Received: 23 April 2019

DOI: 10.1002/mma.5934

LETTER

WILEY

Conceptual modelling: Towards detecting modelling errors in engineering applications

Klaus Gürlebeck[1] | **Dmitrii Legatiuk**[1,2] | **Henrik Nilsson**[3] | **Kay Smarsly**[2]

[1]Chair of Applied Mathematics,Bauhaus University Weimar, Germany

[2]Chair of Computing in Civil Engineering, Bauhaus University Weimar, Germany

[3]Functional Programming Laboratory, School of Computer Science, University of Nottingham, Nottingham, UK

Correspondence
Dmitrii Legatiuk, Department of Applied Mathematics, Computing in Civil Engineering, Bauhaus University, Weimar, Germany.
Email: dmitrii.legatiuk@uni-weimar.de

Communicated by: S. Wolfgang

Funding information
German Research Foundation (DFG), Grant/Award Number: SM 281/9-1

Rapid advancements of modern technologies put high demands on mathematical modelling of engineering systems. Typically, systems are no longer "simple" objects, but rather coupled systems involving multiphysics phenomena, the modelling of which involves coupling of models that describe different phenomena. After constructing a mathematical model, it is essential to analyse the correctness of the coupled models and to detect modelling errors compromising the final modelling result. Broadly, there are two classes of modelling errors: (a) errors related to abstract modelling, eg, conceptual errors concerning the coherence of a model as a whole and (b) errors related to concrete modelling or instance modelling, eg, questions of approximation quality and implementation. Instance modelling errors, on the one hand, are relatively well understood. Abstract modelling errors, on the other, are not appropriately addressed by modern modelling methodologies. The aim of this paper is to initiate a discussion on abstract approaches and their usability for mathematical modelling of engineering systems with the goal of making it possible to catch conceptual modelling errors early and *automatically* by computer assistant tools. To that end, we argue that it is necessary to identify and employ suitable mathematical abstractions to capture an accurate conceptual description of the *process* of modelling engineering systems.

KEYWORDS

abstraction, engineering, formal approaches, modelling, type theory

MSC CLASSIFICATION

00A71; 68N18; 68Q55; 68Q60

1 | INTRODUCTION

The solution of engineering problems typically starts with the development of appropriate models in a step-by-step process: starting at the conceptual level, the models are gradually refined until a sufficient level of detail is achieved. Given the complexity of current engineering problems, the models are usually coupled multiphysics models, integrating submodels that are either physics based or data driven. Frequently, it is also a practical necessity to incorporate *existing* models, whatever their nature might be.

Errors can be introduced at any step in the modelling process, and the more complex the process, the larger the scope of making mistakes leading to errors. If errors are only discovered late in the modelling process, error correction can be very

costly. For example, assumptions that later turn out to be flawed may render the work carried out in subsequent modelling steps invalid: it is perfectly possible to state a conceptually *wrong* model and *solve* it correctly by numerical methods. With the growing complexity of engineering problems, reflected in a corresponding growth of model complexity, it is thus an increasingly pressing need to automatically detect errors as early as possible in the modelling process.

To this end, it is helpful to first identify the phases of the modelling process:

- Conceptual modelling: Concerned with capturing all aspects relevant to the problem under consideration as mathematical formulations.
- Instance modelling: Concerned with the solution procedure for these formulations, which can be done by means of analytic or numerical methods or, in general, by means of simulations.

We can then proceed to classify modelling errors accordingly:

- Conceptual errors: Related to the coherence of a model as a whole.
- Instance errors: Concrete errors related to practical aspects of modelling, eg, approximation quality and implementation.

It is relatively well understood how to find and correct instance errors, and addressing instance errors rarely requires going back to earlier steps in the modelling process. In contrast, conceptual errors are much less well understood, and as conceptual errors appear in the early stages of the modelling process, the impact of conceptual errors tend to be much more profound. Thus, it is necessary to provide techniques and tools for detecting conceptual errors as early as possible in the modelling process.

While trivial, the classical heat equation provides a helpful illustrative example:

$$c\rho\frac{\partial\theta}{\partial t} - \operatorname{div}\left(\lambda(\mathbf{x})\operatorname{grad}\,\theta\right) = 0, \mathbf{x} \in \mathbb{R}^n,$$

where c is the heat capacity, ρ is the material density, and λ is the thermal conductivity. When the thermal conductivity λ is constant, this equation simplifies to

$$c\rho\frac{\partial\theta}{\partial t} - \lambda\Delta\theta = 0.$$

In fact, this is the most common formulation of the heat equation, and because of its prevalence, a typical conceptual error is to take this simplified model to be valid also for nonconstant thermal conductivity by just making the conductivity a function of $\mathbf{x} \in \mathbb{R}^n$:

$$c\rho\frac{\partial\theta}{\partial t} - \lambda\Delta\theta = 0 \implies c\rho\frac{\partial\theta}{\partial t} - \lambda(\mathbf{x})\Delta\theta = 0. \tag{1}$$

This illustrates the nature of conceptual errors well: basic assumptions of models are violated at a later stage in the modelling process.

A formalised modelling process could account for how the simplified formulation is derived from the general formulation. Automatically catching conceptual mistakes like the one above would then at least be conceivable, as the derivation would no longer be valid if the assumption of constant conductivity is changed at a later point. Note that without any way of catching conceptual errors, the fact that there is an error would not become manifest until it is discovered that the model simply does not approximate the reality sufficiently well during validation. Moreover, the validation would not give any clear indication as to where the problem actually lies, resulting in time-consuming checks of the complete model. Detection of conceptual modelling errors prior to implementation thus requires a sound mathematical basis that allows an abstract description of a complete system. In particular, it is necessary to understand the general structure of a complete model and how different submodels, in the case of coupled problems, are coupled to each other.

In this short paper, as a first step towards improving early detection of conceptual errors, we suggest that type theory, in the sense used, eg, in programming languages,[1] should play a more prominent role in the modelling process, including at the early conceptual stages. Types reflect chosen characteristics of objects. When objects are composed to form more general objects, this allows checking that the composition is well formed with respect to the chosen characteristics and, if that is the case, to derive the characteristics of the compound object. Evidently, the type-theoretic approach reflects the basic idea of modelling coupled problems, where several submodels (objects) are coupled (composed) together to obtain a more general object, while also allowing checking of the composition. The objectives of this paper are to explore how models, particularly coupled models, can be described in a more profoundly typed way at the conceptual stage, how such a description can facilitate an early detection of modelling errors, and, expanding work initiated in Legatiuk and Nilsson,[2]

outlining how suitable existing languages, such as the functional language Haskell, can be leveraged to serve as a tool to check the coherence of the models through the approach of language embeddings.

At this initial stage, we only concern ourselves with only physics-based models. Since physics-based models are built upon a solid mathematical foundation, it is expected that the type-theoretic representation of physics-based models will be richer than the representation of data-driven models. Moreover, the advantage of the compositional type-theoretic approach to conceptual modelling over a direct use of computer algebra systems (CASs) is a strict semantic based on types, which is not supported by modern CASs. The absence of strict semantic in CASs implies that although symbolic calculations can be carried out in full generality, the decision upon model correctness and coherence lies on a modeller and is not made automatically by a CAS.

2 | CONCEPTUAL APPROACHES TO MATHEMATICAL MODELLING

Several approaches towards conceptual modelling of engineering systems based on abstract mathematics have been proposed in recent years. Keitel et al[3] proposed an abstract approach towards engineering modelling based on graph theory. In this case, models are considered as vertices of graphs with edges representing model couplings. However, the focus of the work was not related to the detection of conceptual modelling errors, but to practical evaluation of models in the instance modelling phase based on uncertainty and sensitivity analyses. In contrast, Gürlebeck et al[4] proposed a modelling framework based on category theory where models are treated as abstract objects in categories and coupling of models is described by functorial mappings between the categories. The category theory-based approach allows checking the consistency of the modelling process and thus detecting modelling errors. Moreover, the category theory-based modelling methodology has recently been applied to practical engineering problems from the field of aerodynamic analysis of bridges,[5] indicating practical advantages of conceptual modelling.

A formal approach to describing mathematical models in general based on functional analysis has been proposed by Dutailly.[6,7] Mathematical models (or systems) are represented by sets of variables the models contain, and it has been shown that, if the set of variables satisfy some specific conditions, there is an abstract Hilbert space corresponding to such model. Although a formal description of models including evolutional model behaviour has been presented, the idea of detecting modelling errors by this formal approach has not been addressed. Another way to describe the modelling process is to use the tools of logic, such as model theory.[8] However, applying model theory and lower predicate calculus typically requires a more formal construction that makes it less practical. Nonetheless, the connection between system modelling and logical equations has been actively investigated in recent years,[9,10] demonstrating that system models can be turned into logical equations allowing important model properties to be proven by studying the corresponding logical equations. However, the results are related to the use of logic for modelling purposes, focusing on a specific class of models, and do not address the problem of error detection for general models of mathematical physics.

The general task of detecting modelling errors in the conceptual modelling phase necessitates the construction of a framework that lends itself to computer implementation. One possibility is to base the framework on automated theorem proving, requiring a logical description of the corresponding mathematical model.[11] Alternatively, the framework can be established based on type theory. In type-theoretic setting, a particular type is assigned to each basic model component, and a complete model is constructed by combining basic model components, ie, by matching their types. The coherence check of the complete model is then shifted to a type system governing the conditions under which typed components may be combined, inferring the type of the combined entity from this. Moreover, the use of type theory, on the one hand, provides a strong mathematical formalism, and on the other hand, it renders the construction more transparent, compared with logical equations, because only by considering the type restrictions of terms it is possible to deduce whether model components may be combined. That said, it is not a binary choice, but rather a spectrum of possibilities of varying expressiveness: dependent types,[12] in their full generality, allow arbitrary logical predicates to be expressed at the type level, while work such as LiquidHaskell[13] demonstrate one approach to support selected classes of predicates at the type level while retaining full automation through the integration of a satisfiability modulo theories solver (SMT solver). Finally, the idea of composing types is typical for many programming languages, particularly for functional programming languages such as Haskell.[14] Therefore, the type checker of Haskell can be used for coherence checks. Thus, exploring the link between functional programming and modelling, a type theory-based modelling approach can be embedded into Haskell. Moreover, this Haskell embedding will be benefit from a support of symbolic computations simplifying the job of the Haskell type checker. One approach would be to extend the type checker with computer algebra capabilities. Works dealing with a combination of CASs and automated theorem provers have been presented in recent years.[15,16] Thus, a combination of the Haskell type checker with a CAS, such as Maple, could be seen as a distant goal for the

application of type theory to conceptual modelling. Another approach could be to generalise the type with logical predicates, leveraging an SMT solver to ensure that type checking is still fully automatic along the lines of what has been done in LiquidHaskell.[13]

The idea of using an embedding in Haskell for modelling physical systems dates back to functional hybrid modelling (FHM),[17] which is a prototype Haskell-embedded modelling language. Nilsson[18] later proposed a novel approach to a type system for equation systems constructed by composing individual equation system fragments. This work was later refined and given formal semantics by Capper and Nilsson.[19,20] The main purpose was to study how concrete modelling languages, like FHM, can be extended with consistency checks on models, particularly to ensure solvability of the system of equations. The decision upon solvability of the system is done using the Haskell type checker via appropriately constructed structural types. Inspired by these works, in Legatiuk and Nilsson[2] first steps towards the extension of FHM to realistic engineering modelling with partial differential equations was presented. However, here, the focus is not on a concrete language for practical modelling but on conceptual modelling error detection. A further development of the type-theoretic approach to mathematical modelling is proposed in the next section.

3 | MATHEMATICAL MODELLING AND TYPES

This section outlines a type-based approach to mathematical modelling. Following the general concept of FHM,[18] we extend the original idea of embedding differential equations in a functional language through first-class notions of functions and relations on *signals* to a more general class of models, in particular, not restricted to ordinary differential equations and differential-algebraic equations as in the case of FHM, but allowing partial differential equations. The term "signal" was a natural choice in the FHM, since it has been designed for initial value problems represented by differential-algebraic equations with applications in electrical engineering. Problems in mathematical physics are more involved, because the corresponding models may depend on several variables or are time independent. Nonetheless, we continue to use the term *signal* for the unknowns, but keeping in mind that signals are now more general as they can be *temporal*, *spatial*, *spatio-temporal*, or even *constant*. The distinction between different kinds of signals should become clear from the signal type signature.

3.1 | Signals and signal relations

By construction, equations of mathematical physics have a clear physical meaning, which is typically reflected by the solutions. Therefore, a type system that reflects the physical background of a model is an attractive proposition. For example, the type of a function describing one-dimensional displacements could be written as `(Time, Coordinate) → Displacement`, and the type for a temperature field could then be `(Time, Coordinate) → Temperature`. The typing here makes modelling more transparent because the physical interpretations of quantities can be grasped directly from the type signatures. However, such an approach also implies that, in general, a unique set of type is required for each problem of mathematical physics. This significantly limits the reuse of the type system. Therefore, instead of using the explicit "physical" typing, we propose to work directly with functions and mathematical operations used to model a given physical phenomenon. Following this concept, a set of independently typed primitives for model components is introduced, and a model of arbitrary complexity can be obtained by composing model components. Moreover, the lack of a clear physical interpretation of types could be compensated at a later stage by using a more advanced type system keeping track of physical dimensions[21] and possibly also units of measure, eg, preventing confusion between metric and imperial units.

To establish a type system applicable to different problems of mathematical physics, we first introduce the finite-dimensional set $\{\xi_i\}_{i=0}^{n}$ of variables with the conventions that ξ_0 corresponds to the time variable, and $\xi_i, i = 1, \dots, n$, correspond to the spatial variables used in a model, and n being the total number of spatial dimensions. For simplicity, we take the type of the time variable and the spatial variables to be $\mathbb{R}$, and to avoid confusion about the ordering of ξ_i, we assume that the ordering of the spatial variables follows the standard ordering of unit vectors in $\mathbb{R}^n$. It follows that, for now, we make the simplifying assumption that all models in principle are time dependent, while the number of spatial dimensions can vary. This is sufficiently general to discuss a coupling of models of different dimensionality, as well as coupling of time-dependent and static models.

The original problem must thus be reformulated in terms of the new variables ξ_i, $i = 0, 1, \ldots, n$. The reformulation allows us to introduce the polymorphic type of signals as follows:

$$\mathtt{S}\ \alpha = \mathbb{R}^{(n+1)} \to \alpha,$$

where $\mathbb{R}^{(n+1)}$ is the type of a vector of time and space coordinates, and α is the type of values "carried" by the signal, ie, its value at a specific point in time and space. The concrete type α depends on the model under consideration: for scalar-valued quantities, α is a base type; for vector-valued quantities, α is a product type.

A signal thus represents the unknown function in the equation(s) of a mathematical model. Signals therefore only exist implicitly: otherwise, the solution of a problem would already be given from the outset. Consequently, the signal type is *conceptual* and never used explicitly. What is typed explicitly are the *equations* that characterise signals, leading us to the notion of *signal relations* to which we will turn shortly.

For the convenience of future practical realisation of the language proposed in this paper, and to express n-ary relations on signals, we view a signal carrying elements of a product type $\mathtt{S}\,(T_1 \times T_2 \times \cdots \times T_n)$ as isomorphic to a product of signals $(\mathtt{S}T_1) \times (\mathtt{S}T_2) \times \cdots \times (\mathtt{S}T_n)$.

Remark 1. In practice, following the variable convention ξ_i, $i = 0, 1, \ldots, n$ set out above means that a problem formulated in tensor setting,[22] complex setting,[23] or hypercomplex setting[24] needs to be rewritten in a component form. It may seem restrictive to put strong demands on the formulation of models, but after completing the type-theoretic setting for componentwise formulations of problems, the extension of basic type system to more general formulations will be straightforward.

We now introduce the type of *signal relations*: relations on signals of some specific spatial dimensionality n. The type

$$\mathtt{SR}\ (n :: \mathbb{N})\alpha$$

is the type for a relation on a signal of type $(\mathtt{S})\ \alpha$ depending on time and n spatial dimensions with $\mathbb{N} = \{0, 1, 2, \ldots\}$ denoting the set of natural numbers. To relate two or more signals, we rely on the isomorphism between a products of signals and a signals of a products as discussed above. For example. the type of a relation between two signals of n spatial dimensions carrying concrete types T_1 and T_2, respectively, would be $\mathtt{SR}\,n(T_1, T_2)$.

For the general notations for defining relations on signals, we use the following notation originating in λ-abstraction

sigrel *pattern* **where** *equations*.

The *pattern* introduces signal variables that are bound to the value of the corresponding signal at each point in time and space. For example, for a given signal variable p of a given type t, ie, $p :: t$, the notation introduced leads to

sigrel p **where** $\ldots :: \mathtt{SR}\ nt$.

To describe concrete mathematical models, we introduce two kinds of equations:

$$e_1 = e_2,\ sr \diamond e_3,$$

where e_i, $i = 1, 2, 3$ are expressions that are allowed to introduce new variables and sr is an expression denoting a signal relation. The signals characterised by these equations and all involved signal relations are all for the same number n of spatial dimensions as reflected by the type of the overall signal relation. Both equations are further restricted to be well typed: if $e_i :: T_i$, $i = 1, 2, 3$, then $T_1 = T_2$ and $sr :: \mathtt{SR}\,nT_3$. The first kind of equation requires the values of the two expressions to be equal at all points in space and time. The second kind of equation introduces arbitrary relations on signals, where the symbol $\diamond$ is understood as a *relation application* resulting in a constraint that must hold at all points in space and time. Moreover, naturally, the first kind of equation is just a special case of the second kind, as equality is a subset of general relations on signals.

As noted above, it is often required to combine models with different number of spatial dimensions or, more generally, different coordinate systems. In our setting, models are signal relations, so the question then is how to relate the coordinate system of one signal relation to that of another when the systems are not the same. This can be accomplished by an

operation that transforms a signal relation by applying an affine transformation to the (spatial part of) coordinate vectors. Such a transformation might have a type along the lines

$$\texttt{SR}\ m\alpha \to \mathbb{R}^{m\times(n+1)} \to \texttt{SR}\ n\alpha,$$

where $\mathbb{R}^{m\times(n+1)}$ is the type of an m by $n+1$ real-valued matrix representing the affine transformation of coordinates in the space of the resulting signal relation of type $\texttt{SR}\ n\alpha$ (n spatial dimensions) to coordinates in the space of the transformed signal relation of type $\texttt{SR}\ m\alpha$ (m spatial dimensions).

As we do not need any such operation in the following, we do not discuss this point further here. However, we do note that the above type is an example of a *dependent* type,[12] of which we will encounter further examples in the following. In more detail, the actual type of the transformation would be

$$\{mn :: \mathbb{N}\} \to \texttt{SR}\ m\alpha \to \mathbb{R}^{m\times(n+1)} \to \texttt{SR}\ n\alpha.$$

Thus, the dimensions of the signal relations and the size of the transformation matrix depends on the *values* of the first two arguments m and n to the transformation operation, hence, *dependent* type. Following the conventions of the dependently typed language Agda,[25] the two first arguments have been enclosed in braces to indicate that they are *implicit*, meaning that they can be left out whenever they can be inferred from the context, which usually is the case for the number of dimensions of a model.

3.2 | Typing of few basic operators

The practical use of signal relations also requires typing of the mathematical tools used to express a model. We illustrate by returning to our running example, the heat equation:

$$c(\mathbf{x})\rho(\mathbf{x})\frac{\partial\theta}{\partial t} - \operatorname{div}\left(\lambda(\mathbf{x})\operatorname{grad}\theta\right) = f(\mathbf{x},t). \tag{2}$$

Typically, modelling of physical phenomena results in formulations of initial boundary value problems for equations. However, as this study aims at developing a type-theoretic approach for conceptual modelling error detection, it is not necessary to formulate initial/boundary conditions for a single model. Because initial/boundary conditions characterise a particular behaviour of a solution, rather than the model. Therefore, the heat equation (2) can be used together with different types of initial/boundary conditions. However, for coupled problems, the situation is more involved. This is because coupling or transmission conditions that must be satisfied on the boundary (or interface) between the models needs to be formulated. These conditions are derived directly from the modelling assumptions and must be taken into account by the conceptual modelling approach.

Returning to Equation (2), the mathematical operators, such as ∂_t, div, grad, and the unknown signal $\theta(\mathbf{x},t)$, heat capacity $c(\mathbf{x})$, material density $\rho(\mathbf{x})$, and variable thermal conductivity $\lambda(\mathbf{x})$ must be typed. For the sake of completeness, we recall the definitions of divergence and gradient:

$$\operatorname{div}\mathbf{u} \coloneqq \nabla\cdot\mathbf{u} = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \cdots + \frac{\partial u_n}{\partial x_n},\ \operatorname{grad} f \coloneqq \nabla f = \frac{\partial f}{\partial x_1}\mathbf{e}_1 + \frac{\partial f}{\partial x_2}\mathbf{e}_2 + \cdots + \frac{\partial f}{\partial x_n}\mathbf{e}_n,$$

where $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ is a vector-valued function of n variables x_i, $i = 1, \ldots, n$, $f = f(x_1, \ldots, x_n)$ is a scalar-valued function, and $\mathbf{e}_i$, $i = 1, \ldots, n$ are the standard unit vectors in $\mathbb{R}^n$.

Assuming the convention for new variables discussed above, $\{\xi_i\}_{i=0}^n$, the partial differentiation wrt individual variables can be written as a general operator[2]:

$$D_i^s \coloneqq \frac{\partial^s}{\partial\xi_i^s}, \tag{3}$$

where i is the index of a differentiation variable and s is the order of the derivative. The type of the operator D_i^s is

$$D :: \{n :: \mathbb{N}\} \to \mathbb{N}_{\le n} \to \mathbb{N} \to \texttt{SR}\ n(\alpha,\alpha), \tag{4}$$

where $\mathbb{N}_{\le n}$ is the type of natural numbers up to and including n. Note that this is another example of a dependent type, with the dimension argument implicit as it usually will be implied by the context. Both the variable index i and the derivative order s are restricted to natural numbers as we do not consider fractional derivatives at present.

As divergence and gradient require only first-order partial derivatives wrt the *spatial* variables, the divergence and gradient can be defined by a recursive application of operator D_i^1 to a vector-valued signal and via mapping the operator D_i^1 over a vector filled in by a scalar-valued function, respectively. The types of div and grad are

$$\texttt{div} :: \{n :: \texttt{Nat}\} \to \texttt{SR}\ n(\alpha^n, \alpha),\ \texttt{grad} :: \{n :: \texttt{Nat}\} \to \texttt{SR}\ n(\alpha, \alpha^n), \tag{5}$$

where α is a base type and α^n is an n-ary vector of α, ie, a product of n αs. Note that this, again, is a dependent type, and that the number of spatial dimensions, again, is an implicit argument, allowing it to be omitted whenever it is implied by the context, which we exploit in the following.

The composition of these operators is not commutative:

$$\texttt{div} \circ \texttt{grad} :: \texttt{SR}\ n(\alpha, \alpha),\ \texttt{grad} \circ \texttt{div} :: \texttt{SR}\ n(\alpha^n, \alpha^n),$$

where $\circ$ is relational composition. Moreover, $\texttt{div} \circ \texttt{grad}$ defines the classical Laplace operator, which can be applied to scalar-valued functions or mapped over vector-valued functions.

The operators D, div, and grad can now be used to relate signals by means of relation application:

$$D(i)(s) \diamond (f_1, g_1),\ \texttt{div} \diamond (f_2, g_2),\ \texttt{grad} \diamond (f_3, g_3).$$

The meaning of this is given by the following equations:

$$f_1(\xi) = \frac{\partial^s g_1(\xi)}{\partial \xi_i^s},\ f_2(\xi) = \sum_{j=1}^{n} \frac{\partial (g_3)_j(\xi)}{\partial \xi_j},\ f_3(\xi) = \sum_{j=1}^{n} \frac{\partial g_3(\xi)}{\partial \xi_j} \mathbf{e}_j,$$

where $\boldsymbol{\xi}$ denotes the vector of the variables $\xi_0, \xi_1, \ldots, \xi_n$.

To make it easier to write equations, a surface syntax, closer to the classical mathematical notations than the relational syntax introduced above, could be adopted:

$$f_1 = \mathrm{D}(i)(s)\mathrm{g}_1,\ f_2 = \texttt{div}\ g_2,\ f_3 = \texttt{grad}\ g_3.$$

With this notation, we can view application of D, div, and grad to a signal as resulting in a new signal. This is just a more concise, arguably more intuitive, syntax: the underlying meaning is exactly as above.

3.3 | General algorithm for model consistency check

We now outline an algorithm for checking the consistency of a conceptual model, using the heat equation (2) as an example once more. The conceptual error discussed in the introduction can be detected by checking the types of basic modelling assumptions constituting the model of heat conduction that ultimately results in Equation (2), as opposed to checking the final model only. In particular, the couplings in coupled models are typically based on such assumptions, not on the final model form, meaning that a comprehensive coherence check is not possible if only the final model is considered.

The classical model of heat conduction is defined by three main modelling assumptions,[26] representing basic laws in physics:

1. Fourier's law: if the temperature of a body is non-uniform, heat currents arise in the body, directed from points of higher temperature to points of lower temperature. For the heat flux density $\vec{q}$ this reads as

$$\vec{q} = -\lambda\ \mathrm{grad}\ u. \tag{6}$$

Integrating (6) over the surface of an arbitrary reference domain $\Omega_0 \subset \Omega$, we obtain

$$Q_1 = -dt \int_{\partial\Omega_0} \lambda(x) \frac{\partial u}{\partial n} d\Gamma, \tag{7}$$

where Q_1 is the amount of heat flowing through the surface of Ω_0 in the time interval of length dt and $d\Gamma$ stands for the area element on $\partial\Omega_0$.

2. The amount of heat that is needed in order to change the temperature of Ω_0 in the time interval of length dt is equal to

$$Q_2 = dt \int_{\Omega_0} c\rho \frac{\partial u(x,t)}{\partial t} dV. \tag{8}$$

3. The amount of heat generated or absorbed inside the body is given by

$$Q_3 = dt \int_{\Omega_0} F(x,t) dV, \tag{9}$$

where $F(x,t)$ is the density of the heat source or the heat sink and dV is the volume element.

Using formulae (7) to (9) and the principle of conservation of energy, we get

$$\int_{\Omega_0} c\rho \frac{\partial u(x,t)}{\partial t} dV = \int_{\partial\Omega_0} \lambda(x) \frac{\partial u}{\partial n} d\Gamma + \int_{\Omega_0} F(x,t) dV. \tag{10}$$

Additionally, we need to introduce two mathematical relations, playing the role of basic operators. One is the Gauß integral theorem, and the other the concept of defining a density by shrinking Ω_0 to a point x by taking the limit (under certain assumptions for f) $\lim_{\Omega_0 \mapsto \{x\}} \frac{\int_{\Omega_0} f(s) dV}{\int_{\Omega_0} 1 dV} = f(x)$. Finally, applying these two mathematical tools to integral (10) yields the differential equation (2). Once this procedure has been implemented, it becomes easy to check whether transmission conditions on a coupling interface are modelled correctly. It also becomes clear why the derivation of (1) was not correct. Identification of such conceptual modelling errors requires working with basic physical assumptions because the final form of the differential equation does not provide sufficient evidence for their detection.

In our setting, the definite integral in one dimension can be typed straightforwardly as follows:

$$\texttt{integral} :: \{n :: \mathbb{N}\} \to \mathbb{N}_{\leq n} \to (\mathbb{R}, \mathbb{R}) \to \texttt{SR}\ n(\alpha, \alpha), \tag{11}$$

where $\mathbb{N}_{\leq n}$ is the type of the index of the integration variable and $(\mathbb{R}, \mathbb{R})$ is the type of the integral bounds. Multidimensional integration can now be defined by iterative application of (11).

Using the integral above together with the notation and operators defined in the previous section, the basic modelling assumptions (6) to (10) can now be transliterated into our typed setting. However, as the objective of this paper is to initiate a discussion on using type theory for modelling, we omit the details of this translitaration. Instead, we discuss how the typed modelling framework outlined above can be used as a foundation for an algorithm for checking model coherence:

Given:	**Typing of basic modelling assumptions in a form of a library.**
`Step 1:`	Declare concrete types for variables, functions, coefficients;
`Step 2:`	Select basic modelling assumptions from the library;
`Step 3:`	Compiler checks if the resulting expression is well typed;
`Step 4:`	Performing symbolic calculations on the model equations, if possible;
Output:	Final form of the model with type signatures of all its components.

The algorithm has two main assumptions: (a) the basic modelling assumptions are formalised in our typed framework, and (b) the compiler provides support for symbolic calculations. The first assumption has been addressed earlier when introducing typing of basic mathematical operators. The mathematical operators are considered as primitive objects of formalisation of modelling assumptions. Thus, basic modelling assumptions can be introduced as compositions of basic operators, for example,

$$\texttt{Fouriers\_law} := -S \cdot \texttt{integral0}(t1, t2) \circ (\lambda \cdot D11),$$

and the Haskell type checker will infer the types of the modelling assumptions. In this way, the library of typed basic modelling assumptions can be created.

The second assumption on the algorithm is mostly a question of implementation using suitable components such as a host language compiler and a CAS. Consider a formal representation of the model of heat conduction as a composition of basic modelling assumptions:

```
Heat_Model := Diff ∘ (Conservation_energy_principle ∘ (Fourier's_law ∘ Amount_of_heat ∘
Heat_source)),
```

where `Fouriers_law`, `Amount_of_heat`, and `Heat_source` are formalised modelling assumptions; `Conservation_energy_principle` is the formalised principle of conversation of energy leading to the integral formulation (10); and `Diff` is the function converting the integral form of the heat equation into differential form (2) by using the Gauß integral theorem and taking the limit. We use parentheses to avoid making assumptions regarding the associativity of composition.

An alternative, simplified, approach to the typing of basic physical assumptions and providing symbolic calculations capabilities would be to declare directly the differential form of the model of heat conduction:

$$
\begin{aligned}
&\mathit{Heat} :: \mathtt{SR}\ 3(\mathbb{R},\mathbb{R})\\
&\mathit{Heat} = \mathbf{sigrel}\quad (\theta, f)\quad \mathbf{where}\\
&\qquad f = c\cdot\rho\cdot D(0)(1)\theta - \mathtt{div}\circ(\lambda\cdot\mathtt{grad})\theta\\
&\qquad \lambda = \mathit{expression},
\end{aligned}
$$

where material constants c and ρ are understood as constant signals, while the thermal conductivity λ is a signal, which is not necessarily a constant. This alternative approach is less general than the first approach, but perhaps more transparent. Moreover, the support by symbolic calculations is also beneficial: if the expression for λ is indeed a constant, the whole expression can be simplified. We would like to remark that the question of regularity constrains of functions and parameters is not in the scope of current research, although it is important for practical use of models.

Irrespective of the alternative chosen for type-theoretic model representation, the proposed type-theoretic formalism allows avoiding conceptual modelling errors discussed in the introduction. Since the final expression must pass the Haskell type checker, any expression, ie, model, which is not well typed will be rejected as inconsistent. Moreover, the error messages related to type checking will indicate clearly the sources of conceptual modelling errors. Thus, the model coherence can be automatically assured in the conceptual modelling phase, ie, before starting actual implementation of concrete solution procedures.

4 | CONCLUSIONS AND FUTURE WORK

The complexity of modern problems of mathematical physics and engineering necessitates a deeper analysis of coupled mathematical models wrt their global conceptual correctness and coherence than what current modelling approaches can offer. In particular, automated detection of conceptual modelling errors, ie, violation of basic physical laws and assumptions, is a key priority as the impact of conceptual modelling errors on the final model can be profound. However, modern modelling methodologies do not properly address conceptual modelling error detection. Given the coupled nature of modern problems, detection of conceptual modelling errors requires tools of abstract mathematics that adequately can capture the structure and semantics of a model.

To address this problem, we, in this short paper, initiated a discussion on the use of abstract approaches for mathematical modelling of engineering systems. The use of type theory plays a special role in our proposed framework. There are two key reasons for this choice: (a) type theory provides a strong formal foundation for the conceptual modelling, and (b) type-theoretic ideas can be implemented straightforwardly in a functional programming language, such as Haskell or Agda, for example. The type-theoretic approach requires a type system that allows the description of mathematical models. We outlined such a system, illustrating its use on a simple example of a heat equation, and, with that system as a starting point, we identified the key steps of an algorithm for automated detection of conceptual modelling errors. Future work includes further development of the type system for a formalised modelling process in mathematical physics, along with a prototype computer implementation. Further study of the connections between the type-theoretic approach and symbolic computation are required to that end.

ACKNOWLEDGEMENT

This research is partially supported by the German Research Foundation (DFG) through grant SM 281/9-1. The support offered by DFG is gratefully acknowledged. Any opinions, findings, conclusions, or recommendations expressed in this paper are solely those of the authors and do not necessarily reflect the views of DFG. This work does not have any conflicts of interest.

ORCID

Dmitrii Legatiuk https://orcid.org/0000-0002-0028-5793

REFERENCES

1. Pierce BC. *Types and Programming Languages*. Cambridge, Massachusetts: The MIT Press; 2002.
2. Legatiuk D, Nilsson H. Abstract modelling: towards a typed declarative language for the conceptual modelling phase. In: Proceedings of 8Th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools; 2017; Weßling, Germany.
3. Keitel H, Karaki G, Lahmer T, Nikulla S, Zabel V. Evaluation of coupled partial models in structural engineering using graph theory and sensitivity analysis. *Eng Struct*. 2011;33:3726–3736.
4. Gürlebeck K, Hofmann D, Legatiuk D. Categorical approach to modelling and to coupling of models. *Math Methods Appl Sci*. 2017;40(3):523–534.
5. Kavrakov I, Legatiuk D, Gürlebeck K, Morgenthal G. A categorical perspective towards aerodynamic models for aeroelastic analyses of bridges. *R Soc Open Sci*. 2019;6:181848.
6. Dutailly JC. Hilbert spaces in modelling of systems. 47 pages, HAL Id: hal-00974251; 2014.
7. Dutailly JC. Common structures in scientific theories. 34 pages, HAL Id: hal-01003869; 2014.
8. Robinson A. *Introduction to Model Theory and to the Metamathematics of Algebra*. Amsterdam: North-Holland Publishing Company; 1963.
9. Vassilyev SN. Method of reduction and qualitative analysis of dynamic systems: I. *J Comput Syst Int*. 2006;45(1):17–25.
10. Vassilyev SN, Davydov AV, Zherlov AK. Intelligent control via new efficient logics. In: Proceedings of the 17th World Congress The International Federation of Automatic Control Seoul, Korea, July 6-11; 2008.
11. Vassilyev SN, Druzhinin AE, Morozov NY. Derivation of preservation conditions for properties of mathematical models. *Dokl Math*. 2015;92(3):658–663.
12. Barthe G, Coquand T. An introduction to dependent type theory. In: Barthe G, Dybjer P, Pinto L, Saraiva J, eds. *Applied Semantics. APPSEM 2000. Lecture Notes in Computer Science*, Vol. 2395. Berlin, Heidelberg: Springer; 2002.
13. Jhala R, Seidel E, Vazou N. Programming with refinement types. An introduction to LiquidHaskell, https://ucsd-progsys.github.io/liquidhaskell-blog/; 2017.
14. Bird R. *Thinking Functionally with Haskell*. Cambridge, United Kingdom: Cambridge University Press; 2015.
15. Gottliebsen H, Kelsey T, Martin U. Hidden verification for computational mathematics. *J Symbolic Comput*. 2005;39:539–567.
16. Kaliszyk C, Wiedijk F. Certified computer algebra on top of an interactive theorem prover. In: Kauers M, Kerber M, Miner R, Windsteiger W, eds. *Towards Mechanized Mathematical Assistants. MKM 2007*, Calculemus 2007. volume 6546 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer.
17. Nilsson H, Peterson J, Hudak P. Functional hybrid modeling. *Volume 2562 of Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer; 2003.
18. Nilsson H. Type-based structural analysis for modular systems of equations. In: 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools. July 8; 2008; Paphos,Cyprus.
19. Capper J, Nilsson H. Static balance checking for first-class modular systems of equations. In: Page R, Horváth Z, Zsók V, eds. *Trends in Functional Programming: 11th International Symposium, TFP 2010*, May 2010, Revised Selected Papers, volume 6546 of Lecture Notes in Computer Science. Norman, OK, USA: Springer; 2011:50–65.
20. Capper J, Nilsson H. Structural types for systems of equations: type refinements for structurally dynamic first-class modular systems of equations. *High-Order Symb Comput*. 2012;25(2-4):275–310.
21. Kennedy A. Dimension types. In: Proceedings of the 5th European Symposium on Programming; 1994:348–362.
22. Lurie AI. Theory of elasticity. *Foundations of Engineering Mechanics*. Berlin Heidelberg: Springer; 2005. Translated from Russian.
23. Muskhelishvili NI. *Some Basic Problems of the Mathematical Theory of Elasticity*. Dordrecht: Springer Science+Business Media; 1977.
24. Gürlebeck K, Habetha K, Spröig W. *Application of Holomorphic Functions in Two and Higher Dimensions*. Switzerland: Springer; 2016.
25. Norell U, Chapman J. Dependently typed programming in Agda. In: Proceeding AFP'08 Proceedings of the 6th international conference on Advanced functional programming; 2008; Heijen, The Netherlands:230–226. May 19–24.
26. Tikhonov AN, Samarskii AA. *Equations of Mathematical Physics*. New York: Dover Publications, Inc.; 1963.

How to cite this article: Gürlebeck K, Legatiuk D, Nilsson H, Smarsly K. Conceptual modelling: Towards detecting modelling errors in engineering applications. *Math Meth Appl Sci*. 2019;1–10. https://doi.org/10.1002/mma.5934