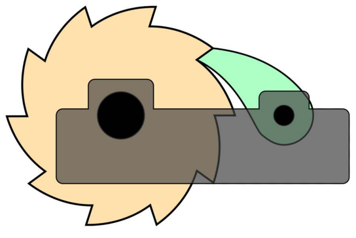
IsamDAE: An implicit Structural Analysis tool for multimode DAE systems

Benoît Caillaud, Mathias Malandain, Joan Thibault

November 25th, 2019 - Aussois - Synchron'19 workshop

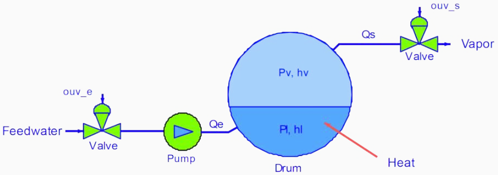
Multimode (aka. hybrid) systems

- Natural models for physical phenomena
  - mechanics (engagement/release of links)



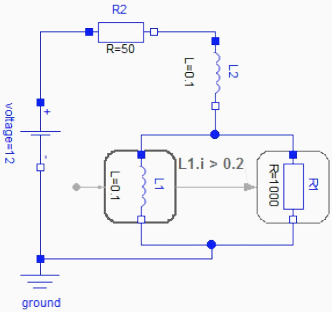
Multimode (aka. hybrid) systems

- Natural models for physical phenomena
  - mechanics (engagement/release of links)
  - thermodynamics (phase (dis)appearance)
  - hydraulics (opening/closing of a valve)
  - electronics (switching diode/transistor)



Multimode (aka. hybrid) systems

- Natural models for physical phenomena
  - mechanics (engagement/release of links)
  - thermodynamics (phase (dis)appearance)
  - hydraulics (opening/closing of a valve)
  - electronics (switching diode/transistor)
- Fault modeling (component break)



Multimode (aka. hybrid) systems

- Natural models for physical phenomena
  - mechanics (engagement/release of links)
  - thermodynamics (phase (dis)appearance)
  - hydraulics (opening/closing of a valve)
  - electronics (switching diode/transistor)
- Fault modeling (component break)
- Reconfigurable systems ((dis)appearance of components)

A sketch of Modelica and its semantics [Fritzson]

- Modelica = DAE + Objects
- Class = container for equations

```
model SimpleDrive
   ..Rotational.Inertia   Inertia1  (J=0.002);
   ..Rotational.IdealGear IdealGear1(ratio=100)
   ..Basic.Resistor       Resistor1 (R=0.2)
      ...
equation
   connect(Inertia1.flange_b, IdealGear1.flange_a);
   connect(Resistor1.n, Inductor1.p);
      ...
end SimpleDrive;
```

```
model Resistor
   package SIunits = Modelica.SIunits;
   parameter SIunits.Resistance R = 1;
   SIunits.Voltage v;
   ..Interfaces.PositivePin p;
   ..Interfaces.NegativePin n;
equation
   0 = p.i + n.i;
   v = p.v - n.v;
   v = R*p.i;
end Resistor;
```

```
type Voltage =
     Real(quantity="Voltage",
          unit    ="V");
```

```
connector PositivePin
   package SIunits = Modelica.SIunits;
   SIunits.Voltage      v;
   flow SIunits.Current i;
end PositivePin;
```

A sketch of Modelica and its semantics [Fritzson]

- Modelica Reference v3.3:

  "The semantics of the Modelica language is specified by means of a set of rules for translating any class described in the Modelica language to a flat Modelica structure"

- Pros:
  - Semantics of continuous-time 1-mode Modelica models: Cauchy problem on the DAE resulting from the inlining of all components
  - DAE $\Rightarrow$ modularity & reusability
  - interconnecting components $=$ algebraic constraints ($\neq$ ODE)

```
model SC2
  class M
    parameter Real S;
    parameter Real C;
    Real u,i;
  equation
    C*der(u) + S*u = i;
  end M;
```

```
  M m1(S=1e-6,C=1e-5);
  M m2(S=2e-5,C=3e-5);
equation
  m1.u = m2.u;
  m1.i + m2.i = 0;
end SC2;
```

A sketch of Modelica and its semantics [Fritzson]

- Modelica supports multimode systems

```
x*x + y*y = 1;
der(x) + u = 0;
u = if x >= 0 then x+y else y;
when x <= 0 do reinit(x,1); end;
when y <= 0 do reinit(y,x); end;
```

- Cons:
  - What about the semantics of multimode systems?
  - Concept of solution incompletely defined
- and, unsurprisingly: Questionable simulations

Objectives and challenges

- Handling variable structure: Take into account the mode dependency of equations and variables in multimode DAE (mDAE) systems

Objectives and challenges

- Handling variable structure: Take into account the mode dependency of equations and variables in multimode DAE (mDAE) systems
- Model representation: Represent structural information of multimode system in a concise way (i.e., no mode enumeration)

Objectives and challenges

- Handling variable structure: Take into account the mode dependency of equations and variables in multimode DAE (mDAE) systems
- Model representation: Represent structural information of multimode system in a concise way (i.e., no mode enumeration)
- Implicit structural analysis and block-triangular decomposition: Adapt existing algorithms so that they handle "all modes at once" (i.e., modes are not enumerated)

Objectives and challenges

- Handling variable structure: Take into account the mode dependency of equations and variables in multimode DAE (mDAE) systems
- Model representation: Represent structural information of multimode system in a concise way (i.e., no mode enumeration)
- Implicit structural analysis and block-triangular decomposition: Adapt existing algorithms so that they handle "all modes at once" (i.e., modes are not enumerated)

To our knowledge, no similar works in the literature.

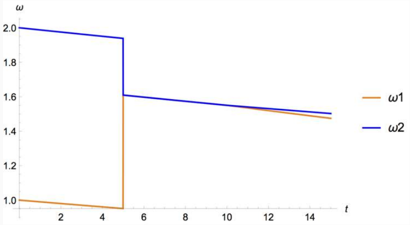
Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes

```
j1*der(w1) = -k1*w1 + f1;
j2*der(w2) = -k2*w2 + f2;
0 = if g then w1-w2 else f1;
f1 + f2 = 0;
```



Singular inconsistent scalar system for f1 = ((if g then w1-w2 else 0.0)) / (-(if g then 0.0 else 1.0)) = -0.502621/-0

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes
- Solution 2: enumerate all modes (separate structural analyses)

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes
- Solution 2: enumerate all modes (separate structural analyses)
  - Patience is a virtue: 2 modes per component $\Rightarrow 10^{15}$ modes for a 50-component system

```
class TrainN
  import Railcar;
  parameter Integer n = 50;
  Railcar railcar[n];
  Modelica.electrical.analog.Interfaces.PositivePin pin_p;
  Modelica.electrical.analog.Interfaces.NegativePin pin_n;
  Real v[n];
  Modelica.Blocks.Interfaces.RealOutput u;
equation
  connect(pin_n,railcar[n].pin_n);
  for i in 1:n-1 loop
    connect(railcar[i+1].pin_p,railcar[i].pin_n);
  end for;
  for i in 1:n loop
    connect(railcar[i].v,v[i]);
  end for;
  connect(pin_p, railcar[1].pin_p);
  n*u + sum(v) = 0;
  annotation(...);
end TrainN;
```

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes
- Solution 2: enumerate all modes (separate structural analyses)
  - Patience is a virtue: 2 modes per component $\Rightarrow 10^{15}$ modes for a 50-component system
- Solution 3: structural analysis at run-time

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes
- Solution 2: enumerate all modes (separate structural analyses)
  - Patience is a virtue: 2 modes per component $\Rightarrow 10^{15}$ modes for a 50-component system
- Solution 3: structural analysis at run-time
  - No diagnosis at run-time, except basic type-checking

- JIT compilation : index reduction, Dulmage-Mendelsohn decomposition and automatic differentiation performed at run-time
- Modelyze [Broman], Modia [Elmqvist]

Analysis of a multimode DAE

- "Solution" 1: "forget" about the mode dependencies (approximate structural analysis)
  - ...possibly pivoting variables that vanish in some modes
- Solution 2: enumerate all modes (separate structural analyses)
  - Patience is a virtue: 2 modes per component $\Rightarrow 10^{15}$ modes for a 50-component system
- Solution 3: structural analysis at run-time
  - No diagnosis at run-time, except basic type-checking

Our idea:

- *Symbolic* structural analysis $\Rightarrow$ represent the structure of a mDAE as *functions* $M \to \mathbb{N}/\mathbb{B}$ of the modes

Structural Analysis of DAE

DAE

General form: $F(x, x', x'', \ldots, t) = 0$

- $x = (x_1, x_2, \ldots, x_n)$ with $x_i = x_i(t)$;
- $F = \{f_1, f_2, \ldots, f_n\}$ set of n functions of t, and of x_i and of finite number of their derivatives.

DAE

General form: $F(x, x', x'', \dots, t) = 0$

- $x = (x_1, x_2, \dots, x_n)$ with $x_i = x_i(t)$;
- $F = \{f_1, f_2, \dots, f_n\}$ set of n functions of t, and of x_i and of finite number of their derivatives.

Define σ_{ij} the highest differentiation order of x_j in equation f_i. The leading variables of F are $x_j^{(\sigma_j)}$ with $\sigma_j = \max_i \sigma_{ij}$.

DAE

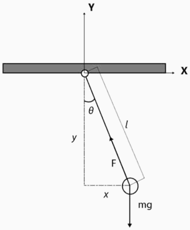
General form: $F(x, x', x'', \ldots, t) = 0$

- $x = (x_1, x_2, \ldots, x_n)$ with $x_i = x_i(t)$;
- $F = \{f_1, f_2, \ldots, f_n\}$ set of n functions of t, and of x_i and of finite number of their derivatives.

Define σ_{ij} the highest differentiation order of x_j in equation f_i. The leading variables of F are $x_j^{(\sigma_j)}$ with $\sigma_j = \max_i \sigma_{ij}$. If $\sigma_j = 0$, variable x_j is said algebraic.

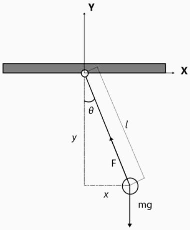
Example: a pendulum (Cartesian coordinates)

$$(S) \quad \begin{cases} x'' + Tx &= 0 \\ y'' + Ty - g &= 0 \\ x^2 + y^2 - l^2 &= 0 \end{cases}$$



Example: a pendulum (Cartesian coordinates)

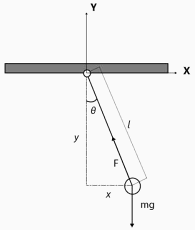
$$(S) \quad \begin{cases} x'' + Tx &= 0 \\ y'' + Ty - g &= 0 \\ x^2 + y^2 - l^2 &= 0 \end{cases}$$



T is an algebraic variable : (S) can not be solved like an ODE.

Example: a pendulum (Cartesian coordinates)

$$(S) \quad \begin{cases} x'' + Tx &= 0 \\ y'' + Ty - g &= 0 \\ x^2 + y^2 - l^2 &= 0 \end{cases}$$



T is an algebraic variable : (S) can not be solved like an ODE. The Jacobian matrix wrt. (x'', y'', T) is:

$$J = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 0 \end{pmatrix}$$

J is singular : the system can not be solved without some transformation.

Example: a pendulum (Cartesian coordinates)

However, if the third equation is differentiated twice:

$$(S') \quad \left\{ \begin{array}{rcl} x'' + Tx & = & 0 \\ y'' + Ty - g & = & 0 \\ 2xx'' + 2(x')^2 + 2yy'' + 2(y')^2 & = & 0 \end{array} \right. \quad ; \quad J' = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{pmatrix}$$

The Jacobian J' is invertible

Example: a pendulum (Cartesian coordinates)

However, if the third equation is differentiated twice:

$$(S') \quad \left\{ \begin{array}{rcl} x'' + Tx & = & 0 \\ y'' + Ty - g & = & 0 \\ 2xx'' + 2(x')^2 + 2yy'' + 2(y')^2 & = & 0 \end{array} \right. \quad ; \quad J' = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{pmatrix}$$

The Jacobian J' is invertible

How can one determine automatically which equations have to be differentiated, and how many times?

Structural Analysis

Principles:

Structural Analysis

Principles:

- structural invertibility of a matrix = almost certainly invertible when its non-zero elements are random variables varying in a small neighborhood

Structural Analysis

Principles:

- structural invertibility of a matrix = almost certainly invertible when its non-zero elements are random variables varying in a small neighborhood
- Checking structural invertibility $\Rightarrow$ no determinant needs to be computed

Structural Analysis

Principles:

- structural invertibility of a matrix = almost certainly invertible when its non-zero elements are random variables varying in a small neighborhood
- Checking structural invertibility $\Rightarrow$ no determinant needs to be computed
- Retains useful information: which variables appear (with what differentiation order) in which equation? (σ_{ij} : highest differentiation order of x_j in f_i)

Structural Analysis

Principles:

- structural invertibility of a matrix = almost certainly invertible when its non-zero elements are random variables varying in a small neighborhood
- Checking structural invertibility $\Rightarrow$ no determinant needs to be computed
- Retains useful information: which variables appear (with what differentiation order) in which equation? (σ_{ij} : highest differentiation order of x_j in f_i)
- Uses graph theoretic algorithms (eg. Pantelides method)

Highlights on several methods

- Structural analysis methods:
  - Pantelides (1988)
  - Weighed Bipartite Graph method [Ding et al. 2008]
  - Σ-method [Pryce 2001]
  - σ-ν method [Chowdhry et al. 2004]
- Several implementations (Modelica tools, Mathematica...)

Σ-matrix representation of the Pendulum

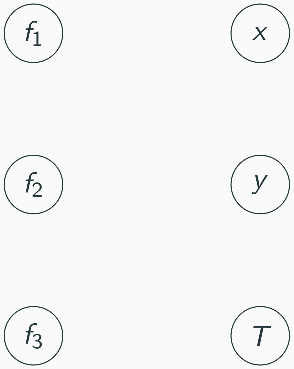
$$\begin{cases} f_1 &= & x'' - Tx \\ f_2 &= & y'' - Ty + g \\ f_3 &= & x^2 + y^2 - L^2 \end{cases}$$

Σ-matrix representation of the Pendulum

$$\begin{cases} f_1 &= x'' - Tx \\ f_2 &= y'' - Ty + g \\ f_3 &= x^2 + y^2 - L^2 \end{cases}$$

Variable order:

$$X = (x, y, T)$$



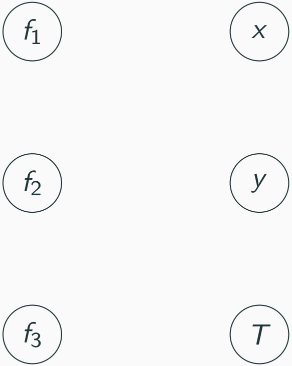
Σ-matrix representation of the Pendulum

$$\begin{cases} f_1 &= x'' - Tx \\ f_2 &= y'' - Ty + g \\ f_3 &= x^2 + y^2 - L^2 \end{cases}$$

Variable order:

$$X = (x, y, T)$$

$$\Sigma = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$



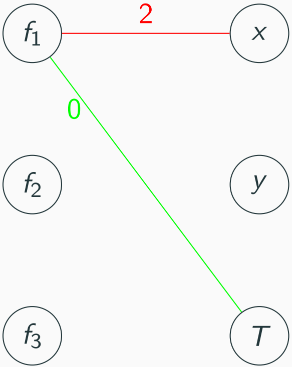
Σ-matrix representation of the Pendulum

$$\begin{cases} f_1 &= x'' - Tx \\ f_2 &= y'' - Ty + g \\ f_3 &= x^2 + y^2 - L^2 \end{cases}$$

Variable order:

$$X = (x, y, T)$$

$$\Sigma = \begin{pmatrix} 2 & - & 0 \\ & & \\ & & \end{pmatrix}$$



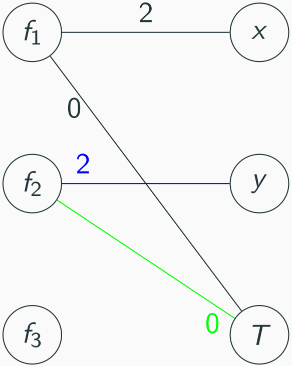
Σ-matrix representation of the Pendulum

$$\begin{cases} f_1 &= x'' - Tx \\ f_2 &= y'' - Ty + g \\ f_3 &= x^2 + y^2 - L^2 \end{cases}$$

Variable order:

$$X = (x, y, T)$$

$$\Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ & & \end{pmatrix}$$



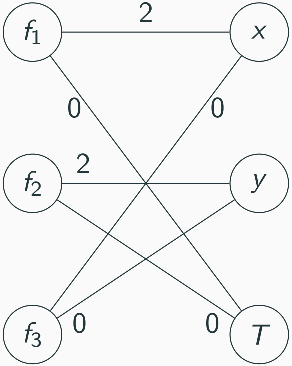
Σ-matrix representation of the Pendulum

$$\begin{cases} f_1 &= x'' - Tx \\ f_2 &= y'' - Ty + g \\ f_3 &= x^2 + y^2 - L^2 \end{cases}$$

Variable order:

$$X = (x, y, T)$$

$$\Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ 0 & 0 & - \end{pmatrix}$$



The Σ-method

Compute the least diff. order c_i of equation f_i st. the Jacobian is structurally invertible

Primal problem: compute a maximal weight transverse of Σ

The Σ-method

Compute the least diff. order c_i of equation f_i st. the Jacobian is structurally invertible

Primal problem: compute a maximal weight transverse of Σ

Dual problem: Compute the minimal solution of the linear program

$$\boxed{\begin{array}{lrl} (\mathbf{P_{off}}) \ : & \min \ \hat{z} = \sum_j d_j - \sum_i c_i & \\ & \text{s.t.} \quad d_j - c_i \geq \sigma_{ij} & \forall (i,j) \in S \\ & c_i \geq 0 & 1 \leq i \leq n \end{array}}$$

with a fixed-point method using the maximal weight transverse

The Σ-method

Compute the least diff. order c_i of equation f_i st. the Jacobian is structurally invertible

Primal problem: compute a maximal weight transverse of Σ

Dual problem: Compute the minimal solution of the linear program

$$\boxed{\begin{aligned} (\mathbf{P_{off}}) \quad : \quad & \min \ \hat{z} = \sum_j d_j - \sum_i c_i \\ & \text{s.t.} \quad d_j - c_i \geq \sigma_{ij} \quad \forall (i,j) \in S \\ & \qquad c_i \geq 0 \quad 1 \leq i \leq n \end{aligned}}$$

with a fixed-point method using the maximal weight transverse

Result:
- c_i = number of times equations must be differentiated

The Σ-method

Compute the least diff. order c_i of equation f_i st. the Jacobian is structurally invertible

Primal problem: compute a maximal weight transverse of Σ

Dual problem: Compute the minimal solution of the linear program

$$\boxed{\begin{aligned}(\mathbf{P_{off}}) \quad : \qquad \min \ \hat{z} = \sum_j d_j - \sum_i c_i & \\ \text{s.t.} \quad d_j - c_i \geq \sigma_{ij} \quad & \forall (i,j) \in S \\ c_i \geq 0 \quad & 1 \leq i \leq n\end{aligned}}$$

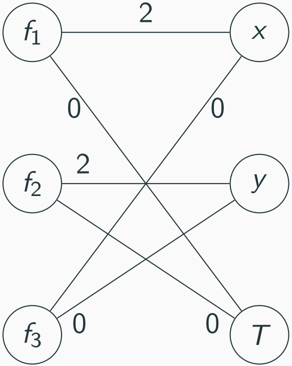
with a fixed-point method using the maximal weight transverse

Result:

- c_i = number of times equations must be differentiated
- d_j = differentiation order of the leading variables in the resulting system

Back to the Pendulum example

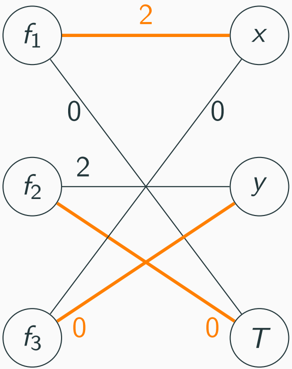
$$\Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ 0 & 0 & - \end{pmatrix}$$



Back to the Pendulum example

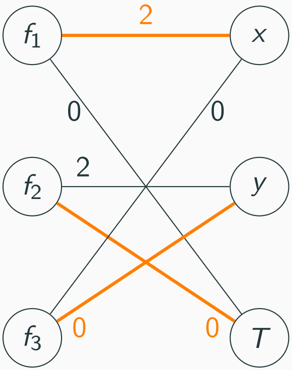
A solution to the Primal problem:

$$\Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ 0 & 0 & - \end{pmatrix}$$



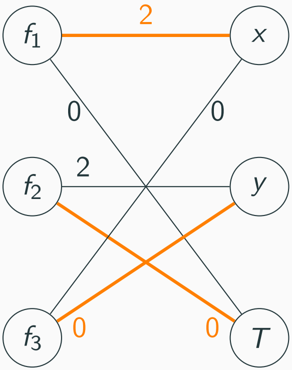
Back to the Pendulum example

$$c_i \left\{ \begin{array}{c|ccc} 0 & \mathbf{2} & & 0 \\ 0 & & 2 & \mathbf{0} \\ 0 & 0 & \mathbf{0} & \\ \hline & & & \\ & \multicolumn{3}{c}{\underbrace{\qquad\qquad\qquad}_{d_j}} \end{array} \right.$$



Back to the Pendulum example

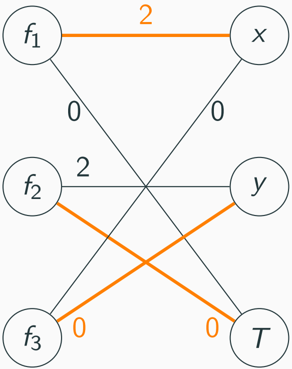
$$c_i \left\{ \begin{array}{c|ccc} 0 & \mathbf{2} & & 0 \\ 0 & & 2 & \mathbf{0} \\ 0 & 0 & \mathbf{0} & \\ \hline & 2 & & \end{array} \right. \qquad \underbrace{\qquad\qquad}_{d_j}$$



Back to the Pendulum example

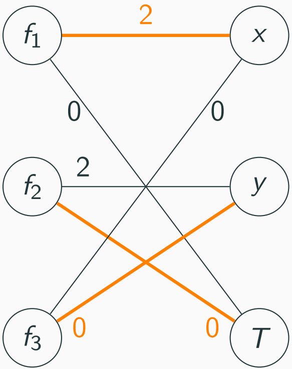
$$c_i \left\{ \begin{array}{c|ccc} 0 & \mathbf{2} & & 0 \\ 0 & & 2 & \mathbf{0} \\ 0 & 0 & \mathbf{0} & \\ \hline & 2 & 2 & \end{array} \right.$$

$$\underbrace{\qquad\qquad}_{d_j}$$



Back to the Pendulum example

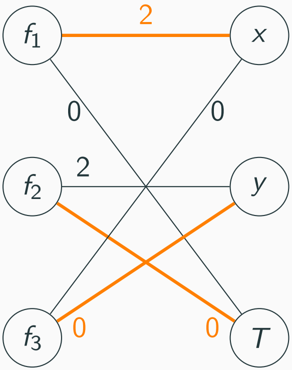
$$
c_i \left\{ \begin{array}{c|ccc}
0 & \mathbf{2} & & 0 \\
0 & & 2 & \mathbf{0} \\
0 & 0 & \mathbf{0} & \\
\hline
 & \multicolumn{3}{c}{\underbrace{2 \quad\; 2 \quad\; 0}_{d_j}}
\end{array} \right.
$$



Back to the Pendulum example

$$c_i\left\{\begin{array}{c|ccc} 0 & \mathbf{2} & & 0 \\ 0 & & 2 & \mathbf{0} \\ 2 & 0 & \mathbf{0} & \\ \hline & 2 & 2 & 0 \end{array}\right.$$

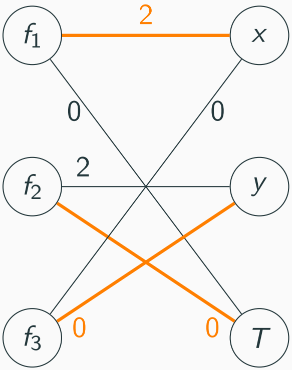
$$\underbrace{\qquad\qquad\qquad}_{d_j}$$



Back to the Pendulum example

$$c_i\left\{\begin{array}{c|ccc} 0 & \mathbf{2} & & 0 \\ 0 & & 2 & \mathbf{0} \\ 2 & 0 & \mathbf{0} & \\ \hline & \multicolumn{3}{c}{\underbrace{2 \quad\; 2 \quad\; 0}_{d_j}} \end{array}\right.$$

Fixed-point has been reached $\Rightarrow$ the solution has been computed



The IsamDAE tool and the MEL language

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests

- Boolean (mode) variables: predicates on real variables

```
g : boolean = x > 1.e-2
```

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests
- Boolean (mode) variables: predicates on real variables

```
g : boolean = x > 1.e-2
```

- Invariants are used to narrow the structural analysis to particular modes

```
invariant liq | gas
```

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests
- Both variables...

```
if !g then xf : real end
```

...and equations...

```
e1 : equation 0 = if g1 & !g2 then x else - y / 2.;
if g1 | g2 then e2t : equation 0 = x + y end;
```

...can be placed in or contain `if` ... `then` ... `else` ... statements

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests

- foreach loops and arrays, to define parametric models

```
foreach k in 1 .. n do
   x[k] : real
done
```

MEL: a toy mDAE modeling language

- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests
- foreach loops and arrays, to define parametric models

```
foreach k in 1 .. n do
   x[k] : real
done
```

- Also: `parameters`, uninterpreted `nonlinear functions`, `when <event> then <statements> end`

MEL: a toy mDAE modeling language

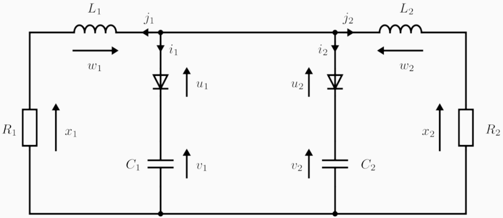
- MEL: *ad hoc* multimode DAE systems language
- Not using Modelica for several reasons:
  - Modelica is an overly complex language
  - Models with mode-dependent number of equations/variables
  - Declaration of invariants, excluding some modes from the structural analysis
  - More flexibility for future experiments & tests

- foreach loops and arrays, to define parametric models

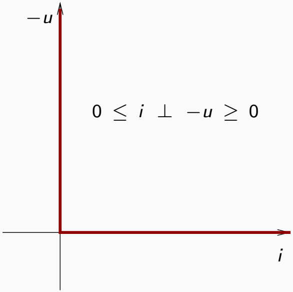
```
foreach k in 1 .. n do
   x[k] : real
done
```

- Also: `parameters`, uninterpreted `nonlinear functions`, `when <event> then <statements> end`

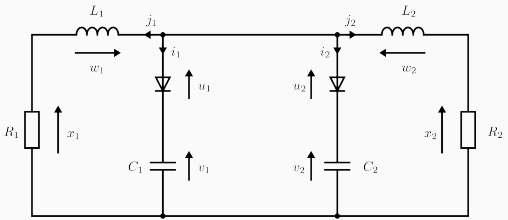
Example: the RLDC2 circuit



- Provided by M. Otter and S. E. Mattsson
- Simple model with 4 modes, 14 equations, 14 variables



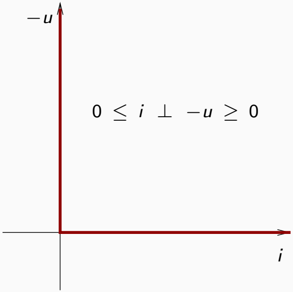
Example: the RLDC2 circuit



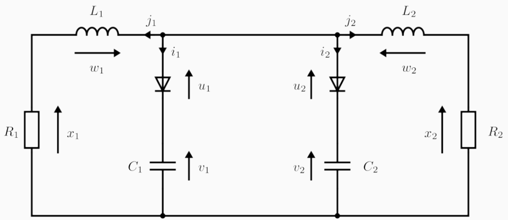
- Provided by M. Otter and S. E. Mattsson
- Simple model with 4 modes, 14 equations, 14 variables

- Currently not handled by Dymola & OpenModelica:

```
Model error - division by zero
```



Example: the RLDC2 circuit

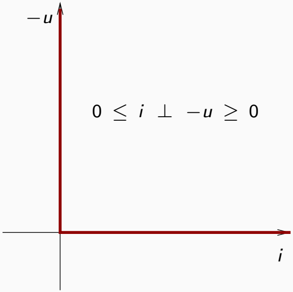


- Provided by M. Otter and S. E. Mattsson
- Simple model with 4 modes, 14 equations, 14 variables

- Currently not handled by Dymola & OpenModelica:

  `Model error - division by zero`

- Ideal diodes modeled as complementarity conditions



Example: the RLDC2 circuit

```
// Kirchhoff laws
K1 : equation 0 = j1+i1+i2+j2;
K2 : equation x1+w1 = u1+v1;
K3 : equation u1+v1 = u2+v2;
K4 : equation u2+v2 = x2+w2;
// Resistors
R1 : equation x1 = R1*j1;
R2 : equation x2 = R2*j2;
// Inductors
L1 : equation w1 = L1*der(j1);
L2 : equation w2 = L2*der(j2);
// Capacitors
C1 : equation i1 = C1*der(v1);
C2 : equation i2 = C2*der(v2);
```

```
// Diode 1
// p1 holds iff left limit
// of s1 is non-negative
p1 : boolean = last(s1);
S1: equation s1 = if p1 then i1 else -u1;
Z1: equation 0 = if p1 then u1 else i1;

// Diode 2
// p2 holds iff left limit
// of s2 is non-negative
p2 : boolean = last(s2);
S2: equation s2 = if p2 then i2 else -u2;
Z2: equation 0 = if p2 then u2 else i2
```

Functional encoding of the structure of a mDAE

Encoding a model (in a nutshell)

- Warning: In this talk we do not deal with mode changes. Assume that solutions are continuous

Encoding a model (in a nutshell)

- Warning: In this talk we do not deal with mode changes. Assume that solutions are continuous
- Everything is encoded as functions of the mode variables

Encoding a model (in a nutshell)

- Warning: In this talk we do not deal with mode changes. Assume that solutions are continuous
- Everything is encoded as functions of the mode variables
- BDDs (Binary Decision Diagrams) are an appropriate framework:
  - Compact and canonical representation of Boolean functions as DAGs
  - Efficient computations on such functions
  - Integer functions: variable-length little-endian binary encoding (list of BDDs)

Encoding a model (in a nutshell)

- Warning: In this talk we do not deal with mode changes. Assume that solutions are continuous
- Everything is encoded as functions of the mode variables
- BDDs (Binary Decision Diagrams) are an appropriate framework:
  - Compact and canonical representation of Boolean functions as DAGs
  - Efficient computations on such functions
  - Integer functions: variable-length little-endian binary encoding (list of BDDs)
- Negation $\neg$ and equality check in $\mathcal{O}(1)$, other operations include:

  Conjunction/disjunction: $\wedge/\vee$

  Existential quantification: $\exists a.\, f(a, b)$

  Universal quantification: $\forall a.\, f(a, b)$

Encoding a model (in a nutshell)

- Warning: In this talk we do not deal with mode changes. Assume that solutions are continuous
- Everything is encoded as functions of the mode variables
- BDDs (Binary Decision Diagrams) are an appropriate framework:
  - Compact and canonical representation of Boolean functions as DAGs
  - Efficient computations on such functions
  - Integer functions: variable-length little-endian binary encoding (list of BDDs)
- Negation $\neg$ and equality check in $\mathcal{O}(1)$, other operations include:

  Conjunction/disjunction: $\wedge/\vee$

  Existential quantification: $\exists a.\, f(a, b)$

  Universal quantification: $\forall a.\, f(a, b)$
- However: very sensitive to variable and computation ordering

Structural Analysis

Pryce's Σ-method (1/2)

Σ-matrix coefficients for a single-mode DAE:

$$
\begin{array}{rcl}
f_1(x_1, x_1', \ldots, x_1^{(\sigma_{1,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{1,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{1,n})}) & = & 0 \\
f_2(x_1, x_1', \ldots, x_1^{(\sigma_{2,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{2,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{2,n})}) & = & 0 \\
\vdots & & \\
f_n(x_1, x_1', \ldots, x_1^{(\sigma_{n,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{n,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{n,n})}) & = & 0
\end{array}
$$

Convention: x_j does not appear in $f_i \Rightarrow \sigma_{i,j} = -\infty$

Pryce's Σ-method (2/2)

John Pryce's two-step structural analysis method:

- **Primal problem:** search for a HVT (Highest-Value Transversal)
  - it is a maximum-weight perfect matching between the equations and variables of the DAE

- **Dual problem:** find the solution $(c_1, \ldots, c_n, d_1, \ldots, d_n)$ of a Linear Programming problem
  - solved thanks to a fixpoint iteration

Result: "solve equations $f_i^{(c_i)}$ for leading variables $x_j^{(d_j)}$"

+ HVT used for scheduling computations

Multimode structural analysis Σ-method (1/4)

Σ-matrix coefficients for a single-mode DAE:

$$
\begin{array}{rcl}
f_1(x_1, x_1', \ldots, x_1^{(\sigma_{1,1,m})}, x_2, x_2', \ldots, x_2^{(\sigma_{1,2,m})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{1,n,m})}) & = & 0 \\
f_2(x_1, x_1', \ldots, x_1^{(\sigma_{2,1,m})}, x_2, x_2', \ldots, x_2^{(\sigma_{2,2,m})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{2,n,m})}) & = & 0 \\
& \vdots & \\
f_n(x_1, x_1', \ldots, x_1^{(\sigma_{n,1,m})}, x_2, x_2', \ldots, x_2^{(\sigma_{n,2,m})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{n,n,m})}) & = & 0
\end{array}
$$

Convention: x_j does not appear in f_i in mode m implies $\sigma_{i,j,m} = -\infty$

Auxiliary functions: $\chi_I : M \times I \to \mathbb{B}$, $\chi_J : M \times J \to \mathbb{B}$ and $\chi_E : M \times E \to \mathbb{B}$ characteristic functions of the set of active equations, variables and incidence edges

Multimode structural analysis (2/4)

The primal problem is solved in the following way:

Multimode structural analysis (2/4)

The primal problem is solved in the following way:

- Encode constraints as functions $M \to \mathbb{B}$
  - μ: *"an active equation must be matched to a variable"*
  - ν: *"...and vice-versa"*
  - Υ: *"an edge can only be part of a matching if it is active"*

Multimode structural analysis (2/4)

The primal problem is solved in the following way:

- Encode constraints as functions $M \to \mathbb{B}$
  - μ: *"an active equation must be matched to a variable"*
  - ν: *"...and vice-versa"*
  - Υ: *"an edge can only be part of a matching if it is active"*
- $X := \Upsilon \wedge \mu \wedge \nu$ describes all perfect matchings in all modes

Multimode structural analysis (2/4)

The primal problem is solved in the following way:

- Encode constraints as functions $M \to \mathbb{B}$
  - μ: *"an active equation must be matched to a variable"*
  - ν: *"...and vice-versa"*
  - Υ: *"an edge can only be part of a matching if it is active"*
- $X := \Upsilon \wedge \mu \wedge \nu$ describes all perfect matchings in all modes
- Apply a (parametrized) ArgMax operator using edge weights

  $\Rightarrow$ only keep maximum weight perfect matchings

Parameterized argmax algorithm (3/4)

Problem:

Given φ and $(w_k)_{k=0\ldots N-1}$, compute:

$$\psi \;=\; \mathrm{ArgMax}_{\vec{V}}(w \,|\, \varphi) \;=\; \{\vec{x} = (x_v)_{v \in \vec{V}} \,|\, \varphi(\vec{x}) \text{ and } w(\vec{x}) \text{ maximal}\}$$

Algorithm:

$$\begin{array}{lcl}
maxb_k(\gamma) & = & \gamma \wedge (\,\pi \iff w_k\,) \text{ with:} \\
 & & \quad \pi \;=\; \exists \vec{V}, \gamma \wedge w_k \\
\psi & = & \psi_0 \\
\psi_k & = & maxb_k(\psi_{k+1}) \text{ for all } k < N \\
\psi_N & = & \varphi
\end{array}$$

Multimode structural analysis (4/4)

The dual problem is solved by ”parametrizing” everything

Multimode structural analysis (4/4)

The dual problem is solved by "parametrizing" everything

- Standard (single-mode) fixpoint iteration:

$$\begin{aligned}\forall j, \quad d_j &\leftarrow \max_i(\sigma_{ij} + c_i) \\ \forall i, \quad c_i &\leftarrow d_{j_i} - \sigma_{i,j_i}\end{aligned}$$

with all c_i's and d_j's initialized to 0

Multimode structural analysis (4/4)

The dual problem is solved by "parametrizing" everything

- Standard (single-mode) fixpoint iteration:

$$\begin{array}{lcl} \forall j, \quad d_j & \leftarrow & \max_i(\sigma_{ij} + c_i) \\ \forall i, \quad c_i & \leftarrow & d_{j_i} - \sigma_{i,j_i} \end{array}$$

  with all c_i's and d_j's initialized to 0

- Parametrized (multimode) fixpoint iteration:

$$\forall j, \quad d_j \quad \equiv \quad \text{if } \chi_J(j) \text{ then } \underbrace{\max_{e=(i,j)} \{\text{if } \chi_E(e) \text{ then } \sigma_{i,j} + c_i \text{ else } 0\}}_{M \to \mathbb{N}} \text{ else } 0$$

$$\forall i, \quad c_i \quad \equiv \quad \text{if } \chi_I(i) \text{ then } \underbrace{\max_{e=(i,j)} \{\text{if } (\chi_J(j) \wedge T(e)) \text{ then } d_j - \sigma_{i,j} \text{ else } c_i\}}_{M \to \mathbb{N}} \text{ else } 0$$

  with all c_i's and d_j's initialized to zero functions

Our results

Good news: everything works as it should

Our results

Good news: everything works as it should

- Same results in every mode as with standard structural analysis, but way faster

Our results

Good news: everything works as it should

- Same results in every mode as with standard structural analysis, but way faster
- Detection and diagnosis of modes in which the system is structurally singular (a list of equations and variables that cannot be consistently matched is returned)

Our results

Good news: everything works as it should

- Same results in every mode as with standard structural analysis, but way faster
- Detection and diagnosis of modes in which the system is structurally singular (a list of equations and variables that cannot be consistently matched is returned)

"Bad" news: everything works as it should...

Our results

Good news: everything works as it should

- Same results in every mode as with standard structural analysis, but way faster
- Detection and diagnosis of modes in which the system is structurally singular (a list of equations and variables that cannot be consistently matched is returned)

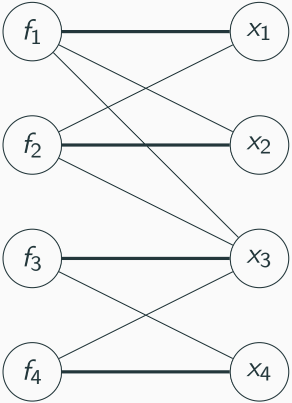
"Bad" news: everything works as it should...

- *Numerical* singularities are (by definition) unseen by *structural* analysis

Dependencies and scheduling

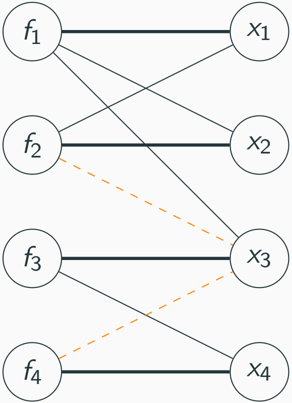
Dependency graph

- (Single-mode) Dependency graph:



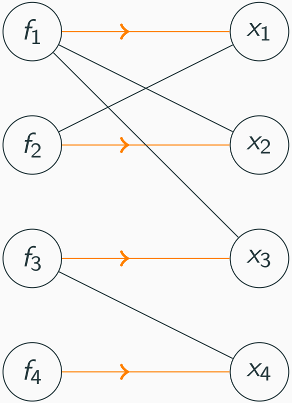
Dependency graph

- (Single-mode) Dependency graph: saturated edges are directed



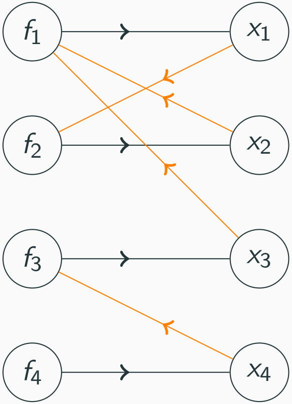
Dependency graph

- (Single-mode) Dependency graph: saturated edges are directed
  - $i \rightsquigarrow j$ for an edge in the chosen transversal



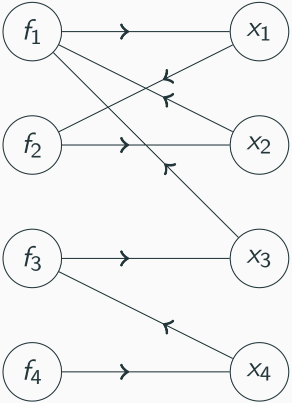
Dependency graph

- (Single-mode) Dependency graph: saturated edges are directed
  - $i \rightsquigarrow j$ for an edge in the chosen transversal
  - $j \rightsquigarrow i$ for other edges



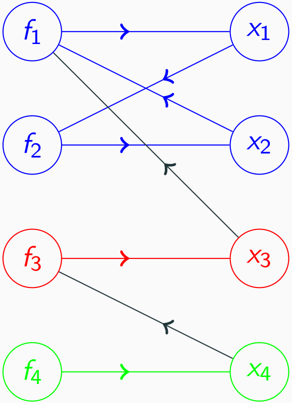
Dependency graph

- (Single-mode) Dependency graph: saturated edges are directed
  - $i \rightsquigarrow j$ for an edge in the chosen transversal
  - $j \rightsquigarrow i$ for other edges
- Symbolic translation is quite straightforward



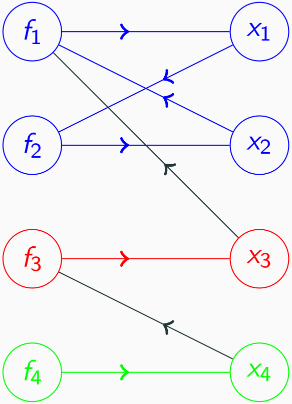
Computing the SCCs

- Strongly Connected Components: minimal blocks of equations for the numerical solving



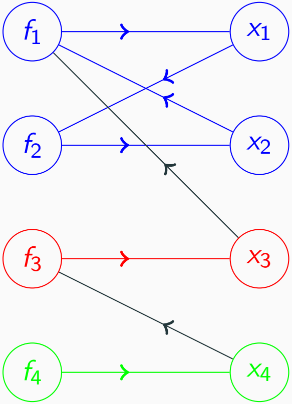
Computing the SCCs

- Strongly Connected Components: minimal blocks of equations for the numerical solving
- Standard tool: Tarjan's algorithm



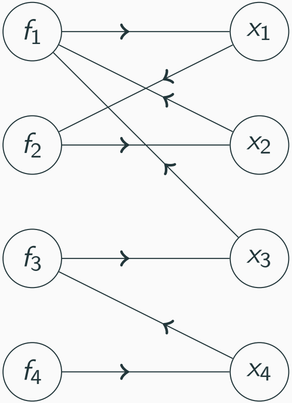
Computing the SCCs

- Strongly Connected Components: minimal blocks of equations for the numerical solving
- Standard tool: Tarjan's algorithm
  - Not suited in multimode: depth-first search approach can require the enumeration of modes



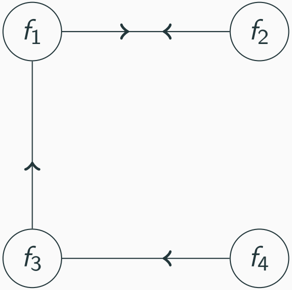
Computing the SCCs

- Parametrizing the naive approach:



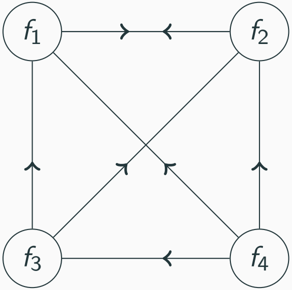
Computing the SCCs

- Parametrizing the naive approach:
  - **Equation dependency graph:**
    $(f \rightsquigarrow g) \Leftrightarrow (f \rightsquigarrow x) \wedge (x \rightsquigarrow g)$



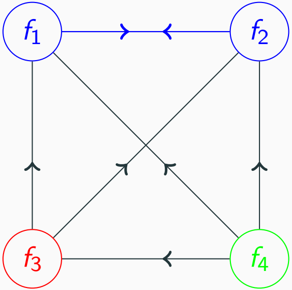
Computing the SCCs

- Parametrizing the naive approach:
  - **Equation dependency graph:** $(f \rightsquigarrow g) \Leftrightarrow (f \rightsquigarrow x) \wedge (x \rightsquigarrow g)$
  - **Transitive closure:** $(f \rightsquigarrow g) \wedge (g \rightsquigarrow h) \Rightarrow (f \rightsquigarrow h)$; iterate until convergence (pretty inexpensive with adapted data structures)



Computing the SCCs

- Parametrizing the naive approach:
  - **Equation dependency graph:** $(f \rightsquigarrow g) \Leftrightarrow (f \rightsquigarrow x) \wedge (x \rightsquigarrow g)$
  - **Transitive closure:** $(f \rightsquigarrow g) \wedge (g \rightsquigarrow h) \Rightarrow (f \rightsquigarrow h)$; iterate until convergence (pretty inexpensive with adapted data structures)
  - **SCCs:** $g \in SCC(f) \Leftrightarrow (f \rightsquigarrow g) \wedge (g \rightsquigarrow f)$ Equivalence relation, i.e., function $M \times E \times E \rightarrow \mathbb{B}$



Mode-dependent scheduling graph

- Several "splitting" steps:

Mode-dependent scheduling graph

- Several "splitting" steps:
    - Create the equation blocks (i.e., SCCs; implicitly declared until this step)

Mode-dependent scheduling graph

- Several "splitting" steps:
  - Create the equation blocks (i.e., SCCs; implicitly declared until this step)
  - Give each equation its differentiation order c_i

Mode-dependent scheduling graph

- Several ”splitting” steps:
  - Create the equation blocks (i.e., SCCs; implicitly declared until this step)
  - Give each equation its differentiation order c_i
  - Look at the inputs and outputs of the block (essential for code generation)

Mode-dependent scheduling graph

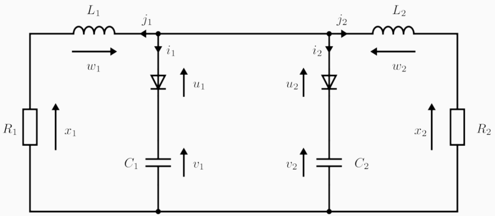
- Several "splitting" steps:
  - Create the equation blocks (i.e., SCCs; implicitly declared until this step)
  - Give each equation its differentiation order c_i
  - Look at the inputs and outputs of the block (essential for code generation)
- Not computationally expensive, actually:
  - Blocks tend to be localized, i.e., a few mode variables are involved for each block

Mode-dependent scheduling graph

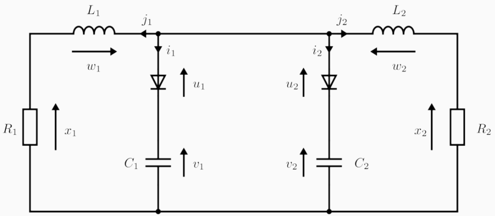
- Several "splitting" steps:
  - Create the equation blocks (i.e., SCCs; implicitly declared until this step)
  - Give each equation its differentiation order c_i
  - Look at the inputs and outputs of the block (essential for code generation)
- Not computationally expensive, actually:
  - Blocks tend to be localized, i.e., a few mode variables are involved for each block
- Check the dependencies between the blocks

The RLDC2 example

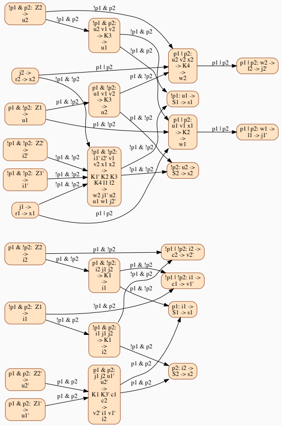
The RLDC2 example



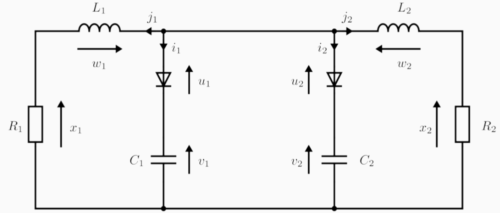
The RLDC2 example



- Conditional block dependency graph

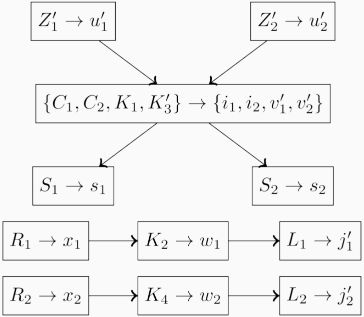


The RLDC2 example

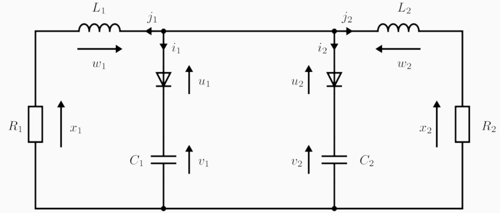


- Conditional block dependency graph
- Strong mode dependency on equation blocks and their structure

When both diodes are passing:

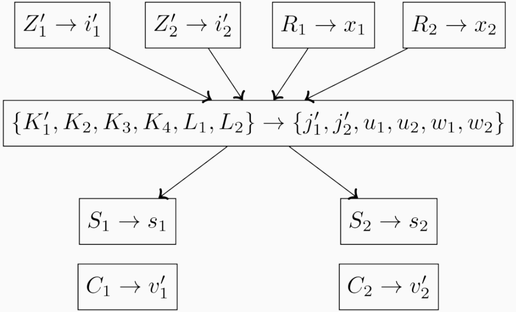


The RLDC2 example

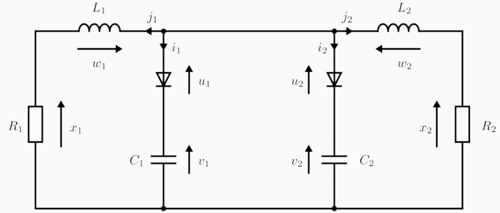


- Conditional block dependency graph
- Strong mode dependency on equation blocks and their structure

When both diodes are blocking:

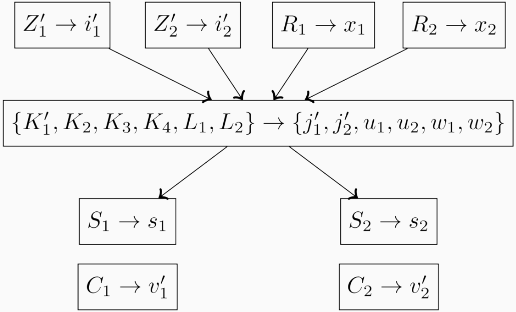


The RLDC2 example



- Conditional block dependency graph
- Strong mode dependency on equation blocks and their structure

When both diodes are blocking:



Executing the RLDC2 model (1/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2

L.V.

i_1	i_2	j_1	j_2	s_1	s_2	u_1	u_2	v_1	v_2	w_1	w_2	x_1	x_2

- Fixed-size state vector with all possible state variables
  - maximal values of the d_j's throughout the modes are known

- Fixed-size leading variables vector: one per variable
  - actual d_j implied (given by the block dependency graph)

Here, simulation is performed on two threads with shared memory; no assumption about a strategy for picking the next block to solve

Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

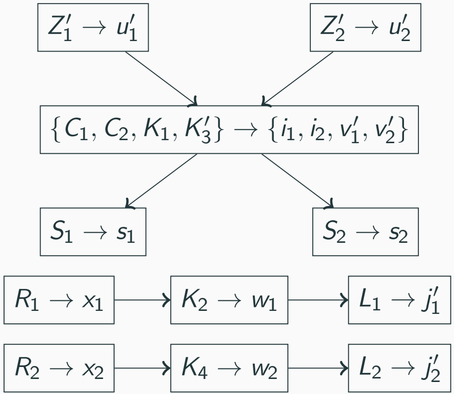
i_1	i_2	j_1	j_2	s_1	s_2	u_1	u_2	v_1	v_2	w_1	w_2	x_1	x_2

Thread 1:

$Z_1' \to u_1'$

Thread 2:

$Z_2' \to u_2'$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

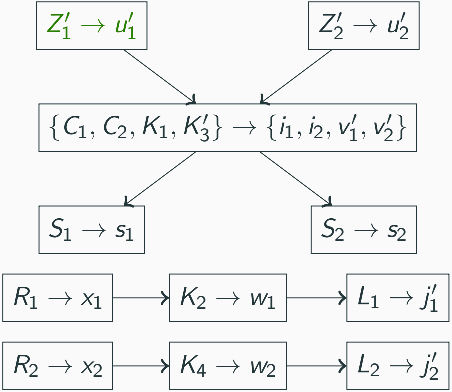
i_1	i_2	j_1	j_2	s_1	s_2	u_1'	u_2	v_1	v_2	w_1	w_2	x_1	x_2
						$\star$							

Thread 1:

$Z_1' \to u_1'$

Thread 2:

$Z_2' \to u_2'$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

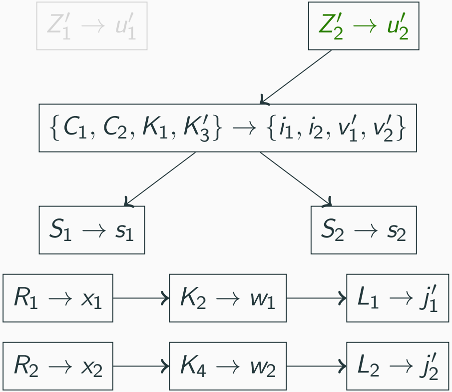
i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v_1	v_2	w_1	w_2	x_1	x_2
						$\star$	$\star$						

Thread 1:

$j_1 : R_1 \to x_1$

Thread 2:

$Z'_2 \to u'_2$



Executing the RLDC2 model (2/2)

S.V.

		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2

L.V.

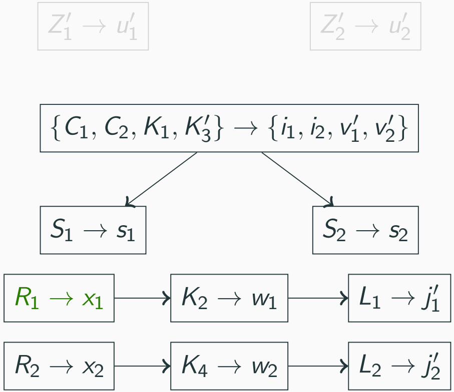
						$\star$	$\star$					$\star$	
i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v_1	v_2	w_1	w_2	x_1	x_2

Thread 1:

$j_1 : R_1 \to x_1$

Thread 2:

$j_1, j_2, u'_1, u'_2 : \{C_1, C_2, K_1, K'_3\} \to \{i_1, i_2, v'_1, v'_2\}$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

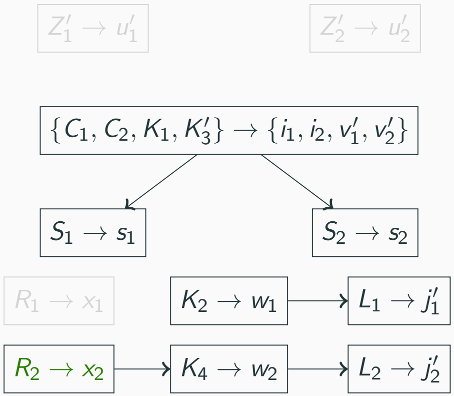
i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v_1	v_2	w_1	w_2	x_1	x_2
						$\star$	$\star$					$\star$	$\star$

Thread 1:

$j_2 : R_2 \rightarrow x_2$

Thread 2:

$j_1, j_2, u'_1, u'_2 : \{C_1, C_2, K_1, K'_3\} \rightarrow \{i_1, i_2, v'_1, v'_2\}$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

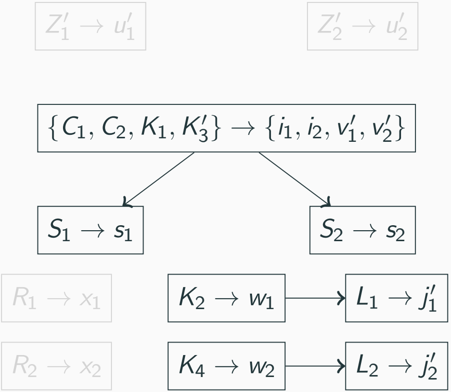
i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v_1	v_2	w_1	w_2	x_1	x_2
						$\star$	$\star$			$\star$		$\star$	$\star$

Thread 1:

$u_1, v_1, x_1 : K_2 \to w_1$

Thread 2:

$j_1, j_2, u'_1, u'_2 : \{C_1, C_2, K_1, K'_3\} \to \{i_1, i_2, v'_1, v'_2\}$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

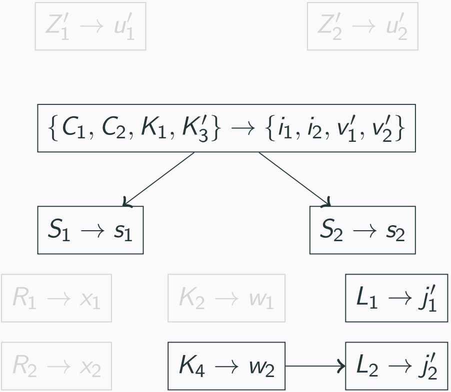
i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2
$\star$	$\star$					$\star$	$\star$	$\star$	$\star$	$\star$		$\star$	$\star$

Thread 1:

$u_2, v_2, x_2 : K_4 \rightarrow w_2$

Thread 2:

$j_1, j_2, u'_1, u'_2 : \{C_1, C_2, K_1, K'_3\} \rightarrow \{i_1, i_2, v'_1, v'_2\}$



Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

i_1	i_2	j_1	j_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2
$\star$	$\star$					$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

Thread 1:

$u_2, v_2, x_2 : K_4 \to w_2$

Thread 2:

$w_1 : L_1 \to j'_1$

$Z'_1 \to u'_1$ $\quad$ $Z'_2 \to u'_2$

$\{C_1, C_2, K_1, K'_3\} \to \{i_1, i_2, v'_1, v'_2\}$

$\boxed{S_1 \to s_1}$ $\quad$ $\boxed{S_2 \to s_2}$

$R_1 \to x_1$ $\quad$ $K_2 \to w_1$ $\quad$ $\boxed{L_1 \to j'_1}$

$R_2 \to x_2$ $\quad$ $\boxed{K_4 \to w_2} \longrightarrow \boxed{L_2 \to j'_2}$

Executing the RLDC2 model (2/2)

S.V.

		⋆	⋆	⋆	⋆	⋆	⋆
i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2

L.V.

⋆	⋆	⋆				⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
i_1	i_2	j'_1	j_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2

$Z'_1 \to u'_1$ $Z'_2 \to u'_2$

Thread 1:

$\{C_1, C_2, K_1, K'_3\} \to \{i_1, i_2, v'_1, v'_2\}$

$w_2 : L_2 \to j'_2$

$\boxed{S_1 \to s_1}$ $\boxed{S_2 \to s_2}$

Thread 2:

$R_1 \to x_1$ $K_2 \to w_1$ $\boxed{L_1 \to j'_1}$

$w_1 : L_1 \to j'_1$

$R_2 \to x_2$ $K_4 \to w_2$ $\boxed{L_2 \to j'_2}$

Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

i_1	i_2	j'_1	j'_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2
$\star$	$\star$	$\star$	$\star$			$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

Thread 1:

$w_2 : L_2 \to j'_2$

Thread 2:

$i_1 : S_1 \to s_1$

$\boxed{Z'_1 \to u'_1}$ $\boxed{Z'_2 \to u'_2}$

$\boxed{\{C_1, C_2, K_1, K'_3\} \to \{i_1, i_2, v'_1, v'_2\}}$

$\boxed{S_1 \to s_1}$ $\boxed{S_2 \to s_2}$

$\boxed{R_1 \to x_1}$ $\boxed{K_2 \to w_1}$ $\boxed{L_1 \to j'_1}$

$\boxed{R_2 \to x_2}$ $\boxed{K_4 \to w_2}$ $\boxed{L_2 \to j'_2}$

Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

i_1	i_2	j'_1	j'_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2
$\star$	$\star$	$\star$	$\star$	$\star$		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

$\boxed{Z'_1 \rightarrow u'_1}$ $\boxed{Z'_2 \rightarrow u'_2}$

Thread 1:

$\boxed{\{C_1, C_2, K_1, K'_3\} \rightarrow \{i_1, i_2, v'_1, v'_2\}}$

$i_2 : S_2 \rightarrow s_2$

$\boxed{S_1 \rightarrow s_1}$ $\boxed{S_2 \rightarrow s_2}$

Thread 2:

$\boxed{R_1 \rightarrow x_1}$ $\boxed{K_2 \rightarrow w_1}$ $\boxed{L_1 \rightarrow j'_1}$

$i_1 : S_1 \rightarrow s_1$

$\boxed{R_2 \rightarrow x_2}$ $\boxed{K_4 \rightarrow w_2}$ $\boxed{L_2 \rightarrow j'_2}$

Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

i_1	i_2	j_1'	j_2'	s_1	s_2	u_1'	u_2'	v_1'	v_2'	w_1	w_2	x_1	x_2
$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

Thread 1:

$i_2 : S_2 \rightarrow s_2$

Thread 2:

$Z_1' \rightarrow u_1'$ $Z_2' \rightarrow u_2'$

$\{C_1, C_2, K_1, K_3'\} \rightarrow \{i_1, i_2, v_1', v_2'\}$

$S_1 \rightarrow s_1$ $\boxed{S_2 \rightarrow s_2}$

$R_1 \rightarrow x_1$ $K_2 \rightarrow w_1$ $L_1 \rightarrow j_1'$

$R_2 \rightarrow x_2$ $K_4 \rightarrow w_2$ $L_2 \rightarrow j_2'$

Executing the RLDC2 model (2/2)

S.V.

i_1	i_2	j_1	j_2	u_1	u_2	v_1	v_2
		$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

L.V.

i_1	i_2	j'_1	j'_2	s_1	s_2	u'_1	u'_2	v'_1	v'_2	w_1	w_2	x_1	x_2
$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$

Thread 1:

$Z'_1 \rightarrow u'_1$ $\quad$ $Z'_2 \rightarrow u'_2$

$\{C_1, C_2, K_1, K'_3\} \rightarrow \{i_1, i_2, v'_1, v'_2\}$

Thread 2:

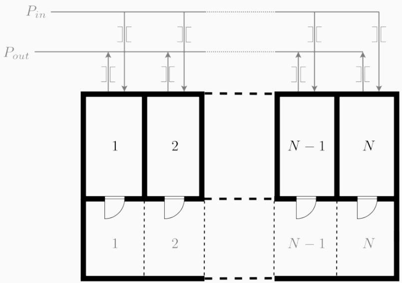
$S_1 \rightarrow s_1$ $\quad$ $S_2 \rightarrow s_2$

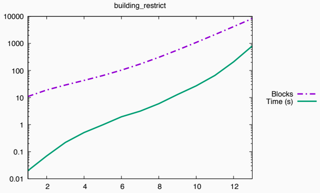
$R_1 \rightarrow x_1$ $\quad$ $K_2 \rightarrow w_1$ $\quad$ $L_1 \rightarrow j'_1$

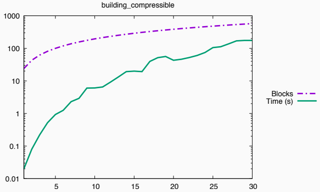
$R_2 \rightarrow x_2$ $\quad$ $K_4 \rightarrow w_2$ $\quad$ $L_2 \rightarrow j'_2$

Scalability

A thermal model of an office building







Two variants:

- Incompressible air: singular when all doors closed and does not scale up
- Compressible air: scales up (number of blocks linear in N)

Conclusion

Conclusion

Results:

- Structural analysis methods for multimode DAE systems
- Extending Pryce's Σ-method
- Index reduction for all modes, with no mode enumeration
- Handles varying dimension, varying structure, varying index systems
- Software: IsamDAE `https://allgo18.inria.fr/apps/isamdae`

Conclusion

IsamDAE:

- Implementation in OCaml based on Arlen Cox's MLBDD package
- Tested on moderately large models (10^3 equations, 2^{80} modes)
- Please try the web version `https://allgo18.inria.fr/apps/isamdae`
- To do:
    - Structural analysis of mode changes (strong assumption: logico-numerical fixed-point equations are rejected, ie. requires infinitesimal delay between guard and equation)
    - Detecting impulsive mode changes
    - Interfacing with Dymola (collab. with Dassault Systèmes)

Conclusion

Open questions:

- Compositional structural analysis (divide and conquer approach) exploiting the topology of the model
- Handling linear equations with integer coefficients (connectors, Kirchhoff equations...)
- Understanding the relationship btw. mDAE and Complementarity Systems (Christelle Kozaily's PhD work)

Thank you

Questions?