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)

- Reconfigurable systems (appearance/disappearance of components)
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

```plaintext
model SimpleDrive
  .Rotational.Inertia Inertial (J=0.002);
  .Rotational.IdealGear IdealGear1(ratio=100)
  .Basic.Resistor Resistor1 (R=0.2)
  ...
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;
```

```plaintext
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 $= \text{algebraic constraints} (\neq \text{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

  \[
  x^2 + y^2 = 1; \\
  \text{der}(x) + u = 0; \\
  u = \text{if } x \geq 0 \text{ then } x+y \text{ else } y; \\
  \text{when } x \leq 0 \text{ do reinit}(x,1); \text{ end}; \\
  \text{when } y \leq 0 \text{ do reinit}(y,x); \text{ end};
  \]

- **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)

- 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
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)
• "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 ⇒ $10^{15}$ modes for a 50-component system

```plaintext
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.BlocksInterfaces.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 ⇒ $10^{15}$ 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 ⇒ \(10^{15}\) 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 ⇒ $10^{15}$ 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 ⇒ represent the structure of a mDAE as functions $M \rightarrow \mathbb{N}/\mathbb{B}$ of the modes
Structural Analysis of 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.
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 \( \sigma_{ij} \) the highest differentiation order of \( x_j \) in equation \( f_i \). The leading variables of \( F \) are \( x_j^{(\sigma_j)} \) with \( \sigma_j = \max_i \sigma_{ij} \).
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 \( \sigma_{ij} \) 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)

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

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

\[
J = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

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

\[ (S) \begin{align*}
    x'' + Tx &= 0 \\
    y'' + Ty - g &= 0 \\
    x^2 + y^2 - l^2 &= 0
\end{align*} \]

\( T \) 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:

\[
\begin{aligned}
S' &= \begin{cases}
  x'' + Tx &= 0 \\
  y'' + Ty - g &= 0 \\
  2xx'' + 2(x')^2 + 2yy'' + 2(y')^2 &= 0
\end{cases} \\
J' = \begin{pmatrix}
1 & 0 & x \\
0 & 1 & y \\
2x & 2y & 0
\end{pmatrix}
\end{aligned}
\]

The Jacobian $J'$ is invertible.
Example: a pendulum (Cartesian coordinates)

However, if the third equation is differentiated twice:

\[
(S') \begin{cases}
  x'' + Tx &= 0 \\
y'' + Ty - g &= 0 \\
2xx'' + 2(x')^2 + 2yy'' + 2(y')^2 &= 0
\end{cases}
\]

The Jacobian \( J' \) is invertible

How can one determine automatically which equations have to be differentiated, and how many times?
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? \( \sigma_{ij} \): highest differentiation order of \( x_j \) in \( f_i \).

- Uses graph theoretic algorithms (e.g., Pantelides method).
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 ⇒ no determinant needs to be computed

• Retains useful information: which variables appear (with what differentiation order) in which equation? ($\sigma_{ij}$ : highest differentiation order of $x_j$ in $f_i$)
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? ($\sigma_{ij}$: highest differentiation order of $x_j$ in $f_i$).
- Uses graph theoretic algorithms (e.g., Pantelides method).
Highlights on several methods

- Structural analysis methods:
  - Pantelides (1988)
  - Weighed Bipartite Graph method [Ding et al. 2008]
  - $\Sigma$-method [Pryce 2001]
  - $\sigma$-$\nu$ method [Chowdhry et al. 2004]

- Several implementations (Modelica tools, Mathematica...)
Σ-matrix representation of the Pendulum

\[
\begin{align*}
  f_1 &= x'' - Tx \\
  f_2 &= y'' - Ty + g \\
  f_3 &= x^2 + y^2 - L^2
\end{align*}
\]
Σ-matrix representation of the Pendulum

\[
\begin{align*}
  f_1 &= x'' - T x \\
  f_2 &= y'' - T y + g \\
  f_3 &= x^2 + y^2 - L^2
\end{align*}
\]

Variable order:

\[X = (x, y, T)\]
Σ-matrix representation of the Pendulum

\[
\begin{align*}
    f_1 &= x'' - Tx \\
    f_2 &= y'' - Ty + g \\
    f_3 &= x^2 + y^2 - L^2
\end{align*}
\]

Variable order:

\[X = (x, y, T)\]
\[ \begin{align*}
    f_1 &= x'' - Tx \\
    f_2 &= y'' - Ty + g \\
    f_3 &= x^2 + y^2 - L^2
\end{align*} \]

Variable order:

\[ X = (x, y, T) \]

\[ \Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 0 & - \end{pmatrix} \]
$\Sigma$-matrix representation of the Pendulum

\[
\begin{align*}
    f_1 &= x'' - Tx \\
    f_2 &= y''' - Ty + g \\
    f_3 &= x^2 + y^2 - L^2
\end{align*}
\]

Variable order:

\[X = (x, y, T)\]

\[
\Sigma = \begin{pmatrix}
    2 & -2 & 0 \\
    -2 & 2 & 0
\end{pmatrix}
\]
\[ \begin{align*}
  f_1 &= x'' - Tx \\
  f_2 &= y'' - Ty + g \\
  f_3 &= x^2 + y^2 - L^2
\end{align*} \]

Variable order:

\[ X = (x, y, T) \]

\[ \Sigma = \begin{pmatrix}
  2 & - & 0 \\
  - & 2 & 0 \\
  0 & 0 & -
\end{pmatrix} \]
The $\Sigma$-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 $\Sigma$
The \( \Sigma \)-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 \( \Sigma \)

**Dual problem:** Compute the minimal solution of the linear program

\[
(P_{off}) : \quad \min \hat{z} = \sum_j d_j - \sum_i c_i \\
\text{s.t.} \quad d_j - c_i \geq \sigma_{ij} \quad \forall (i, j) \in S \\
c_i \geq 0 \quad 1 \leq i \leq n
\]

with a **fixed-point** method using the maximal weight transverse
The \( \Sigma \)-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 \( \Sigma \)

**Dual problem:** Compute the minimal solution of the linear program

\[
(\text{P}_\text{off}) : \quad \min \; \hat{z} = \sum_j d_j - \sum_i c_i \\
\text{s.t.} \quad d_j - c_i \geq \sigma_{ij} \quad \forall (i,j) \in S \\
\quad c_i \geq 0 \quad 1 \leq i \leq n
\]

with a fixed-point method using the maximal weight transverse

**Result:**  
- \( c_i \) = number of times equations must be differentiated
**The $\Sigma$-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 $\Sigma$

**Dual problem:** Compute the minimal solution of the linear program

$$(P_{\text{off}}): \min \, \hat{z} = \sum_j d_j - \sum_i c_i$$

s.t. $d_j - c_i \geq \sigma_{ij} \quad \forall (i,j) \in S$

$c_i \geq 0 \quad 1 \leq i \leq n$

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

\[ \Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ 0 & 0 & - \end{pmatrix} \]
A solution to the Primal problem:

\[ \Sigma = \begin{pmatrix} 2 & - & 0 \\ - & 2 & 0 \\ 0 & 0 & - \end{pmatrix} \]
Back to the Pendulum example

\[ c_i \begin{cases} 
0 & 2 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 
\end{cases} \]

\[ d_j \]

\[ \begin{array}{c}
\text{\( f_1 \)} \\
\text{\( f_2 \)} \\
\text{\( f_3 \)} \\
\end{array} \quad 2 \quad \begin{array}{c}
\text{\( x \)} \\
\text{\( y \)} \\
\text{\( T \)} \\
\end{array} \]

\[ \text{\( f_1 \)} \quad 2 \quad \text{\( f_2 \)} \quad \text{\( f_3 \)} \]

\[ \text{\( x \)} \quad 0 \quad \text{\( y \)} \quad 0 \]

\[ \text{\( T \)} \quad 0 \quad \text{\( T \)} \quad 0 \]
Back to the Pendulum example

\[
c_i \begin{cases} 
0 & 2 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\hline
2 & & \\
\end{cases}
\]

\[
d_j
\]
Back to the Pendulum example

\[
\begin{align*}
\begin{array}{c|cc}
  c_i & 2 & 0 \\
  0 & 2 & 0 \\
  0 & 0 & 0 \\
  2 & 2 & \\
\end{array}
\end{align*}
\]

\[
d_j
\]

\[
\text{Diagram:}
\begin{align*}
  f_1 & \rightarrow x \\
  f_2 & \rightarrow y \\
  f_3 & \rightarrow T
\end{align*}
\]
Back to the Pendulum example

\[
\begin{array}{l}
\{ \begin{array}{c}
0 \quad 2 \quad 0 \\
0 \quad 2 \quad 0 \\
0 \quad 0 \quad 0 \\
\end{array} \}
\]

\[
c_i \quad \{ \begin{array}{c}
0 \quad 2 \quad 0 \\
0 \quad 2 \quad 0 \\
0 \quad 0 \quad 0 \\
\end{array} \}
\]

\[
d_j
\]

\[
\begin{array}{c}
f_1 \quad 2 \quad x \\
f_2 \quad 2 \quad y \\
f_3 \quad 0 \quad T \\
\end{array}
\]
Back to the Pendulum example

\[ c_i \begin{cases} 
0 & 2 & 0 \\
0 & 2 & 0 \\
2 & 0 & 0 \\
2 & 2 & 0 
\end{cases} \]

\[ d_j \]
Back to the Pendulum example

\[
\begin{pmatrix}
\begin{array}{c|ccc}
  & 2 & 0 \\
\hline
0 & 2 & 0 \\
0 & 2 & 0 \\
2 & 0 & 0 \\
\end{array}
\end{pmatrix}
\]

Fixed-point has been reached ⇒ 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

  \[ g : \text{boolean} = x > 1.e-2 \]
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
  \[ g : \text{boolean} = x > 1.\text{e}{-2} \]

- Invariants are used to narrow the structural analysis to particular modes
  \[ \text{invariant liq | gas} \]
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...

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

...and equations...

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

```plaintext
foreach k in 1 .. n do
  x[k] : real
done
```
MEL: a toy mDAE modeling language

- MEL: \textit{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 \textit{flexibility} for future experiments & tests

- foreach loops and arrays, to define parametric models

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

- Also: \texttt{parameters}, uninterpreted \texttt{nonlinear functions}, \texttt{when <event>} \texttt{then <statements>} \texttt{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

\[ \begin{align*}
0 \leq i & \perp -u \geq 0 \\
- & \quad \quad \\
\end{align*} \]
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

\[ 0 \leq i \perp -u \geq 0 \]
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

\[ 0 \leq i \perp -u \geq 0 \]
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 : boolean = last(s1);
S1 : equation s1 = if p1 then i1 else -u1;
Z1 : equation 0 = if p1 then u1 else i1;

// Diode 2
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
**Warning:** In this talk we do not deal with mode changes. Assume that solutions are continuous.
• **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
• **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 $O(1)$, other operations include:
  - Conjunction/disjunction: $\land/\lor$
  - Existential quantification: $\exists a. f(a, b)$
  - Universal quantification: $\forall a. f(a, b)$
• **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: $\land/\lor$

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

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

• However: very sensitive to variable and computation ordering
Structural Analysis
Σ-matrix coefficients for a single-mode DAE:

\[
\begin{align*}
    f_1(x_1, x_1', \ldots, x_1^{(\sigma_{1,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{1,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{1,n})}) &= 0 \\
    f_2(x_1, x_1', \ldots, x_1^{(\sigma_{2,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{2,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{2,n})}) &= 0 \\
    \vdots \\
    f_n(x_1, x_1', \ldots, x_1^{(\sigma_{n,1})}, x_2, x_2', \ldots, x_2^{(\sigma_{n,2})}, \ldots, x_n, x_n', \ldots, x_n^{(\sigma_{n,n})}) &= 0
\end{align*}
\]

Convention: \( x_j \) does not appear in \( f_i \) \( \Rightarrow \sigma_{i,j} = -\infty \)
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 $\Sigma$-method (1/4)

$\Sigma$-matrix coefficients for a single-mode DAE:

\[
\begin{align*}
   f_1(x_1, x'_1, \ldots, x_1^{(\sigma_{1,1,m})}, x_2, x'_2, \ldots, x_2^{(\sigma_{1,2,m})}, \ldots, x_n, x'_n, \ldots, x_n^{(\sigma_{1,n,m})}) &= 0 \\
   f_2(x_1, x'_1, \ldots, x_1^{(\sigma_{2,1,m})}, x_2, x'_2, \ldots, x_2^{(\sigma_{2,2,m})}, \ldots, x_n, x'_n, \ldots, x_n^{(\sigma_{2,n,m})}) &= 0 \\
   \vdots \\
   f_n(x_1, x'_1, \ldots, x_1^{(\sigma_{n,1,m})}, x_2, x'_2, \ldots, x_2^{(\sigma_{n,2,m})}, \ldots, x_n, x'_n, \ldots, x_n^{(\sigma_{n,n,m})}) &= 0 
\end{align*}
\]

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

Auxiliary functions: $\chi_I : M \times I \rightarrow \mathbb{B}$, $\chi_J : M \times J \rightarrow \mathbb{B}$ and $\chi_E : M \times E \rightarrow \mathbb{B}$
characteristic functions of the set of active equations, variables and incidence edges
The **primal problem** is solved in the following way:

- Encode constraints as functions $M \rightarrow B$.
- $\mu$: "an active equation must be matched to a variable".
- $\nu$: "...and vice-versa".
- $\Upsilon$: "an edge can only be part of a matching if it is active".

$X := \Upsilon \land \mu \land \nu$ describes all perfect matchings in all modes.

Apply a (parametrized) ArgMax operator using edge weights $\Rightarrow$ only keep maximum weight perfect matchings.
The primal problem is solved in the following way:

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

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

- $X := \Upsilon \land \mu \land \nu$ describes all perfect matchings in all modes
The primal problem is solved in the following way:

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

- $X := \Upsilon \land \mu \land \nu$ describes all perfect matchings in all modes
- Apply a (parametrized) ArgMax operator using edge weights
  $\Rightarrow$ only keep maximum weight perfect matchings
Problem:
Given $\varphi$ and $(w_k)_{k=0}^{N-1}$, compute:

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

Algorithm:

$$\text{maxb}_k(\gamma) = \gamma \land (\pi \iff w_k)$$
with:

$$\pi = \exists \vec{V}, \gamma \land w_k$$

$$\psi = \psi_0$$

$$\psi_k = \text{maxb}_k(\psi_{k+1}) \text{ for all } k < N$$

$$\psi_N = \varphi$$
The dual problem is solved by "parametrizing" everything.
The dual problem is solved by "parametrizing" everything:

- Standard (single-mode) fixpoint iteration:
  \[ \forall j, \quad d_j \leftarrow \max_i (\sigma_{ij} + c_i) \]
  \[ \forall i, \quad c_i \leftarrow d_j - \sigma_{i,j} \]

with all \( c_i \)'s and \( d_j \)'s initialized to 0
The dual problem is solved by "parametrizing" everything

- Standard (single-mode) fixpoint iteration:

\[ \forall j, \, d_j \leftarrow \max_i (\sigma_{ij} + c_i) \]
\[ \forall i, \, c_i \leftarrow d_{ji} - \sigma_{i,j} \]

with all \( c_i \)'s and \( d_j \)'s initialized to 0

- Parametrized (multimode) fixpoint iteration:

\[ \forall j, \, d_j \equiv \begin{cases} \text{if } \chi_J(j) & \text{then } \max_e \{ \text{if } \chi_E(e) & \text{then } \sigma_{i,j} + c_i \text{ else } 0 \} \text{ else } 0 \end{cases} \]
\[ \begin{array}{c} \mathcal{M} \rightarrow \mathbb{N} \end{array} \]

\[ \forall i, \, c_i \equiv \begin{cases} \text{if } \chi_I(i) & \text{then } \max_e \{ \text{if } (\chi_J(j) \land T(e)) & \text{then } d_j - \sigma_{i,j} \text{ else } c_i \} \text{ else } 0 \end{cases} \]
\[ \begin{array}{c} \mathcal{M} \rightarrow \mathbb{N} \end{array} \]

with all \( c_i \)'s and \( d_j \)'s initialized to zero functions
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
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
(Single-mode) Dependency graph:

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

$\varepsilon_1 \varepsilon_2 \varepsilon_3 \varepsilon_4$
(Single-mode) Dependency graph: saturated edges are directed
  - $i \leadsto j$ for an edge in the chosen transversal
  - $j \leadsto i$ for other edges
• (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

![Diagram of strongly connected components]
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

\[
\begin{array}{cccc}
  f_1 & x_1 \\
  f_2 & x_2 \\
  f_3 & x_3 \\
  f_4 & x_4 \\
\end{array}
\]
Computing the SCCs

- Parametrizing the naive approach:

\[
(f \rightarrow g) \iff (f \rightarrow x) \land (x \rightarrow g)
\]

- Transitive closure:

\[
(f \rightarrow g) \land (g \rightarrow h) \Rightarrow (f \rightarrow h)
\]

Iterate until convergence (pretty inexpensive with adapted data structures)

- SCCs:

\[
g \in SCC(f) \iff (f \rightarrow g) \land (g \rightarrow f)
\]

Equivalence relation, i.e., function $M \times E \times E \rightarrow B$
Computing the SCCs

- Parametrizing the naive approach:
  - **Equation dependency graph:**
    \[(f \leadsto g) \iff (f \leadsto x) \land (x \leadsto g)\]

- Transitive closure:
  \[(f \leadsto g) \land (g \leadsto h) \Rightarrow (f \leadsto h)\]; iterate until convergence

- SCCs:
  \[g \in SCC(f) \iff (f \leadsto g) \land (g \leadsto f)\]

- Equivalence relation, i.e., function
  \[M \times E \times E \to B\]
Computing the SCCs

- Parametrizing the naive approach:
  - **Equation dependency graph:**
    \((f \leadsto g) \iff (f \leadsto x) \land (x \leadsto g)\)
  - **Transitive closure:**
    \((f \leadsto g) \land (g \leadsto h) \Rightarrow (f \leadsto h)\);
    iterate until convergence
    (pretty inexpensive with adapted data structures)
Computing the SCCs

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

\[ Z'_1 \rightarrow u'_1 \]
\[ Z'_2 \rightarrow u'_2 \]
\[ \{C_1, C_2, K_1, K'_3\} \rightarrow \{i_1, i_2, v'_1, v'_2\} \]
\[ S_1 \rightarrow s_1 \]
\[ S_2 \rightarrow s_2 \]
\[ R_1 \rightarrow x_1 \]
\[ K_2 \rightarrow w_1 \]
\[ L_1 \rightarrow j'_1 \]
\[ R_2 \rightarrow x_2 \]
\[ K_4 \rightarrow w_2 \]
\[ L_2 \rightarrow j'_2 \]
The RLDC2 example

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

When both diodes are blocking:

\[ Z'_1 \rightarrow i'_1 \quad Z'_2 \rightarrow i'_2 \quad R_1 \rightarrow x_1 \quad R_2 \rightarrow x_2 \]

\[ \{K'_1, K_2, K_3, K_4, L_1, L_2\} \rightarrow \{j'_1, j'_2, u_1, u_2, w_1, w_2\} \]

- \( S_1 \rightarrow s_1 \)
- \( S_2 \rightarrow s_2 \)
- \( C_1 \rightarrow v'_1 \)
- \( C_2 \rightarrow v'_2 \)
The RLDC2 example

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

When both diodes are blocking:

\[ Z_1' \rightarrow i_1' \quad Z_2' \rightarrow i_2' \quad R_1 \rightarrow x_1 \quad R_2 \rightarrow x_2 \]

\[ \{K_1', K_2, K_3, K_4, L_1, L_2\} \rightarrow \{j_1', j_2', u_1, u_2, w_1, w_2\} \]

\[ S_1 \rightarrow s_1 \quad S_2 \rightarrow s_2 \]

\[ C_1 \rightarrow v_1' \quad C_2 \rightarrow v_2' \]
Executing the RLDC2 model (1/2)

- 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)

<table>
<thead>
<tr>
<th>S.V.</th>
<th>L.V.</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_1</td>
<td>i_1</td>
</tr>
<tr>
<td>i_2</td>
<td>i_2</td>
</tr>
<tr>
<td>j_1</td>
<td>j_1</td>
</tr>
<tr>
<td>j_2</td>
<td>j_2</td>
</tr>
<tr>
<td>u_1</td>
<td>u_1</td>
</tr>
<tr>
<td>u_2</td>
<td>u_2</td>
</tr>
<tr>
<td>v_1</td>
<td>v_1</td>
</tr>
<tr>
<td>v_2</td>
<td>v_2</td>
</tr>
<tr>
<td>s_1</td>
<td>s_1</td>
</tr>
<tr>
<td>s_2</td>
<td>s_2</td>
</tr>
<tr>
<td>u_2</td>
<td>u_2</td>
</tr>
<tr>
<td>v_2</td>
<td>v_2</td>
</tr>
<tr>
<td>w_1</td>
<td>w_1</td>
</tr>
<tr>
<td>w_2</td>
<td>w_2</td>
</tr>
<tr>
<td>x_1</td>
<td>x_1</td>
</tr>
<tr>
<td>x_2</td>
<td>x_2</td>
</tr>
</tbody>
</table>

Thread 1:

\[ Z'_1 \rightarrow u'_1 \]

Thread 2:

\[ Z'_2 \rightarrow u'_2 \]
Executing the RLDC2 model (2/2)

**Thread 1:**

\[ Z'_1 \rightarrow u'_1 \]

Thread 2:

\[ Z'_2 \rightarrow u'_2 \]
Executing the RLDC2 model (2/2)

Thread 1:

\( j_1 : R_1 \rightarrow x_1 \)

Thread 2:

\( Z'_2 \rightarrow u'_2 \)
Executing the RLDC2 model (2/2)

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)

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. 

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
 & i_1 & i_2 & j_1 & j_2 & u_1 & u_2 & v_1 & v_2 \\
\end{array}
\]

L.V. 

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
 & i_1 & i_2 & j_1 & j_2 & s_1 & s_2 & u_1' & u_2' & v_1 & v_2 & w_1 & w_2 & x_1 & x_2 \\
\end{array}
\]

**Thread 1:**

\[u_1, v_1, x_1 : K_2 \rightarrow w_1\]

**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.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1$</td>
<td>$j_2$</td>
<td>$u_1$</td>
<td>$u_2$</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
</tbody>
</table>

### L.V.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1$</td>
<td>$j_2$</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$u'_1$</td>
<td>$u'_2$</td>
</tr>
</tbody>
</table>

#### 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.  

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1$</td>
<td>$j_2$</td>
<td>$u_1$</td>
<td>$u_2$</td>
<td>$v_1$</td>
<td>$v_2$</td>
<td></td>
</tr>
</tbody>
</table>

L.V.  

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1$</td>
<td>$j_2$</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$u'_1$</td>
<td>$v'_1$</td>
<td>$v'_2$</td>
</tr>
</tbody>
</table>

Thread 1:

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

Thread 2:

$w_1 : L_1 \rightarrow j'_1$

\[
Z'_1 \rightarrow u'_1 \quad Z'_2 \rightarrow u'_2
\]

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

\[
S_1 \rightarrow s_1 \quad S_2 \rightarrow s_2
\]

\[
R_1 \rightarrow x_1 \quad K_2 \rightarrow w_1 \quad L_1 \rightarrow j'_1
\]

\[
R_2 \rightarrow x_2 \quad K_4 \rightarrow w_2 \quad L_2 \rightarrow j'_2
\]
Executing the RLDC2 model (2/2)

<table>
<thead>
<tr>
<th>S.V.</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1$</td>
<td>$j_2$</td>
<td>$u_1$</td>
<td>$u_2$</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L.V.</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$i_2$</td>
<td>$j_1'$</td>
<td>$j_2$</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$u_1'$</td>
<td>$u_2'$</td>
<td>$v_1'$</td>
<td>$v_2'$</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$x_1$</td>
</tr>
</tbody>
</table>

**Thread 1:**

$w_2 : L_2 \rightarrow j_2'$

**Thread 2:**

$w_1 : L_1 \rightarrow j_1'$

$Z_1' \rightarrow u_1'$

$Z_2' \rightarrow u_2'$

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

$S_1 \rightarrow s_1$

$S_2 \rightarrow s_2$

$R_1 \rightarrow x_1$

$K_2 \rightarrow w_1$

$L_1 \rightarrow j_1'$

$R_2 \rightarrow x_2$

$K_4 \rightarrow w_2$

$L_2 \rightarrow j_2'$
Executing the RLDC2 model (2/2)

Thread 1:

\[ w_2 : L_2 \rightarrow j'_2 \]

Thread 2:

\[ i_1 : S_1 \rightarrow s_1 \]
Executing the RLDC2 model (2/2)

Thread 1:

\( i_2 : S_2 \rightarrow s_2 \)

Thread 2:

\( i_1 : S_1 \rightarrow s_1 \)
Executing the RLDC2 model (2/2)

<table>
<thead>
<tr>
<th>S.V.</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_1)</td>
<td>(i_2)</td>
<td>(j_1)</td>
<td>(j_2)</td>
<td>(u_1)</td>
<td>(u_2)</td>
<td>(v_1)</td>
<td>(v_2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L.V.</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_1)</td>
<td>(i_2)</td>
<td>(j_1')</td>
<td>(j_2')</td>
<td>(s_1)</td>
<td>(s_2)</td>
<td>(u_1')</td>
<td>(u_2')</td>
<td>(v_1')</td>
<td>(v_2')</td>
<td>(w_1)</td>
<td>(w_2)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td></td>
</tr>
</tbody>
</table>

**Thread 1:**

\(i_2 : S_2 \rightarrow s_2\)

**Thread 2:**

\(R_1 \rightarrow x_1\)
\(K_2 \rightarrow w_1\)
\(L_1 \rightarrow j_1'\)

\(R_2 \rightarrow x_2\)
\(K_4 \rightarrow w_2\)
\(L_2 \rightarrow j_2'\)

\(Z_1' \rightarrow u_1'\)
\(Z_2' \rightarrow u_2'\)

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

\(S_1 \rightarrow s_1\)
\(S_2 \rightarrow s_2\)
Executing the RLDC2 model (2/2)

Thread 1:

Thread 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
Results:

- Structural analysis methods for multimode DAE systems
- Extending Pryce’s $\Sigma$-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
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, i.e. 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?