Benchmarking the Modular Structural Analysis Algorithm

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

[1]Inria Centre at Rennes University, Campus universitaire de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cedex, France; firstname.lastname@inria.fr

Abstract

In a 2023 Modelica Conference paper, we proposed a novel method for the *modular structural analysis* of DAE systems, in which the structural analysis is not performed on flattened models, but rather at the class level. A new notion of *structural interface* was proposed, in which classes are enriched with context information. That paper developed our approach based on a few illustrative examples.

In this paper, we provide the details of our algorithm. Its performance depends on the system architecture: the analysis of models having a small number of classes (possibly instantiated many times), with a low treewidth system architecture, scales up very efficiently with this approach. We then present additional benchmarks, among which a urban heating network, a representative real-life example on which a near-logarithmic scaling up is shown.

Keywords: DAE, Modelica, object-oriented modeling, structural analysis, index reduction, block-triangular decomposition, interface theory, Difference Bound Matrices

1 Introduction

Modelica advocates an *object-oriented style of programming* (Fritzson 2015), in which subsystem or component models are encapsulated as classes, and instantiated to form large system models. Unfortunately, state of the art structural analysis proceeds by, first, flattening the structured model, and then performing the analysis (Pop et al. 2019). This impedes the storage of model components as executable code or in a format similar to it, which is an crucial feature of compiled programming languages. Additionally, it poses a challenge when attempting to extend DAE-based modeling to higher-order modeling or dynamically changing systems (Broman and Fritzson 2008; Broman 2021). Moreover, as pointed out in (Höger 2015), there are situations in which modeling capabilities are limited by the structural analysis itself, as performance issues arise because of model flattening. Finally, an index-reduction method for DAE systems defined as array equations is proposed in (Otter and Elmqvist 2017). In the same direction, (Fioravanti et al. 2023) proposes an array-aware matching algorithm that scales up to large systems of equations with arrays and loops.

In our previous work (Benveniste et al. 2023) a *structural analysis-aware interface* for components was introduced, thanks to which:

1. structural analysis is performed at the class level;
2. the results of this structural analysis are then instantiated for each component, knowing its context;
3. these component-level structural analyses are composed to derive system-level structural information.

Our approach was presented with the help of a few illustrative examples. In this paper, we provide a formal description of our algorithm. Its performance relates to the following two features of the (otherwise very large) system model: a small number of classes (possibly instantiated many times), and a close-to-tree-shaped system architecture. The urban heating network we present as a benchmark exhibits these features.

The paper is organized as follows. Section 2 provides a background on structural analysis. In Section 4, we introduce Σ-systems, the abstraction of DAE components that this work builds upon, and every function needed to perform the modular structural analysis of models built from these systems. Section 5 presents the core result of this paper, namely, the algorithms used for modular structural analysis. Finally, in Section 6, after addressing implementation details that are key to the performance of the algorithm, we present experimental results obtained from several benchmark models of variable sizes, including an industrial-strength model of the heating network mentioned above.

The full structural analysis also includes the construction of a block-triangular form for the index-reduced system of equations. To simplify our presentation, we ignore this aspect.

2 Background on DAE systems and their structural analysis

We consider square DAE systems, of the form:

$$S \quad : \quad f_j(\text{the } x_i\text{'s and their derivatives}) = 0 \qquad (1)$$

where $X =_{\text{def}} \{x_1, \ldots, x_m\}$ is the set of dependent variables and $F =_{\text{def}} \{f_1, \ldots, f_n\}$ is the set of functions. Throughout this section, we only consider square systems, meaning that $m = n$. Instead of the Pantelides algorithm (Pantelides 1988) for structural index reduction, we will use the Σ-method developed by John Pryce (Pryce 2001). This method is recalled next.

To S as in (1), we associate its (bipartite) *incidence graph* $\mathscr{G}_S = (F \cup X, E)$ having $F \cup X$ as set of vertices, and

having an edge $(f,x) \in E$ if and only if x occurs in function f, regardless of its differentiation degree. To account for differentiations, we augment $\mathscr{G}_S$ to

$$\mathscr{G}_S \quad = \quad (F \cup X, E, \sigma) \tag{2}$$

by adding $\sigma =_{\text{def}} \{\sigma_{f,x} \mid (f,x) \in E\}$, the set of nonnegative integer-valued *weights*, where $\sigma_{f,x}$ is equal to the maximal differentiation degree of variable x in function f. This yields a *weight* for any perfect matching $\mathscr{M}$ of $\mathscr{G}_S$[1] through $\sigma(\mathscr{M}) = \sum_{(f,x)\in\mathscr{M}} \sigma_{f,x}$. The Σ-method consists in solving the following pair of dual linear programming problems:

$$\begin{array}{rll} \text{maximize} & J = \sum_{(f,x)\in E} \sigma_{f,x}\, \xi_{f,x} & \\ \text{subject to} & \sum_{f:(f,x)\in E} \xi_{f,x} = 1 & \forall x \in X \\ \text{and} & \sum_{x:(f,x)\in E} \xi_{f,x} = 1 & \forall f \in F \\ \text{and} & \xi_{f,x} \geq 0 & \forall (f,x) \in E \end{array} \tag{3}$$

$$\begin{array}{rll} \text{minimize} & K = \sum_x d_x - \sum_f c_f & \\ \text{subject to} & d_x - c_f \geq \sigma_{f,x} & \forall (f,x) \in E \\ \text{and} & c_f \geq 0 & \forall f \in F \end{array} \tag{4}$$

Problem (3) is the *primal* and Problem (4) is the *dual*. The former encodes the search for a *maximum weight perfect matching* $\mathscr{M}$ of $\mathscr{G}$, also called *optimal matching* in the sequel. It is proved in (Pryce 2001, Theorem 3.6) that, if a solution exists to the index problem, then a unique element-wise minimal solution for Problem (4) exists. Furthermore, this solution does not depend on the choice of the optimal matching solving the primal problem. The following procedure is proposed in (Pryce 2001) to solve (3,4):

Procedure 1 (Σ-method)

1. *Solve LP* (3)*, which gives an optimal matching* $\mathscr{M}$ *of* $\mathscr{G}$*, described as* $(f,x_f)_{f\in F}$ *or* $(f_x,x)_{x\in X}$*;*

2. *Using* $\mathscr{M}$ *found in step 1, solve for* $(d_x)_{x\in X}$ *the following constraint system:*

$$\begin{array}{rrcl} \forall x \in X: & d_x & \geq & \sigma_{f_x,x} \\ \forall (f,x) \in E: & d_x - d_{x_f} & \geq & \sigma_{f,x} - \sigma_{f,x_f} \end{array} \tag{5}$$

   whereas $c_f = d_{x_f} - \sigma_{f,x_f}$ *yields the equation offsets. Index reduction then consists in differentiating* c_f *times equation* f=0*.*

Notations 1 *We denote by* $\mathscr{L}^{\mathsf{p}}$ *the primal problem* (3) *associated to S, by* $\mathscr{L}^{\mathsf{d}}$ *the problem* (5) *in which matching* $\mathscr{M}=(\xi_{f,x})_{(f,x)\in E}$ *is a solution of primal problem* (3)*, and we write* $\mathscr{L} = (\mathscr{L}^{\mathsf{p}}, \mathscr{L}^{\mathsf{d}})$. □

[1] A perfect matching is a one-to-one assignment of variables to equations, obtained by selecting edges of the incidence graph $\mathscr{G}_S$. The existence of a perfect matching is a criterion for structural nonsingularity.

Using DBMs for the dual problem: In (Pryce 2001), an iterative procedure is proposed to compute the element-wise minimal solution of the dual problem, which is unique. However, handling Problem (5) in a modular way requires additional algebraic manipulations, which are beyond the applicability of the Σ-method. However, we found that Difference Bound Matrices (DBM) (Dill 1989; Miné 2001) turn out to be a subclass of linear programs to which problem (5) belongs.

DBMs are matrices of integer or real coefficients encoding constraint systems like $\mathscr{L}^{\mathsf{d}}$. We generically denote them by A in the sequel. The composition of dual problems $A_i, i = 1,2$ is then encoded by the element-wise minimum of the DBMs: $\mathsf{A}_1 \wedge \mathsf{A}_2$. Eliminating variables from the dual problem is encoded by a matrix denoted by $\Pi_Z(\mathsf{A})$, and the complementary operation (reconstructing after the elimination) is denoted by $\amalg_Z(\mathsf{A})$. Both matrices are easily computed from A.

3 A coupled pendulums model

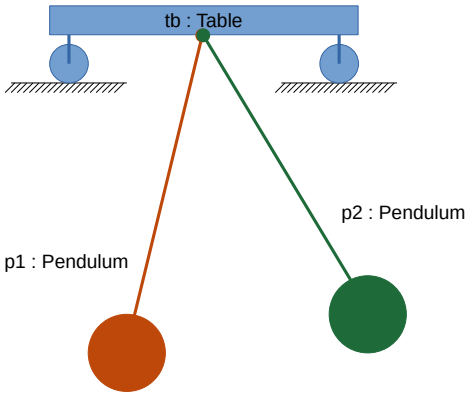


Figure 1. Coupled pendulums, attached to a sliding mass.

Concepts and algorithms introduced presented in the sequel of the paper are illustrated on a model of two 2D coupled pendulums, attached to a mass that can slide on the ground with zero friction, in one dimesion only. The mechanical assembly is depicted Fig. 1, and the Modelica model is defined Fig. 2.

To demonstrate the modularity of the structural analysis method, the model has been as modular as it can be: model `CoupledPendulums` consists of an instance of model `Table`, two instances of model `Pendulum`, and three connecting equations. Two of these connecting equations are trivial, that amount to a variable renaming. Among the three equations, only the third, line 37, a force balance equation, needs to be considered as an equation in the structural analysis. Equations line 9, 26, and 27 are also considered as defining synonyms for the first order derivatives of some state variables. They are not to be considered as equations during the structural analysis.

An important feature of this model is that most variables of models `Table` and `Pendulum` are protected, meaning that they can not be referenced from outside of the model

DOI 10.3384/ecp218175

they belong to. The only variables that remain public are those that are necessary for assembling together the components of the mechanical system, namely: the position of the table `x` and `f`, the projection along the x axis of the forces exerted on the pivot linking the table to each pendulum. As this is explained in the sequel of the paper, this is crucial to the performance of the modular structural analysis.

```
model CoupledPendulums
  model Table
    parameter Real m = 100.;
    public Real x(start=0.,fixed=true);
    protected Real u(start=0.,fixed=true);
    public Real f;
  equation
    m*der(u) - f = 0;
    der(x) = u;
  end Table;
  model Pendulum
    parameter Real m = 1.;
    parameter Real l = 1.;
    parameter Real g = 9.81;
    parameter Real theta0 = 0.;
    public Real x;
    public Real f;
    protected Real a(start=l*sin(theta0),fixed=true);
    protected Real b(start=-l*cos(theta0),fixed=false);
    protected Real u(start=0.,fixed=true);
    protected Real v(start=0.,fixed=true);
    protected Real lambda;
  equation
    m*der(u) - lambda*(a-x);
    m*der(v) - lambda*b + m*g;
    der(a) = u;
    der(b) = v;
    (a-x)^2 + b^2 - l^2 = 0;
    f - lambda*(a-x)/l = 0;
  end Pendulum;
  protected Table tb;
  protected Pendulum p1(theta0=0.1);
  protected Pendulum p2(theta0=0.);
equation
  p1.x = tb.x;
  p2.x = tb.x;
  tb.f + p1.f + p2.f = 0;
end CoupledPendulums;
```

Figure 2. Modelica model of the coupled pendulums.

4 Open DAE systems, Σ-systems, and Σ-interfaces

So far, we focused on closed systems, i.e., systems that operate in isolation. In contrast, open systems are designed to operate in different, yet unspecified environments or contexts. They are the appropriate notion when dealing with modularity. In this section, we introduce open DAE systems and their abstraction for structural analysis, called Σ-systems.

4.1 Open DAE systems and Σ-systems

An *open DAE system* (*oDAE* for short) is a pair $S=(F,X)$, where F is a set of equations $f=0$ and X is a set of variables and their derivatives $x'^k, x \in X$. We require that the cardinalities of these sets satisfy $|X| \geq |F|$. The excess of variables with respect to equations allows capturing interactions with yet unspecified contexts, as shown by the following notion of composition.

Given S_1 and S_2 two oDAEs such that $F_1 \cap F_2 = \emptyset$, their *composition* $S_1 \,\|\, S_2$ is the oDAE $S=(F,X)$ defined by $F = F_1 \uplus F_2$ and $X = X_1 \cup X_2$. Let $S_i, i=1,2$ be two oDAEs. Two solutions $(\mathscr{M}^i, c^i, d^i), i=1,2$ for their respective oDAE index problems are called *complementary* if the union $\mathscr{M}^1 \uplus \mathscr{M}^2$ is a matching, and if variable offsets agree on common variables: $\forall x \in X_1 \cap X_2, d^1_x = d^2_x$. When this holds, matching $\mathscr{M} =_{\text{def}} \mathscr{M}^1 \uplus \mathscr{M}^2$ is equation-complete.

Lemma 1 *Let $S_i, i=1,2$ be two oDAEs, with composition $S = S_1 \,\|\, S_2$. The following properties hold:*

1. *Every solution $(\mathscr{M}, c, d)$ to the oDAE index problem for S projects to a pair of complementary solutions $(\mathscr{M}^i, c^i, d^i), i=1,2$ for the oDAE index problems of $S_i, i=1,2$, where*

$$\mathscr{M}^i = \mathscr{M} \cap (F_i \times X_i),\ c^i = \Pi_{F_i}(c),\ d^i = \Pi_{X_i}(d)\,.$$

2. *Conversely, every pair of complementary solutions $(\mathscr{M}^i, c^i, d^i), i=1,2$ for the oDAE index problems of $S_i, i=1,2$ yields a solution to the oDAE index problem for S by setting*

$$\mathscr{M} = \mathscr{M}^1 \uplus \mathscr{M}^2;\ c_f = c^i_f \text{ if } f \in F_i;\ d_x = d^i_x \text{ if } x \in X_i\,.$$

To account for the fact that the structural analysis of DAEs and oDAEs works on weighted incidence graphs only, we introduce the adequate abstraction for this, called Σ-system. To this end, we distinguish between shared and local variables,[2] by enforcing the following assumption:

Assumption 1 *For $S=(F,X)$ an oDAE, we assume a decomposition $X = X^s \uplus X^\ell$ of its set X of variables into its subsets of* shared *and* local *variables. By convention, local variables of* **S** *are never shared with any other oDAE.*[3]

Local variables are determined locally, by the oDAE equations, whereas shared variables can be determined, depending on the cases, either locally, by the oDAE system, or externally, by the equations of its environment. Interaction with the environment occurs through shared variables only.

Since the structural analysis of oDAE S is built on top of its incidence graph $\mathscr{G}_S$ only, we consider an abstraction of S as $\mathscr{G}_S$, now called a Σ-system:

1. A Σ-*system* is a tuple $\mathbf{S} = (F, X^\ell, X^s, E, \sigma)$, where:

[2] For instance, local variables are declared using the `protected` modifier in Modelica, while shared variables are declared using the default `public` modifier.

[3] We ensure that local variable identifiers are chosen unique, e.g., by performing appropriate renaming to avoid local variable identifier clashes.

- F is a set of functions; X^ℓ and X^s are two disjoint sets of *local* and *shared* variables; we set $X{=}X^s{\uplus}X^\ell$ (the cardinality of F is greater than, or equal to, that of X^ℓ);
- $E \subseteq F \times X$ is the set of *edges*; and
- $\sigma : E \to \mathbb{N}$ is the *weight function*.

2. Given two Σ-systems $\mathbf{S}_1$ and $\mathbf{S}_2$, their *composition* $\mathbf{S}_1 \,\|\, \mathbf{S}_2$ is only defined if $F_1 \cap F_2 = \emptyset$ and yields the Σ-system $\mathbf{S} = (F_1 \uplus F_2, X_1^\ell \uplus X_2^\ell, X_1^s \cup X_2^s, E_1 \uplus E_2, \sigma^1 \uplus \sigma^2)$.

3. Given $x \in X^s$, the *hiding* of x in $\mathbf{S}$ is the Σ-system denoted by $\mathbf{S}{\downarrow_x}$ and obtained by replacing, in $\mathbf{S}$, the pair (X^ℓ, X^s) with $(X^\ell \cup \{x\}, X^s \setminus \{x\})$.

4. Given $x{\in}X^s$ and $y{\notin}X$, the *renaming* of x by y in $\mathbf{S}$ is the Σ-system denoted by $\mathbf{S}[y/x]$ and obtained by replacing x by y everywhere in $\mathbf{S}$.

The combined use of these operators allows to define all Modelica constructs as macros (with the exception of *inner/outer* variable declarations, which imply complex name scoping rules).

Using Section 2, we can associate to a Σ-system $\mathbf{S}$ its oDAE index problem, which, reusing Notations 1, we denote by $\mathscr{L}_\mathbf{S} = (\mathscr{L}_\mathbf{S}^\mathsf{p}, \mathscr{L}_\mathbf{S}^\mathsf{d})$, or simply $\mathscr{L} = (\mathscr{L}^\mathsf{p}, \mathscr{L}^\mathsf{d})$. Its solutions take the form $(\mathscr{M}, c, d)$, where matching $\mathscr{M}$ is equation-complete. The variables matched in $\mathscr{M}$ are determined by the system itself, therefore we call them *dependent variables* and denote the resulting set by X^dep. There are generally several possible choices for $X^\mathsf{dep} \subseteq X$, depending on which variables from X^s are chosen to be part of it.

This choice is parametrized using the notion of selector. Let $\mathbf{S} = (F, X^\ell, X^s, E, \sigma)$ be a Σ-system. Call *selector* any subset $Y \subseteq X^s$, and consider:

- $X^\mathsf{dep} = X^\ell \uplus Y$ and $E^Y = E \cap (F \times X^\mathsf{dep})$;
- σ^Y, the restriction, to E^Y, of the weight function σ;
- $\mathscr{G}^Y = (F \cup X^\mathsf{dep}, E^Y, \sigma^Y)$, the incidence graph; and
- any optimal matching $\mathscr{M}^Y$ for $\mathscr{G}^Y$, if it exists.

Call $\mathscr{M}^Y$ the *selected optimal matching* (SOM) for $\mathbf{S}$ with Y, and *selected optimal weight* its associated weight.

The following result will be instrumental in identifying the appropriate notion of interface required for structural analysis:

Theorem 1 *Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be two Σ-systems, and let $\mathbf{S} = \mathbf{S}_1 \,\|\, \mathbf{S}_2$ be their composition. The following formula holds, where Y is any selector for $\mathbf{S}$ and Y_i is any selector for $\mathbf{S}_i, i{=}1,2$:*

$$\mathsf{J}^Y = \max \left\{ \mathsf{J}_1^{Y_1} + \mathsf{J}_2^{Y_2} \;\middle|\; Y_1 \cap Y_2 = \emptyset \text{ and } Y_1 \cup Y_2 = Y \right\}. \quad (6)$$

We now move to considering the dual problem. For $\mathbf{S}$ a Σ-system and $Y \subseteq X^s$ a selector for it, we denote by $\mathscr{L}^{Y\mathsf{d}}$ the dual problem (5) with $X^\mathsf{dep} = X^\ell \uplus Y$ and $X^\mathsf{free} = X^s \setminus Y$. The following result relates the dual problem of a composition to the dual problems of each of the composed open systems:

Theorem 2 *Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be two open systems and $\mathbf{S} = \mathbf{S}_1 \,\|\, \mathbf{S}_2$ their composition. Denoting by $\mathscr{L}_i^{Y_i\mathsf{d}}, i{=}1,2$ the dual problem of $\mathbf{S}_i$ with selector Y_i, then, for* any *choice of (Y_1, Y_2) achieving the max in (6):*

$$\mathscr{L}^{Y\mathsf{d}} \equiv \mathscr{L}_1^{Y_1\mathsf{d}} \cup \mathscr{L}_2^{Y_2\mathsf{d}}, \quad (7)$$

$$\mathsf{A}^Y = \mathsf{A}_1^Y \wedge \mathsf{A}_2^Y. \quad (8)$$

In (7), the $\cup$ on the right hand side indicates the union of sets of constraints, whereas the symbol $\equiv$ means that the two sides possess identical sets of solutions.

4.2 Σ-interfaces

The Σ-interface is *precisely what is required for interactions with the outside world* — no more, no less. We complement it with the notions of local decomposition and extension, which summarize the missing information in the Σ-interfaceneeded to reconstruct solutions to the structural analysis of the Σ-system.

Notations 2 *In the sequel, given a relation $R \subseteq W \times W$ or a function $f : W \to Z$, and $V \subseteq W$, we denote by*

$$\Pi_V(R) \text{ and } \Pi_V(f)$$

the restriction *of R to $V \times V$ and of f to V.*[4] *We will use this notation with some liberty. For $Y \subseteq X^s$ a selector, we will write $\Pi_Y(\mathsf{J})$ instead of $\Pi_{\mathscr{P}(Y)}(\mathsf{J})$ to mean the restriction of function J to $\mathscr{P}(Y)$, the set of all subsets of Y.*

For the following definition, we reuse the DBM associated to the dual problem introduced in Section 2.

Definition 1 (interface) *To Σ-system $\mathbf{S} = (F, X^\ell, X^s, E, \sigma)$, we associate its* Σ-interface

$$\mathbf{I}_\mathbf{S} = \left(X^s, \mathsf{J}, \Pi_{X^s}(\mathsf{A})\right), \quad (9)$$

where J and $\Pi_{X^s}(\mathsf{A})$ are functions that respectively map any selector $Y \subseteq X^s$ to:

- J^Y*, the optimal weight of $\mathbf{S}$ for the primal problem with selector Y;*
- $\Pi_{X^s}(\mathsf{A}^Y)$*, the projection, over X^s, of the DBM A^Y associated to the dual problem with selector Y.*

In our modular algorithm, we use the following function:

$$\text{INTERFACECOINTERFACE}(\mathbf{S}) = \left(\mathbf{I}_\mathbf{S}, \amalg_{X^s}(\mathsf{A})\right) \quad (10)$$

[4] Our notation is reminiscent of a projection, on purpose.

DOI 10.3384/ecp218175

where $\amalg_{X^s}(\mathsf{A})$ is a function that maps each selector $Y \subseteq X^s$ to $\amalg_{X^s}(\mathsf{A}^Y)$, the extension of A^Y from X^s. Function (10) splits $\mathbf{S}$ as its Σ-interface $\mathbf{I_S}$ and an object $\amalg_{X^s}(\mathsf{A}^Y)$ that makes up for the loss of information in $\mathbf{I_S}$. The latter is stored for subsequent reuse when "reverting the Σ-interface", as we shall see.

To illustrate the concept of Σ-interface, let us consider model `Table` (Fig. 2). It has two public variable, `x` and `f`. Protected variable `u` (line 5) should be seen as a synomym for the first order derivative of `x` (equation line 9). This means that for the structural analysis, this model has two variables and one equation (line 8). As a consequence, the interface $I_t = (\{x,f\}, J_t, A_t)$ of this model admits two valid selectors: $\{x\}$ and $\{f\}$. The optimal weight J_t and the DBM A_t are given in the table below, for each of the two valid selectors:

Y	J_t	A_t
$\{f\}$	0	$A_t^{\{f\}}$
$\{x\}$	2	$A_t^{\{x\}}$

where:

$$A_t^{\{f\}} = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} & -2 & 0 \\ & & \\ & -2 & 0 \end{bmatrix} \end{array} \qquad A_t^{\{x\}} = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} & -2 & 0 \\ & 0 & 2 \\ & & \end{bmatrix} \end{array}$$

Matrix elements that are left blank have the value $+\infty$, and impose no constraint on the corresponding offset variables. Therefore, DBM $A_{\{f\}}$ is equivalent to the system of inequalities $d_x \geq 2, d_f \geq 0, d_x \geq 2 + d_f$. The second DBM $A_{\{x\}}$ encodes $d_x \geq 2, d_f \geq 0, d_f \geq d_x - 2$.

Theorem 3 (hiding) *Let* $\mathbf{S}$ *be a* Σ*-system, with* Σ*-interface* (9). *Let* $\mathbf{T} =_{\text{def}} \mathbf{S}{\downarrow_Z}$*, where* $Z \subseteq X^s$*, and define* $Z^s = X^s \backslash Z$*. Then, the* Σ*-interface of* $\mathbf{T}$ *is*

$$\mathbf{I_T} = \left(Z^s, \Pi_{Z^s}(\mathsf{J}), \Pi_{Z^s}(\mathsf{A})\right),$$

where $\Pi_{Z^s}(\mathsf{J})$ *and* $\Pi_{Z^s}(\mathsf{A})$ *are functions mapping any selector* $Y \subseteq Z^s$ *to* $\Pi_{Z^s}(\mathsf{J}^Y)$ *and* $\Pi_{Z^s}(\Pi_{X^s}(\mathsf{A}^Y))$.

As a direct consequence of the properties of the projection, note that, for every selector Y,

$$\Pi_{Z^s}(\mathsf{A}^Y) \quad = \quad \Pi_{Z^s}(\Pi_{X^s}(\mathsf{A}^Y)) \tag{11}$$

In our modular algorithm, we will use the following function, where $\mathbf{I_S}$ is as in (9):

$$\text{HIDEEXPOSE}(\mathbf{I_S}, Z) = \left(\mathbf{I_T}, \amalg_Z(\mathsf{A})\right) \tag{12}$$

In (12), $\amalg_Z(\mathsf{A})$ is the function mapping each selector $Y \subseteq X^s$, to $\amalg_{Z^s}(\mathsf{A}^Y)$, the extension of A^Y from Z^s. Function (12) replaces the pair $(\mathbf{I_S}, Z)$ by its Σ-interface $\mathbf{I_T}$, and object $\amalg_{Z^s}(\mathsf{A}^Y)$ that makes up for the loss of information in $\mathbf{I_T}$. The latter is stored for subsequent reuse when "reverting the hiding".

Theorem 3 and (11) show that HIDEEXPOSE can be evaluated through the Σ-interfaces $\mathbf{I_S}$ and $\mathbf{I_T}$ only.

Theorem 4 (composition) *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be two* Σ*-systems, with* Σ*-interfaces* $\mathbf{I}_{\mathbf{S}_i} = (X_i^s, \mathsf{J}_i, \mathsf{A}_i), i = 1,2$.

1. *We have* $\mathbf{I}_{\mathbf{S}_1 \| \mathbf{S}_2} = (X^s, \mathsf{J}, \mathsf{A})$*, where:*

$$\begin{aligned} X^s &= X_1^s \cup X_2^s, \text{ and, for } Y \subseteq X^s, \\ \mathsf{J}^Y &= \max \left\{ \mathsf{J}_1^{Y_1} + \mathsf{J}_2^{Y_2} \;\middle|\; \begin{array}{l} Y_i \subseteq X_i^s, i{=}1,2 \\ Y_1 \cap Y_2 = \emptyset \\ Y_1 \cup Y_2 = Y \end{array} \right\} \qquad (13) \\ \mathsf{A}^Y &= \mathsf{A}_1^{Y_1} \wedge \mathsf{A}_2^{Y_2} \qquad (14) \end{aligned}$$

2. *The resulting* A^Y *in* (14) *is independent from the chosen optimizing pair.*

The right-hand sides of (13, 14) depend only on the Σ-interfaces of $\mathbf{S}_1$ and $\mathbf{S}_2$. Hence, we will denote by $\mathbf{I_{S_1}} \| \mathbf{I_{S_2}}$ the so defined operation on Σ-interfaces.

Call *optimizing pair* the function $Y \mapsto (Y_1, Y_2)$, mapping any Y to a pair (Y_1, Y_2) achieving the maximum in (13). In our modular algorithm, we use the following function:

$$\text{COMPOSEDECOMPOSE}(\mathbf{I_{S_1}}, \mathbf{I_{S_2}}) = \begin{pmatrix} \mathbf{I_{S_1}} \| \mathbf{I_{S_2}} \\ Y \mapsto (Y_1, Y_2) \end{pmatrix} \tag{15}$$

computing the optimizing pair as its second component, for subsequent reuse when "reverting the composition".

To illustrate the composition operator, let us consider the composition of model `Table` with the equation "`der(f) + x = 0`". The interface of the former is given above, while the latter has for interface $I_e = (\{x,f\}, J_e, A_e)$, where J_e and A_e are defined as follows:

Y	J_e	A_e
$\{f\}$	1	$A_e^{\{f\}}$
$\{x\}$	0	$A_e^{\{x\}}$

where:

$$A_e^{\{f\}} = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} & 0 & -1 \\ & & \\ & 1 & 0 \end{bmatrix} \end{array} \qquad A_e^{\{x\}} = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} & 0 & -1 \\ & 0 & -1 \\ & & \end{bmatrix} \end{array}$$

The composition $(I_t \parallel I_e) = (\{x,f\}, J, A)$ is as follows: it admits only one valid selector $Y = \{x,f\}$, which optimal partitioning is $Y_t = \{x\}$ and $Y_e = \{f\}$. This gives a weight $J^Y = J_t^{Y_t} + J_e^{Y_e} = 3$. DBM A^Y is the pointwise minimum of $A_t^{Y_t}$ and $A_e^{Y_e}$:

$$A^Y = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} & -2 & -1 \\ & 0 & 2 \\ & 1 & 0 \end{bmatrix} \end{array}$$

4.3 Reverting the Σ-interface

In this section, we formalize what was meant above by "reverting" interfaces, hiding, and composition.

Consider a Σ-system $\mathbf{S} = (F, X^\ell, X^s, E, \sigma)$. Call Σ-*solution of* $\mathbf{S}$ any tuple

$$\nu \quad = \quad \big(Y, (d_x)_{x\in X}, (c_f)_{f\in F}\big)\,, \tag{16}$$

where $Y \subseteq X^s$ is a chosen selector such that $\mathsf{J}^Y > -\infty$, and variable offsets $(d_x)_{x\in X}$ are solution of A^Y, whereas equation offsets $(c_f)_{f\in F}$ are computed as in Procedure 1. We denote by $\mathsf{sol}(\mathbf{S})$ the set of all Σ-solutions of $\mathbf{S}$. We equip $\mathsf{sol}(\mathbf{S})$ with the partial order $\leq$ defined by

$$\nu \leq \nu' \quad \text{iff} \quad Y = Y' \text{ and } \forall x \in X : d_x^Y \leq d_x'^Y\,. \tag{17}$$

The term "minimal" will be referring to this order.

Let $\mathbf{I}$ be a Σ-interface following (9). Call *local solution* of $\mathbf{I}$ any tuple

$$\mu \quad = \quad \big(Y, (d_x)_{x\in X^s}\big)\,, \tag{18}$$

where $Y \subseteq X^s$ is a chosen selector such that $\mathsf{J}^Y > -\infty$, and variable offsets $(d_x)_{x\in X^s}$ are solution of $\Pi_{X^s}(\mathsf{A}^Y)$. Minimality is defined as in (17).

Nota: Σ-solutions are the wanted solutions of the index reduction problem. They are large objects since their cardinality relates to the size of X. In contrast, local solutions are small objects, whose cardinality relates to the size of X^s. Local solutions are the minimal objects from which Σ-solutions for the considered Σ-system can be rebuilt. Our overall objective is to compute Σ-solutions in a modular way, by manipulating only local solutions.

When composing Σ-interfaces, their local solutions fuse into local solutions of the composed interface. Local decomposition is the reverse operation.

Theorem 5 (local decomposition) *Let $\mathbf{I}_i, i{=}1,2$ be two Σ-interfaces and let $\mathbf{I}{=}\mathbf{I}_1 \,\|\, \mathbf{I}_2$ be their composition.*

1. *Let $\mu_i, i{=}1,2$ be a local solution of $\mathbf{I}_i, i{=}1,2$. Say that μ_1 and μ_2 are* compatible *if, with reference to Theorem 4:*

   (a) (Y_1, Y_2) is the optimizing pair for $Y{=}Y_1{\cup}Y_2$, and

   (b) for every $x, y \in X_1^s{\cap}X_2^s$, we have: $d_{1,x} = d_{2,x}$.

   If μ_1 and μ_2 are compatible, we can define their join

$$\begin{array}{rcl} \mu_1 \sqcup \mu_2 & =_{\text{def}} & (Y, (d_x)_{x\in X^s})\,, \text{ where:} \\ Y & = & Y_1 \cup Y_2 \\ d_x & = & \text{if } x \in X_1^s \text{ then } d_{1,x} \text{ else } d_{2,x} \end{array} \tag{19}$$

2. *Conversely, any Σ-solution μ of $\mathbf{I}{=}\mathbf{I}_1 \,\|\, \mathbf{I}_2$* decomposes *as $\mu = \mu_1 \sqcup \mu_2$, where μ_i is a Σ-solution of $\mathbf{I}_i$ and*

   (a) Y decomposes as $Y = Y_1{\cup}Y_2$, where (Y_1, Y_2) is the pair returned by the function COMPOSEDECOMPOSE *defined in (15), and*

   (b) $(d_{i,x})_{x\in X_i^s}$ is the restriction of $(d_x)_{x\in X^s}$ to X_i^s.

The important step in this decomposition is 2a. Step 2b is just a restriction and does not need any side information.

Nota: The decomposition $\mu = \mu_1 \sqcup \mu_2$ may not be unique, since optimizing pairs of selectors can be several. However, step 2b always yields the same result.

We now focus on how to revert the effect of hiding on Σ-interfaces. We use macro (12) together with objects and notations of Lemma 3.

Theorem 6 *Let $\mathbf{I}$ be a Σ-interface as in (9), and let $Z \subseteq X^s$ and $Z^s = X^s \setminus Z$. Consider the interface $\mathbf{J}$ of the hiding $\mathbf{I}{\downarrow_Z}$.*

1. *Let $\mu = (Y, (d_x)_{x\in X^s}, \preceq)$ be a local solution of $\mathbf{I}$ such that $Y \cap Z = \emptyset$. The* hiding of Z in μ, *defined by*

$$\mu{\downarrow_Z} \quad = \quad (Y, (d_x)_{x\in Z^s})\,, \tag{20}$$

   is a local solution of $\mathbf{J}$ and every local solution of $\mathbf{J}$ is obtained in this way. Say that $\mu{\downarrow_Z}$ extends to *μ.*

2. *Conversely, every local solution μ of $\mathbf{J}$ extends to a unique local solution $\mu^\star$ of $\mathbf{I}$, which is minimal among the set of all local solutions of $\mathbf{I}$ extending μ. This minimal extension is computed by using only $\amalg_Z(\mathsf{A})$, stored as the second component of the function* HIDEEXPOSE *introduced in* (12).

Theorem 7 *Let $\mathbf{S}$ be a basic Σ-system with Σ-interface $\mathbf{I}_\mathbf{S}$, and let $\nu = \big(Y, (d_x)_{x\in X}, (c_f)_{f\in F}\big)$ be a Σ-solution of $\mathbf{S}$.*

1. *The* hiding of X^ℓ in ν, *defined by*

$$\nu{\downarrow_{X^\ell}} \quad = \quad (Y, (d_x)_{x\in X^s})\,,$$

   is a local solution of $\mathbf{I}_\mathbf{S}$ and every local solution of $\mathbf{I}_\mathbf{S}$ is obtained in this way. Say that $\nu{\downarrow_{X^\ell}}$ extends to *ν.*

2. *Conversely, every local solution μ of $\mathbf{I}_\mathbf{S}$ extends to a unique Σ-solution $\nu = \mu^\star$ of $\mathbf{S}$, which is minimal among the set of all Σ-solutions of $\mathbf{S}$ extending μ. This minimal extension is computed by using only the pair $\amalg_{X^s}(\mathsf{A})$, which was stored as the second component of the function* INTERFACECOINTERFACE *in* (10).

This completes the toolbox for computing a Σ-solution of the Σ-system in a modular way. With the exception of basic Σ-systems, only shared variables are involved. Hence, these concepts remain as small as the set of shared variables of the associated Σ-systems.[5]

[5] Referring to the Modelica taxonomy, we expect all programs to have a set of *public* variables of small cardinality. In addition, we expect basic classes to possess a small set of variables.

DOI 10.3384/ecp218175

5 Modular algorithm

This section contains the most significant and practical contribution of this work, namely: a term algebra for the modular definition of Σ-systems, and a modular algorithm performing their structural analysis.

While the structural interface theory developed in Section 4.2 provides the needed pillars, it is not sufficient to construct an efficient and scalable algorithm, taking advantage of certain assumptions regarding the DAE systems we target.

Indeed, the large DAE systems that are typically seen in the digital twins of industrial systems are sparse. In addition, they generally result from a composition of a large number of instances of basic models in much smaller number, e.g., taken from libraries. Said differently, our targeted DAE systems contain a large number of structurally identical components, in which only parameter values may differ.

Our modular algorithm ensures that structural analyses are computed only once for each representative component, and then properly fused to get the global analysis. Achieving this relies on the interface theory developed in Section 4.2, complemented with adequate algorithmic techniques and data structures that we explain in this section.

Algorithm 1 Upward sweep: computing interfaces and cointerfaces with memoization

global $\mathscr{I}C : \mathscr{A} \to \mathscr{I} \times \mathscr{C}$ ▷ memoization table
▷ initially empty
function INTERFACE(a)
  if $\mathscr{I}C(a)$ is defined **then** ▷ if already memoized
    $(i,\cdot) \leftarrow \mathscr{I}C(a)$ ▷ then retrieve interface
  else
    if $a = (a_1 \parallel a_2)$ **then** ▷ composition
      $i_1 \leftarrow$ INTERFACE(a_1) ▷ compute interfaces
      $i_2 \leftarrow$ INTERFACE(a_2) ▷ of operands
      $(i,c) \leftarrow$ COMPOSEDECOMPOSE(i_1,i_2)
    else if $a = a'\downarrow_x$ **then** ▷ hiding
      $i' \leftarrow$ INTERFACE(a') ▷ interface of operand
      $(i,c) \leftarrow$ HIDEEXPOSE(i',x) ▷ compute hiding
    else if $a = [y/x]a'$ **then** ▷ renaming
      $i' \leftarrow$ INTERFACE(a') ▷ interface of operand
      $(i,c) \leftarrow$ RENAMEREVERSE(i',y,x)
    else if $a = b$ **then** ▷ basic Σ-system
      $(i,c) \leftarrow$ INTERFACECOINTERFACE(S_b)
    end if
    $\mathscr{I}C(a) \leftarrow (i,c)$ ▷ memoize (co-)interface
  end if
  return i ▷ return interface
end function

Algorithm 2 Downward sweep: computing local solutions

global $\mathscr{I}C$ ▷ initialized by function INTERFACE
  $\mathscr{S}ol \subseteq \mathscr{A} \times \mathscr{S}$ ▷ set of local solutions
  ▷ initially empty
procedure SOLUTION(a,μ)
  if $(a,\mu) \notin \mathscr{S}ol$ **then** ▷ visited yet ?
    add (a,μ) to $\mathscr{S}ol$ ▷ mark visited
    $(\cdot,c) \leftarrow \mathscr{I}C(a)$ ▷ retrieve cointerface
    if $a = (a_1 \parallel a_2)$ **then** ▷ composition
      $(\mu_1,\mu_2) \leftarrow c(\mu)$ ▷ compute local solutions
      ▷ of both operands
      SOLUTION(a_1,μ_1) ▷ propagate local solutions
      SOLUTION(a_2,μ_2) ▷ to the operands
    else if $a = a'\downarrow_x$ **then** ▷ hiding
      $\mu' \leftarrow c(\mu)$ ▷ compute local solution
      ▷ of the operand
      SOLUTION(a',μ') ▷ propagate to operand
    else if $a = [y/x]a'$ **then** ▷ renaming
      $\mu' \leftarrow c(\mu)$ ▷ compute renamed solution
      SOLUTION(a',μ') ▷ propagate to operand
    else if $a = b$ **then** ▷ basic Σ-system
      $\nu \leftarrow c(\mu)$ ▷ compute Σ-solution
      generate reduced index system of
      S_b for Σ-solution ν
    end if
  end if
end procedure

5.1 A term algebra $\mathscr{A}$ of Σ-systems

In this section, we consider a term algebra of Σ-systems, defined by the following grammar:

$$\mathscr{A} \quad ::= \quad \mathscr{B} \mid (\mathscr{A} \parallel \mathscr{A}) \mid \mathscr{A}\downarrow_{\mathscr{X}} \mid \mathscr{A}[\mathscr{X}/\mathscr{X}] \tag{21}$$

where $\mathscr{B}$ is a predefined set of symbols b denoting basic Σ-system S_b, and $\mathscr{X}$ is a set of variable names x, y, or z. The semantics of $\mathscr{A}$ is a mapping that associates a Σ-system $[\![a]\!]$ to term a. It is defined inductively on the term:

$$\begin{aligned} [\![b]\!] &= S_b \\ [\![(a_1 \parallel a_2)]\!] &= \rho_1[\![a_1]\!] \parallel \rho_2[\![a_2]\!] \\ [\![a\downarrow_x]\!] &= [\![a]\!]\downarrow_x \\ [\![a[y/x]]\!] &= (\rho[\![a]\!])[y/x] \end{aligned} \tag{22}$$

where ρ_k, $k = 1,2$ are any injective renamings of the vertices of $[\![a_k]\!] = (F_k, X_k^\ell \uplus X_k^s, E_k, \sigma_k)$, $k = 1,2$, such that $F_1 \cap F_2 = \emptyset$, $X_1^\ell \cap X_2^\ell = \emptyset$, and ρ_k, $k = 1,2$ are identity functions on X_k^s. Renaming ρ is any mapping such that it is the identity on the shared variables of $[\![a]\!]$ and y is not a variable of $\rho[\![a]\!]$. Our central problem is the following:

Problem 1 *How can one perform the structural analysis of $[\![a]\!]$ without computing $[\![a]\!]$, that is, by considering only term a and the Σ-systems S_b of the basic terms b appearing in a?*

A solution to this problem is developed in this section. The following notations are used in the sequel:

Notations 3 *In the sequel, symbols a, a', a_1, a_2 are use to represent terms; i, i', i_1, i_2 denote Σ-interfaces; c denotes cointerfaces and μ, μ', μ_1, μ_2 represent local solutions, while symbol ν is used to represent Σ-solutions. Also, we will use the generic term of* Cointerface *to denote the complementary components computed by the functions* INTERFACECOINTERFACE, HIDEEXPOSE, *and* COMPOSEDECOMPOSE, *introduced in (10,12,15).* □

5.2 A modular algorithm for the structural analysis of $\mathscr{A}$

In this section, we propose an algorithm solving Problem 1. This algorithm consists in traversing the sub-term graph, by first performing an *upward sweep* using Algorithm 1, followed by a *downward sweep* using Algorithm 2, which we comment next.

Upward sweep Algorithm 1 Σ-interfaces are computed inductively, starting from basic Σ-systems. A *memoization* table $\mathscr{I}C: \mathscr{A} \to \mathscr{I} \times \mathscr{C}$ is used to store the local interface and cointerface for each sub-term. Memoization (Cormen et al. 2022, Section 14.3) is an algorithmic technique that consists in ensuring that a function $f(x)$ is evaluated at most once for each parameter value x. This relies on a data structure that stores $f(x)$ for each value x encountered so far during program execution. Table $\mathscr{I}C$ has a double purpose: it is both used to prevent the evaluation of function INTERFACE(a) whenever this has already been done, and to store the local cointerface associated to a term. This cointerface is used in the downward sweep.

Downward sweep Algorithm 2 Local solutions are computed for each sub-term. In procedure SOLUTION(a, μ), a is a term and μ a local solution, seen as its context. This procedure computes the local solutions of the sub-terms of a, in the context of μ. To do this, the local cointerface c of term a, computed during the first phase, is retrieved from table $\mathscr{I}C$.

In the sequel, we call *modular structural analysis* the successive execution of Algorithms 1 and 2.

5.3 Back to the coupled pendulums example

The coupled pendulums model, introduced Section 3, can be expressed as the following component algebra term:

$$
\begin{aligned}
\text{CoupledPendulums} \;=\; & (\,\text{Table}[tb.f/f] \parallel \\
& \quad (\,\text{ConnectingEquation} \parallel \\
& \qquad \text{Pendulum}[p1.f/f] \parallel \\
& \qquad \text{Pendulum}[p2.f/f] \\
& \quad)\downarrow_{p1.f,p2.f} \\
&)\downarrow_{x,tb.f}
\end{aligned}
$$

where Table and Pendulum are the Σ-systems obtained from the classes `Table` and `Pendulum` defined Fig. 2, and ConnectingEquation is the Σ-system obtained from the connecting equation, line 37.

The execution of Algorithm 1 is detailed Fig. 9. The figure displays the term as an acyclic directed graph where each node represents a subterm of term CoupledPendulums, and each edge represents the containment relation. The algorithm starts with basic Σ-systems, and propagates Σ-interfaces in the opposite direction of the edges. Each time the Σ-interface of a term is computed, its corresponding cointerface is stored in the memoization table.

Figure 10 details the downward sweep of Algorithm 2: partial Σ-solutions are propagated down the subterm graph. The end result consists of Σ-solutions for each basic Σ-system instance, that can be used to perform the index reduction of each of these oDAE systems.

6 Implementation and benchmarking

In this section, we address an important implementation aspect of our modular method, then we benchmark our prototype implementation on several examples: a vibrating string model, an industrial-strength thermofluid model of a urban heat network, and a Heat Equation model defined with arrays and a *for* loop.

6.1 Avoiding the combinatorial explosion due to selectors

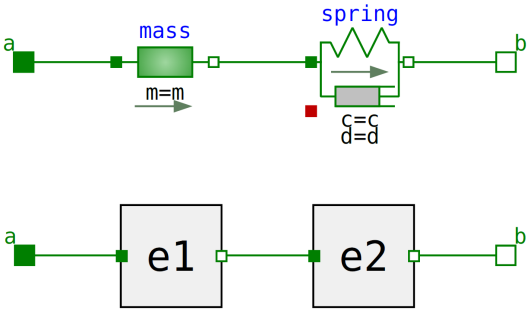
A difficulty in our interface theory is the consideration of selector-dependent structural analysis problems. The primal and dual problems are functions of the selector, that can be numerous, except for systems with only a few public variables. With a mere enumeration of selectors, composition operators can result in an exponential growth of the number of valid selectors.

Indeed, the number of valid selectors usually remains rather small, as the maximum number of shared variables for all the subsystems of a given system is small (below 20 or so). In such cases, enumerating valid selectors is not costly. It is however possible that, for some subsystems, the set of shared variables is larger, and enumerating selectors may become too costly. In Modelica, this typically happens whenever a class has public variables of array types.

In such cases, the set of valid selectors can be represented by its Boolean *characteristic function*. The primal and dual interfaces can also be represented as tuples of Boolean functions In this approach, efficient representations exist, namely Binary Decision Diagrams and their extensions; we use the Reduced-Ordered variant (ROBDD) introduced by (Bryant 1986). In our implementation, we adopted a hybrid approach, where selectors are enumerated for systems with a number of public variables below a given threshold, while the characteristic function approach is used when this threshold is exceeded (Benveniste et al. 2022).

The experimental comparison of the enumerative and symbolic approaches, presented in Section 6.2, turns in

DOI 10.3384/ecp218175




```
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.

favor of the enumerative approach. Nevertheless, the symbolic approach should not be ruled out, as subsystems with a large number of shared variables may occur, and the enumerative approach would fail on these particular instances.

6.2 Benchmarking

Our implementation has been benchmarked on two academic examples and an industrial strength model. The former are a vibrating string model, and a 1-D finite element model of the Heat Equation; the latter is an industrial-strength model of a urban heat network system. All three models have been made parametric so that one can adjust their size, in terms of numbers of equations and variables. For these examples, the performance of the modular structural analysis is compared with a global reference implementation of the Σ-method.

6.2.1 The Vibrating String example

This example is a lumped model of a vibrating string, consisting in a chain of mass-spring-damper elements. The Modelica diagrams and code of this model are given by Figure 3. The mass-spring-damper element is built using the translational mechanics package of the Modelica Standard Library (MSL). The chain is defined by dichotomy, using a recursive class. , exactly like the chain circuit example.

The performance of the modular method was compared with a classical implementation of the Σ-method, where the primal problem is solved using an implementation of the Hungarian method (Munkres 1957), the dual problem is solved by a fixed point iteration, as advocated in (Pryce 2001). This reference implementation also puts the equations in Block Triangular Form (BTF), based on the computation Strongly Connected Components, using Tarjan's

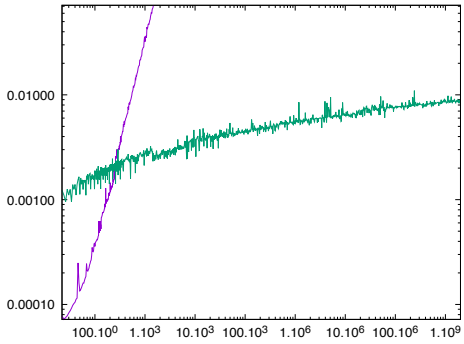


Figure 4. CPU times (seconds) of the modular (in green) and global (in purple) methods for the Vibrating String, as a function of the number of equations.

algorithm (Tarjan 1972). Figure 4 gives the computation times of both methods, for a range of instances of Vibrating String model, up to about 10^9 equations. The computation times for the modular method are logarithmic in the number of equations, because the number of interfaces to be computed is logarithmic in the model parameter value. The modular method runs faster than the global one, except for small models with less than a few hundred equations.

6.2.2 The Urban Heat Network model

This is an industrial-strength thermofluidic model of a urban heat network system, inspired by (Mans et al. 2022). The complete network comprises hot and cold water pipes, heat exchangers and circulation pumps, expansion tanks, and heat sources and sinks. Pipes and heat exchangers are lumped models, consisting in several elements connected in series. Figure 5 shows the variables-to-components incidence graph of the model for parameter value $n = 10$, and

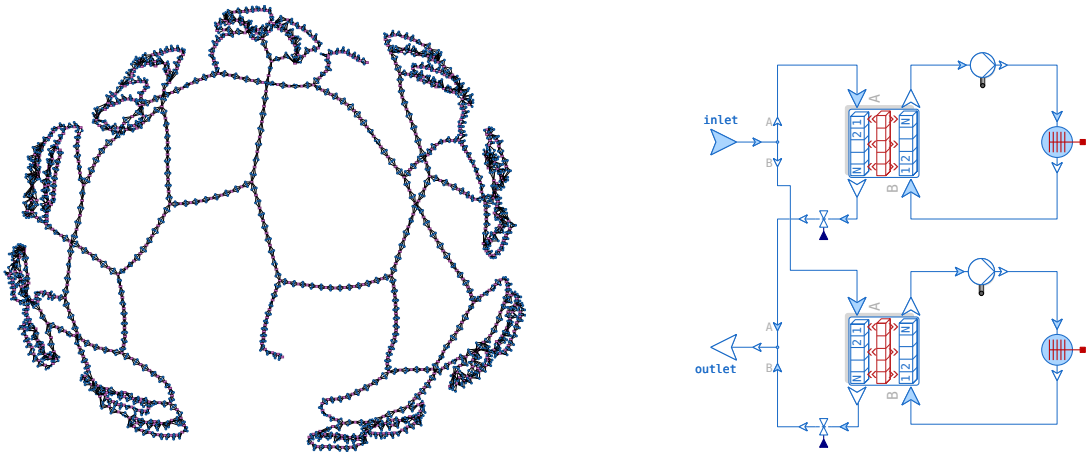


Figure 5. (a) Variables-to-components graph of the Urban Heat Network model for $n = 10$. Blue circles represent variables that are shared between two components; purple boxes represent instances of the basic thermofluidic components. (b) Modelica model of a substation, instantiated n times in the model and interconnected with a network of pipes.

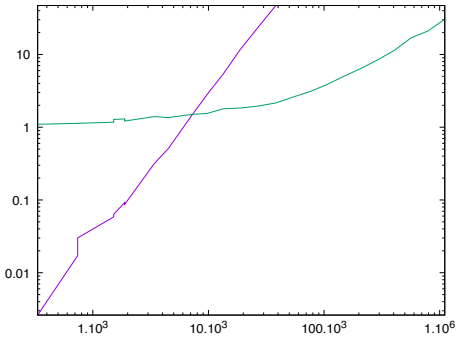


Figure 6. CPU times (seconds) of the modular (in green) and global (in purple) methods, as a function of the number of equations.

```
model Thermal1D
  parameter Integer N = 5;
  parameter Real g = 0.00314785; // W/K
  parameter Real c = 0.2707936; // J/K
  parameter Real Tleft = 400 + 273.15; // K
  parameter Real Tright = 20 + 273.15; // K
  Real T[N](each start=Tright);
equation
  c*der(T[1]) = g * (2*Tleft - 3*T[1] + T[2]);
  for i in 2:N-1 loop
    c*der(T[i]) = g*(T[i-1] - 2*T[i] + T[i+1]);
  end for;
  c*der(T)[N] = g*(T[N-1] - 3*T(N) + 2*Tright);
end Thermal1D;
```

Figure 7. Modelica code of the 1-D Heat Equation, with an array and a loop of equations.

the Modelica model of a substation, that is instantiated n times and connected to a tree-shaped network of hot and cold water pipes. Figure 6 gives the computation times of the global and modular methods for instances of this model up to about 10^6 equations.

Although its measured time complexity is not logarithmic in the size of the model, the modular method runs faster than the global method for instances of the model with more than about 7,000 equations. This is due to the fact that the model has a low treewidth (Bodlaender 1988), as evidenced by Figure 5a.

6.2.3 The Heat Equation model

Figure 7 shows the Modelica code of a finite element model of the 1D Heat Equation. This model, taken from (Fioravanti et al. 2023), consists of an array of real variables and a set of equations defined thanks to a *for* loop. This is a challenging benchmark for two reasons. First, it is a system of ordinary differential equations, on which each of the three steps of the global structural analysis method (primal and dual problems, then BTF decomposition) has a computation time that is linear in the size of the model. Second, this model does not provide a class instantiation tree decomposition of the system of equations; this decomposition has to be defined manually, or computed automatically.

As a reference, we performed the structural analysis using a manually-defined balanced binary tree decomposition of the system of equations, very similar to that of the Vibrating String example. Unsurprisingly, the performances (Figure 8, top graph) show that, above a certain threshold in the size of the system, the modular structural analysis takes only a fraction of the computation time required for the global (classical) structural analysis. As a matter of fact, it should be noticed that the empirical time complexity is logarithmic in the number of equations.

To our knowledge, computing an almost-balanced tree decomposition, that minimizes the number of shared variables between subtrees, is an open problem, in the general case of (possibly nested) loops of equations. It is nevertheless possible to compute tree decompositions, not necessarily balanced, using the heuristics implemented in Snowflake (Thibault 2024; Thibault 2022). Because

DOI 10.3384/ecp218175

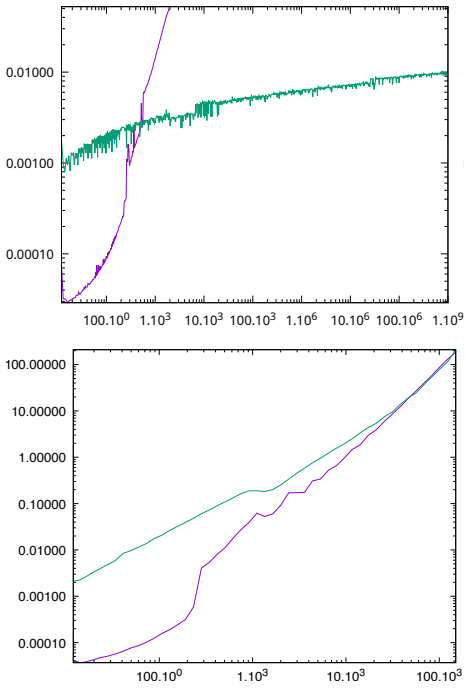


Figure 8. CPU times (seconds) of the modular (in green) and global (in purple) methods for the Heat Equation model, as a function of the number of equations, with balanced (top graph) and unbalanced (bottom graph) tree decompositions.

Snowflake implements a greedy decomposition algorithm, it results in an unbalanced, comb-shaped tree. The performances obtained with this tree decomposition are shown in the bottom graph of Figure 8. Although the modular structural analysis also has a linear time complexity for this model, it can be observed that, for large instances of the model, the computation times of the global and modular methods are almost the same. This examplifies the importance of a balanced tree decomposition for the performance of the modular method.

7 Conclusion and perspectives

Only square and structurally nonsingular DAE models are suitable for simulation. Such models correspond to *closed* systems, considered in isolation. In contrast, *open component models*, possessing more variables than equations, constitute the adequate framework to deal with model composition. Variables in open models are either local (*protected* variables in Modelica, i.e., private to the class), or public (possibly shared with other component models). Open models can be made aware of assumptions on their (unspecified yet) context. Overall, open models address the system view of physical modeling.

In this paper, we propose a notion of Σ-*interface* for open component models. Σ-interfaces involve only shared variables, thus abstracting away equations and local variables. The Σ-interface of an open model carries the information needed to perform modular structural analysis. We propose a hierarchical and modular algorithm that performs the interface-based structural analysis for DAE systems composed of possibly many open component models. Our algorithm exploits the sparsity of the underlying system as reflected in its incidence graph. Furthermore, very much like message passing, its efficiency relates to the model treewidth: the lower, the better.

The modular structural analysis algorithm was benchmarked on two academinc examples and an industrial-strength model of a urban district heating network. We succeeded in performing the structural analysis of DAE systems comprizing from 10^6 to 10^9 equations, depending on the model. Except for small instances of the models, computation times of the modular algorithm were a fraction of the computation times of state-of-the-art global structural analysis algorithms. As expected, sublinear execution times have been measured for several of the use cases.

This Σ-interface theory breaks one of the acknowledged obstacles to the scalability of DAE-based modeling languages, such as Modelica. Our contribution thus provides an important step towards the modular compilation of Modelica.

Of course, some issues remain open. In particular, models with higher treewidth are not handled well by our approach. Such unstructured models typically involve arrays of variables and components, with square regular grids being an example. While our notion of Σ-interface remains a useful candidate abstraction in this case, alternative algorithms for solving the Σ-method in a modular way still have to be found.

References

Benveniste, Albert et al. (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.

Benveniste, Albert et al. (2023-10). “Towards the separate compilation of Modelica: modularity and interfaces for the index reduction of incomplete DAE systems”. In: *Linköping Electronic Conference Proceedings*. Vol. 204. Aachen, Germany, p. 10. DOI: 10.3384/ecp204. URL: https://inria.hal.science/hal-04295096.

Bodlaender, Hans L. (1988). “Dynamic programming on graphs with bounded treewidth”. In: *Automata, Languages and Programming*. Ed. by Timo Lepistö and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 105–118. ISBN: 978-3-540-39291-0.

Broman, David (2021). “Interactive Programmatic Modeling”. In: *ACM Trans. Embed. Comput. Syst.* 20.4, 33:1–33:26. DOI: 10.1145/3431387. URL: https://doi.org/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*. Ed. by Peter Fritzson, François E. Cellier, and David Broman. Vol. 29. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 59–69. URL: http://www.ep.liu.se/ecp/article.asp?issue=029%5C&article=007%5C&volume=.

Bryant, R. 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.

Ⓙ

Y	J	A	Y_1	Y_2
$\{x, tb.f\}$	6	A_{10}	$\{x\}$	$\{tb.f\}$

$$A_{10} = \begin{array}{c} \\ 0 \\ x \\ tb.f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & tb.f \end{array} \\ \begin{bmatrix} -2 & 0 \\ 0 & 2 \\ -2 & 0 \end{bmatrix} \end{array}$$

Ⓘ is identical to Ⓗ, after renaming f by $tb.f$.

Ⓗ

Y	J	A
$\{f\}$	0	A_8
$\{x\}$	2	A_9

$$A_8 = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} -2 & 0 \\ & \\ -2 & 0 \end{bmatrix} \end{array} \qquad A_9 = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{cc} 0 & x \end{array} \\ \begin{bmatrix} -2 & 0 \\ 0 & 2 \\ & \end{bmatrix} \end{array}$$

Ⓖ

Y	J	A
$\{tb.f\}$	4	A_6
$\{x\}$	6	A_7

$$A_6 = \begin{array}{c} \\ 0 \\ x \\ tb.f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & tb.f \end{array} \\ \begin{bmatrix} -2 & 0 \\ -2 & 0 \\ & \end{bmatrix} \end{array} \qquad A_7 = \begin{array}{c} \\ 0 \\ x \\ tb.f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & tb.f \end{array} \\ \begin{bmatrix} -2 & 0 \\ & \\ & 2 \end{bmatrix} \end{array}$$

Ⓔ and Ⓕ are given Fig. 10.

Ⓓ

Y	J	A	Y_1	Y_2
$\{p1.f, p2.f\}$	4	A_3	$\{p1.f\}$	$\{p2.f\}$
$\{x, p1.f\}$	6	A_4	$\{p1.f\}$	$\{x\}$
$\{x, p2.f\}$	6	A_5	$\{x\}$	$\{p2.f\}$

$$A_3 = \begin{array}{c} \\ 0 \\ x \\ p1.f \\ p2.f \end{array} \begin{array}{c} \begin{array}{cccc} 0 & x & p1.f & p2.f \end{array} \\ \begin{bmatrix} -2 & 0 & 0 \\ & & & \\ -2 & 0 & \\ -2 & & 0 \end{bmatrix} \end{array}$$

$$A_4 = \begin{array}{c} \\ 0 \\ x \\ p1.f \\ p2.f \end{array} \begin{array}{c} \begin{array}{cccc} 0 & x & p1.f & p2.f \end{array} \\ \begin{bmatrix} -2 & 0 & 0 \\ 0 & & 2 \\ -2 & 0 & \\ & & \end{bmatrix} \end{array}$$

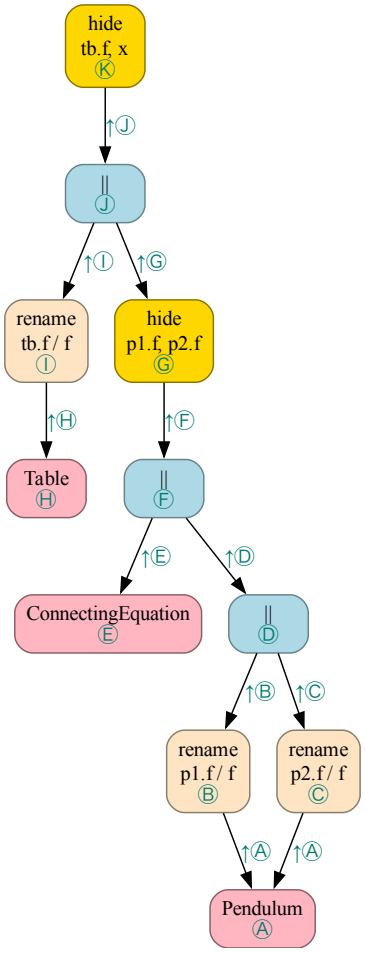
$$A_5 = \begin{array}{c} \\ 0 \\ x \\ p1.f \\ p2.f \end{array} \begin{array}{c} \begin{array}{cccc} 0 & x & p1.f & p2.f \end{array} \\ \begin{bmatrix} -2 & 0 & 0 \\ 0 & 2 & \\ & & \\ -2 & & 0 \end{bmatrix} \end{array}$$

Ⓚ

Y	J	A
$\emptyset$	6	A_{11}

$$A_{11} = \begin{array}{c} \\ 0 \end{array} \begin{array}{c} \begin{array}{c} 0 \end{array} \\ \begin{bmatrix} \end{bmatrix} \end{array}$$

Term Ⓚ has no public variable. It admits one valid selector: the empty set. This means that the Coupled-Pendulum model is structurally nonsingular.



Ⓑ (resp. Ⓒ) is identical to Ⓐ, up to a renaming of f by $p1.f$ (resp. $p2.f$).

Ⓐ

Y	J	A
$\{f\}$	2	A_1
$\{x\}$	4	A_2

$$A_1 = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} -2 & 0 \\ & \\ -2 & 0 \end{bmatrix} \end{array} \qquad A_2 = \begin{array}{c} \\ 0 \\ x \\ f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & f \end{array} \\ \begin{bmatrix} 0 & 0 \\ 0 & 2 \\ & \end{bmatrix} \end{array}$$

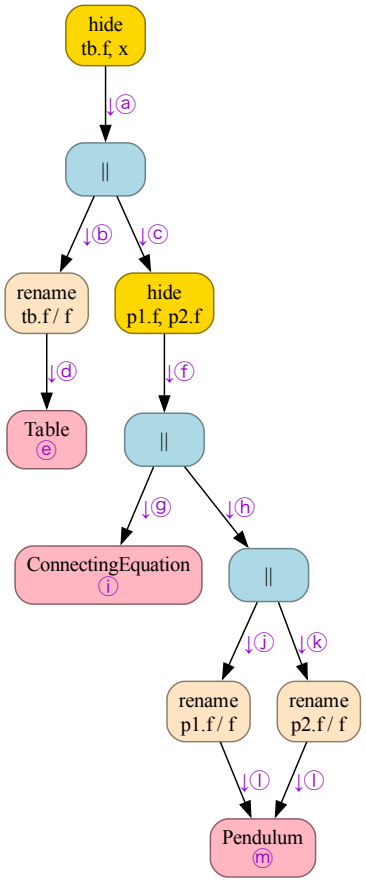
Figure 9. Primal/dual interfaces and primal cointerfaces computed during the upward sweep. Start reading the figure from the bottom. For the sake of clarity, the BTF components of the interfaces/cointerfaces have been omitted.

DOI 10.3384/ecp218175

$$A_{14} = \begin{array}{c} \\ 0 \\ tb.f \\ p1.f \\ p2.f \\ x \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & tb.f & p1.f & p2.f & x \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & \\ & 0 & & -2 \\ & & 0 & -2 \\ & & & \end{bmatrix} \end{array} \quad A_{15} = \begin{array}{c} \\ 0 \\ tb.f \\ p1.f \\ p2.f \\ x \end{array} \begin{array}{c} \begin{array}{cccc} 0 & tb.f & p1.f & p2.f & x \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & \\ & 0 & & -2 \\ & & & \\ & & 2 & 0 \end{bmatrix} \end{array}$$

Ⓕ

Y	J	A
$\{tb.f, p1.f, p2.f\}$	4	A_{14}
$\{tb.f, p1.f, x\}$	6	A_{15}
$\{tb.f, p2.f, x\}$	6	A_{16}
$\{p1.f, p2.f, x\}$	6	A_{17}



Ⓔ

Y	J	A
$\{tb.f\}$	0	A_{11}
$\{p1.f\}$	0	A_{12}
$\{p2.f\}$	0	A_{13}

$$A_{11} = \begin{array}{c} \\ 0 \\ tb.f \\ p1.f \\ p2.f \end{array} \begin{array}{c} \begin{array}{ccc} 0 & tb.f & p1.f & p2.f \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ & & \\ & & \end{bmatrix} \end{array}$$

A_{16} is obtained from A_{15} by exchanging rows and columns $p1.f$ and $p2.f$. A_{17} is obtained from A_{15} by exchanging rows $tb.f$ and $p1.f$, then columns $p1.f$ and $p2.f$.

ⓐ $= \left(\{x, tb.f\}, < d_x = 2, d_{tb.f} = 0 >\right)$ is computed using the cointerface of term ⑪, retrieved from the memoization table.

ⓑ $= \left(\{x\}, < d_x = 2, d_{tb.f} = 0 >\right)$ is computed from ⓐ, using the cointerface of term ⑩.

ⓒ $= \left(\{tb.f\}, < d_x = 2, d_{tb.f} = 0 >\right)$.

ⓓ $= \left(\{x\}, < d_x = 2, d_f = 0 >\right)$ is the reverse renaming of ⓑ.

ⓔ $= \left(\{x\}, < d_x = 2, d_f = 0 >, < c_8 = 0 >\right)$ is the Σ-solution for the instance of model `Table` (Fig. 2). c_8 is the offset of the equation line 8. This means that this equation is not differentiated.

ⓕ $= \left(\{tb.f, p1.f p2.f\}, < d_x = 2, d_{tb.f} = d_{p1.f} = d_{p2.f} = 0 >\right)$.

ⓖ $= \left(\{tb.f\}, < d_{tb.f} = d_{p1.f} = d_{p2.f} = 0 >\right)$.

ⓗ $= \left(\{p1.f, p2.f\}, < d_x = 2, d_{p1.f} = d_{p2.f} = 0 >\right)$.

ⓘ $= \left(\{tb.f\}, < d_{tb.f} = d_{p1.f} = d_{p2.f} = 0 >, < c_{37} = 0 >\right)$. Equation line 37 does not need to be differentiated.

ⓙ $= \left(\{p1.f\}, < d_x = 2, d_{p1.f} = 0 >\right)$.

ⓚ $= \left(\{p2.f\}, < d_x = 2, d_{p2.f} = 0 >\right)$.

ⓛ $= \left(\{f\}, < d_x = 2, d_f = 0 >\right)$.

ⓜ $= \left(\{f\}, < d_x = d_a = d_b = 2, d_f = d_{lambda} = 0, >, < c_{28} = 2, c_{24} = c_{25} = c_{29} = 0 >\right)$. Both instances of model `Pendulum` appear in the context of the same partial solution ①. This model has to undergo only one index reduction, where the only equation to be differentiated is line 28.

A_{12} (resp. A_{13}) is obtained from A_{11} by exchanging rows $tb.f$ and $p1.f$ (resp. $p2.f$).

Figure 10. Partial solutions computed during the downward sweep. Start reading the figure from the top. For the sake of clarity, the BTF components of the partial solutions have been omitted.

Cormen, Thomas H. et al. (2022). *Introduction to Algorithms, 4th Edition*. The MIT Press. ISBN: 9780262046305. URL: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/.

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*. Ed. by Joseph Sifakis. Vol. 407. Lecture Notes in Computer Science. Springer, pp. 197–212. DOI: 10.1007/3-540-52148-8_17. URL: https://doi.org/10.1007/3-540-52148-8%5C_17.

Fioravanti, Massimo et al. (2023-09). "Array-Aware Matching: Taming the Complexity of Large-Scale Simulation Models". In: *ACM Trans. Math. Softw.* 49.3. ISSN: 0098-3500. DOI: 10.1145/3611661. URL: https://doi.org/10.1145/3611661.

Fritzson, Peter (2015). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. 2nd ed. IEEE Press. Hoboken, NJ: Wiley. ISBN: 978-1-118-85912-4. DOI: 10.1002/9781118989166.

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: https://doi.org/10.1016/j.ifacol.2015.05.100. URL: https://www.sciencedirect.com/science/article/pii/S2405896315001019.

Mans, Michael et al. (2022). "Development and Application of an Open-Source Framework for Automated Thermal Network Generation and Simulations in Modelica". In: *Energies* 15.12. ISSN: 1996-1073. DOI: 10.3390/en15124372. URL: https://www.mdpi.com/1996-1073/15/12/4372.

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*. Ed. by Olivier Danvy and Andrzej Filinski. Vol. 2053. Lecture Notes in Computer Science. Springer, pp. 155–172. DOI: 10.1007/3-540-44978-7_10. URL: https://doi.org/10.1007/3-540-44978-7%5C_10.

Munkres, James (1957). "Algorithms for the Assignment and Transportation Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 5.1, pp. 32–38. DOI: 10.1137/0105003.

Otter, Martin and Hilding Elmqvist (2017-05). "Transformation of Differential Algebraic Array Equations to Index One Form". In: *Proceedings of the 12th International Modelica Conference*. Prag, Tschechische Republik: Linköping University Electronic Press. DOI: 10.3384/ecp17132565.

Pantelides, C. 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.

Pop, Adrian et al. (2019). "A New OpenModelica Compiler High Performance Frontend". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4-6, 2019*, 157:071. DOI: 10.3384/ecp19157689.

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

Tarjan, Robert (1972). "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2, pp. 146–160. DOI: 10.1137/0201010.

Thibault, Joan (2022-07). *Constraint System Decomposition*. Research Report RR-9478. Inria Rennes, pp. 1–68. DOI: 10.13140/RG.2.2.13004.49285. URL: https://hal.inria.fr/hal-03740562.

Thibault, Joan (2024). *Snowflake*. https://gitlab.com/boreal-ldd/snowflake [Accessed: 29-04-2025].