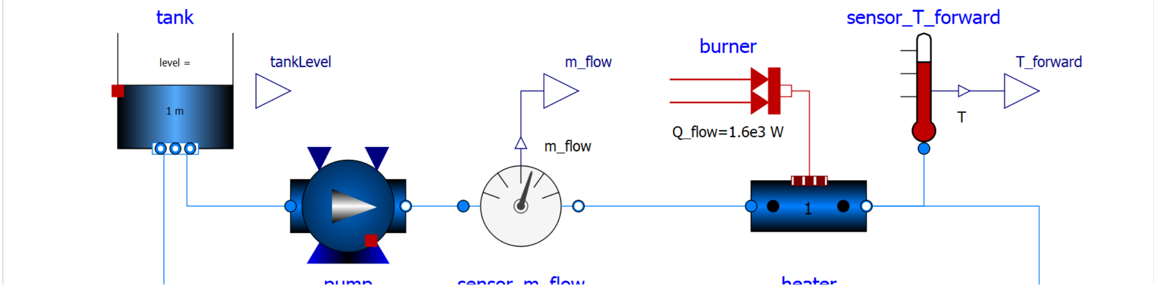


Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

OpenModelica Annual Workshop 2020

Andreas Heuermann Bernhard Bachmann

FH Bielefeld
University of Applied Science
Faculty of Engineering and Mathematics

FH Bielefeld
University of
Applied Sciences

Table of Contents

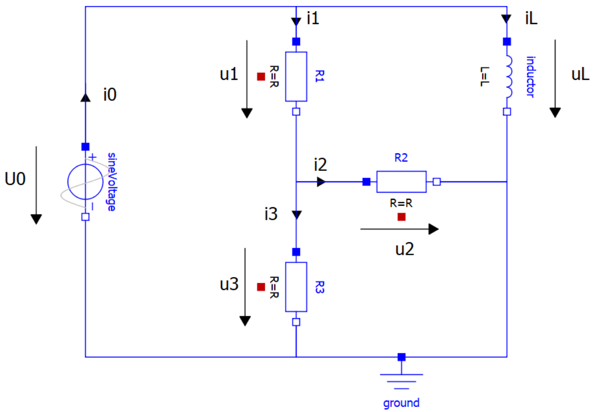
FH Bielefeld
University of
Applied Sciences

Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Modelica Compiler Overview

Modelica Compiler Overview

How to get a mathematical representation from a model



Equations

$$u_0 = A\sin(2\pi f t) \qquad u_0 = u_1 + u_3$$

$$u_1 = R_1 \cdot i_1 \qquad u_L = u_1 + u_2$$

$$u_2 = R_2 \cdot i_2 \qquad u_3 = u_2$$

$$u_3 = R_3 \cdot i_3 \qquad i_0 = i_1 + i_L$$

$$u_L = L \cdot \dot{i}_L \qquad i_1 = i_2 + i_3$$

FH Bielefeld
University of
Applied Sciences

Modelica Compiler Overview

Symbolic Transformation

Steps to perform

- Causalization
  - Assign each variable to exactly one equation
- Matching and sorting
  - Find strong components and sorting
- Adjacency matrix and structural regularity
  - BLT transformation

Group variables

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\underline{\dot{x}}(t) = \begin{pmatrix} \dot{i}_L \end{pmatrix} \quad \underline{x}(t) = \begin{pmatrix} i_L \end{pmatrix}$$

$$\underline{y}(t) = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_L \\ i_0 \\ i_1 \\ i_2 \\ i_3 \end{pmatrix} \quad \underline{p} = \begin{pmatrix} A \\ f \\ R_1 \\ R_2 \\ R_3 \\ L \end{pmatrix} \quad \underline{u}(t) = \begin{pmatrix} \end{pmatrix}$$

FH Bielefeld
University of
Applied Sciences

Modelica Compiler Overview
Symbolic Transformation

- Matching: Assign variables to equations
- Sorting: Construct directed graph
- Get ordered state form

Symbolic transformation

$$\underline{0} = f(\underline{x}(t), \underline{z}(t), \underline{u}(t), t), \quad \underline{z}(t) = \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\underline{z}(t) = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\dot{\underline{x}}(t) = \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$
$$\underline{y}(t) = \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

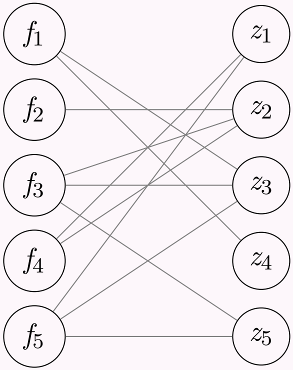
Modelica Compiler Overview

Matching & Sorting

Adjacency matrix

$$
\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array}
\begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\
\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}
$$

Bipartite Graph



FH Bielefeld
University of
Applied Sciences

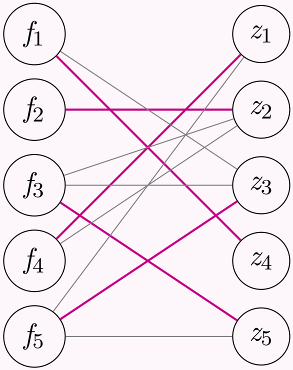
Modelica Compiler Overview

Matching & Sorting

Adjacency matrix

	z_1	z_2	z_3	z_4	z_5
f_1	0	0	1	**1**	0
f_2	0	**1**	0	0	0
f_3	0	1	1	0	**1**
f_4	**1**	1	0	0	0
f_5	1	0	**1**	0	1

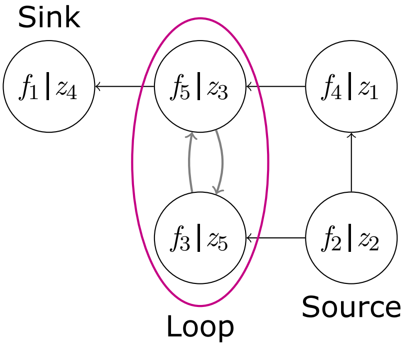
Bipartite Graph



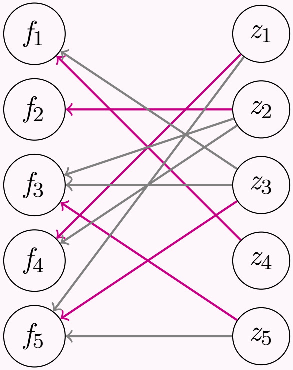
FH Bielefeld
University of
Applied Sciences

Modelica Compiler Overview

Matching & Sorting



Bipartite Graph



FH Bielefeld
University of
Applied Sciences

Modelica Compiler Overview

Block-Lower-Triangular

For our start example we get:

$$
\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_6 \\ f_8 \\ f_{10} \\ f_7 \\ f_5 \\ f_9 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} u_0 & u_1 & i_1 & u_2 & i_2 & u_3 & i_3 & u_L & \dot{i}_L & i_o \end{array} \\
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\end{array}
$$

FH Bielefeld
University of
Applied Sciences

Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Tearing

Overview

Tearing

Target:

- Efficient computation with numerical solver
- Reduces size of algebraic loops

General Idea

- Assume variables are already known (=Tearing variables)
- Causalize equations with this assumption (=inner equations)
- Remaining Equations are residual equations

FH Bielefeld
University of
Applied Sciences

Tearing

Choose good tearing variables

Example

$$u_1 - R_1 \cdot i_1 = 0$$
$$u_2 - R_2 \cdot i_2 = 0$$
$$u_3 - R_3 \cdot i_3 = 0$$
$$u_1 + u_3 = u_0$$
$$u_2 - u_3 = 0$$
$$i_1 - i_2 - i_3 = 0$$

- Find minimal number of iteration variables
  - Mind solvability
  - Problem is NP-hard
- Use heuristics to choose in polynomial time
  - `--tearingMethod = noTearing`
  - `--tearingMethod = minimalTearing`
  - `--tearingMethod = omcTearing`
  - `--tearingMethod = cellier`

Tearing

Choose good tearing variables

Assuem i_3 is known

$$u_3 = R_3 \cdot i_3$$

$$u_1 = u_0 - u_3$$

$$i_1 = \frac{u_1}{R_1}$$

$$u2 = u_3$$

$$i_2 = \frac{u_2}{R_2}$$

Residual equation:

$$0 = i_1 - i_2 - i_3$$

- Try to minimize number of iteration variables
  - Mind solvability
  - Problem is NP-hard
- Use heuristics to choose in polynomial time
  - `--tearingMethod = noTearing`
  - `--tearingMethod = minimalTearing`
  - `--tearingMethod = omcTearing`
  - `--tearingMethod = cellier`

FH Bielefeld
University of
Applied Sciences

Tearing

Hybrid Algebraic Loops

What if I want to disable tearing?

- `omcTearing` or `cellier` to time consuming
- Use sparse non-linear solvers
  - That’s what `--daeMode` is doing
- Debugging and library development

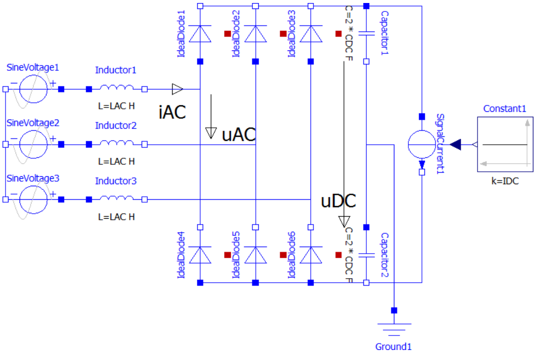
Problem

Hybrid algebraic loops need tearing!

FH Bielefeld
University of
Applied Sciences

Tearing
Hybrid Algebraic Loops

`Modelica.Electrical.Analog.Examples.Rectifier`



- `IdealDiode1.off, ..., IdealDiode6.off` inside algebraic loops
- "Switching state" type: Boolean

FH Bielefeld
University of
Applied Sciences

Tearing

Minimal Tearing

Minimal Tearing

Perform the bare minimum of optimization

Want to causalization of:

- Discrete variables (Boolean, Integer, ...)
- Variables solved inside an algorithm with discrete variables as outputs

General Idea

1. Search all discrete variables, `tearingSelect=never`-variables and variables from algorithms
2. Match found variables to equations of algebraic loop
   $\Rightarrow$ Set as inner variables and equations
3. Remaining variables are iteration variables

FH Bielefeld
University of
Applied Sciences

Minimal Tearing
Implementation

`--tearingMethod=minimalTearing`

- implemented in OpenModelica v1.14
  - Cases for when-equations, if-equations and algorithms will be implemented in v1.16
- Performance: Matching on discrete variables of strong component
  - $\mathcal{O}(|V| \cdot |E|)$ with V discrete variables of loop, E connected equations
- Supersede `--tearingMethod=noTearing`

FH Bielefeld
University of
Applied Sciences

Efficient Minimal Tearing of Hybrid Algebraic Loops for Large-scale System Simulation

Thank you for your attention

Questions

Refrences

- Cellier, E. Francio und Kofman, Ernesto. 2006. Continous System Simulation. New York: Springer Science+Business Media Inc., 2006.
- R. Bulatow, Entwicklung einer Fast-Tearing-Methode zur effizienten Simulation großer differential-algebraischer Gleichungssysteme, Bachelor thesis, 2019