Towards the separate compilation of Modelica: modularity and interfaces for the index reduction of incomplete DAE systems

Albert Benveniste[1] Benoît Caillaud[1] Mathias Malandain[1] Joan Thibault[1]

[1]Inria centre at Rennes University / IRISA, France,
{albert.benveniste,benoit.caillaud,mathias.malandain,joan.thibault}@inria.fr

Abstract

A key feature of the Modelica language is its object-oriented nature: components are instances of classes and they can aggregate other components, so that extremely large models can be efficiently designed as "trees of components". However, the structural analysis of Modelica models, a necessary step for generating simulation code, often relies on the flattening of this hierarchical structure, which undermines the scalability of the language and results in widely-used Modelica tools not being able to compile and simulate such large models.

In this paper, we propose a novel method for the modular structural analysis of Modelica models. An adaptation of Pryce's Sigma-method for non-square DAE systems, along with a carefully crafted notion of component interface, make it possible to fully exploit the object tree structure of a model. The structural analysis of a component class can be performed once and for all, only requiring the information provided by the interface of its child components. The resulting method alleviates the exponential computation costs that can be yielded by model flattening; hence, its scalability makes it ideally suited for the modeling and simulation of large cyber-physical systems.

Keywords: DAE, Modelica, object-oriented modeling, index reduction, structural analysis, linear programming, interface theory, difference bound matrices

1 Introduction

System modeling tools are key to the engineering of safe and efficient Cyber-Physical Systems (CPS). Although ODE-based languages and tools, such as Simulink (MathWorks, Inc. 1994–2023), are widely used in industry, there are two main reasons why DAE-based modeling is best suited to the modeling of such systems: it enables a modeling based on first principles of the physics; it is physics-agnostic, and consequently accomodates arbitrary combinations of physics (mechanics, electrokinetics, hydraulics, thermodynamics, chemical reactions, etc.).

The pioneering work by Hilding Elmqvist (Elmqvist 1978) led to the emergence of the Modelica community in the 1990s, and the DAE-based modeling language of the same name (Modelica Association 2023) has become a *de facto* standard, with its object-oriented nature enabling a component-based modeling style. Its combined use with the port-Hamiltonian paradigm (Rashad et al. 2020) results in a methodology that is instrumental to the scalable modeling of large systems, additionally ensuring that the model architecture preserves the system architecture, in stark contrast to ODE-based modeling (Benveniste, Caillaud, Elmqvist, et al. 2019; Benveniste, Caillaud, and Malandain 2022).

Consequently, DAE-based modeling requires that Modelica tools properly scale up to very large models. However, although Modelica enables the modeling of extremely large systems, its implementations (Dassault Systèmes 2002–2023; Fritzson et al. 2020) are often not capable of compiling and simulating such large models. Scaling has been and still is a subject for sustained effort by the Modelica community (Casella and Guironnet 2021), and although HPC issues belong to the landscape (Braun, Casella, and Bachmann 2017), a more specific issue is of uttermost importance for the Modelica language.

In the first steps of the compilation of a Modelica model, its hierarchical structure is flattened, thanks to a recursive syntactic inlining of the objects composing it.[1] The result of this flattening process is an unstructured DAE that can be exponentially larger than the source model. The *structural analyses* that are required for the generation of simulation code (namely, the *index reduction* of the DAE system, followed by a *block-triangular form* transformation of the reduced-index system) are then performed on this monolithic DAE model. As the compilation process does not fully take advantage of the hierarchical nature of the models it has to handle, the modeling capabilities offered by the Modelica language are undermined by performance issues on the structural analysis itself (Höger 2015; Höger 2019). Additionally, model flattening poses a challenge when attempting to extend DAE-based modeling to higher-order modeling or dynamically changing systems (Broman and Fritzson 2008; Broman 2010; Broman 2021).

In this paper, a new modular structural analysis algorithm is proposed that takes full advantage of the object tree structure of a DAE model. The bedrock of this method is a novel concept of *structural analysis-aware interface* for components. The essence of a component interface is to capture the necessary information about a Modelica class that needs to be exposed, in order to perform the structural

[1]See (Modelica Association 2023), Section 5.6 for a complete definition of the flattening process.

analysis of a component comprizing instances of the former class, while hiding away useless information regarding the equations and all *protected* features it may contain.

In order to compute a component interface, one has to be able to perform the structural analysis of the possibly non-square DAE system that this component encapsulates, and to use the interfaces of the components it aggregates in this analysis. We base our algorithm on Pryce's Σ-method for index reduction (Pryce 2001), which essentially consists in the successive solving of two dual linear integer programs. The striking difference with Pryce's algorithm is that these problems are solved by parts, in a scalable manner.

Putting all of this together, it is then possible to perform a *modular structural analysis*, in which structural analysis is performed at the class level, and the results can then be instantiated for each component of the system model, knowing its context. Hence, structural information at the system level is derived from composing the result of component-level analysis. Modular structural analysis yields huge gains in terms of memory usage and computational costs, as the analysis of a single large-scale DAE is replaced with that of multiple smaller subsystems. Moreover, the analysis is performed at the class level, meaning that a single structural analysis is needed for all system components that are instances of the same class.

To the best of our knowledge, only (Höger 2015) addresses the specific issue of performing the structural analysis of a hierarchical model; Section 2.1 of this paper shows the approach proposed by Höger and explains why we regard it as an efficient but still partial solution. On the related subject of sorting equations, attention is paid in (Zimmermann, Fernández, and Kofman 2019) to methods that avoid unrolling loops and expanding arrays when sorting equations, an issue targeting the same overall objective as both (Höger 2015; Höger 2019) and the present work.

Section 2 introduces two simple examples of DAE systems, to be used for illustrative purposes, and explains the modular approach from (Höger 2015) on one of them. In Section 3, we provide background information on structural analysis, which includes the Σ-method used to perform index reduction.

In Section 4, we introduce Σ-systems as an abstraction of DAE systems suitable for structural analysis. Σ-systems include the linear programming problems associated with the Σ-method and the structural description of the BTF. They also provide a notion of composition that abstracts the composition of DAE systems. This formalization leads us to the main contribution of this paper, presented in Section 5, where we propose the notion of Σ-interface. A Σ-interface exposes the necessary information to assemble partial structural analyses into a system-level structural analysis for a DAE system. We discuss how Σ-interfaces can reduce the computational cost of structural analysis for large systems.

Finally, Section 6 introduces our prototype implementation of the resulting modular method, then presents numerical applications to both examples presented in Section 2.

2 Examples

In this section, we develop two illustrative examples: a slightly modified version of the chained circuit proposed by C. Höger in (Höger 2015), and a homemade chained mass-and-spring system. Both will be used to illustrate how our approach changes and improves the structural analysis process of models where several instances of a same class are connected, while also highlighting different features of our algorithm.

2.1 A chained circuit

2.1.1 The circuit model

This example is a minor modification (with a second inductance added) of the one developed in (Höger 2015). The following text is borrowed verbatim from this reference:

> The Modelica representation of the circuit *[...]* consists of a name (`Circuit`), declarations of parameters (`n`), unknowns (`u` and `i`) and subcomponents (`c`). The physical behavior is defined directly by multiple equations, including the description of sub-circuit interconnection. Without knowledge about the internal structure of `SubCircuit` it is possible to use it inside the larger circuit, as long as it provides the corresponding interface (i.e. variables `u` and `i`). This composition of models is an easy and safe way to create more complex models out of simpler building blocks and is the very foundation of object-oriented modeling.

The `Circuit` and `SubCircuit` Modelica codes and schematics are given in Fig. 1 and 2 respectively.

2.1.2 The approach by C. Höger

While the flattening of the `Circuit` model is linear with `n`, the cost of the structural analysis of the resulting model is super-linear, thus preventing the classical approach from properly scaling up. In (Höger 2015), important contributions are proposed to cope with this problem. As far as we know, this paper is the first one pointing this issue very clearly. A faster method is proposed to perform the structural analysis of a hierarchical model, involving scoping and hiding.

This method is based on Pryce's Σ-method (Pryce 2001), which is explained in Section 3. The Σ-method involves solving a pair of dual Linear Programming problems (*primal* and *dual*), using a specific iterative algorithm for the dual. In Theorem 1 of (Höger 2015), the author provides a complexity argument to support the claim that the dual problem is not a bottleneck on its own.[2] Hence, the author focuses on the primal problem, for which a decomposition method is proposed.

[2] Our own numerical experiments confirm this observation.

```
model Circuit
  parameter Integer n = 10;
  SubCircuit[n] c;
  Real u,i;
equation
  c[1].i = i;
  u = sin(time);
  for j in 2:n loop
    c[j].i = c[j-1].i;
  end for;
  sum(c.u) = u;
end Circuit;
```

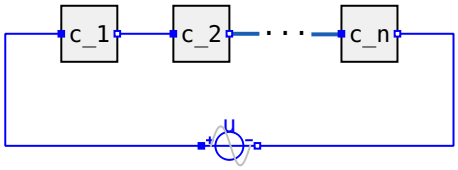


Figure 1. Modelica code and electrical schematics of the chained circuit.

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC=C*der(uC);
  uL1=L1*der(iL1);
  uL2=L2*der(iL2);
  u1 = R1 * i1;
  u2 = R2 * i2;
  u2 = uL1;
  uC=u1+u2;
  i1=i2+iL1;
  u=u1+uL1+uL2;
  i=i1+iC;
  i=iL2;
end SubCircuit;
```

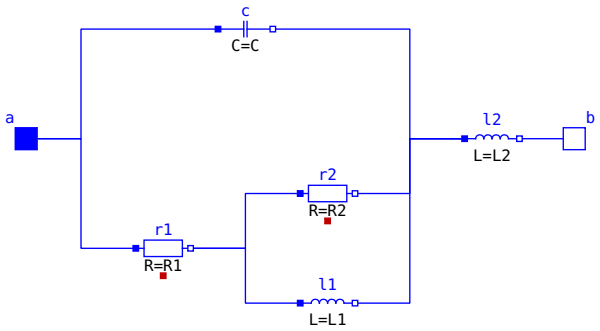


Figure 2. Modelica code of the **SubCircuit** class, and schematics of the corresponding electrical circuit.

2.2 A recursive mass-spring-damper system

Figure 3 shows the model, consisting of a chain of mass-spring-damper elements, defined with the 1D translational components of the Modelica Standard Library (MSL). Although recursive classes are not allowed in Modelica, we use a recursive definition of a chain of elements. This gives a binary-tree structure to the model, that is best suited for the modular structural analysis method presented in the sequel of the paper. Our prototype implementation of the method supports recursive classes, with conditional statements evaluated statically, at compile time.

2.3 Our contribution for these examples

Despite its important contribution, we believe that (Höger 2015) does not provide the ultimate answer. While a hierarchical algorithm for solving the primal problem is a great contribution in terms of computational costs, it still does not fully take advantage of object-oriented modeling. We would like instead to advance towards the separate compilation of components and systems, which consists in:

1. Proposing a notion of interface for model components, that is rich enough to subsequently perform system-level structural analysis; and
2. Proposing a modular structural analysis method, consisting of the needed algorithms performing hierarchical structural analysis based on interface information.

In doing so, both primal and dual problems, as well as the construction of a Block-Triangular Form (BTF) for the Jacobian, need to be addressed—this is also in contrast to the approach of (Höger 2015), where only the primal problem is considered.

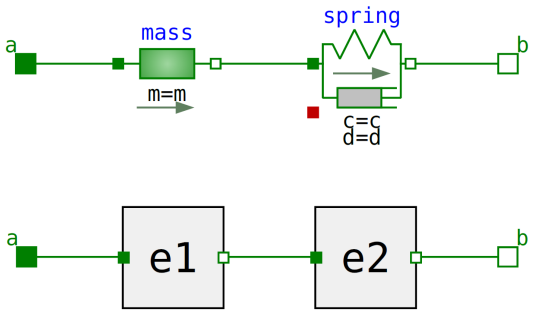
To simplify our presentation, we skip the discussion of the BTF; it will be developed in an extended version of this paper, that is currently in the works.

3 The Σ-method for DAE

The index reduction method proposed by J. Pryce (Pryce 2001), called the Σ-*method*, is an interesting alternative to the classical method originally proposed by C. Pantelides (Pantelides 1988). Its elegant and compact formulation as a pair of dual Linear Programming problems (LP) makes it particularly valuable for an extension to DAE components and architectures. The Σ-method can be summarized as follows:

The Σ-matrix

For $S=(F,X)$ a square DAE system involving equations f=0, where $f\in F$, and variables $x\in X$ and their derivatives, form the Σ-matrix $\Sigma=(\sigma_{f,x})_{(f,x)\in F\times X}$ of the DAE system, where $\sigma_{f,x}$ is the highest differentiation order of variable x in f, or $-\infty$ if x does not appear in f.




```
import ...;
model HarmonicString
   parameter Integer n = 1 "Number of elements";
   parameter Mass m = 1e-3 "Mass";
   parameter Distance l = 1e-2 "Length";
   parameter TranslationalSpringConstant c = 1;
   parameter TranslationalDampingConstant d = 1e-3;
   parameter Distance s0 = 0 "Initial position";
   Flange_a a;
   Flange_b b;
   static if n > 1 then
   protected
     parameter Integer n1 = n/2;
     parameter Integer n2 = n - n1;
     HarmonicString s1(n=n1, m=n1*m/n, ...);
     HarmonicString s2(n=n2, m=n2*m/n, ...);
   else
   protected
     Element e(m=m, l=l, c=c, d=d, s0=s0);
   end if;
equation
   ...
end HarmonicString;
```

Figure 3. A mass-spring-damper system; the element (top-left) is assembled from the 1-D translational mechanical components of the Modelica Standard Library. An assembly of two elements is shown at the bottom-left. A chain of mass-spring-damper elements of length n is defined by the recursive Modelica-like class shown on the right. Although Modelica does not allow for recursive classes, our software prototype allows recursion, provided conditional statements can be evaluated at compile-time.

Primal problem

The primal LP encodes the search for a maximum weight transverse of Σ, that will be described as $(f,x_f)_{f\in F}$ or, equivalently, $(f_x,x)_{x\in X}$ in what follows. The existence of a solution to the primal LP is a success check for the Σ-method.

Dual problem

The variables of the associated dual LP are variable offsets $(d_x)_{x\in X}$ and equation offsets $(c_f)_{f\in F}$, and we search for the minimal non-negative solution to this LP. (Pryce 2001) proves the uniqueness of this solution and proposes a relaxation method for finding it, given a solution to the primal problem.

The dual problem can be rewritten as the following constraint system, involving only the variable offsets $(d_x)_{x\in X}$:

$$\begin{array}{rrcl} \forall x\in X: & d_x & \geq & \sigma_{f_x,x} \\ \forall (f,x)\in E: & d_x - d_{x_f} & \geq & \sigma_{f,x} - \sigma_{f,x_f} \end{array} \quad (1)$$

and the equation offsets are then given by

$$c_f \;=\; d_{x_f} - \sigma_{f,x_f} \;. \quad (2)$$

It is proved in (Pryce 2001) that the set of solutions of (1) does not depend on the particular choice of a solution to the primal problem.

Use of the offsets

Equation offset c_f indicates how many times equation f=0 needs to be differentiated to get the *index-reduced system*, whereas d_x indicates the maximum differentiation order of x in this index-reduced system. In addition, the solution of the primal problem is useful for computing a BTF for the Jacobian of this system.

C. Höger (Höger 2015) explains that the primal problem is the main bottleneck, hence it focuses on solving it efficiently, by decomposing it into smaller subproblems. In contrast, we extend and adapt the Σ-method for *open DAE systems*, which are DAE systems that can possess more variables than equations, and that can be composed with other open DAE systems by unifying their common variables.

4 Structural analysis of open DAE

The set of variables of an open DAE system can be decomposed as

$$X = X^s \uplus X^\ell$$

where X^s and X^ℓ are its sets of *shared* and *local* variables, respectively. For example, the **`SubCircuit`** of Fig. 2 possesses 2 shared variables and 10 local variables (declared as **`protected`** in Modelica).

4.1 Selectors

Since **`SubCircuit`** possesses 11 equations, we can regard it as a square system by assuming that one among the shared variables i, u is dependent and the other one is free (determined by the yet unspecified environment of this circuit):

$$X^s = X^{\mathsf{free}} \uplus Y$$

where $Y \subseteq X^s$ is a subset of the shared variables. The set of dependent variables of the system is then $X^{\mathsf{dep}} = Y \uplus X^\ell$.

If we compose open DAE system S with an environment S' (another open DAE system), then the two selectors Y

and Y' for S and S' must satisfy the condition

$$Y \cap Y' \quad = \quad \emptyset, \tag{3}$$

expressing that S' cannot claim determining variable x if the latter is already claimed by S, i.e., belongs to Y. We say that selectors Y and Y' are *compatible* if (3) holds.

For **SubCircuit**, two possible choices for selectors are

$$Y = \{\mathtt{i}\} \quad \text{or} \quad Y = \{\mathtt{u}\}. \tag{4}$$

4.2 Matching selectors in compositions

If $\{\mathtt{i}\}$ is selected, then **u** is free, i.e., it must be determined by the (yet unspecified) environment. Thus, we expect this environment to allow for a selector containing **u** but not **i**. This means that information (4) has to be exposed by S as part of its interface for structural analysis.

Then, by exposing (4), S sets structural constraints on the interface of any environment for it. We will show that

information (4) is sufficient for characterizing the environments that are compatible with S. (5)

This information, however, is not sufficient to perform the structural analysis in a modular way. In the sequel, we will identify the information that is missing for this purpose.

5 Interfaces for the modular structural analysis of DAE systems

This section introduces the main contribution of this paper, which is the notion of component interface needed for the structural analysis of DAE systems. This notion is called Σ-interface as a reference to the Σ-method itself. Possible ways to perform the modular structural analysis of the **Circuit** example are also shown, as a way to illustrate the benefits of the modular approach.

5.1 The Σ-method for open DAE systems

We first show the result of the Σ-method, applied to the open DAE system **SubCircuit** (see Fig. 2), for the two possible choices of selectors given by (4).

5.1.1 Primal problem

The solutions of the primal problem for both selectors are given in Fig. 4. For each case, the free variable for the considered selector is written in blue. The chosen maximum weight transverse is indicated by highlighting in red the dependent variable associated to each equation, and we give in the third column the differentiation order of this variable in this equation. Two important observations will guide us in defining the primal Σ-interface:

1. The choice of the transverse, which fixes the assignment of variables to equations, *does not depend on the* (yet unspecified) *environment of* **SubCircuit**;
2. The total weight of the maximal transverse of **SubCircuit**, for a given selector, *is added* to the total weight of a maximal transverse of the environment (for a compatible selector), yielding the total weight of the overall transverse (that covers both the component and its environment).

5.1.2 Dual problem

The dual problem is the constraint system (1), whose dependent variables are the offsets $(d_x)_{x\in X}$. For **S** $=_{\text{def}}$ **SubCircuit**, both selectors have to be considered:

- **S** has selector $\{\mathtt{i}\}$. A maximal weight transverse is shown in red on Fig. 4-left. With this transverse, the dual problem (1) is shown in Fig. 5-left.
- **S** has selector $\{\mathtt{u}\}$. A maximal weight transverse is shown in red on Fig. 4-right. With this transverse, the dual problem (1) is shown in Fig. 5-right.

Note that, while edges (f,x) are local (since all equations are local), variables can be shared between components; hence, dual problems only interact via the offsets of their shared variables (**i** and **u** for **SubCircuit**). This observation will guide us in defining the notion of dual Σ-interface.

5.2 Σ-interfaces

5.2.1 Primal Σ-interfaces

We say that a selector for a component is *consistent* if the primal problem has at least one solution for this selector.

Primal Σ-interface of a component Based on the discussion at the end of Section 5.1.1, it suffices to expose,

for each consistent selector, the set of all pairs (selector, maximal transverse weight) (6)

at the interface of the considered component, for the primal problem. Thus, (6) defines the *primal Σ-interface* of an open DAE system. For the **SubCircuit** example, the primal Σ-interface is the set

$$\left\{ \left(Y = \{\mathtt{i}\}, \mathrm{J}_S^{Y=\{\mathtt{i}\}} = 3\right), \left(Y = \{\mathtt{u}\}, \mathrm{J}_S^{Y=\{\mathtt{u}\}} = 2\right) \right\} \tag{7}$$

Composing primal Σ-interfaces Let us consider two open DAE systems S_1 and S_2 whose composition $S = S_1 \cup S_2$ is another open DAE system. Then, solving the primal problem for S for a given selector Y is equivalent to:

1. Solving the primal problems for $S_i, i{=}1,2$, for each selector Y_i, thus producing optimal weights $\mathrm{J}_i^{Y_i}$, then
2. Selecting an *optimizing compatible pair* (Y_1, Y_2) of selectors, i.e., solve

$$\max \left\{ \mathrm{J}_1^{Y_1} + \mathrm{J}_2^{Y_2} \mid Y_1 \cap Y_2 = \emptyset \text{ and } Y_1 \cup Y_2 = Y \right\} . \tag{8}$$

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC = C * der(uC);                    // 1
  uL1 = L1 * der(iL1);                 // 1
  uL2 = L2 * der(iL2);                 // 1
  u1 = R1 * i1;                        // 0
  u2 = R2 * i2;                        // 0
  u2 = uL1;                            // 0
  uC = u1 + u2;                        // 0
  i1 = i2 + iL1;                       // 0
  u = u1 + uL1 + uL2;                  // 0
  iL2 = i1 + iC;                       // 0
  i = iL2;                             // 0
end SubCircuit;
```

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC = C * der(uC);                    // 1
  uL1 = L1 * der(iL1);                 // 1
  uL2 = L2 * der(iL2);                 // 0
  u1 = R1 * i1;                        // 0
  u2 = R2 * i2;                        // 0
  u2 = uL1;                            // 0
  uC = u1 + u2;                        // 0
  i1 = i2 + iL1;                       // 0
  u = u1 + uL1 + uL2;                  // 0
  iL2 = i1 + iC;                       // 0
  i = iL2;                             // 0
end SubCircuit;
```

Figure 4. The primal problem for **SubCircuit**, seen as an open DAE system, for two possible choices for the selector. Left: selector $Y = \{\mathtt{i}\}$ yields a maximal transverse (in red) of weight 3; right: selector $Y = \{\mathtt{u}\}$ yields a maximal transverse (in red) of weight 2. In each case, the contribution of each equation to the weight of the transverse is provided on the right of the equation, and the free variable (to be determined by the environment) is highlighted in blue.

$$
\begin{array}{rcr}
\forall \mathbf{x},\ d_{\mathbf{x}} & \geq & 0 \\
d_{\mathtt{uC}}, d_{\mathtt{iL1}}, d_{\mathtt{iL2}} & \geq & 1 \\
d_{\mathtt{iC}} - d_{\mathtt{uC}} & \geq & -1 \\
d_{\mathtt{uL1}} - d_{\mathtt{iL1}} & \geq & -1 \\
d_{\mathtt{uL2}} - d_{\mathtt{iL2}} & \geq & -1 \\
d_{\mathtt{u1}} - d_{\mathtt{i1}} & \geq & 0 \\
d_{\mathtt{i2}} - d_{\mathtt{u2}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{uL1}} & \geq & 0 \\
d_{\mathtt{uC}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{iL1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{u1}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{uL1}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{u}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{i}} & \geq & 0
\end{array}
\qquad
\begin{array}{rcr}
\forall \mathbf{x},\ d_{\mathbf{x}} & \geq & 0 \\
d_{\mathtt{uC}}, d_{\mathtt{iL1}} & \geq & 1 \\
d_{\mathtt{iC}} - d_{\mathtt{uC}} & \geq & -1 \\
d_{\mathtt{uL1}} - d_{\mathtt{iL1}} & \geq & -1 \\
d_{\mathtt{iL2}} - d_{\mathtt{uL2}} & \geq & +1 \\
d_{\mathtt{u1}} - d_{\mathtt{i1}} & \geq & 0 \\
d_{\mathtt{i2}} - d_{\mathtt{u2}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{uL1}} & \geq & 0 \\
d_{\mathtt{uC}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{iL1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{u1}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{uL1}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{uL2}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{i}} - d_{\mathtt{iL2}} & \geq & 0
\end{array}
$$

Figure 5. The dual problem when **S** has selector $\{\mathtt{i}\}$ (left) and $\{\mathtt{u}\}$ (right).

The primal Σ-interface of S is then obtained by collecting the pairs (Y, J_S^Y) for every consistent selector of S. Given a consistent selector Y for S, we denote by

$$(\pi_1(Y), \pi_2(Y))$$

an optimizing selector pair. Remark that this pair may not be unique. However, the end result of the structural analysis (the offsets d_x and c_f) does not depend upon the choice of $(\pi_1(Y), \pi_2(Y))$.

5.2.2 Dual Σ-interfaces

Dual Σ-interface of a component Based on the discussion at the end of Section 5.1.2, we can now define the *dual* Σ-*interface* as collecting,

> for each consistent selector Y, the projection $\mathscr{D}^Y$ of system (1) on the subset of offsets $(d_x)_{x \in X^s}$. (9)

For the **SubCircuit** example, the dual Σ-interface is the following set (collecting two elements):

$$\left\{ \begin{array}{l} \left(Y = \{\mathbf{i}\},\ \mathscr{D}^{Y=\{\mathtt{i}\}} : \left\{ \begin{array}{l} d_{\mathtt{i}}, d_{\mathtt{u}} \geq 0 \\ d_{\mathtt{i}} \leq d_{\mathtt{u}} + 1 \end{array} \right. \right) \\ \left(Y = \{\mathbf{u}\},\ \mathscr{D}^{Y=\{\mathtt{u}\}} : \left\{ \begin{array}{l} d_{\mathtt{i}}, d_{\mathtt{u}} \geq 0 \\ d_{\mathtt{u}} \leq d_{\mathtt{i}} + 1 \end{array} \right. \right) \end{array} \right\}, \tag{10}$$

In (10), the two constraint systems are obtained by projecting, over the offsets $d_{\mathtt{i}}$ and $d_{\mathtt{u}}$, the constraint systems given in Fig. 5.

Composing dual Σ-interfaces Two open DAE systems $\mathbf{S}_1$ and $\mathbf{S}_2$ can only interact via their shared variables $X_1^s \cup X_2^s$. Hence, for their composition **S**:

> Given a consistent selector Y for **S**, the projection of the dual problem of **S** onto the offsets of $X^s \subseteq X_1^s \cup X_2^s$ is the composition of the projections of the dual problems $\mathscr{D}_i^{\pi_i(Y)}$ of $\mathbf{S}_i, i{=}1,2$ over the offsets of X^s. (11)

5.3 Using Σ-interfaces in DAE systems

In this section, we illustrate the use of Σ-interfaces for the modular structural analysis of DAE systems. To each open DAE system, we associate its Σ-*interface*, by fusing the primal and dual Σ-interfaces defined above: it is a set of triples whose elements are a consistent selector, the weight of a solution of the corresponding primal problem, and the projection of the dual problem on the offsets of the shared variables. We shall then use (8) and (11) to perform the structural analysis of a DAE system in a modular way.

5.3.1 The `Circuit`, seen as a chain

From (7) and (10), one gets the following Σ-interface for `SubCircuit`:

$$\left\{ \begin{array}{l} \left(Y = \{\mathtt{i}\}\,,\, \mathrm{J}_S^Y = 3, \left[\begin{array}{l} d_\mathtt{i}, d_\mathtt{u} \geq 0 \\ d_\mathtt{i} \leq d_\mathtt{u} + 1 \end{array} \right] \right) \\ \left(Y = \{\mathtt{u}\}\,,\, \mathrm{J}_S^Y = 2, \left[\begin{array}{l} d_\mathtt{i}, d_\mathtt{u} \geq 0 \\ d_\mathtt{u} \leq d_\mathtt{i} + 1 \end{array} \right] \right) \end{array} \right\} \tag{12}$$

In what follows, successive instances of `SubCircuit` are chained, as shown in Fig. 1. The structural analysis will be performed by induction on the length n of the chain.

Call $\mathtt{S}_n$ the chain of length n; in particular, $\mathtt{S}_1 =$ `SubCircuit`, denoted by `SC` in what follows. Call `i`, $\mathtt{u}_n$ the shared variables of $\mathtt{S}_n$. Seen an a chain, $\mathtt{S}_n$ is obtained by composing $\mathtt{S}_{n-1}$ with `SC` and adding a Kirchhoff equation for voltages. We regard this last equation as a component with no local variables, denoted by `eq` below.

Note that we also have to rename `u` as `v` in `SC` in order to avoid name clashes, which we write `[u/v]`, and that variable hiding has to be used on the result of the composition, as variables $\mathtt{u}_{n-1}$ and `v` have to be made local in $\mathtt{S}_n$.

As a result, for all $n \geq 2$, $\mathtt{S}_n$ is defined as the following composition:

$$\mathtt{S}_n = \texttt{hide } \mathtt{u}_{n-1}, \mathtt{v} \texttt{ in } \left\{ \begin{array}{l} \mathtt{S}_{n-1} \\ \mathtt{SC[u/v]} \\ \mathtt{u}_n = \mathtt{u}_{n-1} + \mathtt{v} \end{array} \right. \tag{13}$$

For every n, $\mathtt{S}_n$ has exactly one more variable than it has equations. Thus, there are two possible selectors for $\mathtt{S}_n$: $Y_{\mathtt{S}_n} = \{\mathtt{i}\}$ and $Y_{\mathtt{S}_n} = \{\mathtt{u}_n\}$.

Case $Y_{\mathtt{S}_n} = \{\mathtt{u}_n\}$ Then, `i` is free and there is only one triple of compatible selectors in the composition occurring in the right-hand side of (13):

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{u}_{n-1}\}, Y_\mathtt{SC} = \{\mathtt{v}\}, Y_\mathtt{eq} = \{\mathtt{u}_n\}\right) \ .$$

An immediate induction argument shows that the optimal weight for $\mathtt{S}_n$ is $2(n-1)+0+2 = 2n$.

We prove by induction that the dual Σ-interface is

$$\left\{ \begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_n} \geq 0 \\ d_{\mathtt{u}_n} \leq d_\mathtt{i} + 1 \end{array} \right. \tag{14}$$

for every n. This holds for $n = 1$, as (14) then yields the dual Σ-interface of class `SubCircuit` for selector $\{\mathtt{u}\}$ (see (10)). Assuming that (14) holds for $n-1$, the dual Σ-interfaces compose as follows:

$$\texttt{hide } \mathtt{u}_{n-1}, \mathtt{v} \texttt{ in } \left\{ \begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n} \geq 0 \\ d_{\mathtt{u}_{n-1}} \leq d_\mathtt{i} + 1 \\ d_\mathtt{v} \leq d_\mathtt{i} + 1 \\ d_{\mathtt{u}_n} \leq d_{\mathtt{u}_{n-1}} \\ d_{\mathtt{u}_n} \leq d_\mathtt{v} \end{array} \right.$$

which yields (14) when projected on $\mathtt{u}_n$, `i`.

Consequently, the variable offsets of a `SubCircuit` in a `Circuit` are independent of the number of chained instances, and the equation offset of an equation in the k-th component of the chain is equal to that of the same equation in the original class.

Case $Y_{\mathtt{S}_n} = \{\mathtt{i}\}$ Then, $\mathtt{u}_n$ is free and there are two triples of compatible selectors:

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{u}_{n-1}\}, Y_\mathtt{SC} = \{\mathtt{i}\}, Y_\mathtt{eq} = \{\mathtt{v}\}\right) \tag{15}$$

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{i}\}, Y_\mathtt{SC} = \{\mathtt{v}\}, Y_\mathtt{eq} = \{\mathtt{u}_{n-1}\}\right) \tag{16}$$

By adding the contributions of the components, in the order they are written above, in each case, we get:

- In case (15): $[2(n-1)]+0+3 = 2n+1$
- In case (16): $[2(n-1)+1]+0+2 = 2n+1$

Thus, each of the two triples (15) and (16) is optimizing, which brings an important question: *Does the dual Σ-interface depend on the choice of a tuple of selectors?*

If triple (15) is used: System (14) for $n-1$ provides us with the dual Σ-interface of $\mathtt{S}_{n-1}$ for selector $\mathtt{u}_{n-1}$. For $\mathtt{S}_n$, the dual Σ-interfaces then compose as follows:

$$\texttt{hide } \mathtt{u}_{n-1}, \mathtt{v} \texttt{ in } \left\{ \begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n} \geq 0 \\ d_{\mathtt{u}_{n-1}} \leq d_\mathtt{i} + 1 \\ d_\mathtt{i} \leq d_\mathtt{v} + 1 \\ d_\mathtt{v} \leq d_{\mathtt{u}_{n-1}} \\ d_\mathtt{v} \leq d_{\mathtt{u}_n} \end{array} \right.$$

which yields

$$\left\{ \begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_n} \geq 0 \\ d_\mathtt{i} \leq d_{\mathtt{u}_n} + 1 \end{array} \right. \tag{17}$$

If triple (16) is used: We will prove by induction that (17) still holds. Assuming that (17) holds for $n-1$, the dual Σ-interfaces compose as follows:

$$\texttt{hide } \mathtt{u}_{n-1}, \mathtt{v} \texttt{ in } \left\{ \begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n} \geq 0 \\ d_\mathtt{i} \leq d_{\mathtt{u}_{n-1}} + 1 \\ d_\mathtt{v} \leq d_\mathtt{i} + 1 \\ d_{\mathtt{u}_{n-1}} \leq d_\mathtt{v} \\ d_{\mathtt{u}_{n-1}} \leq d_{\mathtt{u}_n} \end{array} \right.$$

which yields again (17).

Recall that solutions to the Σ-method's dual problem are independent of the particular choice of a solution for the primal problem. This property generalizes to the composition of dual interfaces:

> the dual Σ-interface of a composition does not depend on the choice of a particular optimizing tuple. (18)

5.3.2 The `Circuit`, seen as a tree

Instead of a chain, the architecture of the `Circuit` can be regarded as a binary tree:

$$\mathtt{T}_m \ =_{\text{def}} \ \begin{array}{ccc} & \widehat{\mathtt{SC}} & \\ \swarrow & & \searrow \\ \mathtt{T}^-_{m-1} & & \mathtt{T}^+_{m-1} \end{array} \tag{19}$$

where:

- the length ℓ_m of $\mathtt{T}_m$ is defined by $\ell_m = 2\ell_{m-1}+1$, which is exponential in m;
- $\mathtt{T}_m$ has shared variables $\mathtt{i}_m$, $\mathtt{u}_m$;
- $\mathtt{T}^{\pm}_{m-1}$ are two copies of $\mathtt{T}_{m-1}$ with renamings $\mathtt{[i}^{\pm}_{m-1}\mathtt{/i]}$;
- $\widehat{\mathtt{SC}}$ is made of an instance of **SC** and a component (similar to component **eq** above) made of the connection equation $\mathtt{u}_m = \mathtt{u}^{-}_{m-1} + \mathtt{u} + \mathtt{u}^{+}_{m-1}$.

We reuse the Σ-interface (12) of class **SubCircuit**, the optimal weights $2\ell_m$ and $2\ell_m+1$ for selectors $Y_{\mathtt{T}_m}=\{\mathtt{u}_m\}$ and $Y_{\mathtt{T}_m}=\{\mathtt{i}\}$ respectively, and we focus on the dual interface. It is computed by the recursion shown in Fig. 6.

$$Y_{\mathtt{T}_m}=\{\mathtt{u}_m\}: \quad \text{hide } \mathtt{u}, \mathtt{u}^{\pm}_{m-1} \text{ in } \begin{cases} 0 \le d_{\mathtt{i}}, d_{\mathtt{u}^{\pm}_{m-1}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}^{-}_{m-1}} \le d_{\mathtt{i}}+1 \\ d_{\mathtt{u}^{+}_{m-1}} \le d_{\mathtt{i}}+1 \\ d_{\mathtt{u}} \le d_{\mathtt{i}}+1 \\ d_{\mathtt{u}_m} \le d_{\mathtt{u}^{\pm}_{m-1}} \\ d_{\mathtt{u}_m} \le d_{\mathtt{u}} \end{cases}$$

$$\text{which yields} \quad \begin{cases} 0 \le d_{\mathtt{i}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}_m} \le d_{\mathtt{i}}+1 \end{cases}$$

$$Y_{\mathtt{T}_m}=\{\mathtt{i}\}: \quad \text{hide } \mathtt{u}, \mathtt{u}^{\pm}_{m-1} \text{ in } \begin{cases} 0 \le d_{\mathtt{i}}, d_{\mathtt{u}^{\pm}_{m-1}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}^{-}_{m-1}} \le d_{\mathtt{i}}+1 \\ d_{\mathtt{u}^{+}_{m-1}} \le d_{\mathtt{i}}+1 \\ d_{\mathtt{i}} \le d_{\mathtt{u}}+1 \\ d_{\mathtt{u}} \le d_{\mathtt{u}^{\pm}_{m-1}} \\ d_{\mathtt{u}} \le d_{\mathtt{u}_m} \end{cases}$$

$$\text{which yields} \quad \begin{cases} 0 \le d_{\mathtt{i}}, d_{\mathtt{u}_m} \\ d_{\mathtt{i}} \le d_{\mathtt{u}_m}+1 \end{cases}$$

Figure 6. The dual Σ-interface of tree shaped architecture (19).

5.3.3 Discussion

It is worth comparing the chain-based and tree-based approaches above. At first glance, since recursion arguments were used in both cases, the two approaches may seem equivalent in terms of computational costs. However, this impression shall not last once we detail how implementations should proceed.

- For the chain architecture of Section 5.3.1, each induction step consists in the computation of the Σ-interface of prefix $\mathtt{S}_k$ as a function of the Σ-interface of $\mathtt{S}_{k-1}$, for k increasing from 2 to n.

- For the tree-shaped architecture of Section 5.3.2, the induction step expresses the Σ-interface of the root of each subtree $\mathtt{T}_k$ as a function of the Σ-interface of each subtree $\mathtt{T}^{\pm}_{k-1}$. Remark that the Σ-interface of each subtree $\mathtt{T}_k$ is only computed once, as all leaf components are instances of the same class, and all subtrees of height k have, by construction, identical interfaces.

The number of steps is proportional to n in the first case, and $m \sim \log n$ in the second case. Since the computational complexity of each induction step is roughly the same, treating the **Circuit** model as a tree-shaped architecture can dramatically improve performance.

6 Implementation and experimental results

6.1 Implementation considerations

A significant difficulty in our modular structural analysis is the consideration of selector-dependent structural analyses. Both the primal and dual problems are functions of the selectors, which can be numerous for components with a large number of public variables. Although Modelica models are usually sparse, with components exposing only a few variables, a mere enumeration of selectors can result in an exponential growth of the handled data structures.

To deal with this issue, we advocate the approach proposed in (Benveniste, Caillaud, Malandain, and Thibault 2022; Caillaud, Malandain, and Thibault 2020) for multimode DAE systems. In these works, a *dual representation* is introduced, where equations are labeled with a predicate on mode variables, characterizing the modes under which they are active. Using this representation (instead of a direct one, that lists the active equations for each mode of the model) provides a compact representation of the structure of multimode systems. Reduced Ordered Binary Decision Diagrams, or ROBDD (Bryant 1986), provide not only an adapted data structure, but also efficient computation algorithms that can be used for performing the whole structural analysis in an "all-modes-at-once" fashion.

For modular structural analysis, a similar representation can be used, but with selectors instead of modes. The whole structural analysis chain can then be performed in an "all-selectors-at-once" fashion: (Benveniste, Caillaud, Malandain, and Thibault 2022) provides all building blocks for the computation of primal interfaces and their compositions, as presented in Section 5.2.1. The computation and composition of dual interfaces reduce to projecting parametrized constraint systems such as the one illustrated Fig. 5. Such systems are called *Difference Bound Matrices* (DBM) and come equipped with a rich calculus (Dill 1989; Miné 2001) with a polynomial computational complexity. Although there are excellent implementations of DBM, parametric DBM remain to be implemented, possibly using tuples of ROBDD for the representation of matrix elements.

6.2 Benchmarks and measured performance

Experimental results have been obtained on the modular structural analysis method, using two benchmarks: the

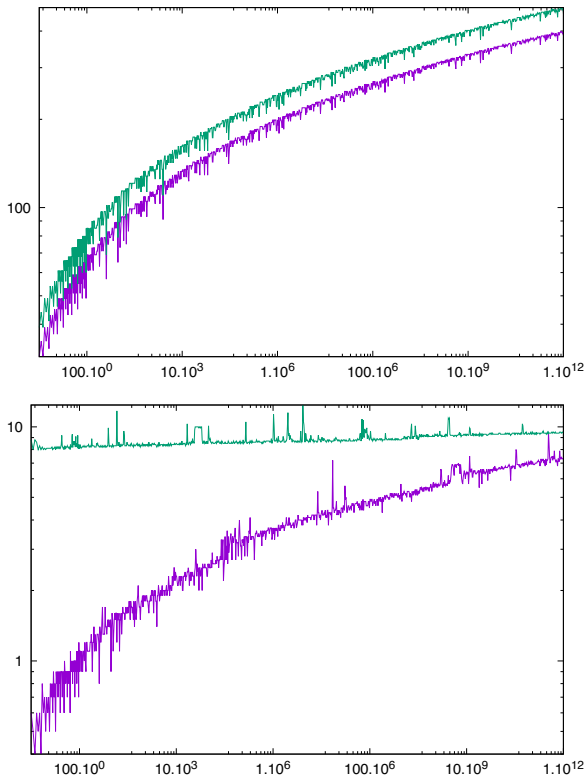


Figure 7. Number of interfaces computed (top plot) and computation time (bottom plot, in ms) for the mass-spring-damper (purple curves) and chain circuit (green curves) models, as functions of parameter n.

chain circuit (Section 2.1, Figure 2) and the mass-spring-damper model (Section 2.2, Figure 3). Models of increasing sizes have be analyzed, up to $n = 10^{12}$.

The experimental results presented below have been obtained with a prototype implementation of the method, based on an enumeration of the selectors. This has a limited impact on performance, for these particular models, since the selector combinatorics is limited to 2 cases for the chain circuit, and 6 cases for the mass-spring-damper model. The key feature of the prototype implementation is the use of a dynamic programming approach for the computation of the interfaces of model tree nodes. For this purpose, a memoization table (see (Cormen et al. 2022), pages 390–392) is used to store the computed interfaces. The software consists in about 10 kLOC of OCaml code and performance has been measured on a MacBook Pro with a 2.4GHz 8-core i-9 Intel processor with 16GB of RAM.

Figure 7 shows the computation times for both models as a function of parameter n. It clearly appears that the empirical time complexity of the method is a logarithmic function of n, like in (Höger 2015). Memory usage is very modest, with about 1MB to store the memoization table.

7 Conclusion and perspectives

In this paper, we present a modular index-reduction method for (possibly incomplete) DAE systems, based on John Pryce's Σ-method (Pryce 2001) and extending the seminal work of Christoph Höger on scalable algorithms for the compilation of Modelica (Höger 2015; Höger 2019). Our method is built upon three key contributions: (i) a concept of *selector* that allows to characterize, from a structural analysis point of view, the possible effects of unknown environments on an incomplete DAE system; (ii) a concept of *interface* for the primal and dual problems of the Σ-method, that encapsulates the minimal information regarding a subsystem that needs to be exposed to its environment in order to perform the structural analysis; and (iii) interface *composition* and *transformation* operators that allow to compute the interface of a system from the interfaces of its parts.

Our modular structural analysis method is well-suited to the Modelica language, since the interface of a class can be computed inductively from the interfaces of the objects contained in the class. Modelica models are often sparse, meaning that each component shares only a few variables with its environment. This guarantees that the interface of a component remains small, independently of the number of components, variables and equations it may contain.

We believe that this concept of interface, and the modular structural analysis method it yields, pave the way towards a genuine separate compilation of Modelica, that can scale up to extremely large models.

The benchmarks performed with our prototype implementation demonstrate that extremely large models, organized in a component tree with sufficient regularity, can be analyzed in a few milliseconds.

Future work shall focus on a more robust implementation of the method, based on our IsamDAE multimode DAE structural analysis software (Benveniste, Caillaud, Malandain, and Thibault 2022; Caillaud, Malandain, and Thibault 2020). We plan to use a functional, BDD-based, representation of interfaces, in order to curb the combinatorics of selectors that is expected when computing interfaces of Modelica classes with a large number of public variables. This functional representation is also a promising approach to the extension of the notion of interface to multimode models.

The structural analysis of Modelica models comprising *for loops* could be done by (i) computing a *tree decomposition* of the component graph obtained from the evaluation of the loops, followed by (ii) the modular structural analysis of the resulting component tree. Benchmarking this approach on the scalable test suite (Casella and Guironnet 2021) would be of great interest.

Several features of the Modelica language, such as *inner/outer* declarations, *expandable* connectors and *over-constrained* connections, may turn out to be difficult to deal with. In the Modelica Language Specification (Modelica Association 2023), the semantics of these features is expressed in terms of an elaboration phase, that is not modular. Devising a modular transformation of these features into the Modelica kernel language is a challenge that needs to be addressed before the modular structural analysis method can be applied to the full Modelica language.

There are also a few fundamental issues, related to the modular structural analysis, that ought to be investigated: (i) Is it possible to decide, by inductive reasoning, whether a parametric model is structurally nonsingular for every valuation of its parameters? (ii) Is it possible to perform the structural analysis of DAE systems that have a large treewidth, such as systems organized as a grid of dimension 2 or higher?

Answers to these questions are key to the design of scalable separate compilation methods for the Modelica language. One could envision the standardization of interfaces, possibly as an extension of the FMI standard for model exchange, so that precompiled Modelica libraries, *equipped with their* Σ*-interfaces,* could be built and reused.

References

Benveniste, Albert, Benoît Caillaud, Hilding Elmqvist, et al. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. Vol. 10000. Lecture Notes in Computer Science. Springer, pp. 283–310. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoît Caillaud, and Mathias Malandain (2022). "From Hybrid Automata to DAE-Based Modeling". In: *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Vol. 13660. Lecture Notes in Computer Science. Springer, pp. 3–20. DOI: 10.1007/978-3-031-22337-2_1.

Benveniste, Albert, Benoît Caillaud, Mathias Malandain, and Joan Thibault (2022). "Algorithms for the Structural Analysis of Multimode Modelica Models". In: *Electronics* 11.17. ISSN: 2079-9292. DOI: 10.3390/electronics11172755. URL: https://www.mdpi.com/2079-9292/11/17/2755.

Braun, Willi, Francesco Casella, and Bernhard Bachmann (2017). "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *Proceedings of the 12th International Modelica Conference*.

Broman, David (2010). "Meta-Languages and Semantics for Equation-Based Modeling and Simulation". PhD thesis. Linköping University, Sweden. URL: https://nbn-resolving.org/urn:nbn:se:liu:diva-58743.

Broman, David (2021). "Interactive Programmatic Modeling". In: *ACM Trans. Embed. Comput. Syst.* 20.4, 33:1–33:26. DOI: 10.1145/3431387.

Broman, David and Peter Fritzson (2008). "Higher-Order Acausal Models". In: *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools, EOOLT 2008, Paphos, Cyprus, July 8, 2008*. Vol. 29. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 59–69. DOI: 10.11128/sne.19.tn.09921.

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

Caillaud, Benoît, Mathias Malandain, and Joan Thibault (2020-04). "Implicit Structural Analysis of Multimode DAE Systems". In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Casella, Francesco and Adrien Guironnet (2021-09). "ScalableTestGrids - An Open-Source and Flexible Benchmark Suite to Assess Modelica Tool Performance on Large-Scale Power System Test Cases". In: *Proceedings of the 14th International Modelica Conference*. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 351–358. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181351.

Cormen, Thomas H et al. (2022). *Introduction to algorithms*. MIT press. ISBN: 9780262046305.

Dassault Systèmes (2002–2023). *Dymola official webpage*. Accessed: 2023-06-12. URL: https://www.3ds.com/products-services/catia/products/dymola/.

Dill, David L. (1989). "Timing Assumptions and Verification of Finite-State Concurrent Systems". In: *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*. Vol. 407. Lecture Notes in Computer Science. Springer, pp. 197–212. DOI: 10.1007/3-540-52148-8_17.

Elmqvist, Hilding (1978). "A structured model language for large continuous systems". PhD thesis. Universiteit i Lund.

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

Höger, Christoph (2015). "Faster Structural Analysis of Differential-Algebraic Equations by Graph Compression". In: *IFAC-PapersOnLine* 48.1. 8th Vienna International Conference on Mathematical Modelling, pp. 135–140. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2015.05.100. URL: https://www.sciencedirect.com/science/article/pii/S2405896315001019.

Höger, Christoph (2019). "Compiling Modelica: about the separate translation of models from Modelica to OCaml and its impact on variable-structure modeling". PhD thesis. TU Berlin. DOI: 0.14279/depositonce-8354.

MathWorks, Inc. (1994–2023). *Simulink official webpage*. Accessed: 2023-06-12. URL: https://www.mathworks.com/products/simulink.html.

Miné, Antoine (2001). "A New Numerical Abstract Domain Based on Difference-Bound Matrices". In: *Programs as Data Objects, Second Symposium, PADO 2001, Aarhus, Denmark, May 21-23, 2001, Proceedings*. Vol. 2053. Lecture Notes in Computer Science. Springer, pp. 155–172. DOI: 10.1007/3-540-44978-7_10.

Modelica Association (2023). *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.6*. Accessed: 2023-06-12. URL: https://modelica.org/documents/MLS.pdf.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Pryce, John D. (2001). "A Simple Structural Analysis Method for DAEs". In: *BIT Numerical Mathematics* 41.2, pp. 364–394. DOI: 10.1023/a:1021998624799.

Rashad, Ramy et al. (2020). "Twenty years of distributed port-Hamiltonian systems: a literature review". In: *IMA J. Math. Control. Inf.* 37.4, pp. 1400–1422. DOI: 10.1093/imamci/dnaa018.

Zimmermann, Pablo, Joaquín Fernández, and Ernesto Kofman (2019). "Set-based graph methods for fast equation sorting in large DAE systems". In: *EOOLT '19: 9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Berlin, Germany, 5 November, 2019*. ACM, pp. 45–54. DOI: 10.1145/3365984.3365991.