Prototyping Composable Simplification Passes for Equation-Oriented Models Using ModelingToolkit.jl

Yingbo Ma | yingbo.ma@juliacomputing.com
Keno Fischer | keno.fischer@juliacomputing.com
Viral Shah | viral@juliacomputing.com
Julia Computing Inc., USA

Chris Rackauckas | chris.rackauckas@juliacomputing.com
Julia Computing Inc., USA
Massachusetts Institute of Technology, USA

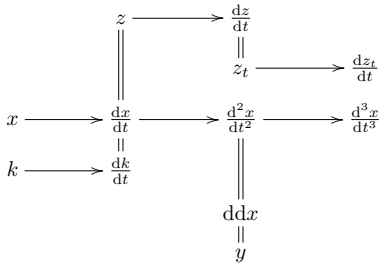
ModelingToolkit.jl (MTK) is an acausal modeling language for high-performance symbolic-numeric computation in scientific computing and scientific machine learning. Like many Modelica implementations, MTK also performs structural transformation passes on differential-algebraic equations. MTK uses alias elimination, partial state selection, and tearing to simplify general DAEs to index 1 semi-implicit DAEs or ODEs. One design that makes MTK distinct from all the other Modelica implementations is that it tries to work with the rest of the Julia language. Thus, users can not only take advantage of the Julia ecosystem, but also apply MTK transformations seamlessly on ordinary models written in Julia without MTK in mind.

MTK was first developed by following Otter and Elmqvist's 2017 paper "Transformation of Differential Algebraic Array Equations to Index One Form" Since then, we have prototyped alternative alias elimination and partial state selection algorithms in comparison to those described in their paper.

Equation-oriented simulation tools need to lower the index of DAEs so that numerical solvers can perform consistent initialization and integrate them. Index reduction algorithms work by adding differentiated equations to the original system, and then to maintain a balanced model, the partial state selection pass needs to select as many dummy derivatives as the number of added equations. Modia.jl's partial state selection pass relies on tearing to detect dummy derivatives which is an elegant approach that bridges Pantelides' index reduction algorithm and tearing.

However, the drawback of this approach is that it's very computationally expensive to perform tearing optimally. Thus, in practice, only heuristic algorithms are used for performance reasons. Unfortunately, a heuristic tearing approach cannot guarantee partial state selection to produce a balanced model, since we must select a precise number of dummy derivatives to keep the DAE system bal-anced. In addition, tearing is oblivious to the numerical rank of the Jacobian matrix of the transformed system as it only has access to structural information. Hence, Modia's approach could produce numerically singular systems even for linear systems with only integer coefficients. To circumvent the aforementioned difficulties, MTK uses the dummy derivative algorithm proposed by Mattsson and Söderlind, and when the Jacobian is integer valued, we use the Bareiss algorithm to select the spanning columns to ensure that the transformed system is always numerically non-singular for this common case. When the Jacobian is non-integer valued, we use the bipartite matching algorithm to pick dummy derivatives that result in at least structurally fully ranked Jacobian.

MTK's alias elimination is strengthened so that it can simplify equivalent differentiated variables as well. After applying the Bareiss algorithm to the linear subsystem to identify aliasing pairs, we may have structures like



where = denotes alias and $\rightarrow$ points to the differentiated variable.

To eliminate redundant differentiated variables, MTK picks the least differentiated variables and aliases all other variables to that particular differentiation chain. In this case, if we pick x as the root variable, we would generate the following aliases

$$z := \frac{\mathrm{d}k}{\mathrm{d}t} := \frac{\mathrm{d}x}{\mathrm{d}t}$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} := z_t := \mathrm{d}\mathrm{d}x := y := \frac{\mathrm{d}^2x}{\mathrm{d}t^2}$$
$$\frac{\mathrm{d}z_t}{\mathrm{d}t} := \frac{\mathrm{d}^3x}{\mathrm{d}t^3}.$$

In this talk, we will present how MTK achieves an easy-to understand implementation of simplification passes by utilizing high-level features of Julia like multiple dispatch and union splitting. We will first review the role of simplification steps in compiling declarative equation-oriented models to executable simulation code, and then demonstrate how these steps are formulated in MTK. Furthermore, we will provide examples of how these novel approaches enable new applications in optimization systems and stochastic differential algebraic equation systems.