

FRIEDRICH-ALEXANDER-UNIVERSITÄT
ERLANGEN-NÜRNBERG

PROFESSUR FÜR WISSENSREPRÄSENTATION
UND -VERARBEITUNG



Knowledge Representation for Modeling and Simulation

– Bridging the Gap Between Informal PDE Theory and
Simulations Practice –

Master's Thesis in Computational Engineering

Author:

Theresa Pollinger
theresa.pollinger@fau.de

Supervisor:

Prof. Dr. Michael Kohlhase

November 9, 2017

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 09. November 2017

Contents

1. Automated Assistants for Mathematical Modeling and Simulation	2
1.1. Related Approaches	2
1.2. The Vision	4
1.3. A Running Example: Solving the Heat Equation	4
1.4. Scope and Aims of This Thesis	6
2. Preliminaries	7
2.1. Mathematical Description of Partial Differential Equations	7
2.1.1. Second Order Linear Partial Differential Operators	7
2.1.2. Boundary Conditions for (Unique) Solvability	9
2.2. Solving PDEs Numerically: The Finite Difference Method	15
2.3. The ExaStencils Framework for Stencil Codes	17
2.4. Theory Graphs	19
2.5. Systems for Mathematical Knowledge Management	22
2.5.1. Semantic Web	22
2.5.2. OMDoc: Open Mathematical Documents	23
2.5.3. $\text{\texttt{gTeX}}$: Semantic Extension for $\text{\texttt{LATEX}}$	23
2.5.4. MMT: Meta-Meta Theories	25
2.6. Active Documents	27
3. Bridging the Gap Between Informal PDE Theory and Simulations Practice	30
3.1. A Software Architecture Proposal	30
3.2. A Dream: The Perfect Interview	33
3.3. Ephemeral Theories	40
3.4. A Theory Graph to Rule Them All	41
3.5. A Prototype Proposal: Conducting an Interview	45
4. Formal and Flexiformal PDE Knowledge	47
4.1. Formalizations	47
4.2. Flexiformalizations	51
4.3. A First Implementation: $\text{\texttt{THEINTERVIEW}}$	52
5. Conclusion and Future Work	54
A. The Full Theory Graph for PDEs	62
B. Formalizations and Code Developed	63

Abstract

This thesis deals with the *knowledge gap* that is commonly encountered with simulations: The simulation knowledge of the person wanting to solve a *partial differential equation (PDE)* is often times not sufficient to set up the solver program, in a way that the model is formally well-defined. These persons are experts in their own domain and not in simulation technology. The knowledge mismatch is usually resolved by intense collaboration with simulation experts.

This problem is also encountered with the PDE solver code **ExaStencils**. It already uses a domain-specific language, **ExaSlang**, attempting to allow for a mathematically natural interface. However, **ExaSlang** is currently lacking the capability to transform the user's *informal model* to a *formal programmatical representation*.

This work points out which elements such a user interface needs to employ: Methods from *mathematical knowledge management (MKM)* are tailored for the task. Conclusively, an architecture for the upper layer of **ExaSlang** is proposed, called the MOSIS architecture. It consists of the *formal and flexiformal knowledge* about PDEs as well as an *active document* to communicate said knowledge to the user. The MOSIS architecture is found to support the central requirements for the desired “translation”.

Based on the idea of a *technical interview*, the MOSIS architecture is made concrete with a MOSIS prototype that uses the MMT and sTeX systems for the formal and flexiformal knowledge representation, respectively.

Finally, THEINTERVIEW is developed as a proof-of-concept implementation of the MOSIS prototype. Chances and current limitations of the MOSIS prototype are evaluated.

1. Automated Assistants for Mathematical Modeling and Simulation

Imagine you want to go to Mars. Not just like that, of course, you probably have made quite some money to actually dare thinking about this – unless you are one of the lucky talented persons to become an astronaut.¹ Anyway, let us just say you have made a considerable fortune and are now in the position to dream about actually having your own rocket built for your very own Mars expedition.

You would probably need to hire a lot of experts for this kind of thing: engineers, physicists, mathematicians, ... There are quite a few disciplines that might be of interest to you, and alone for one single aspect, such as the heat shield, you would need a whole team of smart people. Very importantly though, you would always hire people that are good with large-scale simulations, because every prototype that catches on fire in the real world is literally burnt money.

Now as we have seen before, the facts that many people are involved in the development and that they have different backgrounds can sometimes cause problems. For instance, the varying vocabularies may cause the loss of important information in the course of “translation”. This happened with the Mars Climate Orbiter in 1998: Because of different systems of measure for the impulse – non-SI units of pound (force)-seconds versus SI units of Newton-seconds – the Orbiter was on the wrong trajectory and ultimately crashed into the Mars surface.

Differences in vocabulary can also happen between mathematicians, construction engineers and programmers: looking alone at the word “domain” rings a bunch of different meanings. It may denote the space a function or operator is mapping from (in mathematics), a physical space, such as a structural component made from steel (in construction), or a general discipline (in programming), cf. domain-specific programming language. Obviously, notions that are essential for one person may not be understood by another in professional discussions.

This becomes problematic in situations where there are prerequisites needed for applying a certain method. It may just be so obvious for one person that the finite volume method (FVM) can only be used for the conserved properties in a model, that they would never explicitly point it out to others. And of course, the other persons involved might then try to use the FVM for a non-conserved property and will fail at some point – hopefully soon and in an obvious manner.

1.1. Related Approaches

Currently, the problem of *conserving all the relevant information in collaborative dialogue* between people with different vocabulary is usually resolved by just hiring

¹In that case I would just like to say: Wow, this is amazing! I am really jealous of you.

more smart people to re-think and re-test the setup again and again – we will hear about the *one-brain-barrier* in section 2.5. If you were Elon Musk for example, you could just hope to hire some NASA researchers to support your team.²

In smaller teams with less complex problems, for instance structural engineers simulating the heating efficiency in houses, it is common to rely on out-of-the-box simulation suites. They may use Finite Element solvers such as ANSYS [ansys] to reliably do the job on the designer’s local machine, and use a coarse enough resolution so that it can be handled in reasonable time. More mathematically versed people, such as researchers in structural engineering, will use programs that are formulating this on a more abstract level: MATLAB [MAT] has the functionality `pdepe` [MATpe], and Mathematica [Wol17] offers `NDSolve` [WolND]. Both are functions for solving partial differential equations (PDEs) numerically. They share similar properties: As the programs are closed-source, we cannot say too much about the implementation, in particular the solution algorithm(s) used – apart from the fact that the user cannot choose it themselves. And again, the problems to be solved should be small enough that the computation can be carried out on the user’s local machine.

For simulating something more complex and/or in less time, there are now highly parallel solvers under active development. As an example, the research code **ExaStencils** [Kro+17] for the Finite Difference Method and other stencil-based algorithms can be set up to run on almost any architecture, such as a high performance cluster – we will learn more about **ExaStencils** in section 2.3. Currently it takes some expertise in PDEs to correctly set up the program, either requiring specialist knowledge from people that are experts in something else, or even making it necessary to consult somebody with that knowledge.

You see we had this problem before with our rocket, right?

But looking closely, there are always similar questions that come up in this situation: Did the user enter everything *correctly*? Have they *fully specified* the problem they want to solve? Can there even *exist a sensible solution* to this problem? Can it be obtained with the *chosen method*? What do they need to do to get their results?

From this perspective, there should be some room for automation in this ever-same modeling process.

²If you actually are Elon Musk: I admire your work, and if I can give you a bit of advice, please do try to focus more on electric cars than space.

1.2. The Vision

Thinking on a long time scale, it should be possible to create a generic simulations interface that “knows” about the mathematical backgrounds to PDE problems. By checking the inputs it should be able to give information about the solvability of the problem, just like a human expert would. And it could offer different suitable solution algorithms, both locally and remotely executable. We can even think of an interface that allows the user to understand the modeling process better as they use it, because the components are being explained by the program.

Up until now, there has been little research in this topic, most likely because there was little exchange between the logicians interested in the formalizations of knowledge and their potential “customers”, namely numerical programmers that are tired of talking to people on how to use their algorithms. This led to the fact that only small portions of the – comparably complex – mathematical PDE knowledge were given a formal representation. This may slowly be changing now with the models library [Koh+], where concrete physical problems are beginning to be formalized with MMT, a framework that will be introduced in section 2.5.4.

This thesis however aims at something different from the models library: it wants to automate the process of thinking about PDEs itself.

To illustrate this, we will take a look at the following concrete example.

1.3. A Running Example: Solving the Heat Equation

We have been talking about simulating the heat shield before, but actually we want to concern ourselves with a similar, yet more common, example for now.

Let us say that we know some engineer who has built her house in accordance to every possible architectural optimization. She has been striving to minimize energy losses due to heating – going far beyond the so-called “Energieeinsparverordnung” (German Energy Saving Ordinance). Still, she thinks that there may be room for improvement, which is why she counsels her friend, who is an expert in numerical simulations, to find out what would happen to the temperature distribution in her walls if she applied another layer of insulation at a specific spot on the outside.

As a first approximation, they may set up a simple variant of her problem: the 1-D simulation to describe the temperature distribution throughout a solid wall. We will later talk about the full heat equation in section 2.1.1. The boundary conditions simplify to two values, we can think of them as the warm temperature on the inside of the building complemented by the colder temperature on the outside. Thermal conductivity may vary among layers: for instance, the outermost insulating layer of a building will be of a less conductive material. Additionally, there can be heating pipes midway through our wall which act as sources of heat. The physical domain – time and space – is often times denoted Ω .

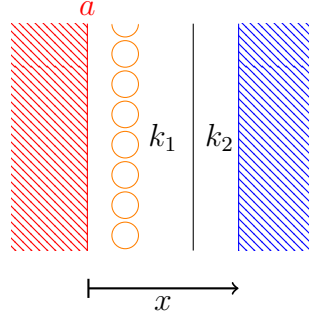


Figure 1: Schematic of the one-dimensional heat conduction problem

Simplifying further, they would consider the stationary solution, where there is no change over time any more. This would correspond to a night with constant (cold) temperature on the outside and our house owner constantly heating on the inside. The equilibrium is reached when the distribution of heat in the wall and therefore the flow of heat through the wall become constant. Then, the actually parabolic heat equation will be elliptic, see section 2.1. The relation is now given by the equation

$$\nabla \cdot (k \nabla T) = \dot{q}_V \quad \text{in } \Omega \quad (1)$$

with

k		the thermal conductivity
\dot{q}_V		the volumetric heat flux / “heat sources” in the material

This is what the simulations programmer would ask their house-owning friend to note down. They would discuss about reasonable values for every parameter, such as the boundary conditions

$$T(a) = T_a, T(b) = T_b \text{ on } \partial\Omega. \quad (2)$$

Only when everything is defined, and the meaning is perfectly clear and sensible to both of them, would the computational scientist give their ok and start the simulation.

As you might have guessed already, the aim of this thesis is to get closer to providing this “automated friend” that helps our engineer solve her problems – only the ones concerned with PDEs of course.

Because if you imagine a world where there is a program helping people to cross the simulations expertise valley, they would suddenly have access to many tools

to advance their science and research. People like our engineer essentially would have a lower barrier to a “playground” – the simulation – which allows them to understand their world a tiny bit better.

And not only that – there would even be a record of the things that happened along the discussion in the form of different types of files, containing the negotiations, the checking, the result. This will make it easier for our engineer to come back after some years and revisit the considerations, and exactly comprehend what happened then.

1.4. Scope and Aims of This Thesis

The example and motives that we have described so far open up a huge scope of approaches that could be pursued to solve the problem in question. Only some of them can be addressed in this thesis, which is why some explicit aims are now introduced as follows:

- O1** to *explore the structure* of reasoning about PDEs
- O2** to provide a *simple implementation* of an interface that allows users to enter their PDE problem in a natural way
- O3** to *use the output* of that interface for *numerical simulation*.

Throughout this thesis, we will stick with the simple heating example to give us a starting point.

First of all, we need to discuss a variety of different aspects connected to our problem – and its possible solutions.

2. Preliminaries

2.1. Mathematical Description of Partial Differential Equations

Definition 2.1 A **partial differential equation** or **PDE** is an equation that contains multivariate functions f_i as well as their partial derivatives $\frac{\partial f_i}{\partial x_j}$. Its **order** is determined by the highest order of partial derivatives occurring in it.

Many processes in nature can be described with PDEs: electromagnetism, fluid mechanics, quantum dynamics. . . . For a thorough introduction to PDE definitions and analysis please refer to *Numerik partieller Differentialgleichungen* [KA00].

As an abbreviation to the partial derivative operator $\frac{\partial}{\partial x}$, often times the short form ∂_x is used.

2.1.1. Second Order Linear Partial Differential Operators

A linear second order PDE applied to some unknown function u can generally be expressed as

$$(\partial_\alpha M_{\alpha\beta} \partial_\beta + \partial_\alpha k_\alpha + r) u = f \quad (3)$$

For determining the type of operator, we are only interested in the first part, which are the second-order partial derivatives, here expressed in matrix notation:

$$\begin{bmatrix} \partial_{x_1} \\ \partial_{x_2} \\ \vdots \\ \partial_{x_d} \end{bmatrix}^\top \overbrace{\begin{pmatrix} A & B & \cdots & X \\ B & C & \cdots & Y \\ \vdots & \vdots & \ddots & \vdots \\ X & Y & \cdots & Z \end{pmatrix}}^{\mathbf{M}} \begin{bmatrix} \partial_{x_1} \\ \partial_{x_2} \\ \vdots \\ \partial_{x_d} \end{bmatrix} \quad (4)$$

The entries A, B, \dots, Z may be variable over the domain. We used the fact that u is assumed to be continuously differentiable to make \mathbf{M} symmetric, in case that it was not symmetric before: $\partial_{\alpha\beta} = \partial_{\beta\alpha}$.

The Heat Equation

The heat equation is a second-order PDE that describes the distribution of heat T in an arbitrary spatial domain Ω over time [Can84].

$$\text{(heat equation)} \quad \left\{ \begin{array}{ll} \rho \, c_p \, \frac{\partial T}{\partial t} - (\nabla \cdot k \, \nabla T) = \dot{q}_V & \text{in } \Omega \\ T = T_0 & \text{in } \Omega \text{ at } t = 0 \\ T = T' & \text{on } \partial\Omega \end{array} \right. \quad (5)$$

Here, the following parameters were used to describe the problem:

ρ	the mass density of the material
c_p	the specific heat capacity
k	the thermal conductivity
\dot{q}_V	the volumetric heat flux / “heat sources” in the material
T'	the temperature profile at the boundary
T_0	the initial temperature distribution

Expanding the nabla/del operators $\nabla = [\partial_{x_1}, \partial_{x_2}, \dots]^\top$ for the gradient ∇ and the divergence $\nabla \cdot$ term, we get the matrix representation of the heat equation. In two space dimensions, it looks like:

$$\rho \, c_p \begin{bmatrix} \partial_t \\ \partial_{x_1} \\ \partial_{x_2} \end{bmatrix}^\top \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} T(x) - \begin{bmatrix} \partial_t \\ \partial_{x_1} \\ \partial_{x_2} \end{bmatrix}^\top \begin{pmatrix} 0 & 0 & 0 \\ 0 & k_{x_1 x_1}(x) & k_{x_1 x_2}(x) \\ 0 & k_{x_2 x_1}(x) & k_{x_2 x_2}(x) \end{pmatrix} \begin{bmatrix} \partial_t \\ \partial_{x_1} \\ \partial_{x_2} \end{bmatrix} T(x) = \dot{q}_V(x) \quad (6)$$

This assumes that $k(x)$ is a tensor that is constant over time.

Types of PDEs

Let us have a look at the nomenclature for different types of PDEs. If the eigenvalues of the matrix all have the same sign on every part of the domain, we call the operator *elliptic*. If additionally there are zero-eigenvalues, we call the operator *parabolic*. And in case the eigenvalues are both positive and negative, the operator is *hyperbolic* [KA00].

So in order to classify the operator provided by the user, we need to know the entries in \mathbf{M} and determine its eigenvalues.

We immediately see that the *Poisson equation*

$$\text{(Poisson Equation)} \quad \left\{ \begin{array}{ll} -\Delta u = f & \text{in } \Omega \\ u(x) = 0 & \text{on } \partial\Omega, \end{array} \right. \quad (7)$$

is elliptic, as the Laplace operator Δ is defined as

$$\partial_{x_1x_1} + \partial_{x_2x_2} + \cdots + \partial_{x_dx_d} = \begin{bmatrix} \partial_{x_1} \\ \partial_{x_2} \\ \vdots \\ \partial_{x_d} \end{bmatrix}^\top \overbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}}^{\mathbf{M}} \begin{bmatrix} \partial_{x_1} \\ \partial_{x_2} \\ \vdots \\ \partial_{x_d} \end{bmatrix} \quad (8)$$

in the matrix notation introduced before: Δ has only eigenvalues of 1 [KA00].

Coming back to our example, the heat equation described above clearly is parabolic, as the eigenvalue associated with time t equals zero. If we then look at the stationary solution to the heat equation, we are only interested in these situations where there is no change over time. As a consequence, all the observations connected to time drop out of the equation, so that the PDE reduces to space dimensions only, and becomes elliptic with positive eigenvalues determined by k – if k is symmetric [Can84].

2.1.2. Boundary Conditions for (Unique) Solvability

For our vision to come true, the well-posedness of the boundary value problem written down by some user needs to be asserted. To this end, it is central to understand what we are actually looking for. For example, it does not suffice to check for every point on the boundary if there is a boundary condition specified. Depending on the kind of problem, this may be too little or already too much information.

Example

For illustration purposes, we are using a very basic example:

If we consider the *Laplace equation*, which is equivalent to the Poisson equation (cf. eq. (7)) with $f(x) = 0$ [KA00]

$$(\text{Laplace equation}) \quad \left\{ \begin{array}{l} \Delta u(x) = 0 \quad \text{in } \Omega \end{array} \right. \quad (9)$$

in just one dimension, it simplifies to an ordinary differential equation

$$\frac{\partial^2 u(x)}{\partial x^2} = 0 \quad \text{in } (0, 1). \quad (10)$$

This can obviously be solved by simply integrating over x twice:

$$u(x) = c_1x + c_2, \quad (11)$$

where $c_1, c_2 \in \mathbb{R}$ are still undetermined. This means that the boundary conditions will have to contain at least two units of information in order to fully determine

u . The fact holds true for other choices of f in the Poisson equation, as all the other variables occurring will be determined by the choice of f .

The different kinds of boundary conditions to supply this information can generally be described as follows. Note however, that existence and uniqueness of solutions need to be proven individually for each type of problem.

Definitions

Definition 2.2 Boundary conditions are equations that need to hold for an unknown function on (parts of) the boundary of its domain. They are used together with partial differential equations to form a **boundary value problem**.

Definition 2.3 The following types of boundary conditions are often times employed for an unknown function u defined on Ω , where $\Gamma \subseteq \partial\Omega$ is the part of the boundary where the condition shall hold.

- **Dirichlet boundary conditions** set u to a certain value f on the boundary:

$$u(x) = f(x), x \in \Gamma$$

- **Neumann boundary conditions** fix the derivative of u with respect to the normal n of $\partial\Omega$ to a certain value g :

$$\frac{\partial u(x)}{\partial n} = \nabla u(x) \cdot n(x) = g(x), x \in \Gamma$$

- **Robin boundary conditions** require that the sum of u and its normal derivative, weighted by c_1 and c_2 , be equal to a certain value h .

$$c_1 u + c_2 \frac{\partial u(x)}{\partial n} = c_1 u + c_2 (\nabla u(x) \cdot n(x)) = h(x), x \in \Gamma$$

- **Cauchy boundary conditions** require both Dirichlet and Neumann boundary conditions to hold on the same part of the boundary.
- **Periodic boundary conditions** require that u and its derivatives have the same value on one side of the domain as the opposite side. They are usually used on one-dimensional, rectangular and cuboid domains only.

Visualization

We would like to illustrate the peculiarities of the different kinds of boundary conditions (BCs) using the Laplace equation example eq. (10).

- For Dirichlet conditions everywhere, existence and uniqueness of a (weak) solution can be proven for elliptic PDEs by way of the *Lax-Milgram* lemma [KA00]. The only additional constraint for applying it is the L_2 - or square-integrability of the right hand side function f .

For our 1-dimensional example this might look like figure 2. The Dirichlet conditions are indicated as dots of a given value.

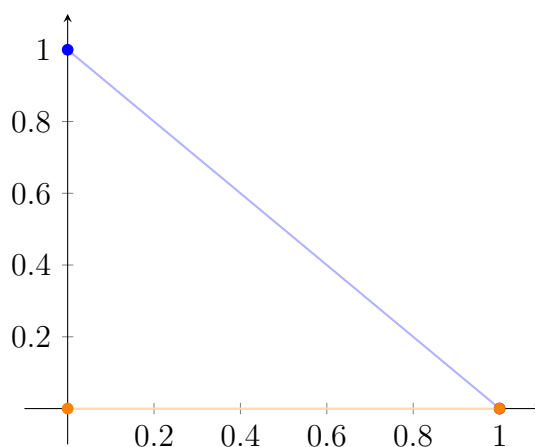


Figure 2: 1-dimensional solution with Dirichlet boundary conditions

- For Neumann boundary conditions everywhere, we cannot obtain an unique solution. What is worse, the conditions may even be contradictory, as we can see in figure 3. Neumann boundary conditions are shown in the graphs as a certain slope on the boundary: The blue ones in figure 3 are consistent but allow for more than one solution. The orange one cannot be fulfilled, because our solution must be linear and therefore a straight line in this graph.

The requirement for consistency with the describing PDE can be obtained by the divergence theorem [KA00]

$$\int_{\Omega} \Delta u \, dV = \oint_{\partial\Omega} \nabla u \cdot n \, dS, \quad (12)$$

meaning that it can only be consistent, if the integral over the Neumann terms is equal to the integral over f . In our simple example, this means that

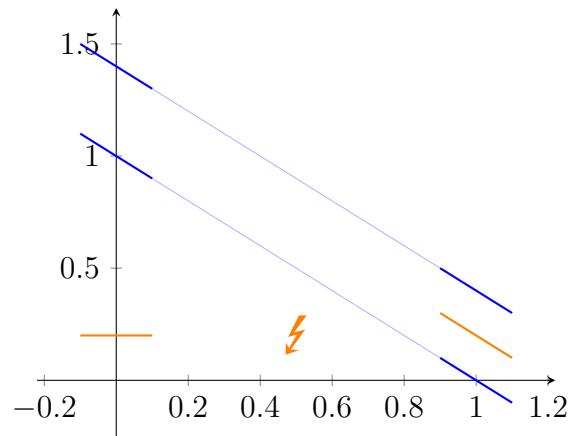


Figure 3: 1-dimensional solution with Neumann boundary conditions

the sum of the two values given for the derivative must equal zero. But in order to make the solution unique, a Dirichlet condition needs to additionally give the “height” of the graph.

- If we have mixed Dirichlet and Neumann boundary conditions, we can always find a unique solution for our simple problem, cf. figure 4. Things can get more complicated when we are going to higher dimensions.

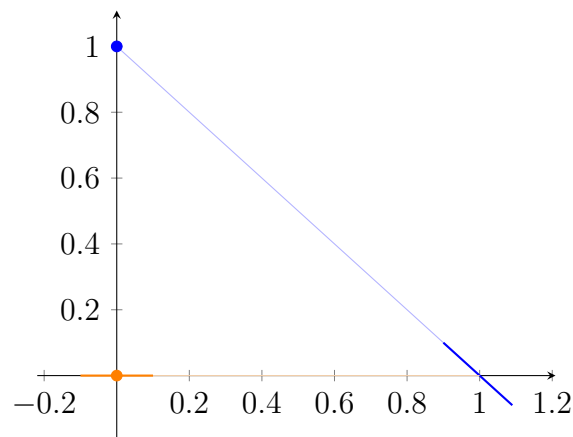


Figure 4: 1-dimensional solution with mixed boundary conditions

- In the case of periodic boundary conditions, we can use the same argument as for Neumann boundary conditions, to find that the sum of the two values given for the normal derivative must equal zero. In fact, we can only find

solutions to the periodic problem, if

$$\int_{\Omega} f(x) dV = 0 \quad (13)$$

holds true. In our linear example, since the boundary points also must have the same value, we implicitly need to have zero derivatives at the boundary. But again there is an undetermined parameter to the solution, as we see in figure 5.

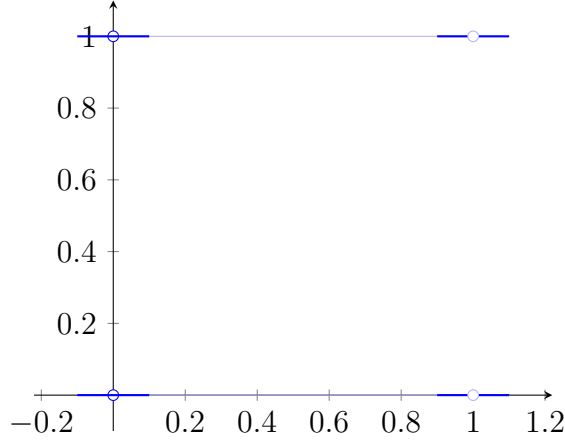


Figure 5: 1-dimensional solution with periodic boundary conditions

Generalization to higher dimensions

As we have seen, a problem's boundary conditions directly influence the solvability of its describing PDEs, even for the most basic cases.

But we can in fact state a bare minimum of information that we need in order for the boundary value problem to be uniquely solvable. Staying with cuboid shaped domains, every single dimension needs to specify boundary conditions defined on a certain measure of the domain boundary

$$A_{required,x} = A_{\perp x} \cdot O_x, \quad (14)$$

where x is one cuboid space dimension, $A_{\perp x}$ denotes the measure of a cut of the domain normal to x (the measure of the “box side” seen from the x direction), and O_x is the order of the PDE with respect to x .

In our simple 1-dimensional example, the cut of the domain perpendicular to the x -Axis gives us a set of measure zero – $A_{\perp x}$ is a point. Since the derivative with respect to x is of order two, we need to specify boundary conditions for two

points. And we have seen in the examples that the boundary sections may coincide (or in higher dimensions, overlap), as long as the conditions are independent.

This leads us to a second requirement: There generally needs to be an “anchoring” on some defined value in order to avoid ambiguities. Therefore, if there are only Neumann or periodic conditions given, we must make sure that a Dirichlet boundary condition is added on at least one point of the boundary. As an example, in the case of the non-stationary parabolic heat equation the anchoring may also be the initial temperature distribution for $t = 0$; then, it suffices to use consistent Neumann boundary conditions for the x coordinate(s).

2.2. Solving PDEs Numerically: The Finite Difference Method

The *finite difference method* or FDM, sometimes also *finite differences* is the earliest approach to solving PDEs. The reason is that the idea is derived from simple difference quotients, as will be shown in the following.

In the FDM, the differential operators are approximated as a system of equations that needs to hold on the points x_i of a regular grid on the domain

$$\bar{x}_{i+h} = \bar{x}_i + h \quad (15)$$

where we call h the grid spacing. In the following, the i indices will be omitted without loss of generality.

The forward difference quotient operator D_+ is well-known to the mathematically versed reader, as its limit for $h \rightarrow 0$ is how we usually define the derivative of any differentiable function u :

$$D_+ u(\bar{x}) \equiv \frac{1}{h} [u(\bar{x} + h) - u(\bar{x})] \quad (16)$$

The backward

$$D_- u(\bar{x}) \equiv \frac{1}{h} [u(\bar{x}) - u(\bar{x} - h)] \quad (17)$$

and central

$$D_0 u(\bar{x}) \equiv \frac{1}{2h} [u(\bar{x} + h) - u(\bar{x} - h)] \quad (18)$$

difference quotients can be written down in an analogous manner. We can easily agree that all of the limits $\lim_{h \rightarrow 0} D$ are the same as the derivative. However, the quality of the approximation is better for the central difference quotient, where the truncation error is of order $O(h^2)$, such that we call the approximation second-order accurate [LeV07, p.5]. Conversely, the discretization error for the forward and backward difference quotients scale in $O(h)$, and we call them first-order accurate [LeV07, p.5].

Now in order to get the standard centered approximation to the second derivative, we just take the difference quotient of difference quotients:

$$D^2 u(\bar{x}) \equiv \frac{1}{h^2} [u(\bar{x} - h) - 2u(\bar{x}) + u(\bar{x} + h)] = D_+ D_- u(\bar{x}) \quad (19)$$

This can also be viewed as a double centered difference quotient on a grid with half the spacing $\frac{h}{2}$

$$\hat{D}_0 u(\bar{x}) = \frac{1}{h} [u(\bar{x} + \frac{h}{2}) - u(\bar{x} - \frac{h}{2})], \quad (20)$$

because

$$\hat{D}_0(\hat{D}_0 u(\bar{x})) = \frac{1}{h} \left[\left(\frac{u(\bar{x} + h) - u(\bar{x})}{h} \right) - \left(\frac{u(\bar{x}) - u(\bar{x} - h)}{h} \right) \right] = D^2 u(\bar{x}) \quad (21)$$

is identical to the standard second order centered approximation D^2 , cf. eq. (19).

And in fact, the error analysis of D^2 by way of the Taylor expansion

$$D^2 u(\bar{x}) = u''(\bar{x}) + \frac{1}{12} h^2 u''''(\bar{x}) + O(h^4) \quad (22)$$

shows that the standard approximation is second-order accurate. We will from here on be concentrating on the standard second order finite difference operator only.

Stencils

Stencils are a way to express a local algorithmic kernel operation. For example, D^2 can be represented as a stencil

$$\frac{1}{h^2} \cdot \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \quad (23)$$

that can be applied to u on an arbitrary grid point \bar{x} to get the approximation to the second derivative at that point.

Now to make for example a Poisson equation (cf. eq. (7)) “come true”, that is to solve it using finite differences, we first initialize a regular grid u , say with value 0 everywhere. We can then access different non-boundary grid points \bar{x} to update the value $u(\bar{x})$ in a way to fulfill the discrete equation

$$\frac{1}{h^2} [u(\bar{x} - h) - 2u(\bar{x}) + u(\bar{x} + h)] = f(\bar{x}), \quad (24)$$

which results in

$$u(\bar{x}) = \frac{1}{2} [-h^2 f(\bar{x}) + u(\bar{x} - h) + u(\bar{x} + h)]. \quad (25)$$

This process provides us with an iterative solver to the Poisson equation. Depending on the “pattern” with which we update u , the algorithmic properties can vary. A commonly known algorithm that can be applied to stencils is the Jacobi method, cf. [KA00], which results in a row-wise access pattern on the u field.

In order to get the discretization of the Laplacian, we see from its definition in eq. (9) that it is the sum of the second derivatives in each direction. Accordingly, the standard second order centered approximation of the Laplace operator in two dimensions can be shown as a two-dimensional stencil

$$\frac{1}{h^2} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \quad (26)$$

2.3. The ExaStencils Framework for Stencil Codes

The ExaStencils project is centered around the aim of writing a code that generates highly optimized stencil codes with the Scala programming language [Kro+17]. The PDE to be solved can be defined in a generic way, while the code that is actually doing the computation is produced automatically and will be heavily parallelized and platform-specific.

As we have seen in the last section 2.2, finite difference schemes can be expressed as stencils, so ExaStencils is a suitable software. In addition to the finite difference method and other stencil-based techniques, there was an algorithm implemented that allows for solving the problem, e.g. the PDE on different scales, which is called the multigrid method [Kro+17]. The finer workings of multigrid are not needed for understanding this thesis and are beyond scope; the interested reader may be referred to *A multigrid tutorial* [BHM00].

The ExaStencils project also developed a domain specific language (DSL), ExaSlang [KK16], for the description of the problem to be simulated, and for the solver to be used. It is designed in different layers of simulation detail as shown in figure 6.

Layer 1 : Continuous model
Layer 2 : Discretization
Layer 3: Solution algorithm
Layer 4: Application specification

Figure 6: The ExaStencils language stack

“Below” the stack, the code generation, compilation and execution happen in a fashion that is adapted to the problem type and system architecture it should run on.

From the layer 2 syntax, we get a glimpse at the ExaSlang DSL:

```
Domain global< [ 0 ] to [ 1 ] >

Field Solution with Real on Node of global = 0.0
Field Solution@finest on boundary = vf_boundaryCoord_x ** 2
Field Solution@(all but finest) on boundary = 0.0

Field RHS with Real on Node of global = 0.0
Operator Laplace_1D from Stencil {
  [-1] => -1.0 / ( vf_gridWidth_x ** 2 )
  [ 0] => 2.0 / ( vf_gridWidth_x ** 2 )
  [ 1] => -1.0 / ( vf_gridWidth_x ** 2 )
}

Equation solEq@finest {
  Laplace_1D * Solution == RHS
}
```

```
Equation solEq@(all but finest) {
  Laplace_1D * Solution == 0.0
}
```

Listing 1: The 1-dimensional Laplace equation eq. (10) in layer 2 syntax

This is a formal – in this case we may call it programmatic – representation of a PDE problem already, and in the lower layers also of a possible solution.

Layer 1, in contrast, should allow to note down the continuous model in a way that is natural for mathematicians. Currently however, the layer 1 information is not processed further, and the other layers of the language need to be filled in by hand. Clearly, the transition from the informal (or possibly flexinformal, cf. section 4.2) PDE model in the user’s mind to the formal one in **ExaSlang** currently requires profound expertise, and is even redundant in some respects: Information that is given in layer 1 could be used to infer most of layers 2 and 3.

First steps towards the automatic generation of layer 2 from layer 1 information were taken with the thesis *Automatische Diskretisierung elliptischer partieller Differentialgleichungen in ExaSlang* [Fla17], where the automatic discretization of the partial differential operators was analyzed and implemented.

There is an essential aspect which has not been investigated so far: **ExaSlang** has no mode of feedback for the user while entering their model into layer 1, as to whether something is unclear, not fully specified or even over-specified. Thus, the user has no means of correcting or formulating the model in a different way.

Also, the means of detecting when the input is unclear, under- or over-specified have not been looked at yet. This would, e.g., be interesting for the boundary condition specification, as we know from section 2.1.2. One obstacle to realizing it is for example that it is not straightforward to have the boundary conditions written down without requiring the user to use a rigid, heavily formalized representation.

So this point – the transition from the informal model in layer 1 to the formal one in layers 2 and 3 – is exactly where our solution approaches must be focused on.

2.4. Theory Graphs

In the course of formalization of mathematical content, it became apparent that there are some patterns occurring regularly when working with mathematical theory.

For instance, we will generally start with basic definitions and a set of assumptions or axioms that we are relying on. We will build more definitions based on that and conclude some more facts. In due course we will apply some renamings in order to capture the current situation better.

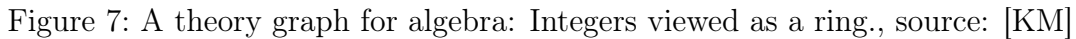
Finally, we will apply abstract analogies to the objects that we are working with: The A that we have been thinking about so far, can also be considered a B , because it has all the same properties – of course then we need to prove that A in some way covers all the assumptions that we had for B . So we can apply something we know about B to A .

These actions that are typically led by intuition can actually be represented in a very structured way: by using theory graphs. A *theory graph* [Koh14] relies on a paradigm of small theories, where basically everything for which we can think of a new name is considered a new theory. Then, each theory will be a node on the graph, and the different kinds of relations – then called *theory morphisms* – can be drawn as arrows on the graph.

This structure can for example be seen in a theory graph for fundamental algebra, cf. figure 7. In the left column, we see the theory of natural numbers evolving, based on the Peano axioms $P1, \dots, P5$. Going upwards, it is continuously extended with the definition of the plus and times operators. Ultimately, the integers are created by extension to negative numbers. On the right-hand side, there is a “background theory” growing on a higher level of abstraction. It starts off with the definition of magmas (as a set and an operator), which with associativity become semigroups, and adding a neutral element makes them monoids.

Until here, we have only extended theories with additional properties, which corresponds to the theory morphism of *inclusions*. Now we can also start to transfer these definitions by way of *views*: The natural numbers, both with the plus and times operators correspond to a monoid, which can be proven by mapping all the symbols in the monoid theory in a suitable manner. This is shown as curled arrows with the ψ and ϕ operations. Additionally, the monoid itself can be considered a monoid by symmetry, which is denoted by the circular view arrow. Now we can show that the naturals with plus and times are not a group yet: both operations are not invertible for all naturals. The dotted arrow shows a *contradiction*: anything that is not a group can never be a group - because the definitions apply to distinct sets.

If now, however, the numbers are extended to the integers, they can be consid-



So we learned that a view is a morphism that can transport meaning between theories, sometimes called a *truth-preserving relation* [Koh14]. Another theory morphism that is not shown in the graph is the *structure* (in theory graphs usually denoted by a normal straight arrow). The structure signifies that all of the contents present in the initial theory are copied to the target theory. Renaming and/or assignment of constants is possible, and is shown in the graph just like the assignments for views, next to the arrow.

After this short recap of elementary algebra and how to structure the thoughts behind it, we will now discuss how these theory graphs also illustrate the generation of new knowledge: the so-called *pushout* theory morphism. A pushout always arises when we have defined something in the more abstract theory which we then can apply to the actual situation theory by way of a view.

20

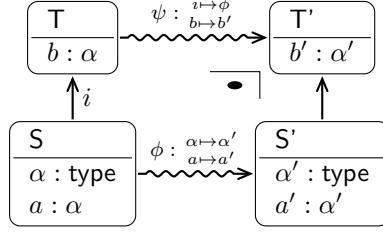


Figure 8: A very simple pushout setup., source: [KM]

hold for S' , so we can directly conclude T' by naming the new objects that arise in the mapping from T , which is denoted by Ψ . Whenever this ladder-like structure occurs, we call the “concludable” part of it a pushout. It is sometimes denoted by a right-angle-with-dot symbol, like in figure 8. Pushouts collect the knowledge that was induced by our theory through views; when all of this accumulated knowledge is written down, we call this process “to flatten” the graph [Koh14].

2.5. Systems for Mathematical Knowledge Management

The research area of mathematical knowledge management (MKM) is concerned with the problems that arise from developments starting in the last century: While we have rapidly growing mathematical knowledge across different disciplines, the abilities of one single person in understanding and knowing about maths remain limited. This phenomenon is called the “one brain barrier”. Or, to describe the problem more technically, there are many researchers – luckily! –, but the information exchange between them just cannot scale well and happens practically only with their direct research partners. It seems obvious that, if all researchers have similar goals, the work done will often times be replicated by others, with varying quality.

This is where MKM comes into play: It studies the possibilities to overcome the one brain barrier by using automated tools. And since researchers, just like everybody else who has an interest in math, use the Internet, one often studied approach is the Semantic Web.

2.5.1. Semantic Web

Around the turn of the Millennium, we have seen the leap to the social web – or Web 2.0 –, a massive change in the way that the Internet was used and perceived. The standard usage has tremendously shifted from consuming to creating and interacting with content.

The next big leap that was predicted to follow was the Semantic Web (or Web 3.0), an infrastructure to provide machine-understandable content alongside the human-understandable content, and make it usable via specialized web services. This would enable the user, among other things, to have these services explain the content of websites to them.

However, the Semantic Web is still waiting for extensive adaptation with the fields of Science, Technology, Engineering, Math (STEM). Catalin et al. [Cat+10] attribute this to the following factors:

1. Both the STEM and Semantic Web user communities are small compared to the social web as a whole.
2. There is significant inherent difficulty and complexity in STEM subject matter.

Because of the expected benefits, it is natural that there is ongoing research in potential systems for the Semantic Web and other fields of MKM.

2.5.2. OMDoc: Open Mathematical Documents

The systems we will be introducing here are based on the OMDoc/MMT ontology – we will get to the MMT part in more detail later on in section 2.5.4. The OMDoc or “Open Mathematical Documents” format [OMDoc] is an XML-based representation for mathematical content – both formal (formulas) and informal (documents). It adds semantic structures to the OpenMath [Bus+04] and/or MathML (“Mathematical Markup Language”) [Aus+10] standards in order to give meaningful context, such as ontologies, to the expressions presented.

A simple example for a snippet of mathematical knowledge formalized in OMDoc is given in listing 2.

```
<theory xml:id="semigroup">
  <symbol name="base-set"/>
  <presentation for="#base-set"><use format="default">M </use></presentation>
  <symbol name="op"/>
  <presentation for="#op"><use format="default">◦</use></presentation>
  <axiom xml:id="closed.ax"><FMP>∀x, y ∈ M .x ◦ y ∈ M </FMP></axiom>
  <axiom xml:id="assoc.ax">
    <FMP>∀x, y, z ∈ M .(x ◦ y) ◦ z = x ◦ (y ◦ z)</FMP>
  </axiom>
</theory>
```

Listing 2: Formalization of semigroup in OMDoc [[URL:omdocspec](#)]

Simply enough, at first the base set is defined and given the symbol M . \circ is determined to be an operator. The closure of the set M under the operator is stated as “closed.ax”, while the most important part, the associativity, is given last, as “assoc.ax”. So, what we see here is just the definition of a semigroup in algebra, which the reader might remember from figure 7.

We immediately observe that this representation, while being straightforward to understand if walked through on a line-by-line basis, is not suited for direct interaction with humans. Even the simplest of statements become long and cluttered with XML, which makes them tedious to read and even more so to write. This is why the OMDoc format is intended to be used by machines, and more human-understandable formats should be used for the creation and retrieval of the information. This is where the $\mathcal{S}\text{TeX}$ and MMT systems come into play.

2.5.3. $\mathcal{S}\text{TeX}$: Semantic Extension for $\mathcal{L}\text{TeX}$

The $\mathcal{S}\text{TeX}$ system [Koh08; CTAN] is aimed at creating *flexiformal content*, that is, both formulas and natural language can be used to describe (mathematical) knowledge [Koh13]. It employs $\mathcal{L}\text{TeX}$ XML [Mil] to transform the contents of semantically marked-up $\mathcal{L}\text{TeX}$ documents to the OMDoc format. $\mathcal{S}\text{TeX}$ is especially useful in building knowledge bases such as lecture notes, as by cross-referencing content in other modules, definitions become relatively easy to look up through

hyperlinks. As an example, the MathHub mathematical glossary [SMG] was built using $\mathsf{\S T E X}$.

For instance, the definition of PDEs in section 2.1 was generated using the following two modules.

```
\begin{modsig}[creators=tp]{partial-differential-equation}
\gimport[smglom/calculus]{partial-derivative}

\syml{partial}{differential}{equation}
\syml{PDE}
\end{modsig}
```

Listing 3: A signature module in $\mathsf{\S T E X}$

The signature gives the module its name, imports other signatures, and states which terms will be declared – and explained – in the module. This structure will in most cases expose an interface to use in all natural languages.

This is different for the natural language modules, which need to specify the language they are written in. Here, we see the English description:

```
\begin{mhmodnl}[creators=tp]{partial-differential-equation}{en}
\begin{definition}
  A \def{partial}{differential}{equation} or \def{PDE} is an equation that contains
  multivariate functions  $\mathsf{\livar{f}{i}}$  as well as their  $\mathsf{\treffis{partial}{derivative}}$ 
 $\mathsf{\pderivative{\livar{f}{i}}{\livar{x}{j}}}$ .
  Its  $\mathsf{\def{order}}$  is determined by the highest order of  $\mathsf{\treffis{partial}{derivative}}$ 
  occurring in it.
\end{definition}
\end{mhmodnl}
```

Listing 4: A natural language description module in $\mathsf{\S T E X}$

Now, we see roughly the same structure as in a $\mathsf{\LaTeX}$ document, including the commonly used **definition** environment. What is different is that we mark the terms to be defined by $\mathsf{\def}$ statements, where the number of trailing “i”s tells how many words the term consists of. Analogously, we reference the terms defined in other modules by $\mathsf{\tref}$ statements. And we can use the notations from imported modules, such as the $\mathsf{\pderivative{ . }{ . }}$. This is a central part of the semantic markup aspect in $\mathsf{\S T E X}$: instead of writing $\mathsf{\frac{\partial . }{\partial .}}$ – which would have given the same typographic result – we are explicitly stating that we mean the partial derivative. In the example, this requires even less syntactical effort.

Also, the MathHub mathematical glossary (SMGloM) [SMG] was built using $\mathsf{\S T E X}$.

There is of course no check whether we used the notations correctly and if all our definitions make sense. This is why with $\mathsf{\S T E X}$, we are still in the domain of informal and flexiformal knowledge. For formal knowledge and the reasoning based on it, we can use the MMT system.

2.5.4. MMT: Meta-Meta Theories

MMT [MMT; RK13] is a foundation-independent system for the processing of logics. It is complementary to \LaTeX in that it deals with *formal contents*.

MMT is suited for formalization of (mathematical) content and reasoning, for example by way of a `jEdit` plug-in [JE]. Results can be stored and reused using the OMDOC markup format. Having formalized axioms and definitions in the MMT surface language, the user can employ MMT to check the validity of many statements by type checking and inference, and to apply knowledge to new situations by means of views – we already know about them from section 2.4.

Of course, even as the system is foundation-independent, there are logical foundations required for every actual formalization. Various of these foundations or “meta theories” can be employed in MMT. In figure 9, there is an example of the formalization of an interval. The meta theory used in the example is called `base:?Logic`, which in turn employs the Edinburgh Logical Framework (LF) [HHP93] as a meta theory. So to be concise, MMT is the framework, while an extended version of LF is the actual “language” used in the example.

MMT document contents

The following description aims at giving a quick overview of syntax and semantics in the MMT surface language. The interested reader may have a look at the MMT tutorial for mathematicians [KM] for further reference.

In figure 9, we see the `jEdit` MMT environment: to the right, there is the normal editor view where files can be written, and the left column (the “sidekick pane”) shows the structure that MMT has found out from parsing and type-checking the file. We are looking at a simplified version of the PDE domain theory that will be explained in section 3.4, and from it we can learn many concepts available in MMT.

In the preamble, we state the current `namespace` and `import` other namespaces in order to use abbreviations for them. The OMDOC/MMT namespaces are given as unique resource identifiers or URIs as described by Dürst and Suignard [DS05] – not to be confused with URLs.

For discussing the main document content, let us first have a look at the general structure: Pieces of information are given in different levels of hierarchy, and each is ended by its own delimiter. On the highest level, the document is partitioned in *modules*: apart from the `namespace` and `import` statements, these can be the logical objects `theory` and `view`. The next lower level is the *declaration*, for example an `include` or the description of a constant; the latter may be divided further into *objects*, giving type, definition or notation for the constant.

The first and quite simple *theory* `GeneralDomains` shows all of these levels. Its name is given, followed by a colon and the meta theory `base:?Logic`, which

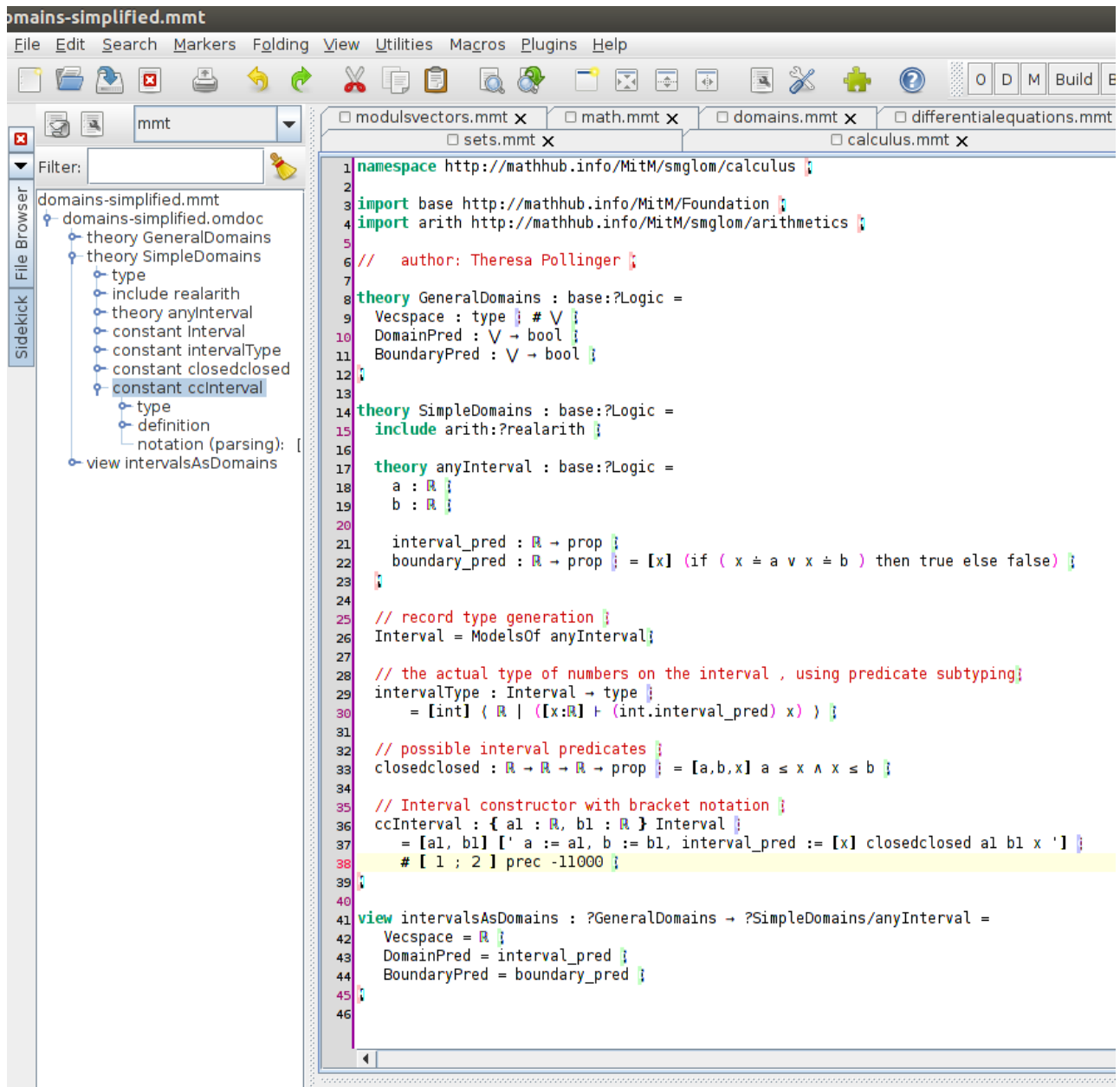


Figure 9: A screenshot of the jEdit editor with MMT plug-in loaded

is a MMT theory itself. The *declarations* follow after the equals sign: There is something called **Vecspace** of type **type** (indicated by the colon, the first *object*), with a notation \bigvee (indicated by the hashtag, the second *object*). **DomainPred** and **BoundaryPred** are both of type $\bigvee \rightarrow \text{bool}$ – some function that returns a bool for every given \bigvee . The idea here is that their return values will indicate whether a

point lies in the domain or at the boundary, respectively.

The next theory `SimpleDomains` is a bit more tricky. It includes the theory `realarith` from the namespace `arith` a.k.a. <http://mathhub.info/MitM/smgglom/arithmetic>. Theories may also be nested, as seen in line 17, where `anyInterval` is declared inside `SimpleDomains`. This is particularly useful for the generation of *record types*. Record types are data types that are generated with all the undefined constants as fields (called a struct or structure in other programming languages). The mechanism is invoked in line 25 with the `modelsOf` statement. So from this point on, we have access to a record type `Interval` with fields `a`, `b`, `interval_pred` and `boundary_pred`.

Moreover, we state the type of all numbers that lie on an interval in lines 29 f. The definition (indicated by the equals sign) tells us that given the interval `int`, the type consists of all $x \in \mathbb{R}$ for which the `int`'s `interval_pred` holds true. The brackets `[x]` are MMT's syntax for a lambda expression applied to some x (cf. lambda calculus).

Now there may be different definitions of whether the boundary points a and b are part of the boundary. To get a closed interval, the predicate `closedclosed` can be used. Its type is $\mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool}$, meaning that we can see it as a function that takes three reals as arguments. Or alternatively we can take, e.g., `closedclosed a b` as a function of type $\mathbb{R} \rightarrow \text{bool}$ – a relation that is called Currying.

And this is just what we are doing in the `ccInterval` “constructor”: we are generating an `Interval` by “filling in” the two defining points `a1` and `b1` as well as the desired predicate using the same points. Also, we add the usual bracket notation for a closed interval.

Finally, we can get these two theories together in the `view` in lines 41ff. The syntax is, not surprisingly, similar to what we have seen in figure 7 in section 2.4: all the constants that were undefined in the base theory `GeneralDomains` are mapped to their counterparts in the target theory `SimpleDomains/anyInterval`.

Now, when we state more information that applies to `GeneralDomains`, we can use it for intervals in particular, cf. section 2.4.

2.6. Active Documents

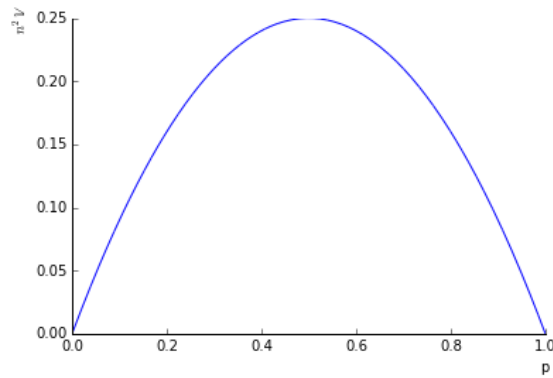
For the interaction with users interested in the STEM disciplines, the last years have shown some increased use of notebook-based programs that allow for the execution of codes, the analysis of formulae and, of course, the visualization of the results. As especially the more recent open source variants Jupyter [Jup] and SageMath [Sage] have made such functionality freely available to teachers and students in the STEM disciplines, teaching and prototyping in this format has now become a standard tool.

The advantages are obvious: Within one user interface, things can be changed and the changes can be visualized immediately. There are no effects to be expected which are not caused by something in the very same document. This makes it easy to set up a notebook illustrating a certain concept without the need for knowledge about other things, such as the system it is running on.

Maximum likelihood estimation

$$\mathbb{V}(\hat{p}) = \frac{p(1-p)}{n}$$

In [3]: `sympy.plot(k*(1-k),(k,0,1),ylabel='$n^2 \mathbb{V}$',xlabel='p',fontsize=28)`



Out[3]: `<sympy.plotting.plot.Plot at 0xcfc25f0>`

In [4]: `b=stats.bernoulli(p=.8)
samples=b.rvs(100)
print var(samples)
print mean(samples)`

0.1659
0.79

Figure 10: Screenshot of a Jupyter notebook illustrating the maximum likelihood estimation, source: [unp12]

To take this even one step further, Kohlhase et al. [Koh+11] have presented the *Active Documents Paradigm* for use with STEM research papers. It aims at combining the executability of presented code with semantic annotations for the content as well as interaction with other users and authors, and services based on these blocks.

The Active Documents Paradigm is in parts matched quite well by the system we have illustrated as our “vision” in section 1.2: As the modeling and configuration generation happen inside the computer explicitly, all steps of the modeling process become storable and therefore replicable.

In fact, current efforts are aimed at making MMT functionality available in

Jupyter by way of a MMT kernel [MMTJup17]. We will therefore see some progress in the direction of more interactive ways of data flow – this will be relevant for the MOSIS system presented in section 3.1.

3. Bridging the Gap Between Informal PDE Theory and Simulations Practice

After reading the last sections, we have a clearer understanding of what the problem initially described in the introductory section 1 entails:

The mathematical modeling and simulation of physical problems is in most cases split up between two groups: On the one side of the great divide, there are physicists, engineers, mathematicians who have a mathematical model in mind. They just want to look at the solution to a PDE in order to infer more general knowledge, like better design decisions, and have no knowledge about or interest in simulation technology. We will call them *domain experts* – or, for our purposes, *users*.

On the other side, there are programmers with some expertise on how to solve PDEs efficiently, and on which kind of problems can be solved with the current simulation technologies. We will call these people *simulation experts*. Sometimes, there is a third kind of interest present: persons that have no mathematical background but still want to simulate using non-standard methods (for example because their problem is very special). They will usually consult some of the aforementioned domain or simulation experts. We will not concern ourselves with them for now, as our aim is to make simulations more easily accessible to mathematically versed persons.

The gap between domain experts and simulation experts is only slowly starting to be populated. As a central step, there were some study degrees set up in the field of Computational Science and Engineering (CSE) since around the turn of the Millennium. Graduates of these programs often times take the role of translators between the needs of the domain experts and the offerings of the simulation experts, and are therefore highly sought after.

But maybe there are ways to use computers to make the “translations” happen in a reliable and therefore verifiable way? This is what will be investigated in the following.

3.1. A Software Architecture Proposal

As pointed out in section 2.3, the knowledge gap between the informal theory in the user’s minds and the formal program lies exactly between layer 1 and 2 of ExaSlang. Furthermore, we learned that active documents are of interest for communicating mathematical contents in section 2.6.

These factors were guiding in the software architecture that is introduced now, and which we will call the Models-to-Simulations Interface System architecture, or short MOSIS, architecture. It is shown graphically in figure 11: Layer 1 is

designed to consist of flexiformal knowledge in the form of theory graph as well as an active document making it accessible to our user.

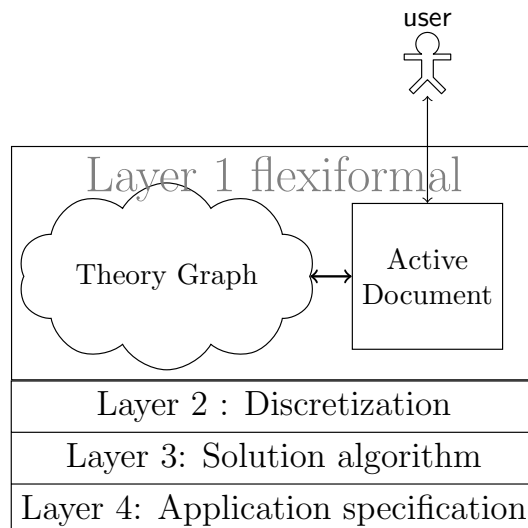


Figure 11: A schematic of the proposed MOSIS architecture

This architecture proposal is based on the assumption that a clear understanding of the domain knowledge is needed, and a (flexi)formal representation of the theory frame may be desirable, to draw definite conclusions about the user’s problem. This is what a human simulations expert would do, based on their knowledge and experience. In the MOSIS architecture, the theory graph is the represented base to taking this task of checking and “understanding”.

The active document is what the user interacts with, and can be realized in a multitude of forms – the more interactive the better. It may request input from the user, or give feedback about what is compatible with the current theory graph, and what is not. In the best case, the active document will also be able to “explain” the reasons for these judgments.

Even more, the output of the MOSIS program can be of interest for the verifiability of the whole process. When there is a good formal representation of the matter, it will be harder to make mistakes: if the input entered does not match the program’s expectation due to the information it has gathered up to that point, it will not accept it. Ideally, it will even tell exactly what the problem was and so it will prevent the mistake from even happening.

We might also find a positive effect towards the long-term reviewability of the simulation, as different files are generated on every step. At a minimum, this must be the layer 2 and 3 files for the ExaSlang stack. If there are files generated

describing the model used, this allows for an easier realization of the models-as-research-data approach envisioned by Kohlhase et al. [Koh+17].

Should it turn out that there was something wrong with the simulation result, it will be easier to find out at which stages errors happened: in the modeling (layer 1), discretization (layer 2), parametrization (layer 3), or in the solver (layer 4 and generated code)? This will be possible even years after the project, when all the scientists involved have long gone on to doing something else.

3.2. A Dream: The Perfect Interview

As pointed out in section 1.2, the all-knowing program is still just a vision. Yet, it is worthwhile to look at the processes that are happening inside the human expert and should therefore happen in our imaginary program as well. This will help us to find out if the architecture introduced in the last section 3.1 could potentially be used in our perfect world, or if it can be extended to one day.

For our initial example from section 1.3 (the engineer who would like to simulate the heat behavior of the walls of her house), the solution process would be clear: She would explain the problem to her friend, describe the physical properties in question, and chat about what else they need to know in order to simulate. When all that is clear, there would probably be different methods of simulation and then simulation parameters offered to her, depending on the problem.

This structure resembles a *technical interview* often times carried out between domain experts and simulations engineers, so this is probably what we want to achieve. Let us dive into the interviewer's head and have a look at the goings-on.

Looking at the result of the interview, we would like to know about the following things:

1. the domain
2. the unknown(s) or codomain(s)
3. the describing PDE(s)
4. the boundary conditions.

The ordering of these items is rising in complexity, and will make it easier for the interviewer to tell the properties of

- the type of PDE
- whether there is enough information in the PDE(s) to allow for solvability
- whether there is enough information in the boundary conditions to allow for solvability
- if it is a special case of boundary value problem for which solvability can be asserted or denied.

To gather all this information, questions will be asked and answers will be given. We will now walk through a symbol-based interview, with different possible answers, and look particularly at the requirements for the program performing it.

<i>Interviewer thoughts</i>	<i>and output</i>	<i>possible user input</i>
<p>1 To begin with, we ask for the physical domain of the simulation, that is time and space variables involved. The input could be given as n-dimensional cuboids as products of closed intervals and unions thereof. This allows to approximate any closed shape arbitrarily well. It should be possible to name different dimensions, and to rename the usual domain symbol Ω into something else.</p> <p style="text-align: right;">$\Omega = ?$</p> <p>If no name is given, the naming of the variable(s) defaults, for example to x_i. We could give feedback like</p> <p style="text-align: right;">$x_1 \in [0; 1]$</p> <p>Note that there are different notations for elementhood and interval separators. We still need to keep track of the order of variables given, and name those without a name. We could give uniform feedback like</p> <p style="text-align: right;"> $t \in [0; 1]$ $x \in [2; 4]$ $y \in [0; 1]$ </p> <p>A common user's expression of the domain may be</p> <p>which would lead us to</p> <p style="text-align: right;"> $t \in [0; 1]$ $x_1 \in [2; 4]$ $x_2 \in [0; 1]$ $x_3 \in [2; 3]$ </p>		<p>$\Omega = [0; 1]$</p> <p> $W =$ $(t \in [0; 1],$ $x = [2, 4],$ $y = [0; 1])$ </p> <p> $\Omega = (t \in [0; 1],$ $x = ([2, 4] \times$ $[0; 1] \times [2, 3]))$ </p>

From these parts of information, we can infer the number of dimensions and the actual size – mathematically speaking: measure – of the domain, and also dimensions, coordinates and measure of its boundary.

- 2** Next up, we would like to ask for the names and types of the unknowns, that is for the functions that we would like to know afterwards. Again, renaming should be possible.

$$u_1 : \Omega \rightarrow ?$$

or

Which we could register as

$$v_1 : \Omega \rightarrow \mathbb{R}$$

$$v_2 : \Omega \rightarrow \mathbb{R}$$

Furthermore, we would like to understand sub-naming of variables

$$u : \Omega \rightarrow \mathbb{R}$$

$$w : \Omega \rightarrow \mathbb{R}$$

The one thing that should never change is the type operator, domain name and arrow “ $: \Omega \rightarrow$ ”. Now we can conclude the number of unknowns, where each dimension can be considered an independent unknown, but their ordering and names of unions of two or more variables need to be kept in memory.

$$v : \Omega \rightarrow \mathbb{R}$$

$$v : \Omega \rightarrow \mathbb{R}^2$$

$$v : \Omega \rightarrow \mathbb{R}^2 \\ = (u, w)$$

- 3** Now we go on to look at the central piece, the partial differential equation. Let’s just have our interviewee write it down.

$$\Delta u = 0$$

We should at this point make explicit what we think it is that the user wrote down, ideally using a collection of formally well-defined knowledge that can expand commonly used derivation operators. For PDEs up to second order, the best format to aim for is a matrix-like representation.

$$\begin{bmatrix} 1 & \partial_{x_1 x_1} & +0 & \partial_{x_1 x_2} \\ +0 & \partial_{x_2 x_1} & +1 & \partial_{x_2 x_2} \end{bmatrix} u = 0$$

Most likely, it is even desirable to have this representation modifiable by the user, as some operators are ambiguous or are implicitly defined on certain domain variables only. This caveat becomes clearer for a slightly more complicated example

which we may justifiably expand to

$$1 \partial_t u + \begin{bmatrix} \mathbf{1} & \partial_{tt} & +0 & \partial_{tx_1} & +0 & \partial_{tx_2} \\ +0 & \partial_{x_1 t} & +1 & \partial_{x_1 x_1} & +0 & \partial_{x_1 x_2} \\ +0 & \partial_{x_2 t} & +0 & \partial_{x_2 x_1} & +1 & \partial_{x_2 x_2} \end{bmatrix} u = 0,$$

while what the user meant was actually

$$1 \partial_t u + \begin{bmatrix} \mathbf{0} & \partial_{tt} & +0 & \partial_{tx_1} & +0 & \partial_{tx_2} \\ +0 & \partial_{x_1 t} & +1 & \partial_{x_1 x_1} & +0 & \partial_{x_1 x_2} \\ +0 & \partial_{x_2 t} & +0 & \partial_{x_2 x_1} & +1 & \partial_{x_2 x_2} \end{bmatrix} u = 0.$$

A simple but very central assertion to take is that the number of PDEs given corresponds to the number of unknowns. In a more advanced setting, it should also be possible to enter PDEs in integral or boundary formulation. This means to have an equation containing (boundary) integrals that hold for arbitrary subsets of the domain, which is often used with conservation laws, e. g. in the finite volume method.

Once the equation is established, we should try to determine the type of PDE described (as introduced in section 2.1). Sometimes however, this may require further knowledge, or the type may change inside the domain, so that the type cannot always be told. In case we can conclude the type, we use this information later on to make better predictions about the solvability.

$$\frac{\partial u}{\partial t} - \Delta u = 0$$

- 4 Finally, we can start to ask for the last yet very fundamental part of our interview: what are the boundary conditions? We already know about the domain and PDE, so we can determine the amount of information needed in each dimension and unknown independently.

The first factor required is the (Lebesgue) measure of the normal cut in this dimension. It can be determined from the domain information, giving us a cuboid lower in dimension than the domain. And the other factor is given in the PDE: find the highest derivative (with respect to this dimension) applied to the variable – occurrences in multi-indices need to be taken into account – and subtract one from it, so that all ambiguous constants can be uniquely determined.

We may communicate this “count” to the user like

BCs for $u(0, x, y)$ or $u(1, x, y)$? $1 \cdot [2 \times 1]$ still required.

BCs for $u(t, 2, y)$ or $u(t, 4, y)$? $2 \cdot [1 \times 1]$ still required.

BCs for $u(t, x, 0)$ or $u(t, x, 1)$? $2 \cdot [1 \times 2]$ still required.

and gradually “count down” as more and more information is entered.

BCs for $u(t, x, 0)$ or $u(t, x, 1)$? $1.5 \cdot [1 \times 2]$ still required.

BCs for $u(t, x, 0)$ or $u(t, x, 1)$? $1 \cdot [1 \times 2]$ still required.

Obviously, the interviewer needs to have a good understanding of euclidean geometry to measure the size of the definition space of the boundary conditions – and to notice if there are some contradictory conditions given for the same part of the boundary. Generally, it may be a good idea to directly sort the given boundary conditions into the different types (as introduced in section 2.1.2), to allow for later analysis.

$$u(t, x, 0) = 0 \\ \text{for } x \leq 3$$

$$\nabla u(t, x, 0) \cdot n \\ = 0 \\ \text{for } x < 3$$

From the last input, we see that also the outside-pointing normal vector n needs to be understood by our system, and therefore it could be a reserved character or special button to push.

As we know from section 2.3, having the boundary conditions written down in a natural way is not an easy task. The syntax presented here was the most natural one that came up during work on this thesis, but we should keep an open mind to allow for as many different representations as possible. As an alternative, we could provide stubs or some kind of form to fill in different types of boundary conditions.

The strictness of handling the openness and closedness of the domains of definition for the boundary conditions is still up to discussion. In a measure theoretical sense, the line $(0, 3, y)$ is irrelevant, but for an accurate numerical representation it may be necessary to evaluate the boundary conditions exactly on this line. In that situation, it would be helpful to have the property defined exactly once on the line, and not undefined or defined twice.

At the end of the boundary conditions negotiation, we must make sure that the boundary conditions do not consist of Neumann and periodic types only, cf. section 2.1.2. If we find that we are certainly running into the problem of ambiguity, we can make sure that the consistency is fulfilled (eq. (12)), and ask the user to determine a Dirichlet-type condition on a single point of the boundary.

-
- 5** So we have finally arrived at the end of the interview. Now we can see if we can find theorems that assert or deny the unique solvability of the boundary value problem. In case the existence of a solution is certain or can not be denied directly, we should propose solution algorithms and make them accessible to the user.
- As a last step, we can invoke the simulation and return its result.
-

These interviews went well – but they were only imaginary of course. However, there is no fundamental reason why it should not be possible to have this kind

of interrogation done programmatically. Of course, a lot of smart tools would be needed to make it work in this extensive way, for example symbolic computations to get the equations into the right shape. Most importantly, there are a lot of notions and notations linked to partial differential equations that the program would need to understand. But it seems like the architecture introduced in section 3.1 can support this, if its parts are made strong enough.

Note that the order of the questions was important to build up our own idea of the model that the user entered. It seems that there is some structure here, and we will be making it more clearly visible in section 3.4.

3.3. Ephemeral Theories

In the interview setting envisioned in section 3.1, we would like to apply our knowledge to new parameters that can be changing with every interview. Imposing this requirement on the MOSIS architecture (cf. section 3.1), said volatile values can obviously not be made part of the original “background knowledge” – they would clutter it and would most likely not be of use for the next active document.

This is why a new mode of creating MMT theories is needed that allows for volatile knowledge which need not be kept across sessions. We call these *ephemeral theories*. Conversely, we will call the non-ephemeral background *persistent theories* for the purposes of this thesis.

In fact, the foundations for this mechanism are already given in the standard for the *Symbolic Computation Software Composability Protocol (SCSCP)* [Fre+] describing the interaction with OpenMath [Bus+04] content:

`store_session`: store an object on the server side (possibly after computing or simplyfying[sic!] it), returning a cookie (actually, an OM reference) pointing to that object. This cookie is then usable within the current session to get access to the actual object

There now exists an extension to the MMT server [Rab] implementing this functionality. It allows for the creation of ephemeral theories and views, as well as adding declarations and assignments to them. It can be found in the MMT repository [Flo15] and is currently called `mmt-interviews`. This feature will be heavily used in THEINTERVIEW implementation discussed in section 4.3.

3.4. A Theory Graph to Rule Them All

Now, having discussed partial differential equations in section 2.1, and theory graphs in section 2.4, as well as ephemeral theories (section 3.3), we can proceed to present a theory graph for the application of PDE knowledge: the PE (Poisson equation) theory graph. A full-sized image can be found in appendix A.

The graph shows the building blocks of the mathematical PDE theory applied to our running heating example from section 1.3; in contrast to section 2.1, it is presented as small theories that are highly structured, to show the logical connections. As in section 2.4, we will be walking through the theory graph step-by-step. This time, we start in the lower right corner, with the domain theory, cf. figure 12.

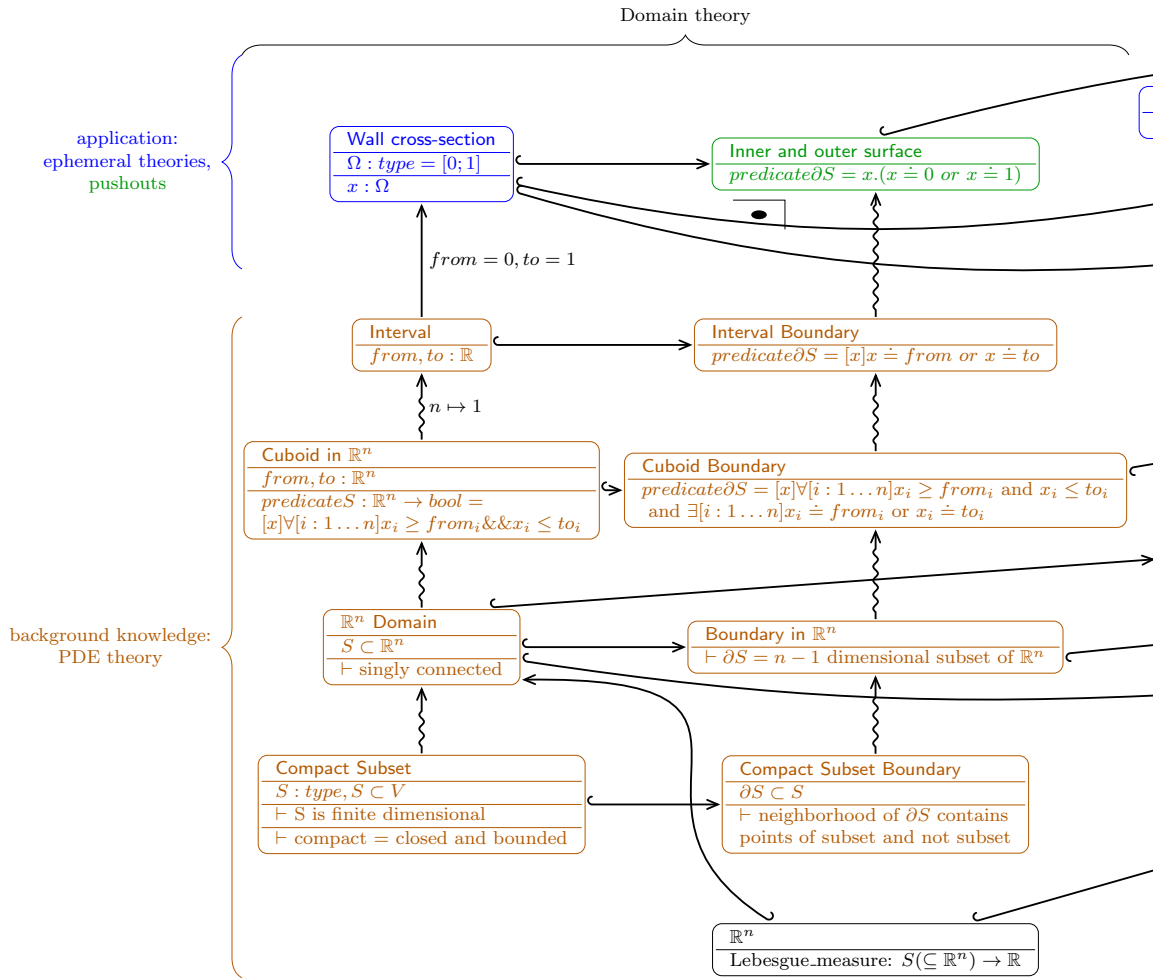


Figure 12: Theory graph for PDEs: Domain theory

On first sight, it becomes apparent that the graph is evolving from general

theories about spaces to the very concrete example of one specific interval. Consequently, the upper “application theory” is marked as ephemeral by the blue coloring, in contrast to the general PDE domain knowledge in orange.

Looking at the details, the graph is building up like a ladder, starting at the very bottom with the compact subspace of a vector space – the corresponding boundary is placed exactly to its right. We are stating that it is finite-dimensional and compact axiomatically with the deduction operator \vdash ; its semantics are “it can be proven that...”, and it creates the type of “proofs that show that...”. On top of the compact subset, there is an arbitrary \mathbb{R}^n Domain, which is *viewed* as a Compact Subset. All the properties in Compact Subset of course also have a mapping in \mathbb{R}^n Domain, but these have been left out in the diagram for convenience. Only one additional property is given: the \mathbb{R}^n Domain is required to be singly connected. This is trivially true for a Cuboid in \mathbb{R}^n , which is uniquely determined by two \mathbb{R}^n points *from* and *to*. The cuboid’s boundary is a union of cuboids of lower dimension, which we get if one coordinate entry x_i is equal to the corresponding *from_i* or *to_i*.

We are restricting ourselves even more to just one dimension, in which the cuboid becomes an Interval, to be “prepared” for our example from the introduction section 1.3. For the interval, the boundary is simplified to just two distinct points. This fact can also be applied to our ephemeral Wall cross-section, so that we can conclude the boundary coordinates by way of a pushout.

Using this structure allows us to formulate properties generally, restricting them to the special case only when needed. For instance, the background knowledge in the next part of the graph – the PDE theory in figure 13 – is connected only to the \mathbb{R}^n Domain.

Here, we see a very central notion of PDEs: even as we do not know the definition of the Unknown yet – because it is just what we will be looking for in the simulation – we need to know the Codomain it maps to, and also some name for it to use in the definition of the PDE. This is what Temperature is illustrating. In contrast, the Parameters Thermal conductivity and Volumetric heat flow are providing a definition which may be constant or varying inside the domain. This resembles the usual scientist’s method to abstract parameters through symbols before applying them to new situations.

The PDE declaration is relatively simple, as we do not know so much about the Parameters involved yet. This only changes when the definition is given in Static heat equation. The Differential Operators symbols, which are based on imported Calculus knowledge, can be directly used for this purpose.

As we know from section 2.1.2, a PDE can only be solvable with the right amount of boundary conditions given, which is why the boundary conditions theory is strongly linked to the solution theory, just like the PDE theory. We can see

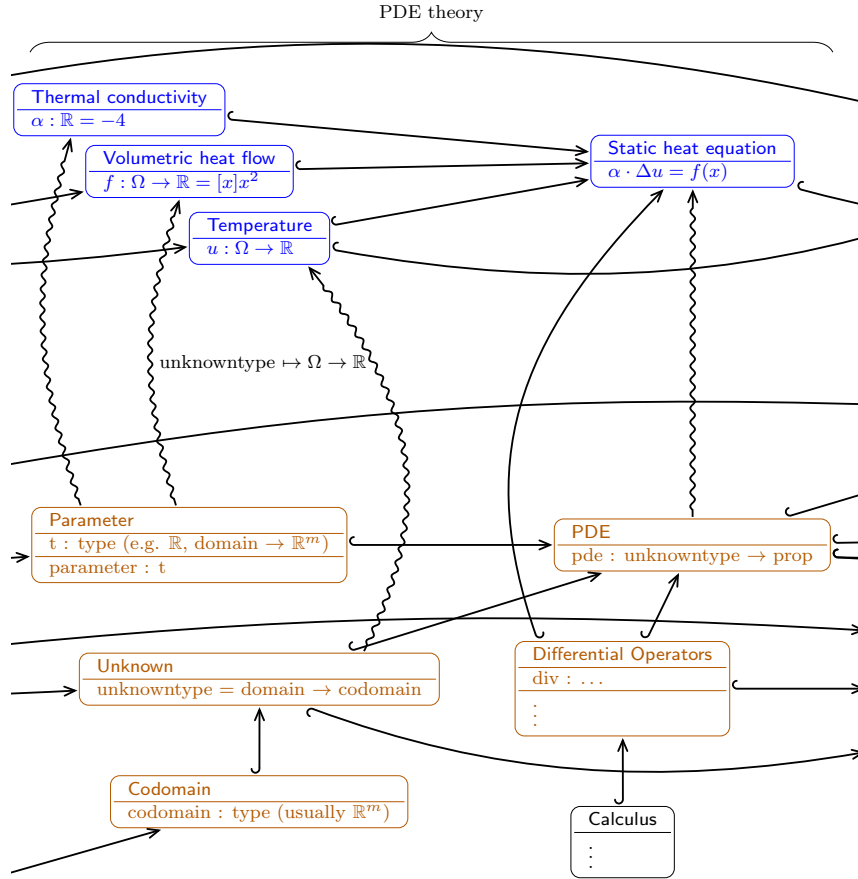


Figure 13: Theory graph for PDEs: PDE theory

these connections in figure 14. Apart from the definition of possible **Boundary Conditions**, we can use our thoughts on required boundary conditions for cuboid domains, which we had developed in section 2.1.2.

Now maybe we can find a proof that the operators of the **PDE** applied to the unknowns are elliptic and linear, in which case we would deal with **Elliptic** and **Linear PDEs**, respectively. All of this, along with the required boundary conditions, sums up to a **Linear Elliptic Boundary Value Problem**.

As it turns out, this is one of the cases where the existence of a unique solution can be asserted by way of the Lax-Milgram lemma [KA00], such that we can create a view. Of course, there are some other combinations of operators and boundary conditions that assert or deny a solution – much more work would be needed to note them all down, which is indicated by the ellipsis ... on the right of figure 13.

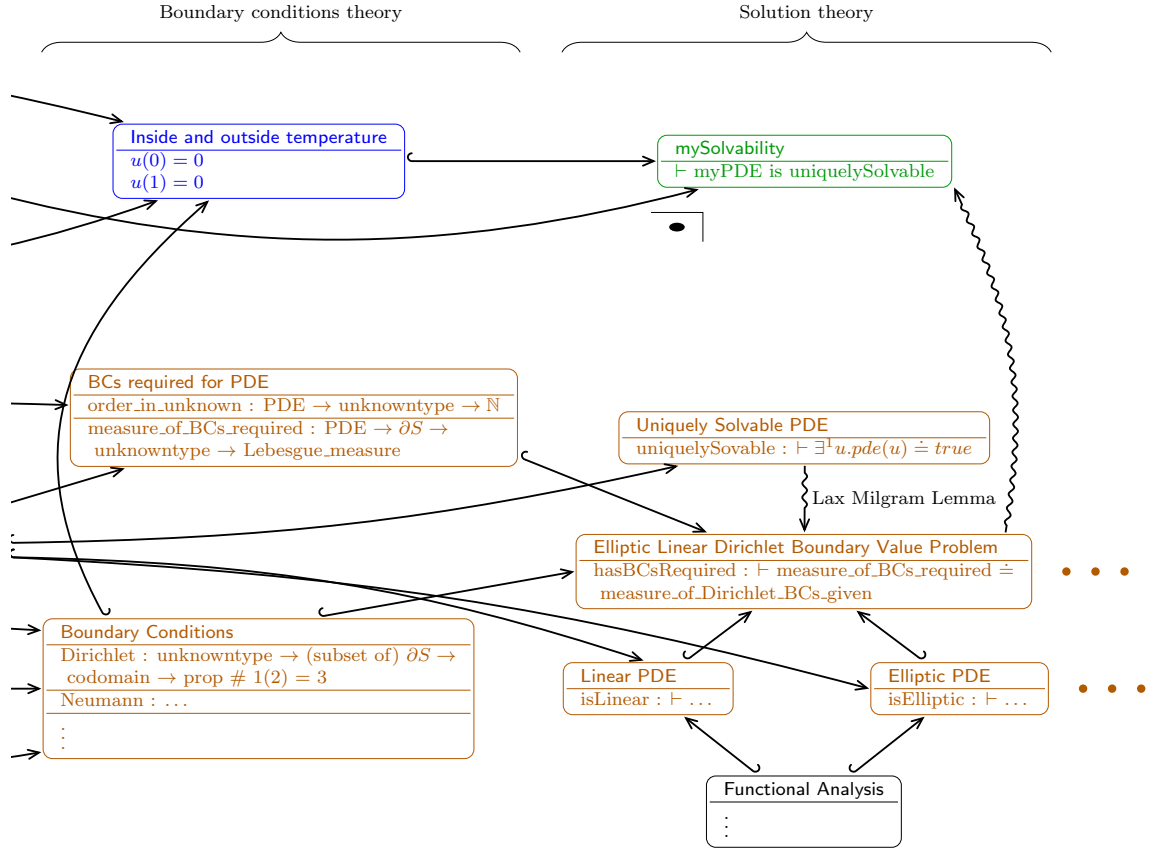


Figure 14: Theory graph for PDEs: Boundary conditions theory

The input entered so far matches this very problem, so we finally can tell – depending on the user input regarding boundary conditions – whether we know a proof of the unique solvability of their problem by way of a pushout.

If we recall the mind experiment of an expert interview from section 3.2, we find that *all the mathematical information needed* to investigate on the user’s model *can be found in the PE theory graph*. This is an essential requirement for the MOSIS architecture.

3.5. A Prototype Proposal: Conducting an Interview

In the last few sections we have “tested” two things in our thought experiments: We now know that an interview can be a way of interacting with our user (cf. section 3.2), and that the PDE knowledge needed for it can be realized as a theory graph (cf. section 3.4). Accordingly, we can now “instantiate” the MOSIS architecture: the theory graph is now managed by MMT, and the active document becomes a technical interview, see figure 15. We will call this the MOSIS prototype.

The user, for example our engineer from section 1.3, “talks” to the interview program, which in turn sends information to and gets replies from the MMT system. MMT “knows” about the theory behind PDEs – depicted by the cloud – and can send back insights to the interview application. During the process, the description of the concrete problem (e.g. the heat equation and the related parameters) is built up and added to the knowledge base. This allows for keeping the ephemeral model description separate and saving it to file independently.

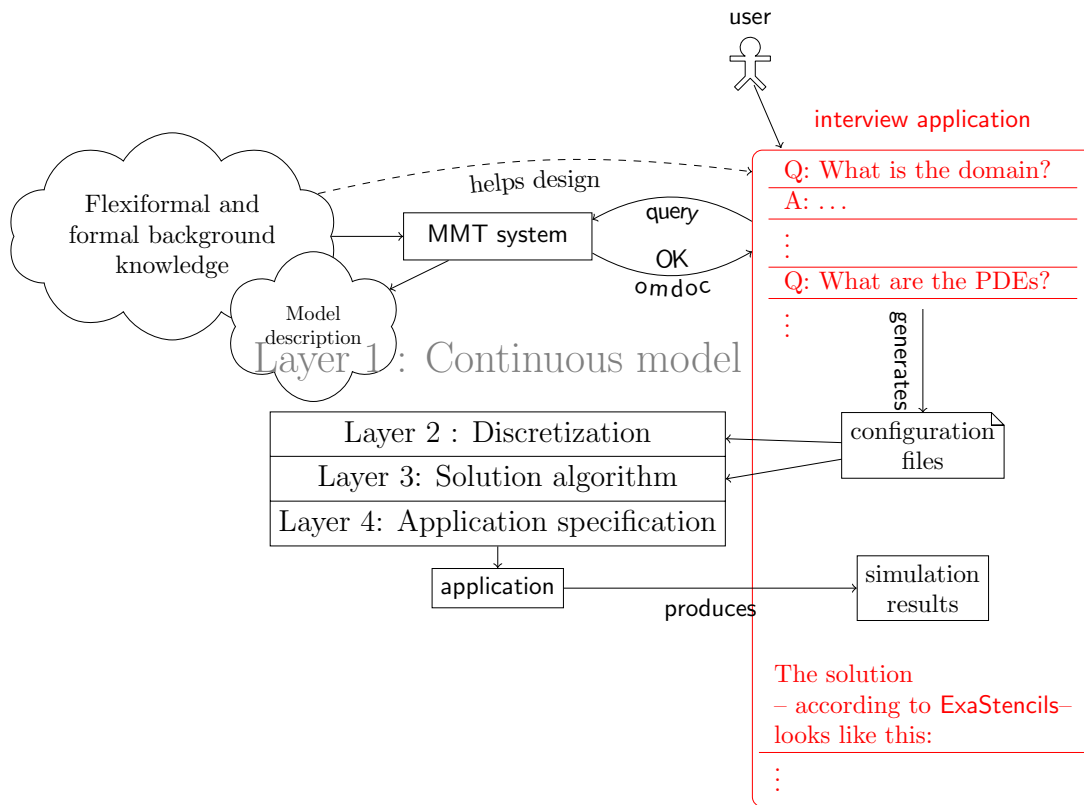


Figure 15: A schematic of the MOSIS prototype

At the end of this process, some configuration files are generated that contain the information relevant for the simulation – as seen in the right part of figure 15. These files make up the **ExaSlang** layer 2 and layer 3 information. As the interview asks about the continuous properties of the model, it is effectively taking the responsibility of **ExaSlang** layer 1.

After the simulation has been executed, the results can be sent back to the user; in the best case, the user has access to different modes of visualization for it. The engineer would maybe see a colorful temperature distribution along the cross-cut of the wall.

Note that all of this is still very general and therefore independent from the actual implementation of the interview program. **THEINTERVIEW** that will be presented in section 4.3 is intended to be a proof of concept and there are of course more rich and immersive variants to be found.

4. Formal and Flexiformal PDE Knowledge

In section 3.4 we introduced a conceptual theory graph for PDE theory. Naturally, the next step is to formalize the PDE knowledge as illustrated in the graph. The results can be found in the MathHub repositories `MitM/smgloM` [MitM], and the flexiformalizations in the `SMGloM` [gitGlo], respectively. And, of course, you may have a look at the static copy in appendix B.

4.1. Formalizations

We will now have a look at some parts of the formalizations, and also discuss the limitations that the MMT system still has to build up the full theory graph.

Structural Particularities

When looking at the full theory graph in figure 23, we see that all the background knowledge is self-contained, as the arrows connecting the `background` and `application` theories are only going from bottom to top. This independence of persistent theories from ephemeral theories allowed us to put an example for a concretely applied ephemeral PDE description into a separate file called `interview_ephemerals.mmt`, with no other theories depending on it.

Furthermore, most of the arrows crossing from the persistent to the ephemeral area are denoting views. If we recall the theory and view syntax from section 2.5.4, we realize that it should be possible to have most ephemeral theories consisting of `includes` only. All the counterparts to the theories that the “parent” background theory includes are also included, and sometimes more, e.g. the `Differential Operators` into `Static heat equation`. Then, every view consists of two parts: It starts with `includes` mapping the respective modules by way of views that were introduced before. After that, we can think of the “new” definitions and assign them to their background counterparts right away. This works nicely for `ephParameter1`, as we can see in figure 16.

So much for the theory – there are some cases where this was not possible in the actual implementation, and we are going to look closely at the factors that are making it hard to keep the structure as simple as described. We will even get to the point where some formalizations were not possible to be carried out without either violating the independence of the background from the application theory or implementing major changes in MMT itself. This part of the implementation has been delayed for now.

Copying in Structures

The observant reader may have noticed that the theory graph shows the morphism

```

theory ephParameter1 : base:?Logic =
  include arith:?realarith ;
  include ?ephDomain ;

view ephParameter1AsParameter : ?Parameter → ?ephParameter1 =
  include ?GeneralDomains = ?ephDomainAsDomain ;
  ptype = ℝ ;
  param = -4 ;

```

Figure 16: Viewing -4 as a parameter to the PDE

between [Interval](#) and [Wall cross-section](#) to be a structure (cf. section 2.4), while the example in section 2.5.4 clearly defines Intervals as record types – making the concrete interval just an instance of it.

The reason for this “workaround” is the strictness with which MMT structures used to copy *all* the constants defined in them – e. g., this applies to the definition of real numbers as well. The contents cannot be identified as being the same between theories any more, and this leads us into trouble. This matter has recently been solved in the new MMT release, but its benefits could not be part of this thesis any more, cf. [\[issue237\]](#).

Also, using record types makes the construction of an Interval nicer in syntax: as we see in figure 17, we can just use the square brackets to create one.

```

theory ephDomain : base:?Logic =
  include ?SimpleDomains ;
  myInterval = [0;1] ; role abbreviation ;
  myDomainPred = ((myInterval).interval_pred) ;
  myBoundaryPred = ((myInterval).boundary_pred) ;
  myDomain = pred (myInterval.interval_pred) ; role abbreviation ;

view ephDomainAsDomain : ?GeneralDomains → ?ephDomain =
  Vecspace = ℝ ;
  DomainPred = [0;1].interval_pred ;
  BoundaryPred = [0;1].boundary_pred ;

```

Figure 17: Viewing the interval $[0; 1]$ as a Domain

On the downside, it also requires to explicitly state the domain and boundary predicates in the view – which in turn are needed for predicate subtyping.

Predicate Subtyping for Derivatives, and Expansions

Predicate subtyping is of high interest for PDE formalization, as it allows to define functions partially, for example only on the [Domain](#). It restricts a type to only

these representatives that fulfill a certain condition (the predicate), and is casually used in mathematical documents:

$$X := \{x \in \mathbb{R} : x \geq 2.2 \wedge x \leq 5\}.$$

This statement reads as “ X consists of all those x in the set \mathbb{R} , that are larger than 2.2 and smaller than 5”. The predicate here is “greater than or equal to 2.2 and smaller than or equal to 5”, and the type that is sub-typed are the reals.

The MMT syntax for this would be

$$\langle \mathbb{R} | [x : \mathbb{R}] \vdash x \geq 2.2 \wedge x \leq 5 \rangle$$

We could also use the constant `pred` which is an abbreviation for the subtyping on reals, which would shorten the expression above to `pred [x] x ≥ 2.2 ∧ x ≤ 5`. The subtyping operator `pred` is also used for defining the derivative, and is actually making the definition of `ephUnknown` a bit cluttered.

```
theory ephUnknown : base:?Logic =
  include base:?Strings
  include ?ephDomain
  include calculus:?higherderivative
  // myCodomain : type = ℝ
  myUnkType : type = myDomain → ℝ
  = pred myDomainPred → ℝ
  myUnk : myUnkType
  uwillbediffable : ⊢ twodiff myUnk = sketch "because this is how it will come out"
  anyuwillbediffable : {u : myUnkType} ⊢ twodiff u

view ephUnknownAsUnknown : ?Unknown → ?ephUnknown =
  include ?GeneralDomains = ?ephDomainAsDomain
  ucodomain = ℝ
  unknowntype = myUnkType
```

Figure 18: Defining the type of the unknown

As the figure 18 shows, we would rather be defining the type of the unknown as `myDomain → ℝ`. This is currently not possible, as the definition of differentiability `anyuwillbediffable` (and later on also the Laplace operator) is expecting the unknown to be some function defined on a predicate subtype (constructed with `pred`) denoting the domain. The definition of `myUnkType` however, is not being expanded enough at this point for the type check to succeed.³

³One might ask as to whether it is really sensible to just state that every function of this type is two times differentiable – and the answer is clearly no. However, differentiation in differential equations is always applied to unknowns, and as the definition says, the unknown will just be defined in a way as to be differentiable. Still, there might be better ways of dealing with this situation.

To cut a long story short: In order to resolve this problem, MMT would need to expand nested type definitions to the right degree at the right time when parsing the differentiability, which it is currently not capable of doing – it is not a trivial task. If MMT had this feature though, we could save some keying in the theory `ephDomain`, cf. figure 17, as we would not need to store all the predicates explicitly.

This is why first experiments were done with an additional type checker cue, `role abbreviation`, which may be added to these constants that should always be expanded whenever used. Further investigations will be necessary to tweak the desired behavior of this tool, and even more will be needed to make the type checking work for all relevant yet intertwined cases. The progress on this issue may be followed up on in the MMT issue tracker [i32617].

The ideas presented in the next paragraph may even help to make the other definitions in `ephDomain` redundant, and as we know, that means that the formalized structure is as tidy as we want it to be.

The Need for More Intricate View Referencing to Fully Use Pushouts

So far, we have discussed those formalization issues that create the need to key in some more lines, or for light restructuring of the theory relations in the MMT surface language. Now, however, we will talk about the point where the formalization could not be continued any more, because the syntax did not allow for the use of pushouts. The reader may remember that we found pushouts to be one of the core concepts for knowledge generation in section 2.4, and not using them in this particular situation would mean to not employ any inference functionality and instead explicitly state everything we know again.

But let us have a closer look at what is going on. The view `ephBCsasBCs` requires us to map the individual boundary conditions

```
view ephBCsasBCs : ?BCsRequired → ?ephBCs =
  include ?GeneralDomains = ?ephDomainAsDomain ;
  include ?Unknown = ?ephUnknownAsUnknown ;
  include ?PDE = ?ephPDEasPDE ;

  firstBC = DirichletBC 0 1 ;
```

Figure 19: Troubles formulating the boundary conditions

to concrete values. But of course the definition of a `DirichletBC` happened in the persistent part of theory already, as it should be reusable to any situation, such as different dimensionalities. So `DirichletBC` is defined in terms of the persistent properties `Boundary`, `ucodomain` and `Domain`, cf. figure 20, but needs to be `included` in the ephemeral theory in order to be mapped to the persistent theory. At the same time, what the persistent terms actually “mean” in this

```

DirichletBC : {where: Boundary, rhs: ucodomain } (Domain → ucodomain) → prop
= [where, rhs] [u] u where = rhs
link meta?metadata?alignment http://mathhub.info/smgglom/theresas-playground/bounda

```

Figure 20: Definition of a Dirichlet boundary condition for a single point

situation, has only been defined in views so far. Consequently, the expressions cannot be replaced with their ephemeral assignments from those views, which is what we would need – for example, to actually find out that 0 indeed is on the `Boundary` for this particular problem. Currently, these are just two unconnected symbols.

This is why a mode and syntax to referencing the information contained in views is needed. We can think for example of something like `ephUnknownAsUnknown/ucodomain` that would replace every occurrence of `ucodomain` with its definiens from the given view. This may become trickier for these cases where the term itself is defined in the persistent theory, and the symbols occurring in the definition are the ones to be mapped by way of views. For instance, `Boundary = vec_pred BoundaryPred` would need to be re-evaluated to know what it means for our concrete situation. But in principle it should be possible to have such a mechanism present in MMT.

The alternative using the currently available features would be to re-formulate the boundary conditions on the ephemeral side of the theory graph (for example called `myDirichletBC`), manually inserting all the types used for the current problem. This can clearly not be what we want, as we would not be using the pushout functionality that makes the small theory approach in MMT so appealing.

4.2. Flexiformalizations

In addition to the formal knowledge, advances have been made in the flexiformal representation of PDE knowledge: Some \LaTeX modules are now there, defining different terms connected to the field – we have learned the definition of a PDE from one of them in section 2.1.

The area of connecting or *aligning* terms that describe the same content in different systems of representation is under active research [Mül+17]. Simple alignments can be used for reformulating mathematical concepts into other logical systems. For the alignments from the formal MMT knowledge to the flexiformal \LaTeX knowledge this means that explanations can be given: For instance, the formalization of a general `Domain` looks quite opaque by itself, as we have seen in figure 9, but knowing the \LaTeX URI, one can look up the definition of a domain.

However, by now it has not been agreed on what exactly should happen when an MMT user wants to look up the URI: should it be opened on the user’s local ma-

chine? This would require them to have all the corresponding modules installed. Maybe it would be sensible to have the respective term in the MathHub mathematical glossary [SMG] opened in the browser? In this case, newly created \LaTeX definitions that are not part of the glossary could not be used. This is why so far only written indications of the alignments have been added to the `.mmt` files with the `link` command. An example of this is given in figure 20. The information is currently not processed any further inside the `jEdit` editor – apart from being added as a tag to the OMDOC module.

4.3. A First Implementation: TheInterview

To show the actual feasibility of the MOSIS architecture introduced in section 3.1, a line-based Python program was implemented, which we will call `THEINTERVIEW`. Not surprisingly, it also is a first approach to realizing the MOSIS prototype as discussed in section 3.5. `THEINTERVIEW` is taking all the responsibilities of the interview in figure 15 but, naturally, is bound to the limitations described in section 4.1.

`THEINTERVIEW` is basically “walking through” the upper, ephemeral part of the theory graph, figure 23, from left to right. It does so by creating something resembling the content of our example file `interview_ephemerals.mmt`, but with the user-defined variables filled in at the spots of interest.

In order to not “reinvent the wheel” here, Python packages were used to make the program a read-eval-print loop [cmd216] – for the line-wise in- and output – as well as a state machine [trans14], for keeping track of which part of the theory graph it is currently looking at.

Formal content is created by the MMT server using the interface described in section 3.3: Ephemeral theories and views can be created, and filled with declarations and assignments. What `THEINTERVIEW` gets back are status codes to know whether the parsing and type checking were successful. Additionally, MMT can be queried for OMDOC descriptions of the modules and constants currently present in the scope via the MMT query language. This is handy for using simplified results, letting most of the parsing be done in MMT. However, to fully exploit that feature, more development into reusable OMDOC parsing in Python is needed, and this is a subject of ongoing research.

One example where the query functionality will be extremely useful is the explanation of context or expressions to the user. As pointed out in section 4.2, it is possible to open aligned \LaTeX references in the browser, once the corresponding terms are referenced in the MathHub mathematical glossary [SMG]. It is therefore one of the next straightforward steps to implement this explanation functionality in one way or another.

From the information entered by the user, configuration files are generated: the

layer 2 and 3 information are placed in a subdirectory, along with the (for our purposes) static layer 4 file. Also, the `knowledge` and `settings` files required by `ExaStencils` are partly generated, while `platform` needs to be manually adjusted to contain information about the current platform.

As pointed out in section 3.1, storing and re-using the information about the model is a desirable aspect. To this end, writing the OMDOC file to disc is another task that is readily doable in `THEINTERVIEW`.

The discretization of the Laplace operator (eq. (9)) that we have seen in listing 1 is currently hard-coded. For details on how to automatically generate a stencil from the operator entered please refer to the thesis by Flad [Fla17], who implemented the automatic discretization – it is just not integrated yet.

```
localadmin@localadmin-TravelMate-P253:~/Desktop/masterarbeit/cmd$ python3 interview.py
Hello, user! I am James, your partial differential equations and simulations expert. Let's set up a simulation together.
How many dimensions would you like to simulate?
(dimensions)1

What is the domain you would like to simulate for?     $\Omega = [?;?]$ 
By the way, you can always try and use LaTeX-type input.
(domain) $\Omega = [ 0 ; 1 ]$ 

Which variable(s) are you looking for?     $u : \Omega \rightarrow ?$ 
(unknowns) $u : \Omega \rightarrow \mathbb{R}$ 
Ok,  $u : \Omega \rightarrow \mathbb{R}$ 
Are these all the unknowns? [Y/n]?
Y
 $u : \text{pred myDomainPred} \rightarrow \mathbb{R}$ 
Would you like to name additional parameters like constants or functions (that are independent of your unknowns)?
 $c : \mathbb{R} = ?$  or  $f : \Omega \rightarrow \mathbb{R} = ?$ 
(parameters) $f : \Omega \rightarrow \mathbb{R} = [x] \cdot x \cdot x$ 
```

Figure 21: A screenshot of `THEINTERVIEW` Python program

5. Conclusion and Future Work

This thesis addresses the knowledge gap in modeling and simulations practice: People with a PDE to be solved, but no background and interest in the numerical solution are still required to feed the PDE model into the simulation (cf. section 1). This often requires detailed system knowledge or intense discussion with simulation experts, or else errors are bound to occur.

For the finite difference solver **ExaStencils**, this problem has so far been approached with the development of a dedicated domain-specific language, **ExaSlang**, cf. section 2.3. The knowledge gap discussed above corresponds to a gap in the **ExaSlang** design: Layer 1 was envisioned as a natural mathematical interface but remained unspecified. This thesis fills the gap by providing an architecture and components for it.

I have proposed the MOSIS architecture in section 3.1. It combines represented knowledge with an active document that the user can interact with. A formal representation of PDE knowledge is required to understand different notions and formulations in the model input, while a flexiformal one helps to communicate what is required of the user.

We established the feasibility of this approach in an envisioned expert interview: All knowledge needed in **THEINTERVIEW** can be traced to representations in the PE theory graph. And we could also use the idea of a technical interview as an active document for the MOSIS prototype, as introduced in section 3.5. It uses MMT (cf. section 2.5.4) and $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ (cf. section 2.5.3) for the representation of formal and flexiformal knowledge.

As a first approach, **THEINTERVIEW** was implemented as a proof-of-concept for the MOSIS prototype, cf. section 4.3. The Python program has a MMT backend through which it can access the PDE knowledge formalized during the course of this work.

It became apparent that a lot more formalization of PDE knowledge will be needed to represent the situations regularly occurring in simulations practice. Especially in section 4.1, some important impulses towards the improvement of the MMT system were given. **THEINTERVIEW** can be extended to use the flexiformal $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ knowledge to communicate definitions to the user.

So indeed some steps towards our initial vision have been taken: The MOSIS prototype's aim of automating a technical interview to set up a simulation has proven possible; it works generally, because the design of the interview is backed up by the formal representation of the PDE model. The reader may recall that this covers the first two parts of this thesis' mission

- O1** to *explore the structure* of reasoning about PDEs ✓
- O2** to provide a *simple implementation* of an interface that allows users to enter their PDE problem in a natural way ✓
- O3** to *use the output* of that interface for *numerical simulation*. □

For the third point, I have gone part of the way. Its completion remains open and leaves an incentive to investigate further.

My thesis was largely exploratory. The MOSIS architecture, and its prototypical showcase in THEINTERVIEW, are tangible core contributions, along with the formalized knowledge in the form of the PE theory graph and MMT formalizations. But they only break the ground on an exciting new project; the most immediate next steps include the following.

Horizontal Scaling of (Flexi-)Formalizations Many more formalizations will be needed to make our architecture work for real-life scenarios, most notably the generalization of domain and codomain spaces to \mathbb{R}^n . The HOL Light multivariable library [jrh15] will be of help here. This is a potentially lengthy but not-so-difficult task, which will be rewarding for other formalization efforts as well.

Integration with Model Pathway Diagrams Model pathway diagrams or MPDs are a recent idea by Kohlhase et al. [Koh+17] to visualize the flow of information in simulation models. In an MPD, the physical quantities – or unknowns and parameters – are put into circles, and the equations describing them are shown in rectangles, with an edge connecting it to every variable occurring in the equation. Accordingly, there are only edges connecting variables and equations. From the number of paths, equations and leaf quantities, it can be inferred whether the model is fully determined [Koh+17]. The current MPD viewer and examples can be found at the MPDHub [MPDHub]; a screenshot is shown in figure 22.

Not surprisingly, the model pathway diagram is equivalent to a theory graph describing the physical model: The unknowns and parameters all have their own theory, type and in the case of parameters, values. The equations **include** all of the physical quantities described in them. For our program, this means that the way to generating MPDs from the interviews is not far-fetched at all: the ephemeral “user-filled” theories already have the same structure, the data exchange just needs to be granularly implemented, possibly even allowing for visualization right as the user enters their problem. This may be part of the next bigger step:

Making the Active Document Stronger THEINTERVIEW implementation was only a showcase; for the MOSIS architecture to reach its potential, there will be stronger implementations needed.

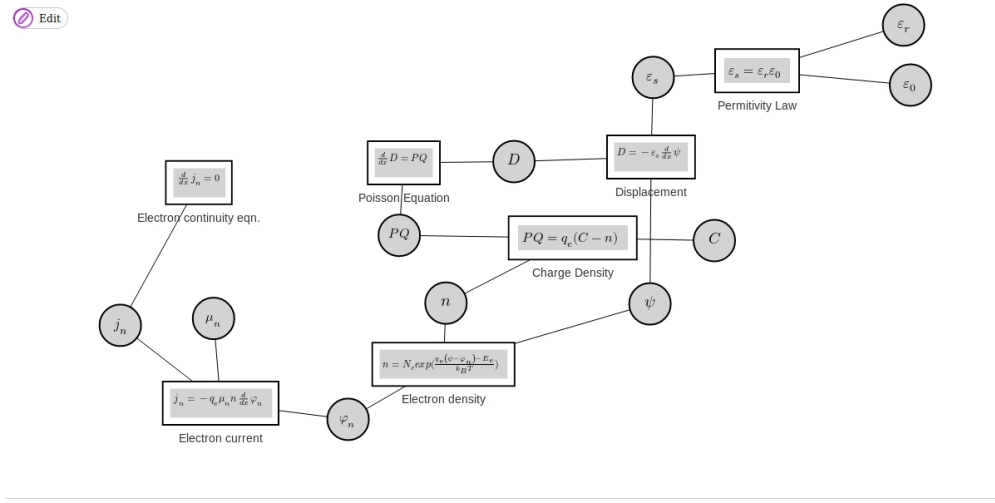


Figure 22: Web viewer interface for model pathway diagrams, source: [Koh+17]

One of the next steps here is the integration into an active document notebook format (cf. section 2.6), such as Jupyter [Jup]. The research on MMT integration has already started and its progress can be followed in the *mmt_jupyter_kernel* repository [MMTJup17].

Another important point to making MOSIS come true is that a mode of storing the ephemeral OMDOC theories can be added to the program in a straightforward manner. And future implementations should also use the explanation capabilities manifested in the flexiformal knowledge, cf. section 4.2.

Finally, it can be said that this thesis presents one approach of making simulations easier to set up. Our vision is useful as well as possible, but currently limited technically. To fully realize the MOSIS architecture, a lot more research will be needed from different disciplines – our exploration showed a plethora of open research questions that are connected, and some of them already ongoing.

We can now help out the engineer from the introductory example in section 1.3 for her first simple problem. There is still some work to do to help her simulate the whole house, and we may likely see the first privately-owned spacecraft going to Mars without the help of our envisioned MOSIS program. While it may not be useful for your first Mars expedition, but maybe for the one after that. And maybe other people who have a mathematical PDE problem would be grateful to have help in solving their own “rocket scientist problem”?

Acknowledgements

Thanks and eternal gratitude go to Michael Kohlhase for talking the author into and helping her through this topic. He has provided a good and open environment for all kinds of ideas.

Thanks also goes to Dennis Müller, who, apart from mathematical discussions and technical help, provided the author with *the* best distractions. ...and of course to Jonas Betzendahl and Tom Wiesing, for making the group a cozy place to be, and helping the author out in manifold ways.

References

- [ansys] *Simulation Software Products / ANSYS*. URL: <http://www.ansys.com/products/all-products> (visited on 10/20/2017).
- [Aus+10] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: <http://www.w3.org/TR/MathML3> (visited on 09/10/2017).
- [BHM00] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.
- [Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: <http://www.openmath.org/standard/om20> (visited on 09/10/2017).
- [Can84] John Rozier Cannon. *The One-Dimensional Heat Equation*. Google-Books-ID: XWSnBZxbz2oC. Cambridge University Press, Dec. 28, 1984. 520 pp. ISBN: 978-0-521-30243-2.
- [Cat+10] David Catalin et al. “eMath 3.0: Building Blocks for a social and semantic Web for online mathematics & ELearning”. In: *1st International Workshop on Mathematics and ICT: Education, Research and Applications*. (Bucharest, Romania, Nov. 3, 2010). Ed. by Ion Mierlus-Mazilu. 2010. URL: <http://kwarc.info/kohlhase/papers/malog10.pdf> (visited on 09/10/2017).
- [cmd216] *cmd2: Quickly build feature-rich and user-friendly interactive command line applications in Python*. Feb. 10, 2016. URL: <https://github.com/python-cmd2/cmd2> (visited on 11/04/2017).
- [CTAN] Michael Kohlhase. *sTeX – An Infrastructure for Semantic Preloading of LaTeX Documents*. URL: <https://www.ctan.org/pkg/stex> (visited on 09/10/2017).
- [DS05] Martin Dürst and Michel Suignard. *Internationalized Resource Identifiers (IRIs)*. RFC 3987. Internet Engineering Task Force (IETF), 2005. URL: <http://www.ietf.org/rfc/rfc3987.txt> (visited on 07/10/2017).
- [Fla17] Ewald Flad. *Automatische Diskretisierung elliptischer partieller Differentialgleichungen in ExaSlang*. B.Sc. Thesis. Supervised by Prof. Dr. Jürgen Teich, Christian Schmitt. 2017.
- [Flo15] Florian Rabe. *The MMT Language and System*. June 2, 2015. URL: <https://github.com/UniFormal/MMT> (visited on 10/24/2017).
- [Fre+] Sebastian Freundt et al. *Symbolic Computation Software Composability Protocol (SCSCP)*. Version 1.3. URL: https://github.com/OpenMath/scscp/blob/master/revisions/SCSCP_1_3.pdf (visited on 08/27/2017).
- [gitGlo] *SMGloM Git Repository*. URL: <http://gl.mathhub.info/smgglom/smgglom> (visited on 10/07/2017).
- [HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. “A framework for defining logics”. In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.
- [i32617] *more expansions, please · Issue #236 · UniFormal/MMT*. GitHub. Oct. 18, 2017. URL: <https://github.com/UniFormal/MMT/issues/236> (visited on 11/08/2017).

- [JE] Florian Rabe. URL: <http://uniformal.github.io/doc/applications/jedit.html> (visited on 05/12/2017).
- [jrh15] jrh13. *hol-light: The HOL Light theorem prover (moved from Google code)*. Oct. 19, 2015. URL: <https://github.com/jrh13/hol-light> (visited on 11/08/2017).
- [Jup] *Project Jupyter*. URL: <http://www.jupyter.org> (visited on 08/22/2017).
- [KA00] Peter Knabner and Lutz Angermann. *Numerik partieller Differentialgleichungen*. DOI: 10.1007/978-3-642-57181-7. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. ISBN: 978-3-540-66231-0 978-3-642-57181-7.
- [KK16] Sebastian Kuckuk and Harald Köstler. “Automatic Generation of Massively Parallel Codes from ExaSlang”. In: *Computation* 4.3 (Aug. 4, 2016), p. 27. ISSN: 2079-3197. DOI: 10.3390/computation4030027. URL: <http://www.mdpi.com/2079-3197/4/3/27> (visited on 10/09/2017).
- [KM] Michael Kohlhase and Dennis Müller. *omdoc/mmt tutorial for mathematicians*. URL: <https://gl.mathhub.info/Teaching/KRMT/blob/master/source/tutorial/mmt-math-tutorial.pdf> (visited on 09/10/2017).
- [Koh+] Michael Kohlhase et al. *A Case study for Active Documents and Formalization in Math Models: The van Roosbroeck Model*. URL: <https://mathhub.info/MitM/models> (visited on 10/05/2017).
- [Koh+11] Michael Kohlhase et al. “The Planetary System: Web 3.0 & Active Documents for STEM”. In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhiro Sato et al. Finalist at the Executable Paper Grand Challenge, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063. URL: <http://kwarc.info/kohlhase/papers/epc11.pdf>.
- [Koh+17] Michael Kohlhase et al. *Mathematical models as research data via flexiformal theory graphs*. WIAS Preprint 2385. 2017. DOI: 10.20347/WIAS.PREPRINT.2385.
- [Koh08] Michael Kohlhase. “Using L^AT_EX as a Semantic Markup Format”. In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: <https://kwarc.info/kohlhase/papers/mcs08-stex.pdf>.
- [Koh13] Michael Kohlhase. “The Flexiformalist Manifesto”. In: *International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. Ed. by Andrei Voronkov et al. Timisoara, Romania: IEEE Press, 2013, pp. 30–36. ISBN: 978-1-4673-5026-6. URL: <http://kwarc.info/kohlhase/papers/synasc13.pdf>.
- [Koh14] Michael Kohlhase. “Mathematical Knowledge Management: Transcending the One-Brain-Barrier with Theory Graphs”. In: *EMS Newsletter* (June 2014), pp. 22–27. URL: <http://www.ems-ph.org/journals/newsletter/pdf/2014-06-92.pdf>.

- [Kro+17] Stefan Kronawitter et al. “A Scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms”. In: *International Journal of Computational Science and Engineering* 14.2 (2017), p. 150. ISSN: 1742-7185, 1742-7193. DOI: 10.1504/IJCSE.2017.10003829. URL: <http://www.inderscience.com/link.php?id=10003829> (visited on 10/09/2017).
- [LeV07] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. DOI: 10.1137/1.9780898717839. Society for Industrial and Applied Mathematics, Jan. 2007. ISBN: 978-0-89871-629-0 978-0-89871-783-9. URL: <http://epubs.siam.org/doi/book/10.1137/1.9780898717839> (visited on 05/19/2017).
- [MAT] *MATLAB - MathWorks*. URL: <https://mathworks.com/products/matlab.html> (visited on 10/20/2017).
- [MATpe] *Partial Differential Equations - MATLAB & Simulink - MathWorks Deutschland*. URL: <https://de.mathworks.com/help/matlab/math/partial-differential-equations.html> (visited on 10/09/2017).
- [Mil] Bruce Miller. *LaTeXML: A L^AT_EX to XML Converter*. URL: <http://dlmf.nist.gov/LaTeXML/> (visited on 09/07/2010).
- [MitM] *MitM/smgglom*. URL: <https://gl.mathhub.info/MitM/smgglom> (visited on 11/01/2017).
- [MMT] Florian Rabe. *The MMT System*. URL: <https://uniformal.github.io/doc/> (visited on 10/16/2017).
- [MMTJup17] Tom Wiesing. *mmt_jupyter_kernel: A Jupyter kernel for MMT*. Oct. 16, 2017. URL: https://github.com/UniFormal/mmt_jupyter_kernel (visited on 11/08/2017).
- [MPDHub] *MPDHub wiki*. URL: <https://github.com/WIAS-BERLIN/MPDHub/wiki/> (visited on 03/22/2017).
- [Mül+17] Dennis Müller et al. “Alignment-based Translations Across Formal Systems Using Interface Theories”. In: *Fifth Workshop on Proof eXchange for Theorem Proving - PxTP 2017*. 2017. URL: <http://jazzpirate.com/Math/AlignmentTranslation.pdf> (visited on 09/30/2017).
- [OMDoc] Michael Kohlhase. *OMDOC: An open markup format for mathematical documents*. Version 1.2. URL: <http://www.omdoc.org/pubs/omdoc1.2.pdf> (visited on 09/10/2017).
- [Pip17] Joachim Piprek. *How to get your simulation paper accepted*. NUSOD Blog. Jan. 4, 2017. URL: <https://nusod.wordpress.com/2017/01/04/how-to-get-your-simulation-paper-accepted/> (visited on 10/24/2017).
- [Rab] Florian Rabe. *The MMT Server*. URL: <http://uniformal.github.io/doc/applications/server.html> (visited on 09/11/2017).
- [RK13] Florian Rabe and Michael Kohlhase. “A Scalable Module System”. In: *Information & Computation* 230 (2013), pp. 1–54. URL: <http://kwarc.info/frabe/Research/mmt.pdf>.
- [Sage] The Sage Developers. *SageMath, the Sage Mathematics Software System*. URL: <http://www.sagemath.org> (visited on 09/30/2016).

- [SMG] *SMGloM Glossary*. URL: <http://mathhub.info/mh/glossary> (visited on 04/21/2014).
- [trans14] *transitions: A lightweight, object-oriented finite state machine implementation in Python*. Oct. 12, 2014. URL: <https://github.com/pytransitions/transitions> (visited on 11/04/2017).
- [unp12] unpingco. *Python-for-Signal-Processing: Notebooks for "Python for Signal Processing" book*. Apr. 18, 2012. URL: <https://github.com/unpingco/Python-for-Signal-Processing> (visited on 11/07/2017).
- [Wol17] Wolfram Research Inc. *Wolfram Mathematica 11*. 2017. URL: <https://www.wolfram.com/mathematica/> (visited on 10/20/2010).
- [WolND] *NDSolve—Wolfram Language Documentation*. URL: <http://reference.wolfram.com/language/ref/NDSolve.html> (visited on 10/20/2017).

A. The Full Theory Graph for PDEs

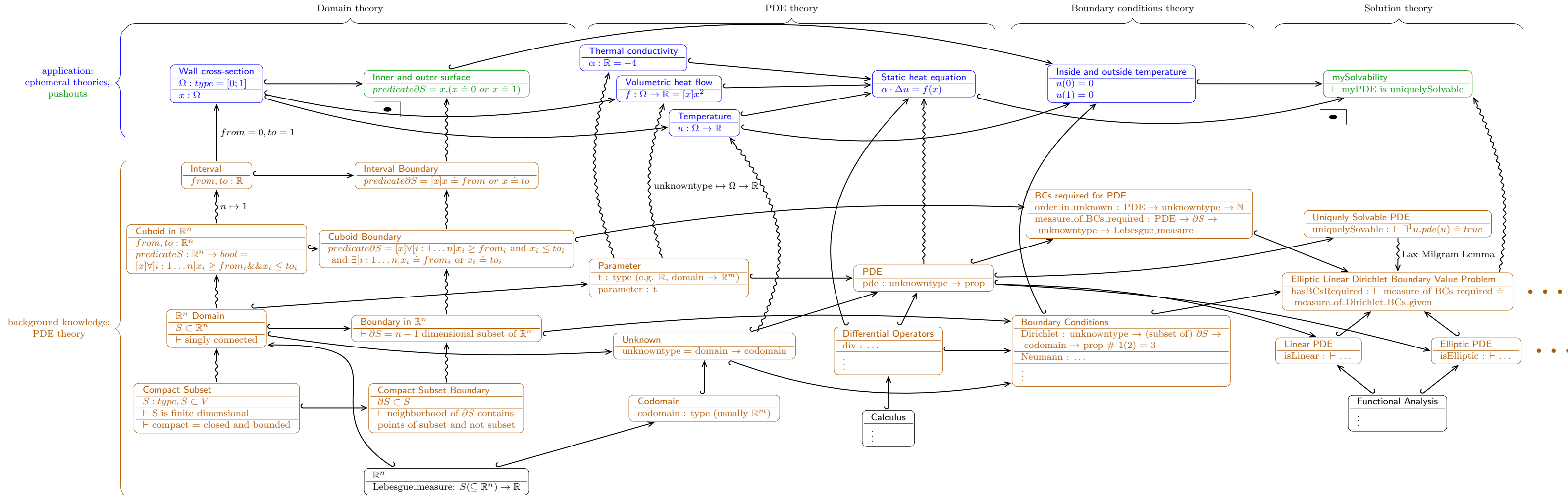


Figure 23: Theory graph for PDEs

B. Formalizations and Code Developed