Dependent Types for Modelling and Simulation

Challenges and Opportunities

Henrik Nilsson

School of Computer Science

University of Nottingham

This Talk

This Talk

Goal:

Tell you a bit about an application area where (aspects of) Dependent Types would be useful.

This Talk

Goal:

Tell you a bit about an application area where (aspects of) Dependent Types would be useful.

Outline:

- Modelling and modelling languages
- Dependent types in current modelling languages
- Functional Hybrid Modelling
- Dependent types to help constructing well-formed models

What kind of modeling? (1)

Our interest: Languages for modeling and simulation of ***physical*** systems; for example:

- electrical circuits
- gear boxes
- robot manipulators
- chemical plants
- ...

Significant ***continuous*** aspects.

What kind of modeling? (2)

Specifically, languages that support:

What kind of modeling? (2)

Specifically, languages that support:

- ***declarative, non-causal*** modelling: models expressed as systems of undirected Differential Algebraic Equations (DAE)

What kind of modeling? (2)

Specifically, languages that support:

- ***declarative, non-causal*** modelling: models expressed as systems of undirected Differential Algebraic Equations (DAE)
- ***hybrid*** modeling: models exhibiting both continuous-time and discrete time behaviour; e.g., structural changes at discrete points in time.

Causal vs. non-causal modeling (1)

Causal or ***block-oriented*** modeling: model is ODE in ***explicit*** form:

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$$

Causality, i.e. cause-effect relationship, given by the modeler. Cf. Functional Programming.

Causal modeling is the dominating modeling paradigm; languages include Simulink.

Causal vs. non-causal modeling (2)

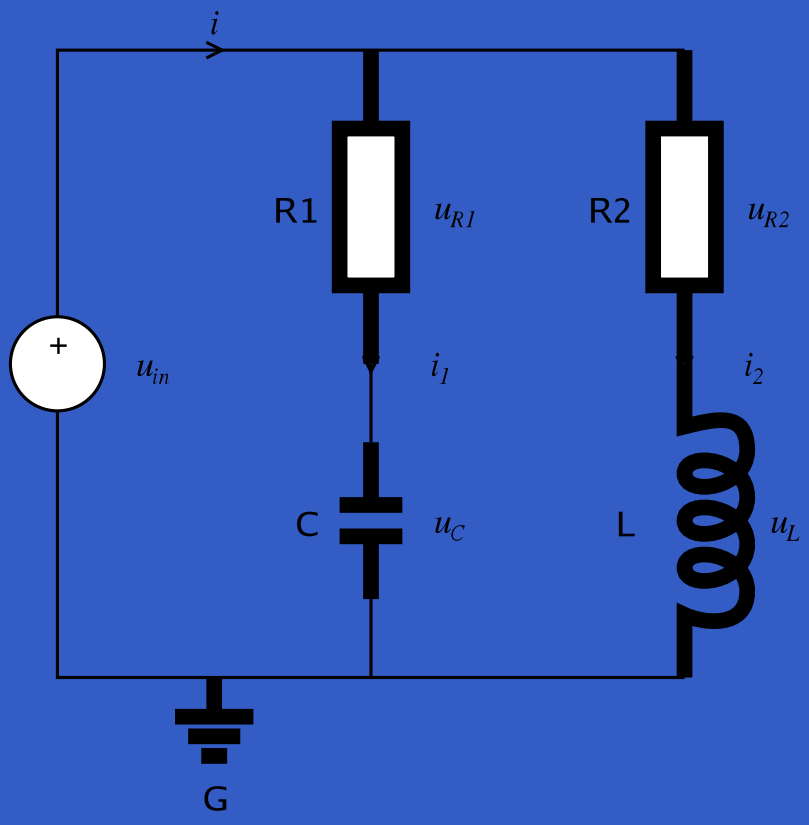
Non-causal or "***object-oriented***" modeling: model is DAE in implicit form:

$$\mathbf{f}(\mathbf{x}, \mathbf{x}', \mathbf{w}, \mathbf{u}, t) = \mathbf{0}$$

Causality inferred by simulation tool from usage context. Cf. Logic Programming.

Non-causal modeling is a fairly recent development; languages include Dymola and Modelica.

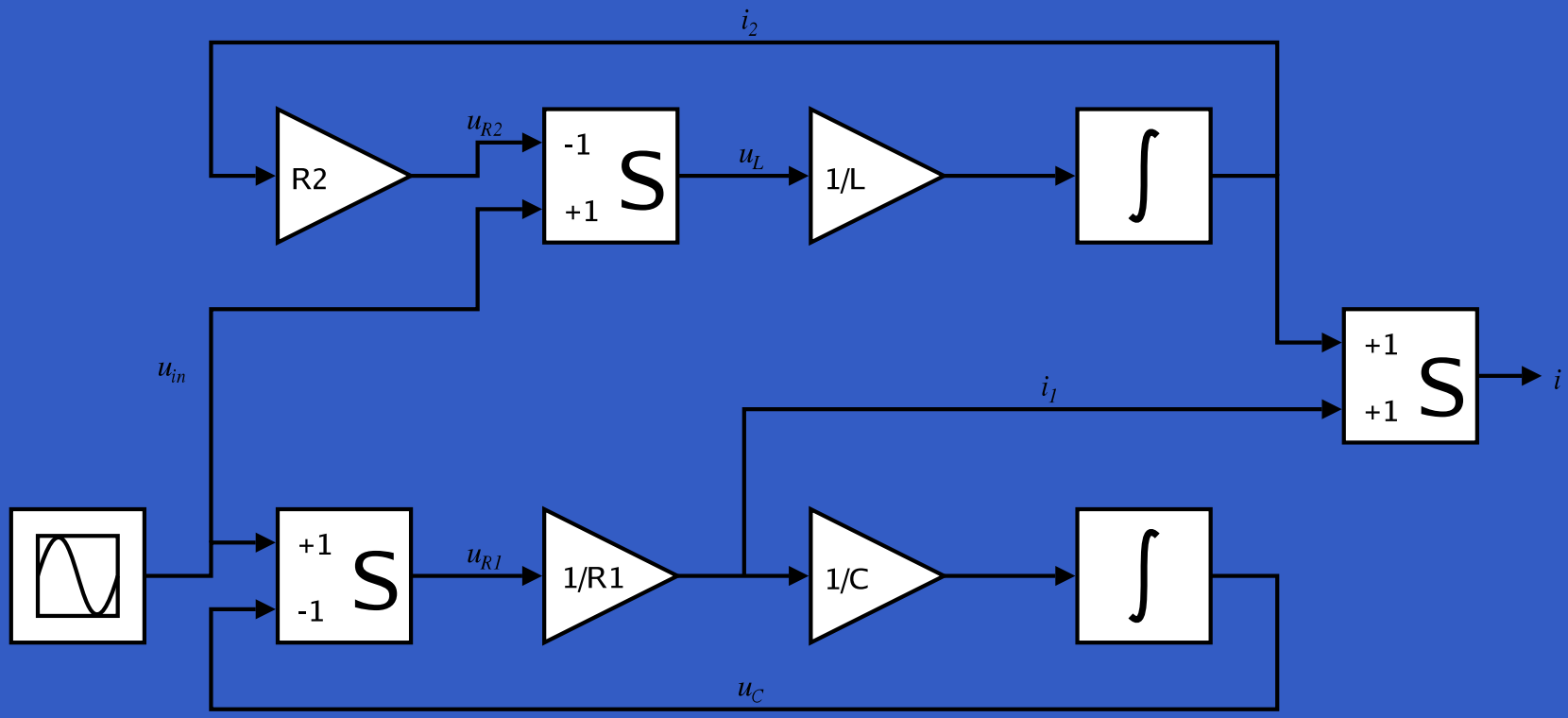
Causal modeling: example (1)



$$
\begin{aligned}
u_{R_2} &= R_2 i_2 \\
u_L &= u_{in} - u_{R_2} \\
{i_2}' &= \frac{u_L}{L} \\
u_{R_1} &= u_{in} - u_C \\
i_1 &= \frac{u_{R_1}}{R_1} \\
{u_C}' &= \frac{i_1}{C} \\
i &= i_1 + i_2
\end{aligned}
$$

Causal modeling: example (2)

Or, as a block diagram:



Drawbacks of causal modeling

Casual modeling bad fit for fundamentally non-causal domains like physical modeling:

- Structure of model and modeled system does not agree.
- Model not simple composition of models of physical components.
- Fixed causality hampers reuse.
- Burden of deriving a non-causal model rests with the modeler.

Non-causal modeling: example (1)

Non-causal resistor model:

$$\begin{aligned} u &= v_p - v_n \\ i_p + i_n &= 0 \\ u &= R i_p \end{aligned}$$

Non-causal inductor model:

$$\begin{aligned} u &= v_p - v_n \\ i_p + i_n &= 0 \\ u &= L i_p{}' \end{aligned}$$

Non-causal modeling: example (2)

A non-causal model of the entire circuit is created by ***instantiating*** the component models: copy the equations and rename the variables.

The instantiated components are then ***composed*** by adding connection equations according to Kirchhoff's laws, e.g.:

$$\begin{aligned} v_{R_1,n} &= v_{C,p} \\ i_{R_1,n} + i_{C,p} &= 0 \end{aligned}$$

Very direct: can be accomplished through a drag-and-drop GUI.

Modelica (1)

- Modelica a current de-facto standard.
- Supported by a number of sophisticated implementations.
- In widespread use; e.g. in automotive industry (Ford, Toyota, . . .).

Modelica (2)

Issues:

- “Declarative”, but imperative origins show (both syntactically and semantically).
- Very complicated, rather ad-hoc type system.
- Limited hybrid modelling capabilities.

Modelica (2)

Issues:

- “Declarative”, but imperative origins show (both syntactically and semantically).
- Very complicated, rather ad-hoc type system.
- Limited hybrid modelling capabilities.

Can we do better (in some ways)?

Modelica (2)

Issues:

- “Declarative”, but imperative origins show (both syntactically and semantically).
- Very complicated, rather ad-hoc type system.
- Limited hybrid modelling capabilities.

Can we do better (in some ways)?

- Let’s have a peek at some Modelica type system aspects
- Will come back to the other points.

Quiz: Spot the Dependent Types!

(No prize, I'm afraid!)

- Modelica provides a rich sublanguage for expressing and operating on arrays.
- The number of dimensions and their sizes are considered to be part of the type.

```
function joinThreeVectors
  input Real v1[:],v2[:],v3[:];
  output Real v[size(v1,1)+size(v2,1)+size(v3,1)]
algorithm
  v := cat(1,v1,v2,v3);
end joinThreeVectors;
```

Array Checking in Modelica

- The Modelica standard promises very little:

Array Checking in Modelica

- The Modelica standard promises very little:
  - Size constraints will be checked at some point.

Array Checking in Modelica

- The Modelica standard promises very little:
  - Size constraints will be checked at some point.
  - When is a “quality-of-implementation issue”.

Array Checking in Modelica

- The Modelica standard promises very little:
  - Size constraints will be checked at some point.
  - When is a “quality-of-implementation issue”.
  - The common approach is to generate an assertion unless the involved sizes happen to be known statically.

Array Checking in Modelica

- The Modelica standard promises very little:
  - Size constraints will be checked at some point.
  - When is a “quality-of-implementation issue”.
  - The common approach is to generate an assertion unless the involved sizes happen to be known statically.
- But at least the need for flexible array sizes and checking them for consistency is recognized: the basic scaffolding is there for doing a better job.

Dimensional Types in Modelica (1)

Modelica has some support for “unit types”:

Dimensional Types in Modelica (1)

Modelica has some support for “unit types”:

- Units given as annotations (strings), e.g.:
  - `"N.m"`
  - `"kg.m/s2"`.
  - `"kg.m.s-2"`.

Dimensional Types in Modelica (1)

Modelica has some support for “unit types”:

- Units given as annotations (strings), e.g.:
  - `"N.m"`
  - `"kg.m/s2"`.
  - `"kg.m.s-2"`.
- No guarantees that dimension annotations will be checked for consistency (but some tools do do limited checking).

Dimensional Types in Modelica (2)

- Clearly a rather ad-hoc treatment.

Dimensional Types in Modelica (2)

- Clearly a rather ad-hoc treatment.
- While basic dimensional type checking does not require dependent types, dependent types might provide a good framework.

Dimensional Types in Modelica (2)

- Clearly a rather ad-hoc treatment.
- While basic dimensional type checking does not require dependent types, dependent types might provide a good framework.
- In its full generality, a dimension type may depend on data; e.g.:

$$x^n$$

The dimension of the result depends on the value of n.

Challenges (1)

Need to keep the end-users in mind. Type-checking must be essentially fully automatic.

Challenges (1)

Need to keep the end-users in mind. Type-checking must be essentially fully automatic.

- Power users/library developers do write code in the traditional sense.

Challenges (1)

Need to keep the end-users in mind. Type-checking must be essentially fully automatic.

- Power users/library developers do write code in the traditional sense.
- But they are experts in their domains, not Computer Scientists.

Challenges (1)

Need to keep the end-users in mind. Type-checking must be essentially fully automatic.

- Power users/library developers do write code in the traditional sense.
- But they are experts in their domains, not Computer Scientists.
- And they have a job to do.

Challenges (1)

Need to keep the end-users in mind. Type-checking must be essentially fully automatic.

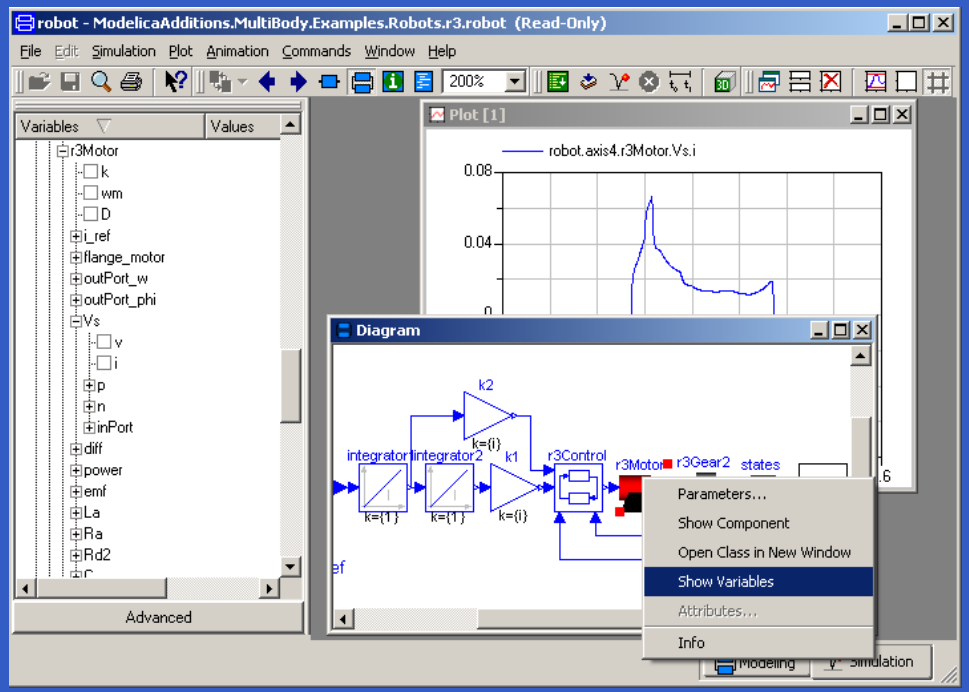
- Power users/library developers do write code in the traditional sense.
- But they are experts in their domains, not Computer Scientists.
- And they have a job to do.
- So unlikely to be impressed if asked to prove e.g.

$$n + m = m + n \quad \text{or} \quad m(n+1) = m + mn$$

(and might not know how).

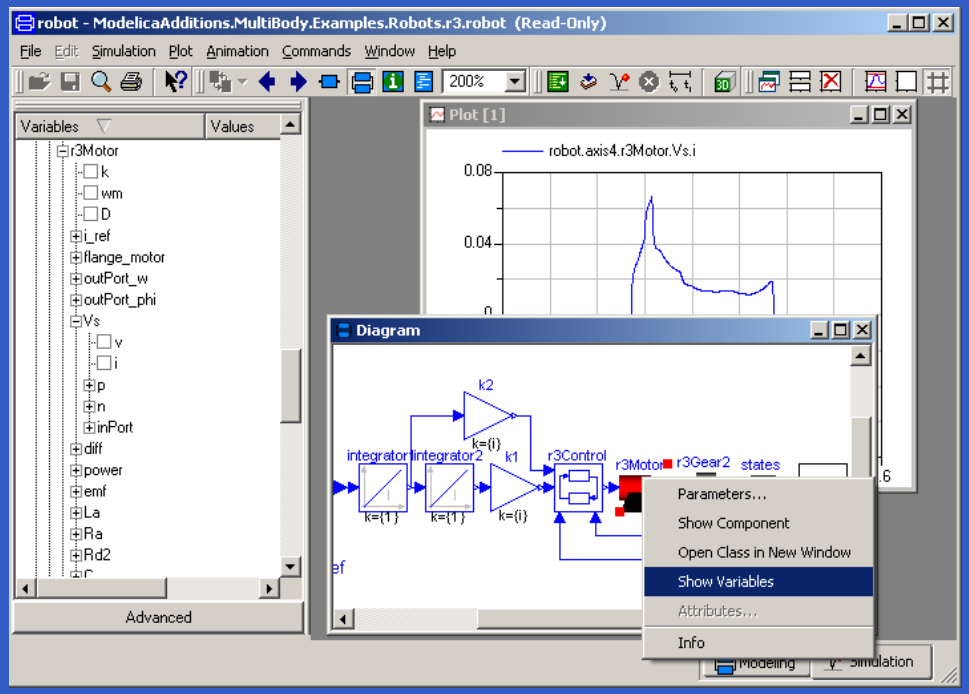
Challenges (2)

- Ordinary users construct models exclusively through drag-and-drop GUIs:



Challenges (2)

- Ordinary users construct models exclusively through drag-and-drop GUIs:



- They don’t expect prove things any more than when they wire real hardware together.

Functional Hybrid Modelling (1)

Recap: Modelica Issues

- “Declarative”, but imperative origins show (both syntactically and semantically).
- Very complicated, rather ad-hoc type system.
- Limited hybrid modelling capabilities

Functional Hybrid Modelling (1)

Recap: Modelica Issues

- “Declarative”, but imperative origins show (both syntactically and semantically).
- Very complicated, rather ad-hoc type system.
- Limited hybrid modelling capabilities

Goal for FHM:

Creating a powerful, fully-declarative modeling language by integrating key notions suitable for multi-domain, hybrid modeling into a functional language.

Functional Hybrid Modelling (2)

Why?

- Purely declarative language as a starting point.
- Semantic simplicity and insight.
- Expressiveness: with a better fundamental design, it might be possible to handle more general models.
- In particular, better hybrid capabilities.

Functional Hybrid Modelling (3)

Design inspiration: Yampa (Functional Reactive Programming):

- Yampa (in principle) supports ***causal*** modelling through functions on continuous signals.
- Such ***Signal Functions*** are first class entities in Yampa, allowing models to be ***computed*** using a functional language (Haskell).

Functional Hybrid Modelling (3)

- First class signal functions give:
  - (Some) meta modelling for free.
  - Very flexible hybrid modelling:
    - compute model
    - simulate until need to reconfigure
    - repeat until done

Functional Hybrid Modeling (4)

Same conceptual structure as Yampa, but:

Functional Hybrid Modeling (4)

Same conceptual structure as Yampa, but:

- First-class ***relations*** on signals instead of functions on signals to enable non-causal modeling.

Functional Hybrid Modeling (4)

Same conceptual structure as Yampa, but:

- First-class ***relations*** on signals instead of functions on signals to enable non-causal modeling.
- Employ state-of-the-art symbolic and numerical methods for sound and efficient simulation.

Functional Hybrid Modeling (4)

Same conceptual structure as Yampa, but:

- First-class ***relations*** on signals instead of functions on signals to enable non-causal modeling.
- Employ state-of-the-art symbolic and numerical methods for sound and efficient simulation.

For the purpose of this talk, think of a signal relation as a (fragment of) a DAE system of equations.

First class signal relations

The type for e.g. a binary signal relation:

$$\texttt{SR}\ (\texttt{Real}, \texttt{Real})$$

E.g. the derivative relation:

$$der :: \texttt{SR}\ (\texttt{Real}, \texttt{Real})$$

Defining relations

The following tentative construct denotes a signal relation:

sigrel $pattern$ **where** $equations$

The pattern introduces ***signal variables*** which at each point in time are going to be bound to to a "sample" of the corresponding signal.

Given $p :: t$, we have:

sigrel p **where** ... :: SR t

Modeling electrical components (1)

The type `Pin` is assumed to be a record type describing an electrical connection. It has fields v for voltage and i for current.

$twoPin$ :: `SR` (`Pin`, `Pin`, `Voltage`)

$twoPin =$ **sigrel** (p, n, v) **where**

$$v = p.v - n.v$$

$$p.i + n.i = 0$$

Modeling electrical components (2)

$$
\begin{aligned}
&resistor :: \texttt{Resistance} \rightarrow \texttt{SR}\,(\texttt{Pin}, \texttt{Pin}) \\
&resistor(r) = \textbf{sigrel}\ (p, n)\ \textbf{where} \\
&\qquad\qquad twoPin \diamond (p, n, v) \\
&\qquad\qquad r \cdot p.i = v \\
&inductor :: \texttt{Inductance} \rightarrow \texttt{SR}\,(\texttt{Pin}, \texttt{Pin}) \\
&inductor(l) = \textbf{sigrel}\ (p, n)\ \textbf{where} \\
&\qquad\qquad twoPin \diamond (p, n, v) \\
&\qquad\qquad l \cdot \textbf{der}(p.i) = v
\end{aligned}
$$

A Problem . . .

How can we ensure that the DAEs we “glue together” actually can be solved?

A Problem . . .

How can we ensure that the DAEs we “glue together” actually can be solved?

The most basic, necessary but not sufficient condition: the number of variables (unknowns) and equations must agree.

. . . and a first cut at a solution

Idea: index SR by the number of equations it provides for the variables of its interface:

$$
\begin{aligned}
&foo :: SR\ (Real, Real, Real)\ 2\\
&foo = \textbf{sigrel}\ (x, y, z)\ \textbf{where}\\
&\qquad f_1(x, y, z) = 0\\
&\qquad f_2(x, y, z) = 0
\end{aligned}
$$

$$
\begin{aligned}
&fie :: SR\ Real\ 1\\
&fie = \textbf{sigrel}\ w\ \textbf{where}\\
&\qquad foo \diamond (u, v, w)\\
&\qquad g(u, v, w) = 0
\end{aligned}
$$

A possible refinement (1)

Counting variables and equations is rather blunt:

$$\begin{aligned} x + y + z &= 0 \\ z &= 0 \\ z &= 0 \end{aligned}$$

The equations are not linearly independent; i.e., the ***determinant*** of the matrix of coefficients is 0.

A possible refinement (1)

Counting variables and equations is rather blunt:

$$\begin{aligned} x + y + z &= 0 \\ z &= 0 \\ z &= 0 \end{aligned}$$

The equations are not linearly independent; i.e., the ***determinant*** of the matrix of coefficients is 0.

But we cannot in general hope to be able to compute the determinants statically. And the DAEs are in general not going to be linear anyway.

A possible refinement (2)

We ***can*** do the second best thing: look at the ***incidence matrix***:

Equations

$$\begin{aligned} f_1(x, y, z) &= 0 \\ f_2(z) &= 0 \\ f_3(z) &= 0 \end{aligned}$$

Incidence Matrix

$$\begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

A possible refinement (2)

We ***can*** do the second best thing: look at the ***incidence matrix***:

Equations Incidence Matrix

$$\begin{aligned} f_1(x, y, z) &= 0 \\ f_2(z) &= 0 \\ f_3(z) &= 0 \end{aligned}$$

$$\begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Unless each variable can be paired with an equation in which it occurs, there is no hope of solving the equations.

A possible refinement (3)

So maybe we can index signal relations with incidence matrices?

$$foo :: SR\ (Real, Real, Real) \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$foo = \textbf{sigrel}\ (x, y, z)\ \textbf{where}$$
$$f_1(x, y, z) = 0$$
$$f_2(y, z) = 0$$

A possible refinement (4)

Incidence matrices are joined in the obvious way:

$$foo :: SR\ (Real, Real, Real) \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Equations

$$foo \diamond (u, v, w)$$
$$foo \diamond (w, x, y)$$

Incidence Matrix

$$\begin{array}{c} u\ \ v\ \ w\ \ x\ \ y \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

A possible refinement (5)

The difficult bit is when we need to choose which equations to pair with local variables:

$$
\begin{aligned}
&bar :: SR\ (Real, Real)\ \textit{???}\\
&bar = \textbf{sigrel}\ (u, y)\ \textbf{where}\\
&\qquad foo \diamond (u, v, w)\\
&\qquad foo \diamond (w, x, y)
\end{aligned}
$$

Depending on how the four available equations are paired with v, w, y, we get two possible incidence matrices for u and y: $(1\ 0)$, $(0\ 1)$.

A possible refinement (6)

- The ultimate choice depends on properties of the ***final, global*** system of equations, and numerical considerations.
- Somehow it is necessary to abstract away from that choice by some form of approximation.
- Note that $(1\ 1)$ is a safe approximation (we're back at basic variable and equation counting).

Summary

- Dependent types can help enforce useful model invariants.
- The type systems of existing modelling languages like Modelica already have dependent aspects and could benefit from a more principled treatment of those aspects.