Symbolic Manipulation for the Simulation of Object-Oriented Models

Francesco Casella (francesco.casella@polimi.it)

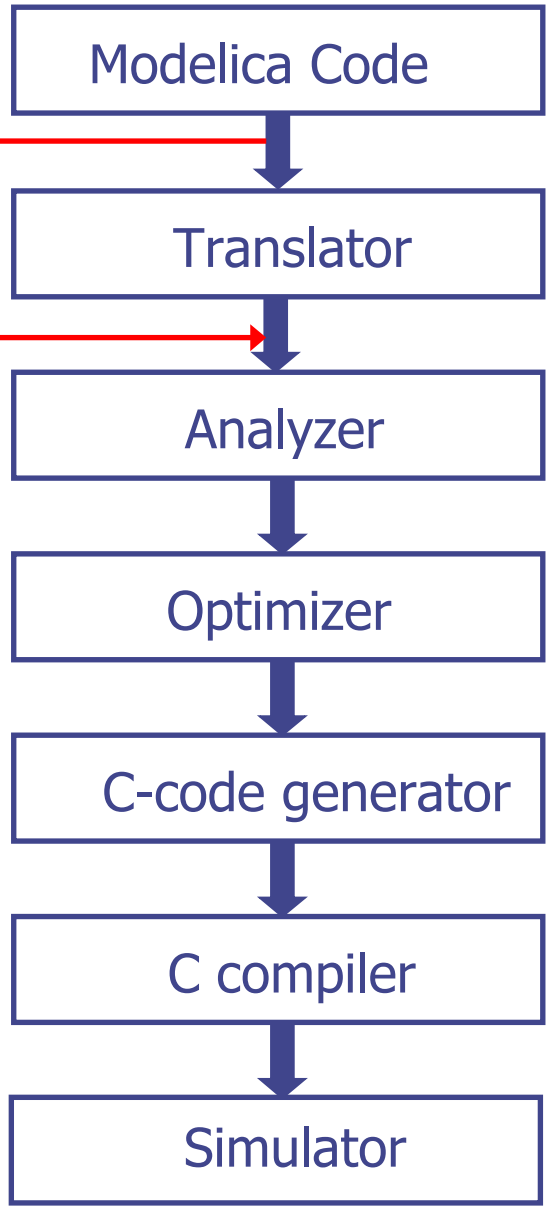
From the Model to the Simulator

- Parsing of the code
- Type check.
- Class expansion (inheritance)
- Generation of the connection equations (`connect`).

"Flattened" system of:

- Equations.
- Constants.
- Variables.
- Functions.

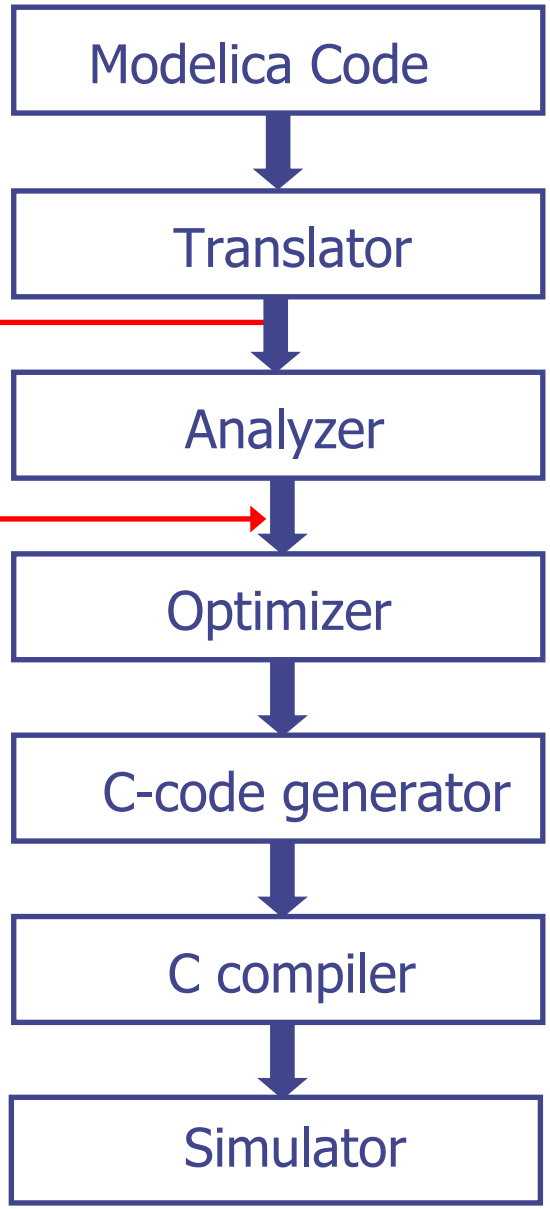
(~~Object-Oriented structure~~ ⇒ Dot Notation)



From the Model to the Simulator

- Equation ordering and structural analysis.
- ***BLT*** (**B**lock **L**ower **T**riangular) transformation

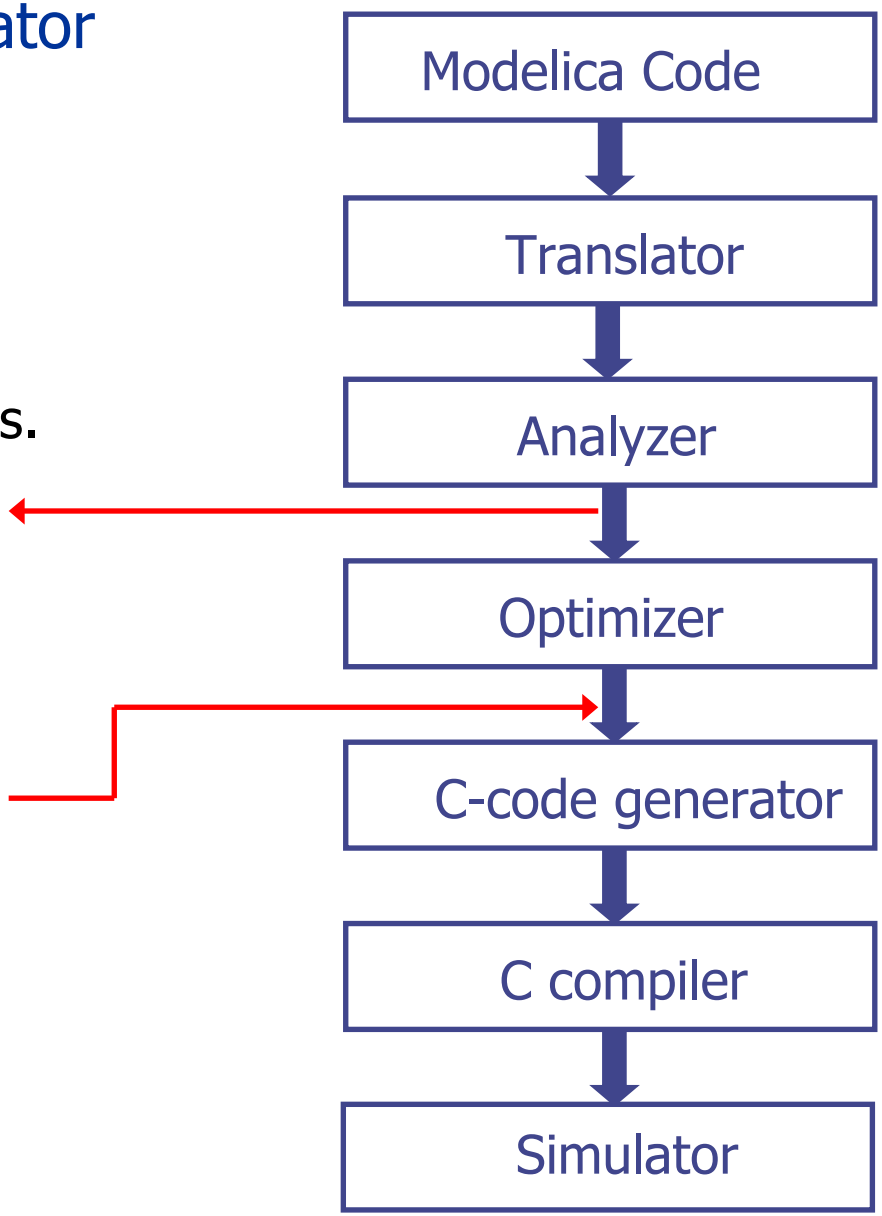
Set of ordered equations



From the Model to the Simulator

- Elimination of trivial equations.
- Common subexpression elimination
- Index reduction.
- Analytical solution of simple equations

Minimal set of equations in efficient state-space form.

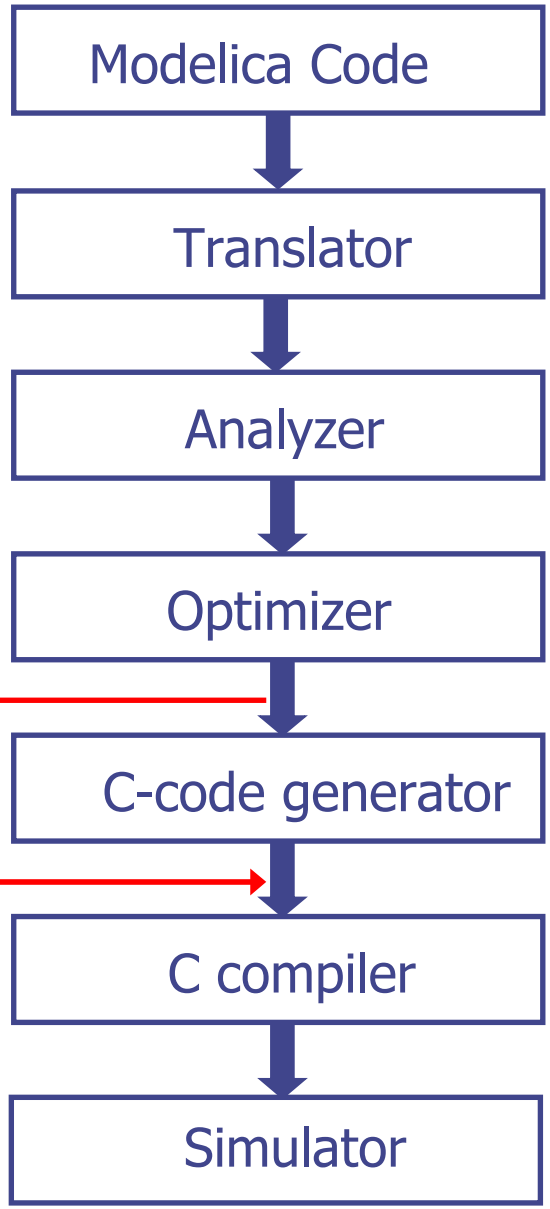


From the Model to the Simulator

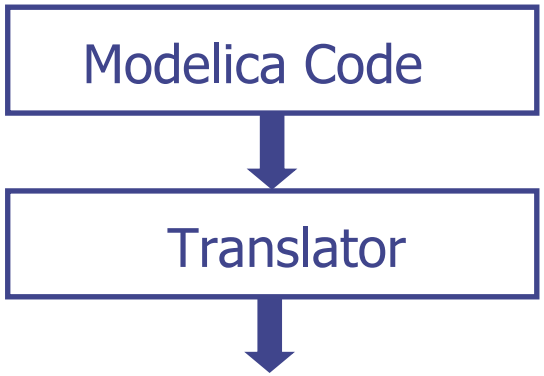
- C code generation.
- Linking with a numerical solver code.

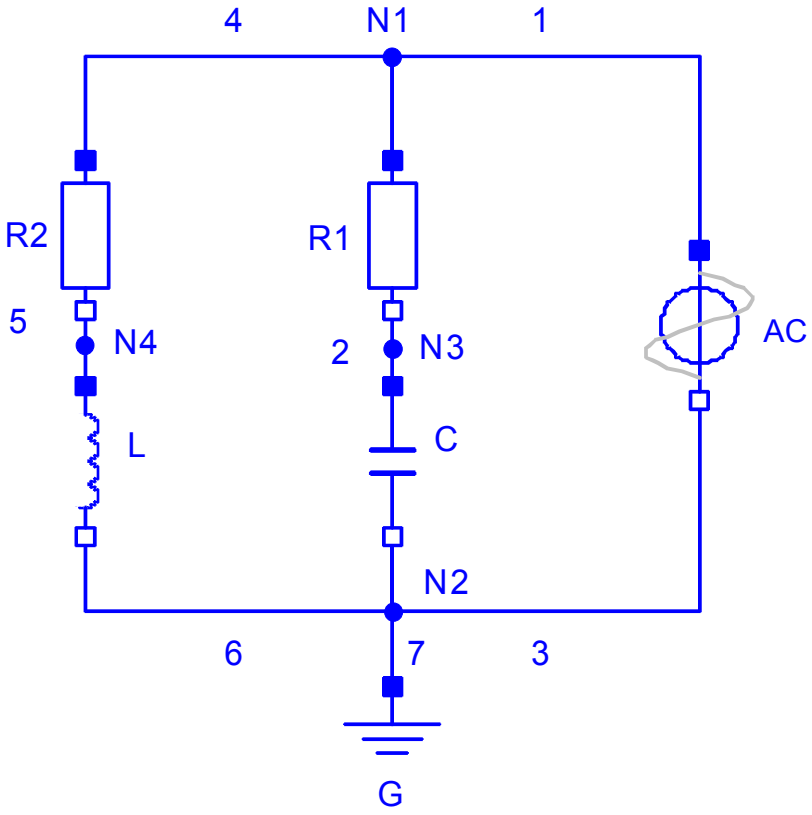
C code.

Often the result is more efficient than "hand-written" code, thanks to the symbolic optimization phase.



Flattening: a Representative Example





Translation rules

- For each instantiated class the translator adds a copy of the class equations to the system DAE.
- For each connection between instances, the translator adds the connection equations to the system DAE.

Flattening: Set of Flattened Equations

32 equations

AC	0 = AC.p.i+AC.n.i AC.v = AC.p.v-AC.n.v AC.i = AC.p.i AC.v = AC.VA* sin(2*AC.PI* AC.f*time)	L	0 = L.p.i+L.n.i L.v = L.p.v-L.n.v L.i = L.p.i L.v = L.L*der(L.i)
R1	0 = R1.p.i+R1.n.i R1.v = R1.p.v-R1.n.v R1.i = R1.p.i R1.v = R1.R*R1.i	G	G.p.v = 0
R2	0 = R2.p.i+R2.n.i R2.v = R2.p.v-R2.n.v R2.i = R2.p.i R2.v = R2.R*R2.i	connections (effort)	R1.p.v. = AC.p.v // 1 C.p.v = R1.v.v // 2 AC.n.v = C.n.v // 3 R2.p.v = R1.p.v // 4 L.p.v = R2.n.v // 5 L.n.v = C.n.v // 6 G.p.v = AC.n.v // 7

C	0 = C.p.i+C.n.i
	C.v = C.p.v-C.n.v
	C.i = C.p.i
	C.i = C.C*der(C.v)
		connections (flow)	0 = AC.p.i+R1.p.i+R2.p.i // N1
			0 = C.n.i+G.p.i+AC.n.i+L.n.i // N2
			0 = R1.n.i+C.p.i // N3
			0 = R2.n.i+L.p.i // N4

Flattening: Set of Variables

32 variables → 30 algebraic + 2 dynamic (state*)

R1	R1.p.i	R1.n.i	R1.p.v	R1.n.v	R1.v	R1.i
R2	R2.p.i	R2.n.i	R2.p.v	R2.n.v	R2.v	R2.i
C	C.p.i	C.n.i	C.p.v	C.n.v	C.v	C.i
L	L.p.i	L.n.i	L.p.v	L.n.v	L.v	L.i
AC	AC.p.i	AC.n.i	AC.p.v	AC.n.v	AC.v	AC.i
G	G.p.i	G.p.v				

** Dynamic variables become state variables, except in the case of higher-index DAE*

Flattening: Set of Parameters, Inputs, Variables

- Parameters → **parameter**

  p = {**R1.R, R2.R, C.C, L.L, AC.VA, AC.f**}

- Inputs → **input**

  this system is autonomous; we could substitute the sinusoidal voltage source with a signal-controlled source, whose voltage would then be determined by the input signal

- Dynamic variables → **der()**

  x = {**C.v, L.i**}

- Algebraic Variables

  y = {**R1.p.i, R1.n.i, R1.p.v, R1.n.v, R1.v, R1.i, R2.p.i, R2.n.i, R2.p.v, R2.n.v, R2.v, R2.i, C.p.i, C.n.i, C.p.v, C.n.v, C.i, L.p.i, L.n.i, L.p.v, L.n.v, L.v, AC.p.i, AC.n.i, AC.p.v, AC.n.v, AC.v, AC.i, G.p.i, G.p.v**}

Structural Analysis

- A structural representation of the system is built, indicating which variables show up in which equations
  - $x(t)$: dynamic variables
  - $y(t)$: algebraic variables
  - $u(t)$: exogenous inputs
  - p : parameters (remain fixed during the simulation)
  - $z(t)$: unknowns of the DAE problem (to be computed as a function of the states x, the inputs u and the parameters p)
- The minimum requirement is to have the same dimension for *f* and *z* (equal number of equations and variables)

$$F(t, x'(t), x(t), y(t), u(t), p) = 0$$

$$f(t, z(t), x(t), u(t), p) = 0 \qquad z(t) = \begin{bmatrix} x'(t) \\ y(t) \end{bmatrix}$$

Alias Variables Elimination

- If the DAE system was obtained by aggregation of object-oriented models, a high percentage of its equations (usually more than 50%) are trivial equations involving algebraic variables, such as:

$$y_j = \pm y_k$$

- The first simplification of the DAE system is obtained by eliminating those equations, and symbolically substituting the alias variables in the remaining equations

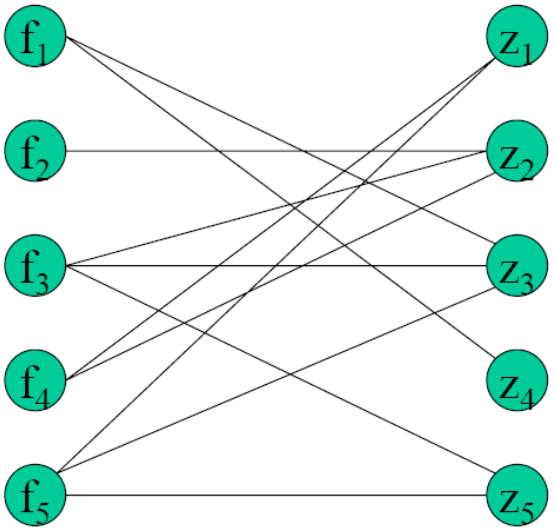
$$y_j \rightarrow \pm y_k$$

Incidence Matrix / E-V Graph

- The structural information is represented by an incidence matrix, equivalent to a bipartite **E**quations – **V**ariables graph

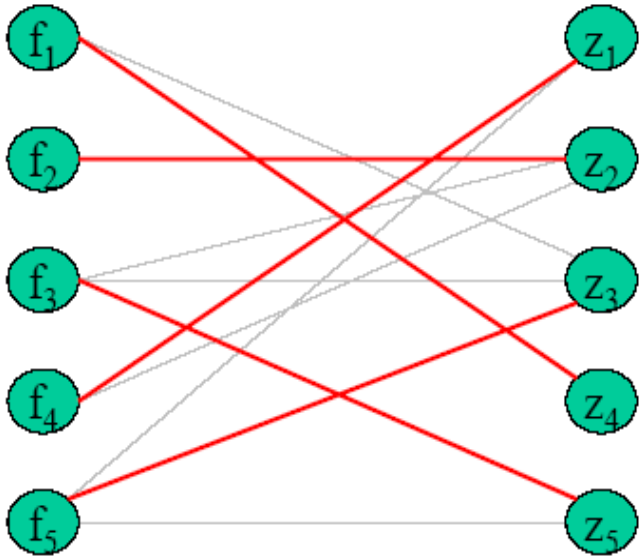
$$
\begin{aligned}
f_1(z_3,z_4) &= 0\\
f_2(z_2) &= 0\\
f_3(z_2,z_3,z_5) &= 0\\
f_4(z_1,z_2) &= 0\\
f_5(z_1,z_3,z_5) &= 0
\end{aligned}
$$

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



Matching Problem

- The matching algorithm is executed: a *graph transversal* is sought, i.e. a set of arcs which makes up a bipartite graph. This graph represents the equations-variables matching
- This is equivalent to finding a permutations of rows and columns of the incidence matrix, in order to get all ones on the main diagonal
- If the algorithm terminates successfully, the system has structural index 1. Otherwise there are two alternatives:
  - structurally inconsistent problem
  - higher-index problem (constraints between dynamic variables)



Matching Problem: Examples of Failure

- Example 1:

$$y_1 + y_2 + y_3 = 0$$
$$y_1 = u_1$$
$$p_1\, y_1 = u_2$$

Nobody assigns the variable y_2!
(Structurally inconsistent problem)

- Example 2:

$$x_1' = x_2 + y_1$$
$$x_2' = x_1 + y_1$$
$$x_1 + x_2 = 0$$

Nobody assigns the variable y_1!
(Index-2 problem)

Index Reduction

- If the matching algorithm fails, Pantelides' algorithm is run – this creates new E-nodes and new arcs by suitably deriving some of the system equations
- If the problem is not structurally singular, Pantelides' algorithm terminates returning a redundant system of index-1 DAE's. The Dummy Derivative algorithm is then run to eliminate the redundant state variables in order to get a balanced system of equations
- When the DD algorithm terminates, the survived dynamic variables are the true state variables of the system, whose initial values can be assigned arbitrarily
- Otherwise, the problem is structurally singular (ill-posed). The structure of the graph can be used to derive diagnostic information in order to facilitate the model troubleshooting

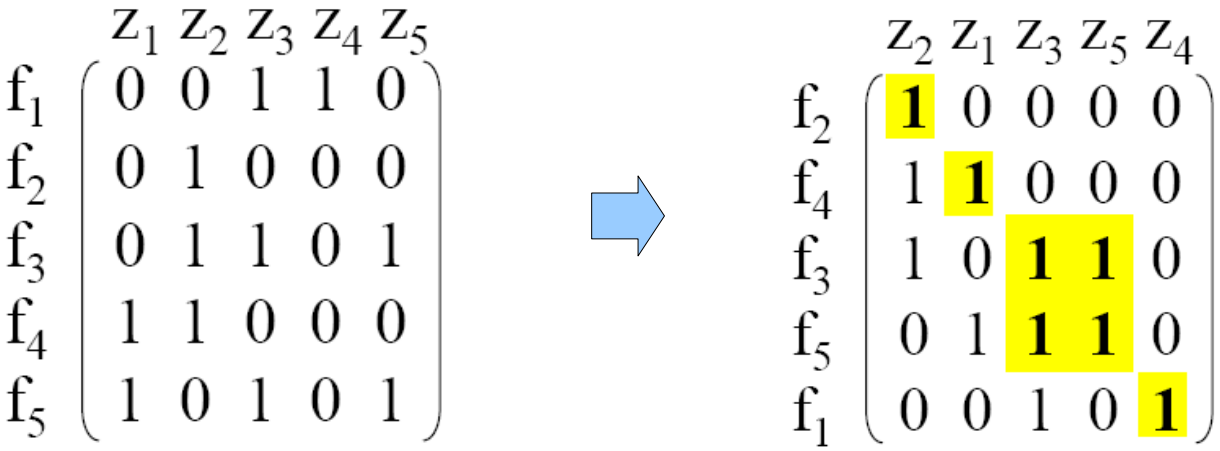
Parameter-Dependent Variables

- By adding to the E-V graph the nodes and arcs corresponding to the parameters (and to their occurrences in the equations), it is possible to determine which variables are structurally dependent on parameters only, and thus become parameters themselves

- The corresponding equations can be solved once and for all during the initialization phase, and eliminated (together with the corresponding variables) from the subsequent analysis, which will only involve time-varying variables.

BLT Transformation (1/2)

- The order of the equation in the function $f(t, z, y, u, p)$ is completely arbitrary, as it depends on the order of the equations in the Modelica code
- The equations must be re-arranged in order to make their solution as simple as possible. This means re-ordering them so that the incidence matrix becomes **B**lock-**L**ower-**T**riangular



BLT Transformation (2/2)

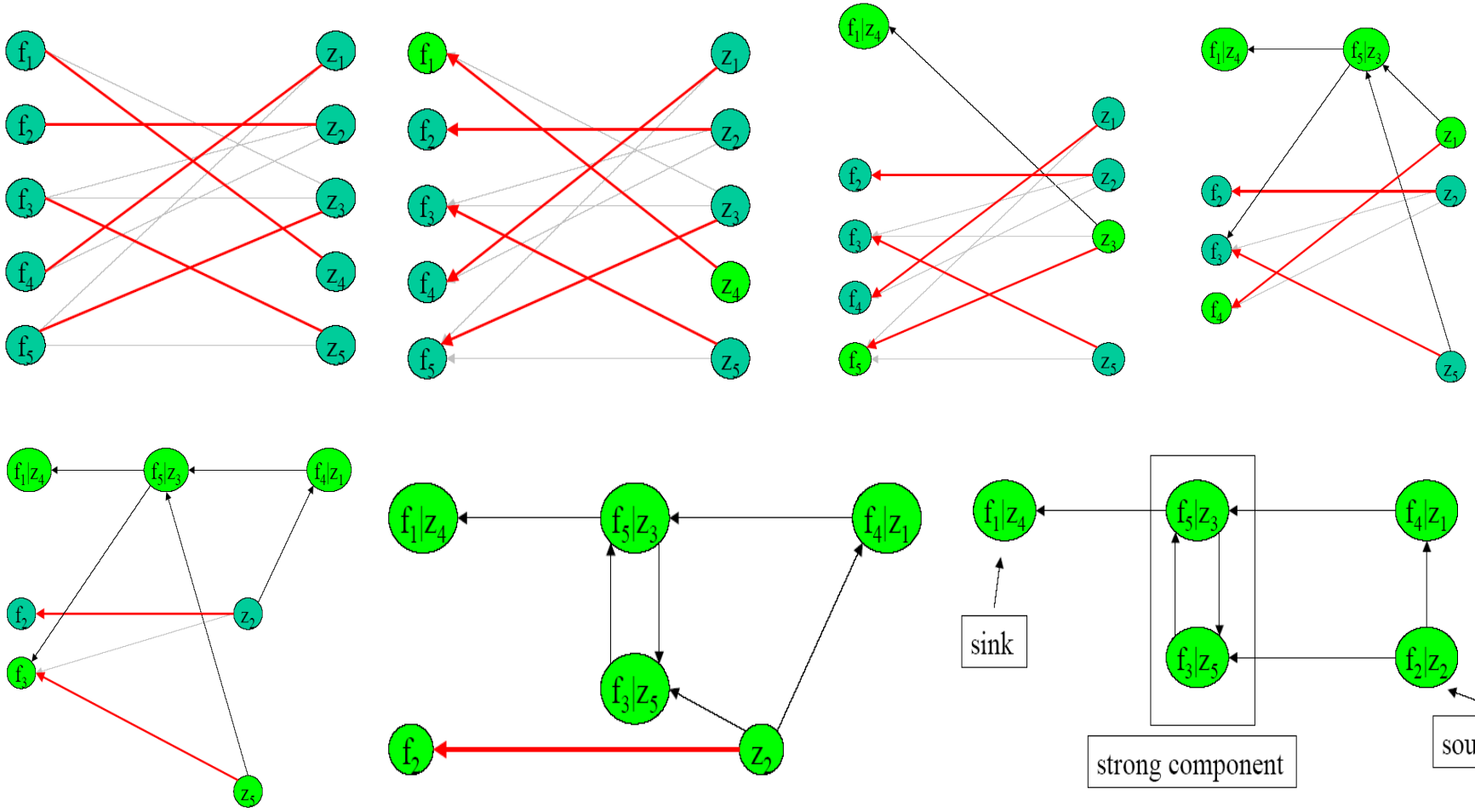
- The solution of the complete system is then reduced to the sequenced solution of many smaller systems, whose dimension corresponds to that of the blocks on the main diagonal
- In this example, instead of solving a system of 5 equation at once, it is possible to sequentially solve 4 systems of 1, 1, 2, 1 equations.

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

- The BLT transformation problem is solved by a famous graph-theoretical algorithm, known as Tarjan's algorithm (1972). Note that the algorithm complexity is only O(*N*)

Tarjan's Algorithm

- Starting from the previously found graph transversal, the algorithm searches for the strongly connected components of the graph, that correspond to the blocks on the main diagonal of the BLT matrix



Remarks on the BLT Transformation (1/2)

- In the most favourable cases, the BLT matrix can be strictly lower triangular; if the variables on the main diagonal appear explicitly (i.e. at most, multiplied by a coefficient) in the corresponding equations, its possible to solve the system by a sequence of assignments.
- Example: RLC circuit equations

```
1) C.i = R1.v/R1.R                              // f(R1.v)
2) R1.v = R1.p.v - C.v                          // f(R1.v,R1.p.v)-C.v
3) der(L.i) = L.v/L.L                           // f(L.v,der(L.i))
4) R1.p.v = AC.VA*sin(2*AC.f*AC.PI*time)        // f(R1.p.v)
5) L.v = R1.p.v - R2.v                          // f(L.v,R1.p.v,R2.v)
6) der(C.v) = C.i/C.C                           // f(der(C.v),C.i)
7) R2.v = R2.R*L.i                              // f(R2.v) - L.i
```

	R2.v	R1.p.v	L.v	R1.v	C.i	der(L.i)	der(C.v)
1)	0	0	0	**1**	**1**	0	0
2)	0	**1**	0	**1**	0	0	0
3)	0	0	**1**	0	0	**1**	0
4)	0	**1**	0	0	0	0	0
5)	**1**	**1**	**1**	0	0	0	0
6)	0	0	0	0	**1**	0	**1**
7)	**1**	0	0	0	0	0	0

Remarks on the BLT Transformation (2/2)

- After the BLT Transformation

	R2.v	R1.p.v	L.v	R1.v	C.i	der(L.i)	der(C.v)
7)	1	0	0	0	0	0	0
4)	0	1	0	0	0	0	0
5)	1	1	1	0	0	0	0
2)	0	1	0	1	0	0	0
1)	0	0	0	1	1	0	0
3)	0	0	1	0	0	1	0
6)	0	0	0	0	1	0	1

```
7) R2.v := R2.R*L.i
4) R1.p.v := AC.VA*sin(2*AC.f*AC.PI*time)
5) L.v := R1.p.v - R2.v
2) R1.v := R1.p.v - C.v
1) C.i := R1.v/R1.R
3) der(L.i) := L.v/L.L
6) der(C.v) := C.i/C.C
```

- The process which is usually carried out manually when building dynamic simulators with block-diagram languages (such as Simulink) is carried out automatically in this context

Structurally Regular but Locally Singular Problems

- The successfu termination of the transversal, Pantelides', Dummy Derivatives and BLT algorithm guarantee that the DAE system is *structurally regular,* i.e. generally solvable given the initial values of the state variables.
- It might happen that specific values of the parameters, the state variables or the inputs make the problem singular (or higher-index). This situation is not dealt with by the simbolic manipulation algorithm (except the state variable pivoting of the DD algorithm), and leads to numerical run-time errors.

$$\begin{vmatrix} a & b \\ 1 & 1 \end{vmatrix} = a - b = 0$$

$$\Leftrightarrow$$

$$a = b$$

```
model SingularModel
  parameter Real a=1;
  parameter Real b=1;
  Real x,y;
equation
  a*der(x) + b*der(y) = sin(time);
  der(x) + der(y) = cos(time);
end SingularModel
```

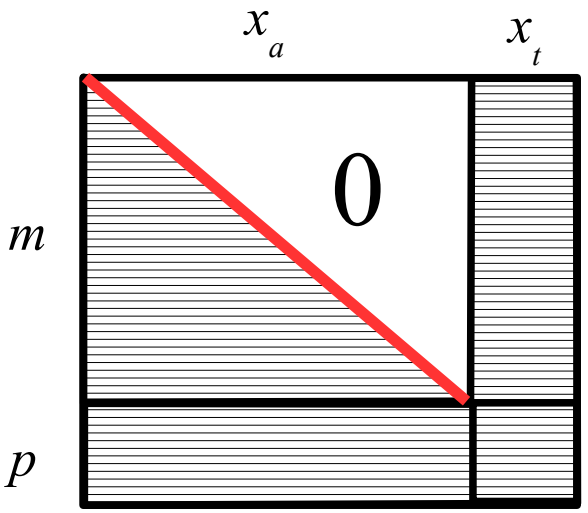
Optimization of the BLT blocks

- After the BLT transformation, the solution of the DAE system is decomposed into a sequenc of smaller problems, corresponding to the main diagonal blocks
- Further symbolic manipulation can be applied to these blocks, to make the solution faster and more numerically robust:
  - *Linear equations*: for specific structures of the equations (0/1 coefficients) it is possible to solve the equation efficiently through symbolic manipulation
  - *Non-linear equations*: the solution of the systems can be made easier by the *tearing* method

Tearing

- The variables that can be solved explicitly in each nonlinear equation are marked as coloured elements in the incidence matrix.
- A row and column permutation is sought, such that the incidence matrix is bordered-lower-triangular, with coloured elements on the main diagonal, and such that the border width is minimized.:

- A guess value is first assigned to x_t: this allows to compute x_a by simple assignments (the numerical results are a function of x_t). The last equations thus form a system of only p variables in the unknowns x_t
- This smaller system can be solved by a Newton-Raphson type an iterative method; finally, the x_a are computed by substitution.



Example

- Let's assume that the equations have been rearranged in this bordered-lower-triangular form:

$$z_1 + \log(z_3) = 0$$
$$z_1^2 - 5z_2^3 = 0$$
$$z_1^2 + z_2^2 - z_3^2 = 0$$

	z1	z2	z3
f1	1	0	1
f2	1	1	0
f3	1	1	1

- z_3 is the tearing variable. Given an initial guess value, it is possible to go through the assignments:

```
z1 := -log(z3)
z2 := (z1^2 / 5 )^1/3
fz3 := z1^2 + z2^2 +z3^2
```

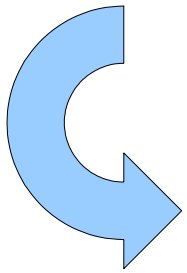
- The solution of the system has been reduced to the solution of $f(z_3) = 0$, for which we have an algorithm to compute the residual. The Jacobian $df(z_3)/dz_3$ can be computed numerically, or possibly by symbolically differentiating the assignments.
- Once z_3 has been found, the first two assignments return z_1, z_2

Remarks

- Contrary to the previous algorithm (whose complexity is polynomial or even linear in the number of equations), the search of the optimal tearing (minimizing p) is an NP-complete problem.
- Since an exhaustive search would take too much time, even for small systems, it is necessary to formulate good heuristic algorithms for structures deriving from typical physical problems.
- The result is a very efficient way of solving large non-linear systems of equations, since the non-linear equation to be solved iteratively has a reduced dimension:
  - easier convergence
  - Jacobian computation and inversion is less expensive
- The initialization of the iterative algorithm is simpler as well, since initial guesses are needed *for the tearing variables only*. However, note that the set of tearing variables can easily change when the structure of the model is modified (e.g. by connecting a new component to an existing object-oriented model).

Common Subexpression Elimination

- During the symbolic manipulation, the same expression may appear in different places
  - example: connection of two thermo-hydraulic components, computing the outlet and the inlet temperature: both are a function of the same pressure and enthalpy
- The symbolic manipulation can apply CSE (**C**ommon **S**ubexpression **E**limination) algorithms, defining temporary variables where the result of the expression is stored, and then use them wherever necessary
- This is extremely useful in the symbolic computation of derivatives, where parts of expressions appear multiple times



```
z  := f(g(t))*h(t);
zd := fd(g(t))*gd(t)*h(t)+f(g(t))*hd(t);

t1 := g(t);
t2 := f(t1);
t3 := h(t);
z  := t2*t3;
zd := fd(t1)*gd(t)*t3 + t2*hd(t);
```

State Selection (1/2)

- In some cases (e.g. thermo-hydraulic models that use fluid property computation models), it is convenient to use state variables that are not differentiated in the original model
- In the following model, the first equation is explicit only if p,h are state variables; otherwise it has to be solved iteratively, which is not a good idea, since the computation of rho(p,h) can be very time-consuming

```
model ControlVolume
  parameter Volume V;
  Pressure p(
    stateSelect = StateSelect.prefer);
  SpecificEnthalpy h(
    stateSelect = StateSelect.prefer);
  Density d;
  Mass M;
  Energy E;
  FluidPort in, out;
```

```
equation
  d = rho(p,h);
  M = V*d;
  E = M*h - p*V;
  der(M) = in.w + out.w;
  der(E) = in.w * in.h +
           out.w * h;
  in.p = p;
  out.p = p;
  out.h = h;
end ControlVolume;
```

State Selection (2/2)

- The index reduction algorithm will try to differentiate the algebraic equations so that the derivatives of the requested state variables (p,h) show up, while E', M' will become dummy derivatives

```
d = rho(p,h);
M = V*d;
E = M*h - p*V;
der(M) = in.w + out.w;
der(E) = in.w * in.h + out.w * h;
```

```
d = rho(p,h);
M = V*d;
E = M*h - p*V;
V*(drho_dp(p,h)*der(p)+drho_dh(p,h)*der(h)) = in.w + out.w;
V*(drho_dp(p,h)*der(p)+drho_dh(p,h)*der(h))*h +
  M * der(h) - V*der(p) = in.w * in.h + out.w * h;
```

- The functions for the computation of the partial derivatives of density can be computed symbolically (if possible), or can be manually specified through **`annotation(Derivative)`**

Result of the Symbolic Manipulation

- The result of the process described so far is a system of linear and non-linear equations that represent the original DAE system in ODE form:
  - given the values of the state, the parameters and the time
  - by solving the systems obtained from the symbolic manipulation
  - the algebraic variables and the derivatives of the state variables of the system (reduced to index-1 form) are computed
- In fact, this consitutes an efficient procedure to compute the f and g functions of the system, reduced to state-space form:

$$x'(t) = f(x(t), u(t), t, p)$$
$$y(t) = g(x(t), u(t), t, p)$$

- The first of these two function can be linked with any code for numerical integration of ODE (or DAE) systems
- If the algorithm is implicit, the Jacobian $\partial f / \partial x$ will be computed as well, usually by numerical differentiation.

Alternative Strategies

- This strategy is adopted by the tools Dymola and OpenModelica
- Since the final result is in the form of ODE + output equations, the model of a system which has only a causal interface (inputs and outputs) can be translated into an S-function, and exported in block-oriented simulation environments such as Simulink or Scicos, which will take care of integrating them.
- The corresponding code is very efficient, thanks to all the previously described optimizations.
- Other simulation environments (e.g. PSEnterprise's gPROMS, which does not use Modelica) just remove the trivial equations, possibly apply the BLT, and then make use of specific DAE solvers for highly sparse problems
- Other Modelica tools (e.g. Equa Simulation's IDA) pre-compile the single models, which are connected at run-time. Since a global index reduction phase is missing, it is mandatory to use solvers which can integrate (at least) index 2 problems.

Initialization (1/3)

- In order to completely determine the initial conditions of the index-1 DAE problem, a number of conditions n is needed, equal to the number of state variables that survived the index reduction phase.:

  A Default initialization: the state variables get their initial value from their **`start`** attribute

  B Explicit initialization: n **`initial equation`** are adde to the Modelica model.

- In case **A**, the initial values $x(0), u(0), p$, are known, therfore it is possible to compute consistent values of $y(0)$ by using the result of the symbolic manipulation of the DAE system;
  the value of $x(0)$ is then used for the first call to the ODE integration routine

Initialization (2/3)

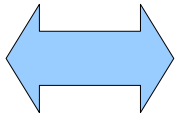
- In the case B, an initialization problem, different from the dynamic problem, is formulated. This problem contains::
  - all the equations of the DAE system, reduced to index 1
  - all the initial equations
- This problem has the initial values $x(0), x'(0), y(0)$ as unknowns, and a corresponding number of equations
- To solve trimming problems (= compute the constant value of the inputs to get certain constant values at the outputs) it is possible to connect the inputs of the system to constant signal generators, whose values are parameters with a **`fixed = false`** attribute, and then add further initial equations to specify the desired output values
- In this case the unknowns of the problems are $x(0), x'(0), y(0), p_{trim}$

Initialization (3/3)

- All the previously discussed methods (structural analysis, BLT, tearing) can be applied to this system; however, note that the fact that the state variables are now unknown makes the incidence matrix bigger and less sparse, and thus significantly reduces the advantages obtained by applying such techniques.
- The numerical solution of case B is therefore much harder in general, at least for strongly non-linear problems. On the other hand, it is only performed once at the beginning of the simulation.
- To attain convergence, it is very important that all the iteration (tearing) variables of the initialization problem have a start value not too far from the sought solution.
- Note that *it is not sufficient to set start values for the states,* because some algebraic variables can be tearing variables as well.

Modelling vs. Simulation

- These methods allow to decouple two distinct tasks:

MODELLING  **SIMULATION**

- The modeller should care of writing models which are
  - Readable (thus self-documenting)
  - Re-usable
  - Without worring too much about how they will be solved
- The simulation tool takes care of all the issues needed to produce an efficient simulation code, e.g.:
  - Scaling of the variables
  - Rearrangement of the equations
  - Index reduction
  - Tearing and symbolic solution of implicit equations
  - Jacobian computations
  - Efficient computation of common subexpressions

References

Cellier, F.E. and Elmqvist, H.: Automated Formula Manipulation Supports Object Oriented Continuous System Modelling. IEEE Control System, April 1993, pp. 28-38.

S.E. Mattsson, G. Söderlind (1993), "Index reduction in differential-algebraic equations using dummy derivatives", Siam J. Sci. Comput., pp. 213-231.

E. Carpanzano, F. Formenti (1994), "Manipolazione simbolica di sistemi DAE: algoritmi, sviluppo del software ed applicazioni, Tesi di Laurea, Politecnico di Milano.

Maffezzoni, C., Girelli, R. and Lluka, P.: Generating Efficient Computational Procedures from Declarative Models. Simulation Practice and Theory 4, 1996, pp. 303-317.

Maffezzoni, C., Girelli, R.: MOSES: modular modelling of physical systems in an object-oriented database, Mathematical and Computer Modelling of Dynamical Systems, v 4, n 2, June 1998, p 121-47

E. Carpanzano, C. Maffezzoni, "Symbolic manipulation techniques for model simplification in object-oriented modelling of large scale continuous systems", Mathematics and Computers in Simulation, Transactions of IMACS, N° 48,1998, pp. 133-150.

Carpanzano, E. Order Reduction of General Nonlinear DAE Systems by Automatic Tearing. Mathematical And Computer Modelling of Dynamical Systems, Vol. 6, N° 2, 2000, pp. 145-168.

S.E. Mattson, H. Olsson, H. Elmqvist, "Dynamic Selection of States in Dymola", Proc. Modelica Workshop 2000, Lund, Sweden, pp. 61-67. http://www.modelica.org/modelica2000/proceedings.html

Fritzson, P: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley 2003.

B. Bachmann (2005), Mathematical Aspects of Object-Oriented Modeling and Simulation, Modelica Conference Tutorial. http://www.modelica.org/events/Conference2005/online_proceedings/Modelica2005-Tutorials.zipB.

Cellier, F.E., Kofman, E: Continuous system simulation. Springer, 2006.