Multi-Mode DAE Systems with Varying Index

Sven Erik Mattsson[1], Martin Otter[2], Hilding Elmqvist[1]

[1]Dassault Systèmes, Sweden, {SvenErik.Mattsson, Hilding.Elmqvist}@3ds.com
[2]Institute of System Dynamics and Control, DLR, Germany, Martin.Otter@dlr.de

Abstract

This paper discusses an approach to handle multi-mode Differential Algebraic Equation (DAE) systems described by continuous-time state machines where mode-dependent state constraints are present. The goal is to perform static symbolic analysis and to generate efficient run-time code. This technique extends the class of multi-mode systems that can be handled by Modelica tools.

Keywords: Multi-mode, DAE, varying index, continuous-time state machine, variable structure system, symbolic transformation.

1 Introduction

1.1 Multi-Mode Systems

In (*Elmqvist et al., 2014*) a proposal was presented to extend the synchronous Modelica 3.3 state machines (*Elmqvist et al., 2012*) to continuous-time state machines having continuous-time models as "states". Every model can be a "state" of a state machine and in particular acausal models. These new types of models are called "*multi-mode systems*". An example of such kind of system is shown in Figure 1.

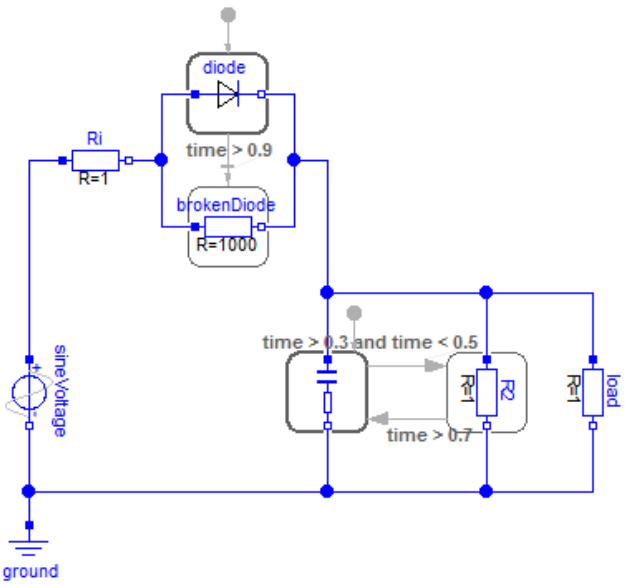


Figure 1. Circuit with two acausal state machines, from (*Elmqvist et al., 2014*).

Additionally, a method was developed to map connections to connectors of states in a particular way. The resulting equations can be processed basically by the *standard* symbolic algorithms supported by Modelica 3.2 tools. This approach already allowed handling a large class of useful variable structure systems with *dynamically changing number of continuous-time states*.

However, models could not be handled with this new method if connections between state and non-state components lead to constraints on continuous-time state variables that vary for the different state machine states. For example, the model in Figure 2 could not be handled. This circuit describes a capacitor C1 that is destroyed when the voltage becomes too large. The destroyed capacitor is modelled with a small resistor R1.

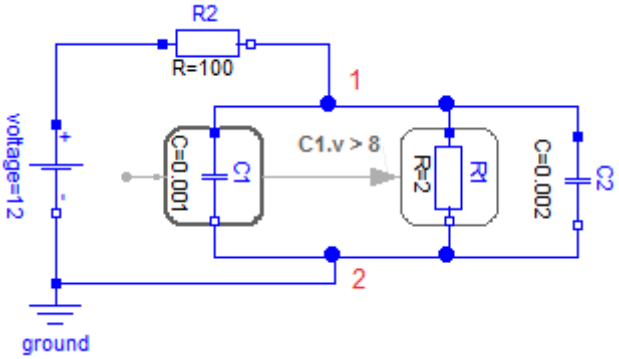


Figure 2. Circuit that could not be handled previously due to different state constraints in the different state machine states; slightly adapted from (*Elmqvist et al., 2014*).

The goal of this work is to extend the class of multi-mode DAE systems that can be simulated by Modelica tools and prototype the technique in Dymola (*Dassault Systèmes, 2015*).

In the EU research project RealSim, algorithms had been developed to symbolically process variable structure DAE systems and simulate the generated code (*Mattsson et al., 2001*). In particular, also a certain class of DAEs could be treated where Dirac impulses occur when switching the structure. These developments had not been incorporated into the release version of Dymola, because they had not been mature enough for a production software. Some ideas from the developments in the RealSim project are now being transferred to multi-mode systems.

1.2 Other Approaches

There are also other approaches to handle variable structure, varying index systems. For example (*Zimmer, 2010*) uses a run-time interpreter that processes the DAE equations at run-time, when the structure and/or the index is changing. The benefit is that a very large class of DAEs with varying index can be handled, at least in principal. The drawback is that the run-time efficiency is one or more orders of magnitude reduced with respect to an approach advocated by this article. Dynamically changing the structural analysis at run-time is also performed by (*Höger, 2014*). Describing variable structure systems with causal state machines is discussed by (*Pepper et al., 2011*).

(*Benveniste et al., 2014*) is tackling the problem from a fundamental point of view: The underlying, precise mathematical description is based on non-standard analysis for discrete-time and hybrid continuous-time/discrete-time multi-mode systems. This approach looks promising. However, it is not yet clear how to utilize this theory practically in a Modelica simulation environment.

2 Prerequisites

In this section several equations and properties are collected together that are prerequisites of the developed methodology, which is introduced in sections 3 and 4.

2.1 Connection equations

In (*Elmqvist et al., 2014*) it was shown how to map physical, acausal connections from components outside of a state machine to components on a state machine. An example is shown in Figure 2 where the components R2 and C2 are connected to the components R1 and C1 that form a *continuous-time state machine*.

Assume a connector c_i present on a state i is defined by one potential variable p_i and one flow variable f_i and that n of these connectors from the same state machine are connected to m connectors $c_{e,j}$ outside of (that is external to) this state machine. Therefore, the following connect statements will be present (for simplicity it is assumed that exactly one of the external connectors, $c_{e,1}$, is connected to all the state machine connectors; if this would not be the case, one could always automatically re-arrange the connect statements):

$$
\begin{aligned}
&\texttt{connect}(c_{e,1}, c_1)\\
&\quad\ldots\\
&\texttt{connect}(c_{e,1}, c_n)\\
&\texttt{connect}(c_{e,1}, c_{e,2})\\
&\quad\ldots\\
&\texttt{connect}(c_{e,1}, c_{e,m})
\end{aligned}
$$

These connect statements are replaced by the following equations, where i characterizes the active state of the state machine:

Connection equations

$$
\begin{aligned}
&\boldsymbol{p} = \{p_1, p_2, \cdots, p_n\}; \quad \boldsymbol{p_e} = \{p_{e,1}, p_{e,2}, \cdots, p_{e,m}\};\\
&\boldsymbol{f} = \{f_1, f_2, \cdots, f_n\}; \quad \boldsymbol{f_e} = \{f_{e,1}, f_{e,2}, \cdots, f_{e,m}\};\\
&i = \text{activeState}();\\
&p_{e,1} = p_i; \qquad \text{// potential equations}\\
&p_{e,1} = p_{e,2:m}\\
&0 = f_i + \textstyle\sum_{j=1}^{m} f_{e,j} \qquad \text{// flow equation}\\
&\text{for } r \text{ in } 1\text{:}\, n-1\\
&\quad k = \text{mod}(i + r - 1, n) + 1\\
&\quad 0 = \text{h}_r(p_k, f_k) \qquad \text{// dummy equations}\\
&\text{end for}
\end{aligned}
\tag{1}
$$

Note, the for-loop generates $n-1$ dummy equations, $0 = \text{h}_r(\ldots)$. These dummy equations are only present in order that the equations of the non-active states form a regular system. The exact form of these equations is irrelevant because they are only used during symbolic analysis and are not present in the generated code. For more explanations, see (*Elmqvist et al., 2014*).

For connections of *two external* connectors to *two states* of a state machine, the connector equations of (1) simplify to:

```
Connection equations for the connection of
two external connectors ce1, ce2 with
two state machine connectors c1, c2:

  // Equations for potential variables
  ce1.p = if activeState(state1) then c1.p else c2.p;
  ce1.p = ce2.p

  // Equation for flow variables
  0 = ce1.f + ce2.f +                                    (2)
      (if activeState(state1) then c1.f else c2.f);

  // Dummy equation for not connected state
  0 = if activeState(state1) then
          h1(c2.p, c2.f) else h2(c1.p, c1.f);
```

The function activeState(*name*) is the Modelica 3.3 built-in function to inquire whether the instance *name* is the current active state of a state machine or not. Equations (2) are used in all following examples to map connections to state machines in to equations.

Input/output connections to states of a state machine can also be handled. Due to the known causality, this results in a much simpler approach as with acausal, physical connectors, and is not discussed in this paper.

2.2 Sinks and sources

The dummy equations (see last equation of (2)) have the drawback that they introduce algebraic loops between the states of a state machine and therefore make the analysis more difficult and the generated code less efficient (due to larger algebraic systems of

DOI 10.3384/ecp1511889

equations). These equations can be removed if either *one external potential* variable or *all external flow* variables are constants or known functions of time, in other words if the state machine states are connected to sink or source components. In such cases, equations (2) can be rewritten to:

Connection equations if *one external potential* variable ce1.p is a constant or a known time function:

```
// Equations for potential variables
ce2.p := ce1.p
c1.p := if activeState(state1) then ce1.p else last(c1.p)
c2.p := if activeState(state2) then ce1.p else last(c2.p)

// Equation for flow variables
0 = ce1.f + ce2.f +
    (if activeState(state1) then c1.f else c2.f);
```
(3)

or to

Connection equations if *all external flow* variables ce1.f, ce2.f are constants or known time functions:

```
// Equations for potential variables
ce1.p = if activeState(state1) then c1.p else c2.p;
ce1.p = ce2.p

// Equation for flow variables
c1.f := if activeState(state1) then ce1.f + ce2.f
                               else last(c1.f)
c2.f := if activeState(state2) then ce1.f + ce2.f
                               else last(c2.f)
```
(4)

Here last(v) is a conceptual function (only used during symbolic analysis) to indicate the value of variable v from the last time instant where the corresponding state was active. For the symbolic analysis, last(v) is a known value. The ":=" operator in the equations indicates the computational causality (= the left hand side is computed from the right hand side).

The proof of equations (3) is straightforward (a proof of equations (4) can be performed in a similar way): Start from (2) and recognize that the dummy equations of the not connected states, h1(..) and h2(..), can be arbitrarily selected, as long as the equations of the not connected states together with these dummy equations are *structurally consistent*[1]. In (3) it is implicitly assumed that state_i together with the rest of the system is *structurally consistent* if the state is active. If state_i is not active, one can assume that keeping the causality of the connector (= identical to the case where the state is active) will still keep this non-active state together with its other dummy equations *structurally consistent*. In other words, under the assumption that ce1.p is a constant or a known function of time, the last equation of (2) can be replaced by:

```
// Dummy equation for not connected state
0 = if activeState(state1) then
      c2.p - last(c2.p) else c1.p - last(c1.p);
```
(5)

Rearranging the other equations of (2) together with (5) results in (3).

As a concrete example, lets analyze the circuit of Figure 2: At node 2 the state machine connectors C1.n, R1.n and the external connectors C2.n, voltage.n, and ground.p are connected together:

```
connect(ground.p, R1.n)
connect(ground.p, C1.n)
connect(ground.p, C2.n)
connect(ground.p, voltage.n)
```
(6)

Since the potential of the ground component is given, ground.p = 0, one external potential variable of the connection set is a constant and therefore equations (3) can be utilized resulting in the following equations that are equivalent to (6):

Connection equations at node 2 of Figure 2:

```
// Equations for potential variables
R1.n.v = 0
C1.n.v = 0
C2.n.v = 0
voltage.n.v = 0

// Equation for flow variables
0 = ground.p.i + voltage.n.i + C2.i +
    (if activeState(R1) then R1.n.i else C1.n.i);
```
(7)

2.3 Differentiating dummy equations

When equations must be differentiated using a generalized form of the Pantelides algorithm (*Pantelides, 1988*), see section 3 and 4, dummy equations of non-connected states might need to be differentiated as well. For example assume that the equation for flow variables in (2) needs to be differentiated. When this equation is differentiated, the time derivatives of ce1.f, ce2.f, c1.f, c2.f are introduced. This in turn means that also the dummy equation in (2) needs to be differentiated. Differentiating this dummy equation would utilize the newly introduced derivatives of the flow variables, but would also introduce *new derivatives* of the potential variables c1.p and c2.p. This in turn might trigger other (unnecessary) differentiations.

We are rather free to select the dummy equations. They are only used to keep the equation sets of non-connected states *structurally consistent*. To avoid unnecessary state constraints of the dummy equations, and in turn unnecessary differentiations of equations, the dummy equations are actually defined in such a way that they provide a relationship between the actually occurring highest derivatives of the potential and flow variables.

For example, when the flow equation in (2) needs to be differentiated and the time derivatives of ce1.f, ce2.f,

[1] A DAE is "structurally inconsistent", if a unique solution cannot exist, or stated differently, if the *Pantelides algorithm* does not converge. (*Pantelides, 1988*) provides an algorithm to test for this property.

c1.f, c2.f are introduced, then the dummy equation is changed to:

```
// Dummy equation for not connected state
0 = if activeState(state1) then                    (8)
      h1(c2.p, der(c2.f)) else h2(c1.p, der(c1.f));
```

Therefore, they provide a relationship between the differentiated flow variables and the non-differentiated potential variables.

2.4 Standard Pantelides and BLT algorithms

The *Pantelides algorithm* (*Pantelides, 1988*) is the key algorithm in this paper and will be generalized for multi-mode systems[2]. It is summarized here for Modelica 3.2 DAEs, that is hybrid DAEs but without (discrete or continuous-time) state machines:

The flattened equations of a Modelica 3.2 model are described by the following equations:

Flattened equations of a Modelica 3.2 model

$$\begin{aligned} &(a)\quad \mathbf{w} = \{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t, \mathbf{m}, \mathbf{m}^-\} \\ &(b)\quad \mathbf{0} = \mathbf{f}(\mathbf{w}, \mathrm{relation}(\mathbf{w})) \\ &(c)\quad \mathbf{m} = \mathbf{g}(\mathbf{w}, \mathrm{relation}(\mathbf{w})) \end{aligned} \tag{9}$$

where

t	The independent (real) variable
$\mathbf{x}(t)$	Variables of type Real, appearing differentiated
$\mathbf{y}(t)$	Variables of type Real, appearing not differentiated (= algebraic variables)
$\mathbf{m}(t_{ev})$	Variables of type discrete Real, Boolean, Integer. They change their values only at event instants t_{ev}. At an event instant, $\boldsymbol{m}^-$ is the value of $\mathbf{m}$ at the previous event iteration at this time instant. During continuous integration, that is between events, $\mathbf{m}$ is fixed (does not change) and $\boldsymbol{m}^- := \mathbf{m}$
$\mathrm{relation}(\mathbf{w})$	All the relations in the model, for example $x_2 > y_5$. During continuous integration all relations are fixed (do not change).

If (directly or indirectly) constraints between variables $\mathbf{x}$ are present, the Jacobian of (9b) with respect to the unknowns of type Real is singular:

$$\det\left(\begin{bmatrix} \frac{\partial f_i}{\partial \dot{x}_j} & \frac{\partial f_i}{\partial y_j} \end{bmatrix}\right) = 0 \tag{10}$$

This means that (9b) cannot be algebraically solved for the unknowns $\dot{\mathbf{x}}$ and $\mathbf{y}$. When using an explicit integration method to solve (9b) between events, these unknowns must be computed. Consequently, if (10) holds, explicit integration methods cannot be used and initialization is problematic[3].

The *Pantelides algorithm* solves this problem by differentiating singular subsets of equations. Since only equations are under consideration that are integrated, the starting point of the algorithm is equation (9b) where the discrete variables $\mathbf{m}, \mathbf{m}^-, \mathrm{relation}(\mathbf{w})$ are kept constant (because they do not change during continuous integration) and therefore their dependencies are ignored:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) \tag{11}$$

In particular this means that all when-clauses are removed and the dependency of the equations from conditions of if-clauses is ignored. The variable and equation structure of (11) is described in the following way:

- All variables appearing in (11) are collected in vector $\mathbf{v} = \{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}\}$, $\mathbf{v} \in \mathbb{R}^{n_v}$ and (11) are $n_{f0} = n_x + n_y$ equations: $\mathbf{0} = \mathbf{f}(\mathbf{v}, t)$
- The variable association list $\mathbf{V}$ is an Integer vector that defines if a variable v_j is the derivative of a variable i: $V_j = i$, if $\frac{dv_j}{dt} = v_i$. If no derivative of variable v_j is present, then $V_j = 0$ (i.e., these variables have the highest occurring derivatives).
- The equation association list $\mathbf{F}$ is an Integer vector that defines if equation f_j is the derivative of an equation i: $F_j = i$, if $\frac{df_j}{dt} = f_i$. If no derivative of equation f_j is present, then $F_j = 0$ (i.e., these equations have the highest occurring derivatives).

After termination of the *Pantelides algorithm* the following two sets of equations are present:

$$0 = f_i(\mathbf{v}, t), \quad F_i = 0, \quad \frac{\partial f_i}{\partial v_j} \text{ structurally regular for } V_j = 0, \tag{12}$$

and

$$0 = f_k(\mathbf{v}, t), \quad F_k > 0, \quad V_j > 0 \tag{13}$$

In other words, (12) are n_{f0} equations (the highest derivative equations) in n_{f0} unknowns (the highest derivative variables), and the highest derivative variables can be computed from the highest derivative equations (provided the Jacobian is not only structurally regular but also regular).

(13) are $n_f - n_{f0}$ equations describing the constraint equations between the potential states $\mathbf{x}$. They are also called invariants. The highest derivatives of the variables, v_j with $V_j = 0$, do not appear in these equations. These constraint equations can either be used to compute appropriate dummy derivatives with the Dummy Derivative method of (*Mattsson and*

[2] Most likely, the alternative formulation from (*Pryce, 2001*) can be used instead of the Pantelides algorithm as well.

[3] see (*Pantelides, 1988*)

DOI 10.3384/ecp1511889

Söderlind, 1993), or these equations can be used to project the solution of (12) to the invariants (13) when the drift becomes too large.

For the new algorithm, the highest derivative equations (12) must be sorted to determine the execution order to compute the highest derivative unknowns. This includes determining the algebraic loops of this equation system. This is a standard algorithm for Modelica models and will be abbreviated as BLT (Block Lower Triangular) as it is usually done.

3 Basic Idea

In this section the basic idea to symbolically process multi-mode systems with varying state constraints (and therefore varying DAE index) is sketched at hand of the circuit of Figure 2. In the follow-up section 4 this idea is generalized and described in more detail.

Intuitively, when activeState(C1) is true, there are two capacitors in parallel, C1 and C2, and this results in a state constraint between the potential states C1.v and C2.v. The constraint equation must be differentiated, which means that the potential variables of the node 1 connection set, such as C2.p.v, *must be differentiated* as well.

When activeState(R1) is true, there is a capacitor C2 and a resistor R1 in parallel, and no state constraint is present. Therefore, the potential variables of the node 1 connection set, such as C2.p.v, *need not to be differentiated*. Since variables of external connectors to a state variable must be differentiated in one mode, and need not to be differentiated in the other mode, it is unlikely that it is possible to build one common equation system for all modes.

Such types of systems can be handled by an obvious *brute force method*: For every possible mode the equations of the complete system are generated for this particular mode and during simulation the model is switched between these equation sets. In the case of the circuit in Figure 2 there would be two equation sets. For small systems with only a few possible modes, this approach might be feasible. However, for large systems with many state machines and several states per state machine, the number of equation sets would be growing exponentially and the generated code would quickly become unmanageable. So this brute force method is not practical for the general case.

For this reason another approach is used that is inspired by (*Mattsson et al., 2001*). It is based on the property that differentiated equations contain the solution set of the non-differentiated equations. In the circuit of Figure 2 the potential variables of node 1 need to be differentiated when in state C1. We can accept this fact and use these differentiated potential variables also when in state R1. As a consequence, the potential equation R1.p.v = C2.p.v is an invariant that must hold during simulation of its differentiated form. The benefit is that only one equation set can be constructed for all modes. Lets' analyze this approach in more detail for the circuit in Figure 2:

The connection equations at node 2 are given by (7). The connection equations at node 1 are:

Connection equations at node 1 of Figure 2:

```
// Equations for potential variables
C2.p.v = if activeState(R1) then R1.p.v else C1.p.v
R2.n.v = C2.p.v

// Equation for flow variables
0 = C2.p.i + R2.n.i +                                        (14)
    (if activeState(R1) then R1.p.i else C1.p.i)

// Dummy equation for not connected state
0 = if activeState(R1) then
      h1(C1.p.v, C1.p.i) else h2(R1.p.v,  R1.p.i);
```

Collecting all equations together and applying the *Pantelides algorithm* shows that no equation must be differentiated. The reason is that the if-clauses in (14) hide the state constraint between the two capacitors from the structural algorithm. BLT partitioning of the equations, taking into account the zeros in (7) and alias elimination, results in:

Sorted equations of Figure 2:

```
inputs = {C1.v, C2.v}  // continuous-time states

R2.v = voltage.V-C2.v
R2.v = R2.R*voltage.i

algebraicLoop
  unknowns = {R1.p.v, R1.p.i, C1.i}

  // Local equations of R1
  if activeState(R1) then
    R1.p.v = R1.R*R1.p.i
  end if;

  // Potential connections to the state machine
  C2.v = if activeState(R1) then R1.p.v else C1.v

  // Dummy equations for inactive states                      (15)
  0 = if activeState(R1) then
        h1(C1.v, C1.i) else h2(R1.p.v,  R1.p.i)
end algebraicLoop

// Flow connection to the state machine
C2.i+voltage.i =
      -(if activeState(R1) then R1.p.i else C1.i)
C2.i = C2.C*der(C2.v)

if activeState(C1) then
  C1.i = C1.C*der(C1.v)
end if;

// Flow connection to the state machine
ground.p.i-C2.i-voltage.i =
          (if activeState(R1) then R1.p.i else -C1.i)
```

For BLT blocks with one variable and one equation, the equation with the variable to solve for is type-set in bold. In this example there is first a sequence of two such equations. After them there is an algebraic loop with 3 unknowns. The equations to use include local equations from the two states and external connection equations to them.

Every algebraic loop that contains connection equations to a state of a state machine must be analyzed whether it is (structurally) non-singular in all modes. For this, it is tried to make an assignment for every particular mode that can occur in the algebraic loop at hand. The algebraic loop in (15) gives rise to two modes: activeState(R1) is true or false. In both cases the relevant equations need to be extracted and the unknowns not present in this mode need to be removed:

```
Algebraic loop of (15) for activeState(R1) == true:

inputs = {C2.v}  // continuous-time states

algebraicLoop
   unknowns = {R1.p.v, R1.p.i}

   R1.p.v = R1.R*R1.p.i

   // Potential connections to the state machine
   C2.v = R1.p.v
end algebraicLoop
```
(16)

The algebraic loop in this mode consists of two equations in two unknowns. An assignment is possible, because R1.p.v can be assigned in the second equation and R1.p.i in the first equation. Therefore this set of equations is structurally regular.

```
Algebraic loop of (15) for activeState(R1) == false:

inputs = {C1.v, C2.v}  // continuous-time states

algebraicLoop
   unknowns = {C1.i}

   // Potential connections to the state machine
   C2.v = C1.v
end algebraicLoop
```
(17)

The algebraic loop in this mode consist of one equation in one unknown. Since the unknown C1.i does not appear in this equation, the algebraic loop is structurally singular.

The approach of Pantelides is to differentiate equations if the smallest possible set of equations has more equations as unknowns. In the new method we differentiate additionally the potential or flow connector equations from external connectors to connectors on a state of a continuous-time state machine if

> these connector equations belong to an algebraic loop and in one mode this algebraic loops is (a) singular and (b) the connector equation is present in this singular case.

In the above example, only equation

```
C2.v = if activeState(R1) then R1.p.v else C1.v
```
(18)

fulfills these requirements (it is a potential connector equation present in an algebraic loop, in mode activeState(R1) == false this loop is singular, and the equation is part of this singular loop). We differentiate this equation and adapt the corresponding dummy equation:

```
der(C2.v) = if activeState(R1) then
                der(R1.p.v) else der(C1.v)
0 = if activeState(R1) then                          (19)
       h1(der(C1.v), C1.i) else
       h2(der(R1.p.v),  R1.p.i)
```

We take the original equations (15), remove (18) and its corresponding dummy equation and add (19). The (standard) *Pandelides algorithm* is performed on the resulting system. In this case the algorithm differentiates equations and variables. After termination, the highest derivative equations are structurally regular. Performing BLT partitioning on this structurally regular subset results in the following equations:

```
Sorted equations of highest derivative equations

inputs = {C1.v, C2.v, R1.p.v}
         // continuous-time states + dummy states

R2.v = voltage.V-C2.v
R2.v =  R2.R*voltage.i
R1.p.v = R1.R*R1.p.i

algebraicLoop
   unknowns = { der(C1.v), der(C2.v), der(R1.p.v),
                C1.i, C2.i}

   C1.i = C1.C*der(C1.v)
   C2.i = C2.C*der(C2.v)                             (20)
   der(C2.v) = if activeState(R1) then der(R1.p.v)
                                  else der(C1.v)
   C2.i – R2.i = -if activeState(R1) then R1.p.i
                                     else C1.i
   0 = if activeState(R1) then
          h1(der(C1.v), C1.i) else
          h2(der(R1.p.v),  R1.p.i)
end algebraicLoop

ground.p.i = C2.i-R2.i+ (if activeState(R1) then
                            R1.p.i else C1.i)
```

Analyzing the newly occurring algebraic loop reveals that this loop is structurally regular in all modes (if this would not be the case, again potential or flow equations in the loop would be differentiated). Therefore, the overall algorithm can be terminated. The final equations have the property that the highest derivative equations are *structurally regular in all*

DOI 10.3384/ecp1511889

modes. The further processing and code generation is performed nearly in the same way as for Modelica 3.2 models. Especially, the dummy derivative method is applied (*Mattsson and Söderlind, 1993*) and a BLT of all highest derivative equations together with all discrete equations (9c) is performed as function of all unknowns. During code generation one has to additionally take into account that equations need to be deactivated when their corresponding state is not active.

Simulation results with the Dymola prototype are shown in the next figure:

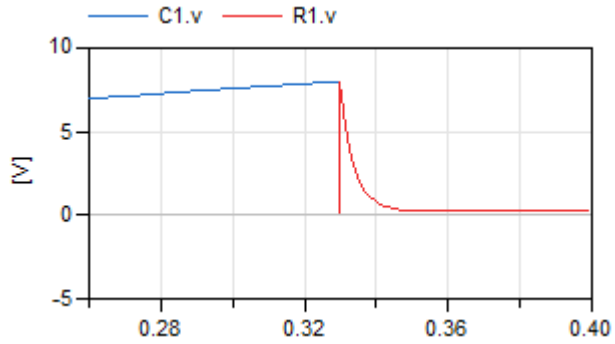


Figure 3. Simulation results of the circuit of Figure 2**.**

Since variable values of non-active states can have non meaningful values, Dymola only displays them in the time period, where the corresponding state is active. Therefore, C1.v is displayed only in the time range [0s, 0.33s] and R1.v is displayed only in the time range [0.33s, 0.6s].

4 Multi-Mode Pantelides Algorithm

The approach sketched in the previous section is more formally defined:

Starting point is a Modelica 3.2 model with one or more continuous-time state machines. As in section 2.4 all discrete equations and discrete variables are ignored during the following analysis. This also means that all transition conditions, such as C1.v > 8 in Figure 2, are ignored in this phase and the symbolic analysis is performed on equation (11). This equation is seen as a function of all (continuous-time) variables of type Real that appear in all states of all state machines and in all equations outside of all state machines.

Multi-Mode Pantelides Algorithm

1. Perform the standard *Pantelides algorithm* on (11) until convergence.
2. Perform BLT partitioning on the highest derivative equations (12) with respect to the highest derivative unknowns.
3. Analyze the algebraic loops detected in 2., that have at least one potential or flow connection equation (2) in the loop. For every such loop perform an assignment for every mode present in this loop (for the assignment ignore all variables and equations not active in the particular mode).
4. *Stop*, if all algebraic loops in 3. are *structurally regular* for all modes. Otherwise, if an algebraic loop is *structurally singular* for at least one mode, stop the analysis of this loop after this first singular mode was found and *goto 5*.
5. For every loop in 4. that was found to be singular, the potential or flow connection equations that are (a) present in the respective loop and (b) give a structural singularity in the analyzed mode, need (conceptually) to be differentiated. This is indirectly achieved by introducing the differentiated variables of the respective connection equations in the variable association list.
6. Continue with the standard *Pantelides algorithm* by analyzing the highest derivative equations (without taking modes into consideration). After convergence is reached, *goto 2.*

The standard *Pantelides algorithm* differentiates the smallest possible set of equations that has more equations as unknowns. The generalization above additionally differentiates connection equations that are the result of connections to states of state machines, if algebraic loops become structurally singular when the corresponding state is active. After termination of the multi-mode Pantelides algorithm, (12) and (13) hold again. The new property is that (12) is structurally regular with respect to the highest derivative variables *in all modes*!

After step 2. it may happen that a connection equation is present outside of all algebraic loops and that this equation shall be solved for a potential or flow variable defined on one of the states. This is, for example, the case in the example of section 5.2:

```
// Flow connection to the state machine
L2.i = if activeState(diode.open) then                    (21)
          diode.open.p.i else diode.closed.p.i;
```

This equation is structurally singular when state diode.open is active. Therefore, such an equation must also be differentiated in step 5.

The *multi-mode Pantelides algorithm* has a worst case complexity that grows exponentially with the number of possible modes which might be troublesome. However, in practice one can hope that this worst case complexity is usually not reached: Whenever state machines are present that influence each other not dynamically (say ideal diodes in an electrical circuit and friction components in the mechanical part of the model), then different algebraic loops will occur for the different state machines, the possible mode values in the loops will be different, and the analysis of the loops is decoupled.

One question is under which conditions the *multi-mode Pantelides algorithm* is converging (so stops after a finite number of iterations). For the standard

Pantelides algorithm this can be determined by replacing $\dot{\mathbf{x}}$ with $\mathbf{x}$ in (11):

$$\mathbf{0} = \mathbf{f}(\mathbf{x}, \mathbf{x}, \mathbf{y}, t) \qquad (22)$$

and performing an assignment for $\mathbf{x}$ and $\mathbf{y}$. If this is possible, the algorithm converges. If not, the DAE (11) is *structurally inconsistent* and the algorithm does not converge. It is not yet clear how to generalize this property for the *multi-mode Pantelides algorithm*.

5 Examples

In this section further examples are shown that shall demonstrate the *multi-mode Pantelides algorithm* in different situations.

5.1 Varying index with inductors

The circuit in the next figure consists of two inductors in series, L1 and L2, where an over-current destroys L1 (the destroyed case is modeled with a large resistor).

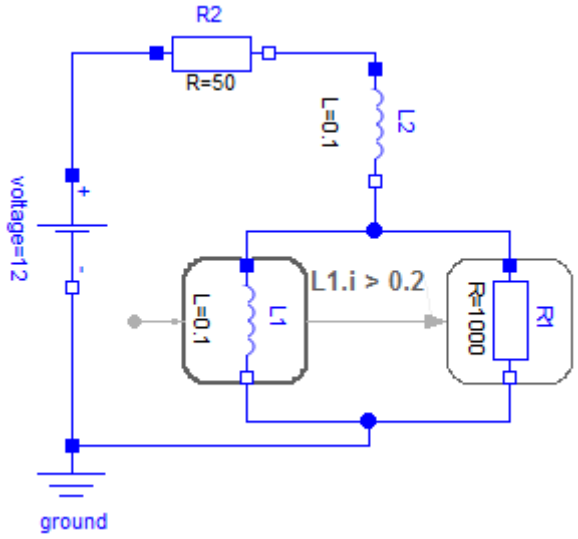


Figure 4. Inductors in series, where one of the inductors is destroyed when the current becomes too large.

When in state L1 the two inductors are in series and there is a constraint between the potential states L1.i and L2.i. When in state R1, this constraint is no longer present. The *multi-mode Pantelides algorithm* operates in a similar way as for the circuit in Figure 2. Simulation results are shown in the next figure:

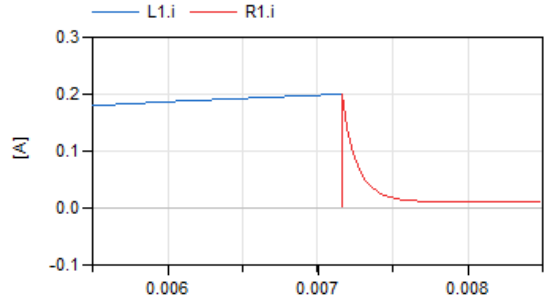
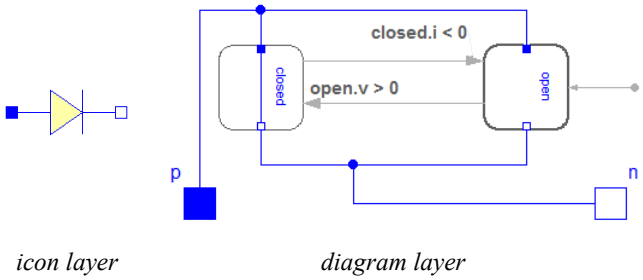


Figure 5. Simulation results of the circuit in Figure 4.

5.2 Varying index with inductor and diode

With continuous-time state machines it is possible to model ideal electrical switches, and in particular ideal diodes:



icon layer *diagram layer*

Figure 6. Ideal diode modelled with a continuous-time state machine.

The diode is modeled as a state machine where the first state is modeling a broken or open line and the second state is modeling an ideal line without resistance. For most situations there is no difference in using this diode model or the one from package Modelica (Modelica.Electrical.Analog.Ideal.IdealDiode) and setting Ron = Goff = 0. However, if varying state constraints occur this is different. Let us consider for example an inductor in series with a diode:

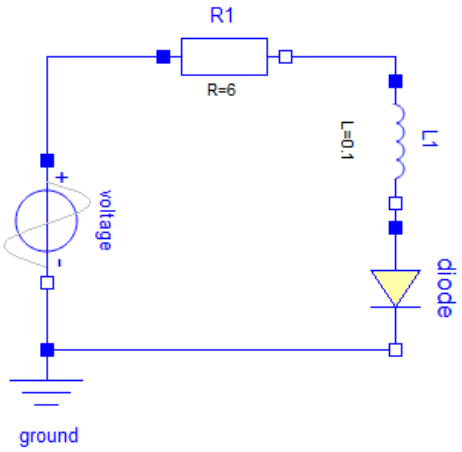


Figure 7. Inductor in series to an ideal diode model.

The current through the inductor, L1.i, is a state when the diode is in state "closed". When the diode is in state "open", the current through the diode is zero, which poses a state constraint forcing also the current though the diode, L1.i, to be zero, which means L1.i cannot be a state in that mode. Such circuits can now be handled with the *multi-mode Pantelides algorithm*, whereas using the ideal diode model from package Modelica would give a singular system during simulation.

Application of the standard *Pantelides algorithm* on the version with the ideal diode model of Figure 7 does not lead to differentiated equations. BLT does not lead to algebraic loops (provided the zero potential at the ground object is utilized). However, the sorted equations contain equation (21), as already discussed

DOI 10.3384/ecp1511889

in section 4. Since this equation is structurally singular when state diode.open is active, the differentiated connector variables, der(diode.open.p.i) and der(diode.closed.p.i) are newly introduced in the variable association list (der(L1.i) is already present). In the next iteration of the algorithm an algebraic loop occurs which is structurally regular in both modes and the algorithm terminates.

5.3 Varying index with capacitor and diode

It is also possible to simulate the case of an ideal diode that is in parallel to a capacitor:

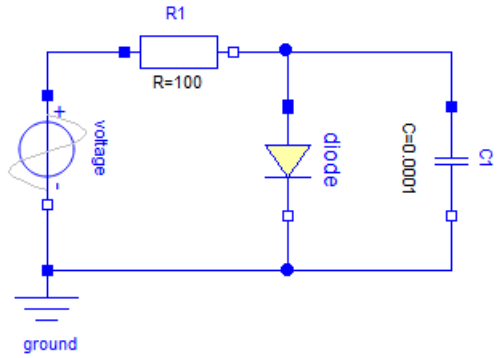


Figure 8. Capacitor in parallel to an ideal diode model.

When the diode is open, C1.v is a state, when it is closed, it is no state. The *multi-mode Pantelides algorithm* handles this system as well. It is interesting to compare simulations of this ideal diode model with the approximate ideal diode model of package Modelica:

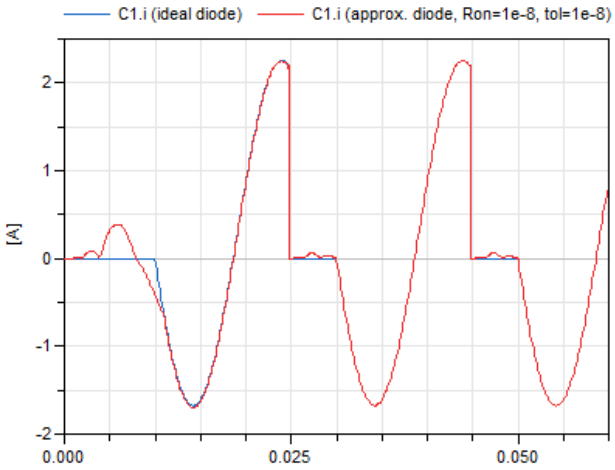


Figure 9. Simulation results of Figure 8.

Even for small values Ron = Goff = 10^{-8} and strict relative error tolerances of 10^{-8} unphysical vibrations occur that are not present with the ideal diode model of Figure 6 giving the correct mathematical solution.

5.4 Varying index with breaking shaft

In Figure 10 a breaking shaft model is shown that could not be handled in (*Elmqvist et al., 2014*): In the beginning two inertias are rigidly connected together. When the absolute value of the cut-torque tau = inertia2.flange_b.tau becomes too large, the shaft breaks and two not-connected inertias remain. This is a case where three iterations of the *multi-mode Pantelides algorithm* are needed: In the first iteration the potential equations of the inertias (= flange angles) are differentiated, in a second iteration these differentiated equations are differentiated again, and in the third iteration it is recognized that the highest derivative equations are structurally regular for all modes. The Dymola prototype selects variables inertia1.phi and inertia1.w statically as states and then there are two conditional state selections for inertia2.phi and inertia2.w.

6 Limitations

The central result of this paper, the *multi-mode Pantelides algorithm*, was tested with several simple examples. However, much more tests especially with large models are needed. It might still be the case that improvements of the algorithm are needed. The following limitations are already known:

When using continuous-time state machines it is easy to model systems where Dirac impulses occur. For example, replacing the diode in Figure 8 by an electrical switch and closing this switch when the voltage drop is not zero, will result in a Dirac impulse. Simulation is usually successful. However, the "propagation" of impulses is not taken into account and therefore in many cases the simulation results will not be correct.

Another issue are the transition conditions: When they are functions of the state connector variables and these variables are differentiated, then the transition conditions might need to be differentiated as well. For example, friction can be modeled with the state machine of Figure 11 (the orange lines are mechanical connections that have angular velocity and not angle as potential variables).

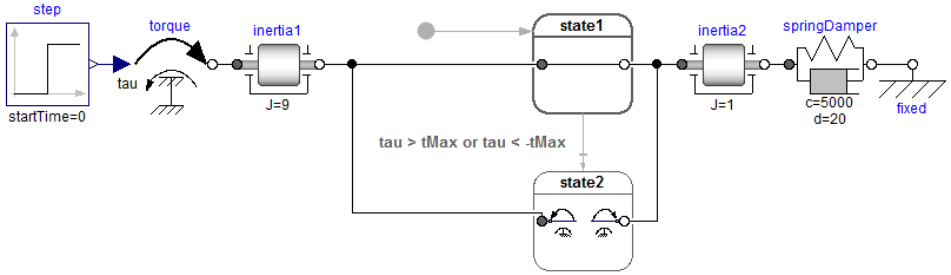


Figure 10. Shaft that breaks due to an overload toque tau > tMax or tau < -tMax

The transition conditions from sliding to Stuck mode are the critical part: w_rel > 0 or w_rel < 0. When in Stuck mode, the constraint variable, w_rel, will be zero or close to zero and when switching from Stuck to Forward or Backward mode then small numerical errors will give different results, especially if dynamically coupled friction elements are present. It is well-known that for this switching direction the derivative of w_rel has also to be taken into account. It is not yet clear how to deduce this with an algorithm.

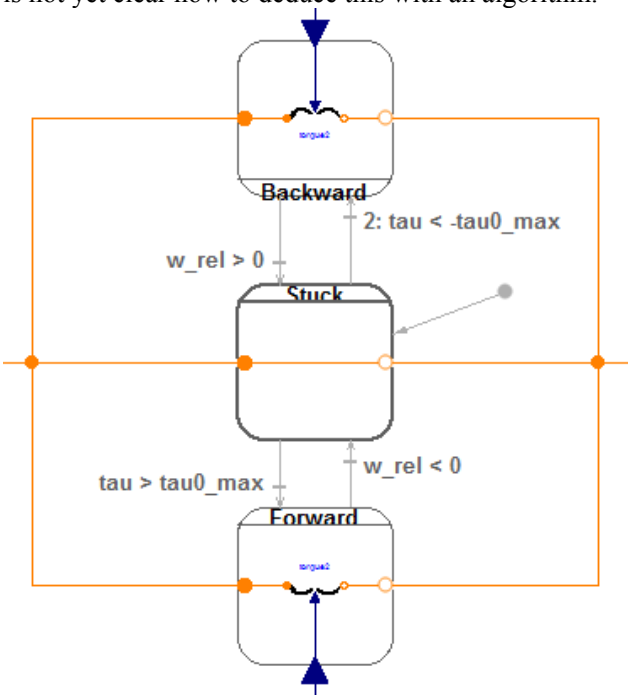


Figure 11. Model of a Coulomb friction element that cannot be handled with the approach of this paper.

7 Conclusions and outlook

In (*Elmqvist et al., 2014*) a new approach was developed to define variable structure systems with varying number of continuous-time states in a convenient way with acausal continuous-time state machines. With a rather simple technique it was possible to symbolically analyze and simulate such systems. In the current paper the limitations of the previous approach have been reduced by generalizing the *Pantelides algorithm* for multi-mode systems. It is then possible to handle continuous-time state machines where state constraints can vary when switching to a new state. There are still unresolved issues and further development is needed before a robust and reliable solution becomes available for the user.

Acknowledgements

This paper is based on research performed within the ITEA2 project MODRIO. Partial financial support of the Swedish VINNOVA and the German BMBF are highly appreciated.

Additionally, inspiring discussions with Albert Benveniste and Benoit Caillaud about their approach to handle multi-mode systems are appreciated as well.

References

Albert Benveniste, Timothy Bourke, Benoît Caillaud, Marc Pouzet (2014): **On the index of multi-mode DAE Systems (also called Hybrid DAE Systems)**. [Research Report] RR-8630, Inria; ENS. <hal-01084069>. Download: https://hal.inria.fr/hal-01084069/document

Dassault Systèmes (2015): **Dymola 2016**. http://www.Dymola.com

Elmqvist H., Gaucher F., Mattsson S.E., Dupont F. (2012): **State Machines in Modelica**. Modelica'2012 Conference, Munich, Germany, Sept. 3-5, 2012. Download: http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf

Elmqvist H., Mattsson S.E., Otter M. (2014): **Modelica extensions for Multi-Mode DAE Systems**. Proceedings of the 10th International Modelica Conference, March 10-12, Lund, Sweden, pp. 183-193. Download: http://www.ep.liu.se/ecp/096/019/ecp14096019.pdf

Höger C. (2014): **Dynamic Structural Analysis for DAEs**. Proceedings of the 2014 SCS Summer Simulation Multiconference. Download: http://dl.acm.org/ft_gateway.cfm?id=2685629&ftid=1511015&dwn=1&CFID=532067289&CFTOKEN=59766485

Mattsson, S.E. and G. Söderlind (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14, pp. 677-692.

Mattsson S.E., Olsson H., Elmqvist H. (2001): **Methods and Algorithms for Varying Structure Hybrid DAE Simulation.** EC IST Project Realsim. Contract number: IST-1999-11979, Internal Report 2.2, Dynasim, Lund, Sweden.

Modelica Association (2014): **Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3, Revision 1**. June 11. Download: https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf

Pantelides C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, 9(2), pp. 213–231.

Pepper P., Mehlhase A., Höger C., Scholz L. (2011): **A Compositional Semantics for Modelica-style Variable-structure Modeling**. 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. ETH Zürich, Switzerland. Download: http://www.ep.liu.se/ecp/056/006/ecp1105606.pdf

Pryce J.D. (2001): **A simple structural analysis method for DAEs**. BIT Numerical Mathematics, Vol. 41, No. 2, pp. 364–394.

Zimmer D. (2010): **Equation-Based Modeling of Variable-Structure Systems**. Dissertation, ETH Zürich, No. 18924. Download: http://www.inf.ethz.ch/personal/fcellier/PhD/zimmer_phd.pdf