\xi') = \perp \end{array}\right\} \quad \Rightarrow \quad \sigma(\xi) = \perp \tag{3.9}$$

(3.8) expresses that ξ cannot be defined unless all its arguments are defined and (3.9) 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 $\Xi \subseteq \Xi$ of Scott variables by the symbol "!":

$$\sigma : !\Xi \qquad (\text{“}\Xi \text{ is } \textit{enabled} \text{ in } \sigma\text{”}) \tag{3.10}$$

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

Satisfaction: Let γ : *prop* be a propositional formula in some tuple of predicates. Write

$$\sigma \models \gamma \qquad (\text{“}\sigma \textit{ satisfies } \gamma\text{”}) \tag{3.11}$$

to indicate that, either $\sigma(\gamma) = \star$, or $\sigma(\gamma) = \mathtt{T}$. For an equation $e \equiv$ "$\mathtt{when}\ \gamma\ \mathtt{do}\ f(X) = 0$", write

$$\begin{array}{ll} \sigma \models e & (\text{“}\sigma \textit{ satisfies } e\text{”}) \\ \text{if} & \left\{ \begin{array}{rl} \text{either} & \sigma(e) = \star \\ \text{or} & \sigma(\gamma) = \mathtt{F} \\ \text{or} & \sigma(\gamma) = \mathtt{T} \text{ and } X = \mathtt{T} \end{array} \right. \end{array} \tag{3.12}$$

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

$$\sigma \models S \qquad (\text{“}\sigma \textit{ satisfies } S\text{”}) \tag{3.13}$$

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 not dead in σ (i.e., $\sigma(e) \neq \bot$).

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 fo index reduction. Thus the system S we consider has the form:

$$S = \widehat{S}\ ;\ \widetilde{S} \tag{3.14}$$

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

Leading, useful, and dummy variables: *Leading variables* in status σ are those having the form $x^{\bullet k}$ for $k = d^o_\sigma(x)$, where $d^o_\sigma(x)$ is the degree of x in σ, for the original system $\widehat{S}$. Let:

$$\lfloor \Xi \rfloor_\sigma \subset \Xi \tag{3.15}$$

be the subset of *useful* Scott variables of S in status σ, consisting of all numerical variables of the form $x^{\bullet k}$ with $k \leq d^o_\sigma(x)$. Numerical variables of Ξ not belonging to $\lfloor \Xi \rfloor_\sigma$ are called *dummy* at σ; they arise from $\widetilde{S}$ in (3.14).

Definition 1 (constructive semantics) *The* constructive semantics *of a system S is the set of all maximal sequences*[2] *of statuses, called* runs,

$$\sigma_0, \sigma_1, \ldots, \sigma_k, \sigma_{k+1}, \ldots, \sigma_K \tag{3.16}$$

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

1. *Status σ_k is coherent, see (3.7), and satisfies S.*
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).*

The considered run is successful *if no useful variable has the value $\star$ in the final status σ_K.* □

Dummy variables (not belonging to $\lfloor \Xi \rfloor_{\sigma_K}$) must be eliminated via existential quantification. For our constructive semantics, evaluating them is thus optional and done only if necessary to evaluate all variables belonging to $\lfloor \Xi \rfloor_{\sigma_K}$. If the constructive semantics succeeds, the system can proceed to executing its next reaction.

[2]"Maximal" refers to prefix order.

Algorithms

We are now ready to describe the symbolic algorithms computing the constructive semantics of S.

With reference to Table 3.1, 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*

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

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.

Formally, with reference to (3.6) and the discussion thereafter:

$$\begin{array}{c} \forall \sigma \in \Sigma, \forall E \subseteq \mathbb{E}_S : \forall e \in E \Rightarrow \left\{ \begin{array}{l} \sigma(\gamma(e)) = \mathsf{T} \\ \sigma(e) = \star \end{array} \right. \\ \beta(\sigma, E) =_{\text{def}} \{x \in X(E) \mid \sigma(x) = \star\} \end{array} \tag{3.17}$$

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

$$\sigma : !E \tag{3.18}$$

if solving E for the variables belonging to $\beta(\sigma, E)$ belongs to the class of atomic actions—see Section 3.3.2. 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:*

$$\begin{array}{ll} & \textit{Algorithm 6 for performing a } \text{tick} \\ < & \textit{Algorithm 3 for enforcing invariants} \\ < & \textit{Algorithm 4 for the evaluation of blocks} \\ < & \textit{Algorithm 5 for the evaluation of predicates} \end{array} \tag{3.19}$$

Theorem 1 *Algorithm 1 terminates in finitely many micro-steps, and is confluent with respect to nondeterministic choices performed in its sub-Algorithm 4.* □

Proof 1 *See the argument following Definition 2.*

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:*

$$coherent(\sigma_0) \quad \text{and} \quad \sigma_0 \models S \tag{3.20}$$

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

$$\begin{aligned} &\sigma_0 : !\Phi_0, \ \text{where} \\ &\Phi_0 =_{\text{def}} \{p(X) : \text{pred} \mid \forall x \in X \Rightarrow \sigma_0(x) \neq \star\} \end{aligned} \tag{3.21}$$

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 (3.20) is a system of equations that needs to be solved—it is called the *consistency* condition. For a subsequent instant, a status satisfying (3.20) is provided by the previous instant.

With reference to Definition 1 and formula (3.16), 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 (invariants) *If $\sigma(\iota) =$ 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 (3.16) for the definion of K. Otherwise, σ is validated and the constructive semantics can further proceed.* □

Algorithm 4 (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.* □

Algorithm 5 (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 (3.7) and satisfies:*

$$\forall \xi \in X : \sigma(\xi) = \text{T} \quad \Longrightarrow \quad \sigma^\circ(\varphi) \in \{\text{T}, \text{F}\} \tag{3.22}$$

A defined value for some more propositions follows. Equations e whose guard got the value $\bot$ or F *are discarded 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 (3.22) 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.

Algorithm 6 (tick) *This algorithm applies if the current status σ satisfies the following conditions:*

1. *σ meets the invariants: $\sigma(\iota) \neq \mathsf{F}$ for every invariant ι;*
2. *$\sigma(\xi) \neq \star$ for every Scott variable ξ of S that either belongs to $\widehat{S}$ (see (3.14)) 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 $\sigma_K \rightarrow \sigma_0^\bullet$, such that, for every x:*

$$\sigma_0^\bullet(x) = \left\{ \begin{array}{rl} \textbf{if } \sigma_K(x^\bullet) \in \{\mathsf{F}, \mathsf{T}\} \textbf{ then} & \sigma_K(x^\bullet) \\ \textbf{else} & \star \end{array} \right. \tag{3.23}$$

$\sigma_0^\bullet$ is initial status for the next instant. It satisfies (3.20) by construction. □

Note that, in (3.23), "dead" is mapped to "undefined". The following obvious result holds:

Lemma 1 *A run is successful in the sense of Definition 1 if and only if it terminates through Algorithm 6.* □

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.

The Pantelides algorithm

The title of this section is an over-claim, since we do not propose here an optimized algorithm to search for the structural index—this is left for future work. Our aim is only to show how the Constructive Semantics itself provides a way to find the structural index. Let us first recall what the Pantelides algorithm is supposed to do. Referring to (3.14), we consider a system $\widehat{S}$ and augment it by adding shifted versions of it following the rules of Figure 3.2, to form what Campbell and Gear call the *array system*:

$$S = \widehat{S}\ ;\ \widehat{S}^\bullet\ ;\ \ldots\ ;\ \widehat{S}^{\bullet k}\,.$$

Following (3.15), we distinguish in S the useful variables $\lfloor\Xi\rfloor \subset \Xi$, which is the subset of Scott variables of S collecting all numerical variables of the form $x^{\bullet k}$ with $k \leq d^o(x)$, where $d^o(x)$ is the maximal shifting degree of variable x in the original system $\widehat{S}$. Numerical dummy variables of Ξ, i.e., those not belonging to $\lfloor\Xi\rfloor$, result from the successive shiftings of $\widehat{S}$ in (3.14). We are not interested in the value taken by the dummy variables, so we must eliminate them (by existential quantification), in a structural sense.

It turns out that the Constructive Semantics of S itself does this for free, provided that the depth k of the array is chosen large enough such that:

$$k \geq d^o(\widehat{S}), \quad \text{where} \quad d^o(\widehat{S}) = \max_{x,\sigma} d^o_\sigma(x) \tag{3.24}$$

for x ranging over the set of all numerical variables of $\widehat{S}$ and σ ranging over the set of all possible statuses of S. The motivation for (3.24) is that, as long as $k < d^o(\widehat{S})$, the shifted system $\widehat{S}^{\bullet k}$ may bring useful variables, so we have to make sure that we include this shifted version of $\widehat{S}$.

Suppose that this Constructive Semantics succeeds in the sense of Definition 1. Then, all useful variables have been successfully evaluated. We then conclude that the structural index of $\widehat{S}$ is at most k. The procedure for finding the structural index follows, using the rules of Table 3.2 for shifting:

Algorithm 7 (Pantelides algorithm)

1. *Begin with $k = d^o(\widehat{S})$ in $S_k = \widehat{S}\ ;\ \widehat{S}^{\bullet}\ ; \ldots ;\ \widehat{S}^{\bullet k}$;*
2. *While the constructive semantics of S_k fails, update $k \to k+1$;*
3. *Stop when the constructive semantics of S_k succeeds.*

Observe that this procedure may not terminate, in which case the structural index is infinite. If the procedure terminates with $k > d^o(\widehat{S})$, then k is the structural index. If, however, the procedure terminates at instant 1, $d^o(\widehat{S})$ may overestimate the structural index. To recover a tighter upper bound for it, we check which equations have been used to evaluate all the useful variables of S. If, for some $k < d^o(\widehat{S})$, the array system $\widehat{S}; \ldots ; \widehat{S}^{\bullet k}$ is sufficient to evaluate all useful variables of $\widehat{S}$, then the structural index is bound by k.

This is clearly not an optimized procedure. To optimize it, we would need to find a way to decide which equations should be shifted forward when the constructive semantics fails. This is left open.

Chapter 4

Multi-mode DAE systems

This chapter sketches the derivation of an execution scheme for multi-mode DAE systems, based on the semantics of multi-mode dAE systems, presented in the previous chapter, and using several basic properties of nonstandard analysis. It is incomplete, and further work is required to specify how the continuous dynamics is expressed during modes, and how impulsive behaviour is handled during mode changes. Roughly, our approach for the structural analysis of multi-mode DAE systems is illustrated on Figure 4.1.

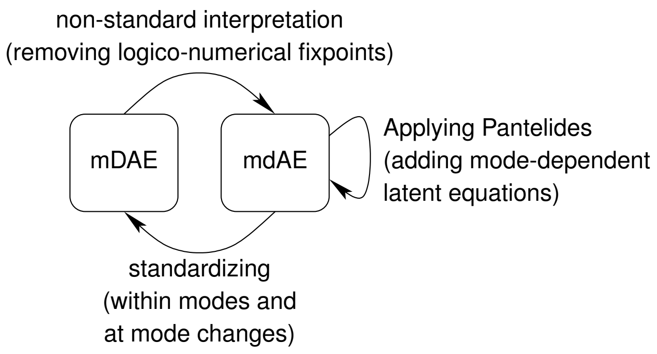

Figure 4.1: Our approach for the structural analysis of multi-mode DAE systems.

4.1 Syntax

We consider the **mDAE** language defined in Table 4.1, compare with the **mdAE** language of Table 3.1. The only difference between **mDAE** and **mdAE** lies in the first line of the syntax. Here, instead of the forward shift, we have two operators, namely

- the derivative x'; and
- the right-limit x^+, which we re-interpret as "next" at mode changes.

The corresponding non-standard interpretation clarifies the meaning of these operators:

$$x' = \frac{1}{\partial}(x^\bullet - x) \quad ; \quad x^+ = x^\bullet \tag{4.1}$$

Performing (4.1) maps a system of **mDAE** to a system of **mdAE**.

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.

$$
\begin{array}{rcl}
\textit{var} & ::= & \Xi \mid \textit{var}' \mid \textit{var}^{+} \\
\textit{Vars} & ::= & \textit{var} \mid \textit{var}, \textit{Vars} \\
\textit{prop} & ::= & \texttt{T} \mid \texttt{F} \mid \textit{var} \mid \texttt{not}\ \textit{prop} \mid \\
 & & \textit{prop}\ \texttt{and}\ \textit{prop} \mid \textit{prop}\ \texttt{or}\ \textit{prop} \\
\textit{num} & ::= & \texttt{num} \\
\textit{pred} & ::= & \texttt{pred}\ \text{“predicate on } \textit{Vars}\text{”} \\
\textit{inv} & ::= & \texttt{inv}\ \textit{prop} \\
\textit{eqn} & ::= & \texttt{when}\ \textit{prop}\ \texttt{do}\ \text{“equation on } \textit{Vars}\text{”} \\
\textit{term} & ::= & \textit{num} \mid \textit{pred} \mid \textit{inv} \mid \textit{eqn} \\
\textit{def} & ::= & \textit{var} : \textit{term} \\
\textit{system} & ::= & \textit{def} \mid \textit{def}\ ;\ \textit{system}
\end{array}
$$

Well formed programs satisfy the same conditions as in Table 3.1.

Table 4.1: Syntax of the **mDAE** language. Keywords are in red. Compare with Table 3.1.

4.2 Step-based versus event-based semantics

So far our presentation of the constructive semantics is step-based. Since numerical solvers can be only called for some positive duration, step-based constructive semantics is not enough.

Furthermore, we know that we may possibly need to provision a numerical integration scheme for all possible modes of the system, which requires listing all the modes. This does not scale up when the system in consideration is the parallel composition of many sub-systems.

Can we help fixing these two problems? In the following we suggest that the answer is positive. Our analysis is rather informal. Further work is needed to formalize this with the objective of generating efficient simulation schemes.

4.2.1 Reachable modes

We are particularly interested in finding, as part of our structural analysis, the reachable modes for which a numerical integration scheme must be provisioned. Recall that a *path* is a finite sequence of successive transitions and a *circuit* is a path starting and ending at identical statuses. With this interpretation, we conjecture the following:

Conjecture 1 *The reachable modes for which numerical integration schemes must be provisioned coincide with the circuits of the constructive semantics, originating from and ending at an initial status.* □

In fact, the characterization given by the above conjecture must probably be refined. Informally, we need to make sure that, while traversing the considered circuit, the changes in the variables involve no discontinuity. Symbolic criteria should be developed that are similar to the techniques used in [3] for the discrete/continuous typing of equations and variables.

4.2.2 Event-based semantics

As the previous discussion suggests, each circuit of the constructive semantics is associated with a particular *stationary regime* of the system, to which a specific numerical integration scheme is associated and the corresponding solver activated. So, we are interested in the *events* of the constructive semantics, at which the execution leaves a given circuit toward a different one. This can occur at a tick or in the course of the current circuit. Event-based (also called DEVS) simulation [11] seems therefore attractive. Even more so in combination with QSS solvers.

4.2.3 Using QSS solvers

In classical numerical integration methods, one integration step let the time progress by the same amount for all the system variables. This global time step is generally variable and adjusted to account for the system stiffness around the considered instant. A rich set of variations exist for numerical integration methods, to account for sparsity. These are essential to handle large scale DAE systems.

Large classes of Modelica models, however, are sparse with local interactions and local stiffness. Classical integration methods are not well adapted to local stiffness [14, 10]. *Quantized State System (QSS) Solvers* have been advocated as alternatives, in which system states are quantized and time progresses locally for each variable, until the current quantized value must be left [11, 14]. QSS solvers thus implement Discrete Event (DEVS) approximations of continuous time systems. Being event-based rather than time-based, QSS methods accomodate the execution scheme of the constructive semantics much better.

Chapter 5

Conclusion

We have proposed a fundamentally new approach to the compilation of multi-mode dAE / DAE systems. The constructive semantics, a concept borrowed from synchronous languages, was central in our approach. The constructive semantics is a systematic technique for decomposing discretization instants into a partial order of properly scheduled atomic actions.

In synchronous languages, the class of atomic actions was straightforward (evaluating expressions), the difficulty being the scheduling. For multi-mode dAE / DAE systems, however, a new essential issue arises, namely: clarifying the classes of atomic actions. As expected, solving regular systems of smooth algebraic equations is an atomic action. We have also investigated the consequences of including regular complementarity conditions within the class of atomic actions, thus including non-smooth dynamical systems within our scope.

So far existing compilation methods have problems when transition conditions must be involved in the process of index reduction. Our approach overcomes this.

Most existing advanced compilation methods have limitations. It is, however unclear, why a program should be accepted or refused. Our approach tells exactly when and why a compilation fails (failure of the Constructive Semantics, letting the index increasing to infinity). Of course, we do not claim that this mathematical answer always speaks to the user's intuition. Still, it is a mathematical answer.

Our approach is systematic and general. It is implemented by the SUNDAE tool by Benoît Caillaud, which is a mockup generating and running the constructive semantics for the **mdAE** language (corresponding code generator with proper calls to numerical solvers remains to be developed). Efficient compilation algorithms are yet to obtain. This is in contrast to the most recent compilation approaches for Modelica, where efficiency remains a driving factor.

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, "A hybrid synchronous language with hierarchical automata: static typing and translation to synchronous code," in *EMSOFT*, S. Chakraborty, A. Jerraya, S. K. Baruah, and S. Fischmeister, Eds. ACM, 2011, pp. 137–148.

[4] ——, "Nonstandard semantics of hybrid systems modelers," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 877–910, 2012.

[5] ——, "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

[6] 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.

[7] 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.

[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. Casella, "Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives," in *Proc. of the Int. Modelica Conference*, H. Elmqvist and P. Fritzson, Eds. Versailles, France: Modelica Association, Sep. 2015.

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

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

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

[14] 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

[15] 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.

[16] 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.

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

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

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

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

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

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

[23] 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.