Examples and Results from a BSc-level Course on Domain Specific Languages of Mathematics

Patrik Jansson
Chalmers Univ. of Technology
patrikj@chalmers.se

Sólrún Halla Einarsdóttir
Chalmers Univ. of Technology
slrn@chalmers.se

Cezar Ionescu
Chalmers Univ. of Technology
cezar@chalmers.se

At the workshop on Trends in Functional Programming in Education (TFPIE) in 2015 Ionescu and Jansson presented the approach underlying the "Domain Specific Languages of Mathematics" (DSLsofMath) course even before the first course instance. We were then encouraged to come back to present our experience and the student results. Now, three years later, we have seen three groups of learners attend the course, and the first two groups have also continued on to take challenging courses in the subsequent year. In this paper we present three examples from the course material to set the scene, and we present an evaluation of the student results showing improvements in the pass rates and grades in later courses.

Keywords: functional programming, computer science education, calculus, didactics, formalisation, correctness, Haskell, types, syntax, semantics, scope

1 Introduction

For the last few years we have been working on the border between education and functional programming research under the common heading of "Domain Specific Languages of Mathematics" (DSLsofMath). This activity started from a desire to improve the mathematical education of computer scientists and the computer science education of mathematicians. In 2014 Ionescu and Jansson applied for a pedagogical project grant to develop a new BSc level course, and from 2016 on the course has been offered to students at Chalmers and University of Gothenburg.

At the workshop on Trends in Functional Programming in Education (TFPIE) in 2015 Ionescu and Jansson [3] presented the approach underlying the DSLsofMath course even before the first course instance. We were then encouraged to come back to present our experience and the student results. Now, three years later, we have seen three groups of learners attend the course, and the first two groups have also continued on to take mathematically challenging compulsory courses in the subsequent year.

The course focus is on types and specifications and on the syntax and semantics of domain specific languages used as tools for thinking (for more details see appendix A). In this paper we present three examples from the course material to set the scene, and we present an evaluation of the student results.

The DSLsofMath activity has also lead to other developments not covered in this paper: presentations at TFPIE 2015, the Workshop on Domain Specific Languages Design and Implementation (DSLDI 2015), IFIP Working Group 2.1 on Algorithmic Languages and Calculi meeting in 2015, and two BSc thesis projects (one in 2016 about Transforms, Signals, and Systems [6] and one in 2018 called "Learn You a Physics"[13] — see appendix B).

To appear in EPTCS.

2 Static checking: Scope and Types in Mathematics

The DSLsofMath lecture notes [4] have evolved from raw text notes for the first instance to 152 pages of PDF generated from literate Haskell + LaTeX sources for the third instance. To give the reader a feeling for the course contents, we will show two smaller and, in the next section, one larger examples from the lecture notes. The two smaller examples are limits and derivatives.

In many of the chapters we start from a textbook definition and "tease it apart" to identify parameters, types, and to help the students understand exactly what it means. When we first presented the course [3], we stressed the importance of syntax and semantics, types and specifications. As the material developed we noticed that variable binding (and scope) is also an important (and in mathematical texts often implicit) ingredient. In our first example here, limits, we show the students that an innocent-looking "if A then B" can actually implicitly bind one of the names occurring in A.

2.1 Case 1: Scoping Mathematics: The limit of a function

This case is from Chapter two of the DSLsofMath lecture notes which talks about the definition (from Adams & Essex [1]) of the limit of a function of type $\mathbb{R} \to \mathbb{R}$:

> We say that $f(x)$ **approaches the limit** L as x **approaches** a, and we write
>
> $$\lim_{x \to a} f(x) = L,$$
>
> if the following condition is satisfied:
> for every number $\varepsilon > 0$ there exists a number $\delta > 0$, possibly depending on ε, such that if $0 < |x - a| < \delta$, then x belongs to the domain of f and
>
> $$|f\ (x) - L| < \varepsilon.$$

The *lim* notation has four components: a variable name x, a point a, an expression $f(x)$, and the limit L. The variable name and the expression can be combined into just the function f and this leaves us with three essential components: a, f, and L. Thus, *lim* can be seen as a ternary (3-argument) predicate which is satisfied if the limit of f exists at a and equals L. If we apply our logic toolbox we can define *lim* starting something like this:

$$lim\ a\ f\ L = \forall\ \varepsilon > 0.\ \exists\ \delta > 0.\ P\ \varepsilon\ \delta$$

It is often useful to introduce a local name (like P here) to help break the definition down into more manageable parts. If we now naively translate the last part we get this "definition" for P:

$$\textbf{where}\ P\ \varepsilon\ \delta = (0 < |x - a| < \delta) \Rightarrow (x \in Dom\ f \wedge |f\ x - L| < \varepsilon))$$

Note that there is a scoping problem: we have f, a, and L from the "call" to *lim* and we have ε and δ from the two quantifiers, but where did x come from? It turns out that the formulation "if ... then ..." hides a quantifier that binds x. Thus we get this definition:

$$\begin{aligned} &lim\ a\ f\ L = \forall\ \varepsilon > 0.\ \exists\ \delta > 0.\ P\ \varepsilon\ \delta \\ &\quad \textbf{where}\ P\ \varepsilon\ \delta = \quad \forall\ x.\ Q\ \varepsilon\ \delta\ x \\ &\qquad\qquad Q\ \varepsilon\ \delta\ x = (0 < |x - a| < \delta) \Rightarrow (x \in Dom\ f \wedge |f\ x - L| < \varepsilon) \end{aligned}$$

The predicate *lim* can be shown to be a partial function of two arguments, f and a. This means that each function f can have *at most* one limit L at a point a. (This is not evident from the definition and proving it is a good exercise.)

2.2 Case 2: Typing Mathematics: derivative of a function

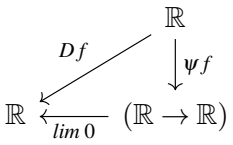
The lecture notes include some other material in between this example and the previous one, but here we jump directly to dissecting one of the classical definitions of the derivative (also from [1]). We now assume limits exist and use *lim* as a function from a and f to L.

> The **derivative** of a function f is another function f' defined by
>
> $$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
>
> at all points x for which the limit exists (i.e., is a finite real number). If $f'(x)$ exists, we say that f is **differentiable** at x.

We can start by assigning types to the expressions in the definition. Let's write X for the domain of f so that we have $f : X \to \mathbb{R}$ and $X \subseteq \mathbb{R}$ (or, equivalently, $X : \mathscr{P}\,\mathbb{R}$). If we denote with Y the subset of X for which f is differentiable we get $f' : Y \to \mathbb{R}$. Thus, the operation which maps f to f' has type $(X \to \mathbb{R}) \to (Y \to \mathbb{R})$. Unfortunately, the only notation for this operation given (implicitly) in the definition is a postfix prime. To make it easier to see we use a prefix D instead and we can thus write $D : (X \to \mathbb{R}) \to (Y \to \mathbb{R})$. We will often assume that $X = Y$ so that we can can see D as preserving the type of its argument.

Now, with the type of D sorted out, we can turn to the actual definition of the function $D\,f$. The definition is given for a fixed (but arbitrary) x. The *lim* expression is using the (anonymous) function $g\ h = \frac{f\,(x+h) - f\,x}{h}$ and that the limit of g is taken at 0. Note that g is defined in the scope of x and that its definition uses x so it can be seen as having x as an implicit, first argument. To be more explicit we write $\varphi\ x\ h = \frac{f\,(x+h) - f\,x}{h}$ and take the limit of $\varphi\ x$ at 0. So, to sum up, $D\,f\ x = lim\ 0\ (\varphi\ x)$. We could go one step further by noting that f is in the scope of φ and used in its definition. Thus the function $\psi\ f\ x\ h = \varphi\ x\ h$, or $\psi\ f = \varphi$, is used. With this notation we obtain a point-free definition that can come in handy: $D\,f = lim\ 0 \circ \psi\ f$. To sum up, here are the steps again, now with typed helpers:

$$\begin{array}{lllll}
D\,f\ x = lim\ 0\ g & \textbf{where} & g\ h = \frac{f\,(x+h) - f\,x}{h}; & g : & \mathbb{R} \to \mathbb{R} \\
D\,f\ x = lim\ 0\ (\varphi\ x) & \textbf{where} & \varphi\ x\ h = \frac{f\,(x+h) - f\,x}{h}; & \varphi : & \mathbb{R} \to \mathbb{R} \to \mathbb{R} \\
D\,f = lim\ 0 \circ \psi\ f & \textbf{where} & \psi\ f\ x\ h = \frac{f\,(x+h) - f\,x}{h}; & \psi : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R} &
\end{array}$$



The key here is that we name, type, and specify the operation of computing the derivative (of a one-argument function). This operation is used quite a bit in the rest of the lecture notes, but here are just a few examples to get used to the notation.

$$\begin{array}{l}
D : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R}) \\
sq\ x \quad\ = x\hat{}2 \\
double\ x = 2 * x \\
c_2\ x \quad\ = 2
\end{array}$$

Then we have the following equalities:

$sq' == D\ sq == D\ (\lambda x \to x\hat{}2) == D\ (\hat{}2) == (2*) == double$
$sq'' == D\ sq' == D\ double == c_2 == const\ 2$

What we cannot do at this stage is to actually *implement D* in Haskell. If we only have a function $f : \mathbb{R} \to \mathbb{R}$ as a "black box" we cannot really compute the actual derivative $f' : \mathbb{R} \to \mathbb{R}$, only numerical approximations. But if we also have access to the "source code" of f, then we can apply the usual rules we have learnt in calculus.

3 Type inference and understanding: Lagrangian case study

The lecture notes goes on through several other definitions related to derivatives, including partial derivatives $D_i = \partial f / \partial x_i$ of type $(\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$. As an example, we can define D_3 in terms of D:

$D_3 f\ (x_1, x_2, x_3) = D\ g\ x_3$
  where $g\ x = f\ (x_1, x_2, x)$ -- $g : \mathbb{R} \to \mathbb{R}$ keeps x_1 and x_2 constant

Our third case study from the lecture notes is the analysis of Lagrangian equations, also studied in Sussman and Wisdom 2013 [10] in their prologue on "Programming and Understanding".

> A mechanical system is described by a Lagrangian function of the system state (time, coordinates, and velocities). A motion of the system is described by a path that gives the coordinates for each moment of time. A path is allowed if and only if it satisfies the Lagrange equations. Traditionally, the Lagrange equations are written
>
> $$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = 0$$
>
> What could this expression possibly mean?

In order to answer Sussman and Wisdom's question, we start by inferring the types of the elements involved:

1. The use of notation for "partial derivative", $\partial L / \partial q$, suggests that L is a function of at least a pair of arguments:

   $$L : \mathbb{R}^i \to \mathbb{R}, i \geqslant 2$$

   This is consistent with the description: "Lagrangian function of the system state (time, coordinates, and velocities)". So, if we let "coordinates" be just one coordinate, we can take $i = 3$:

   $$L : \mathbb{R}^3 \to \mathbb{R}$$

   The "system state" here is a triple (of type $S = (T, Q, V) = \mathbb{R}^3$) and we can call the three components $t : T$ for time, $q : Q$ for coordinate, and $v : V$ for velocity. (We use $T = Q = V = \mathbb{R}$ in this example but it can help the reading to remember the different uses of $\mathbb{R}$.)

2. Looking again at the same derivative, $\partial L/\partial q$ suggests that q is the name of a real variable, one of the three arguments to L. In the context, which we do not have, we would expect to find somewhere the definition of the Lagrangian as

$$\begin{aligned} &L : (T, Q, V) \to \mathbb{R} \\ &L\ (t, q, v) \quad = \ ... \end{aligned}$$

3. therefore, $\partial L/\partial q$ should also be a function of the same triple of arguments:

$$(\partial L \,/\, \partial q) : (T, Q, V) \to \mathbb{R}$$

It follows that the equation expresses a relation between *functions*, therefore the 0 on the right-hand side is *not* the real number 0, but rather the constant function *const* 0:

$$\begin{aligned} &\mathit{const}\ 0 : (T, Q, V) \to \mathbb{R} \\ &\mathit{const}\ 0\ \ (t, q, v) \quad = 0 \end{aligned}$$

4. We now have a problem: $d \,/\, dt$ can only be applied to functions of *one* real argument t, and the result is a function of one real argument:

$$(d \,/\, dt)\,(\partial L \,/\, \partial \dot{q}) : T \to \mathbb{R}$$

Since we subtract from this the function $\partial L/\partial q$, it follows that this, too, must be of type $T \to \mathbb{R}$. But we already typed it as $(T, Q, V) \to \mathbb{R}$, contradiction!

5. The expression $\partial L/\partial \dot{q}$ appears to also be malformed. We would expect a variable name where we find $\dot{q}$, but $\dot{q}$ is the same as dq/dt, a function.

6. Looking back at the description above, we see that the only immediate candidate for an application of d/dt is "a path that gives the coordinates for each moment of time". Thus, the path is a function of time, let us say

$$w : T \to Q \quad \text{-- with } T = \mathbb{R} \text{ for time and } Q = \mathbb{R} \text{ for coordinates } (q : Q)$$

We can now guess that the use of the plural form "equations" might have something to do with the use of "coordinates". In an n-dimensional space, a position is given by n coordinates. A path would then be a function

$$w : T \to Q \quad \text{-- with } Q = \mathbb{R}^{\mathrm{n}}$$

which is equivalent to n functions of type $T \to \mathbb{R}$, each computing one coordinate as a function of time. We would then have an equation for each of them. We will use $n = 1$ for the rest of this example.

7. Now that we have a path, the coordinates at any time are given by the path. And as the time derivative of a coordinate is a velocity, we can actually compute the trajectory of the full system state (T, Q, V) starting from just the path.

$$\begin{aligned} &q : T \to Q \\ &q\ t = w\ t \qquad \text{-- or, equivalently, } q = w \end{aligned}$$

$\dot{q} : T \to V$
$\dot{q}\ t = dw\ /\ dt$ -- or, equivalently, $\dot{q} = D\ w$

We combine these in the "combinator" *expand*, given by

$expand : (T \to Q) \to (T \to (T, Q, V))$
$expand\ w\ t = (t, w\ t, D\ w\ t)$

8. With *expand* in our toolbox we can fix the typing problem in item 4 above. The Lagrangian is a "function of the system state (time, coordinates, and velocities)" and the "expanded path" (*expand w*) computes the state from just the time. By composing them we get a function

   $L \circ (expand\ w) : T \to \mathbb{R}$

   which describes how the Lagrangian would vary over time if the system would evolve according to the path w.
   This particular composition is not used in the equation, but we do have

   $(\partial L\ /\ \partial \dot{q}) \circ (expand\ w) : T \to \mathbb{R}$

   which is used inside $d\ /\ dt$.
9. We now move to using D for $d\ /\ dt$, D_2 for $\partial\ /\ \partial q$, and D_3 for $\partial\ /\ \partial \dot{q}$. In combination with *expand w* we find these type correct combinations for the two terms in the equation:

   $D\ ((D_3\ L) \circ (expand\ w)) : T \to \mathbb{R}$
   $(D_2\ L)\ \circ (expand\ w)\ : T \to \mathbb{R}$

   The equation becomes

   $D\ ((D_3\ L) \circ (expand\ w)) - (D_2\ L) \circ (expand\ w) = const\ 0$

   or, after simplification:

   $D\ (D_3\ L \circ expand\ w) = D_2\ L \circ expand\ w$

   where both sides are functions of type $T \to \mathbb{R}$.
10. "A path is allowed if and only if it satisfies the Lagrange equations" means that this equation is a predicate on paths (for a particular L):

    $Lagrange\ (L, w) = \qquad D\ (D_3\ L \circ expand\ w) == D_2\ L \circ expand\ w$

    where we use $(==)$ to avoid confusion with the equality sign $(=)$ used for the definition of the predicate.

So, we have figured out what the equation "means", in terms of operators we recognise. If we zoom out slightly we see that the quoted text means something like: If we can describe the mechanical system in terms of "a Lagrangian" ($L : S \to \mathbb{R}$), then we can use the equation to check if a particular candidate path $w : T \to \mathbb{R}$ qualifies as a "motion of the system" or not. The unknown of the equation is the path w, and as the equation involves partial derivatives it is an example of a partial differential equation (a PDE).

It is instructive to compare our treatment to Sussman and Wisdom's own answer to their question. Although their approach is very similar in spirit ("we use computer programming in a functional style to encourage clear thinking", [10, page *xv*]), we and our students have found their explanations hard to follow. Their main vehicle for expressing computations, the programming language Scheme, is dynamically typed, and encourages a style in which the typing information is left implicit. As this example illustrates, we have found that having explicit typing is the better choice.

4 Evaluation and results

Now we turn to the empirical evaluation of the course in terms of the students' pass rates and grades in related courses. We considered student results for students from the CSE programme at Chalmers[1] who started their studies in 2014 and 2015. In the spring of their second year at Chalmers (2016 and 2017), these students had the option of either taking the DSLsofMath course or a course on Concurrent Programming (see Table 1).

	Fall	Spring
Year 1	Compulsory courses	Compulsory courses
Year 2	Compulsory courses	DSLsofMath OR ConcProg
Year 3	TSS + Control	...

Table 1: CSE programme structure (simplified)

We considered only "active" students, that is, students who had signed up for at least half of the compulsory courses in the CSE programme during the semesters being considered (Fall 2014 - Fall 2017). This amounted to 145 students, where 53 signed up for the DSLsofMath course (whereas 92 did not) and 34 of those 53 passed the course.

We had access to data on these students' results in all compulsory courses in the programme as well as in the more common elective courses (21 courses in all). At Chalmers students pass a course with a grade of 3, 4, or 5, with 5 being the highest grade, or fail the course with no specified grade. Students usually have two yearly opportunities to retake exams from courses they took in past semesters, to attempt to obtain a passing score or to improve their grades.

The enrolment and results of the DSLsofMath course itself was as follows:

- 2016: 28 students, pass rate: 68%
- 2017: 43 students, pass rate: 58%
- 2018: 39 students, pass rate: 89%

Note that this also counts students from other programmes (mainly SE and Math) while the rest only deals with the CSE students.

4.1 Student results before and after taking DSLsofMath

Our hope was that taking our course would help prepare the students for the math-intensive compulsory courses in the subsequent year with which many students struggle; Transforms, signals and systems (TSS[2]) and Control theory (Control[3])

In Table 2 we see the pass rate and the mean grade (of those who passed) for the above mentioned courses, where PASS represents the students who took the DSLsofMath course and passed it, IN is the group of students who took DSLsofMath (whether or not they passed), and OUT is the students who did not sign up for DSLsofMath.

As we can see, the students who took DSLsofMath had higher mean grades in the third-year courses and were more likely to pass, in particular those who managed to pass the DSLsofMath course. The

[1] Computer Science and Engineering (CSE) is a five-year BSc+MSc programme at Chalmers. It is called "Datateknik" (abbreviated "D") in Swedish.

[2] In Swedish: SSY080 Transformer, signaler och system

[3] In Swedish: ERE103 Reglerteknik

	PASS	IN	OUT
TSS pass rate	77%	57%	36%
TSS mean grade	4.23	4.10	3.58
Control pass rate	68%	45%	40%
Control mean grade	3.91	3.88	3.35

Table 2: Pass rate and mean grade in third year courses for students who took and passed DSLsofMath and those who did not.

correlation between taking, and especially passing, DSLsofMath and success in the third year courses is clear. But perhaps the students who chose to take our course did so because they enjoyed mathematics, and were already more likely to succeed in the subsequent math-heavy courses regardless of whether they took our course. To explore this, we looked at the students' results from their first three semesters at Chalmers, prior to having the option of taking DSLsofMath. We were particularly interested in students' performance in the compulsory mathematics and physics courses (see Table 3, and appendix C).

	PASS	IN	OUT
Pass rate for first 3 semesters	97%	92%	86%
Mean grade for first 3 semesters	3.95	3.81	3.50
Math/physics pass rate	96%	91%	83%
Math/physics mean grade	4.01	3.84	3.55

Table 3: Pass rate and mean grade for courses taken prior to taking (or not taking) DSLsofMath.

Here we can see that there seems to be a small positive bias: those students who choose the DSLsofMath course (the IN group) were a bit more successful in the first three semesters than those in the OUT group. And (not surprisingly) those who pass the course fare even better, on average. Given that many other factors also vary it is not easy to prove that taking DSLsofMath is a significant factor in improving future results, but it does seem likely.

4.2 Students' course assessment and resulting changes

Each course instance has been evaluated with a standard questionnaire sent to all participants as part of the university-wide course evaluation process. The evaluation of the first instance, with Cezar Ionescu as lecturer, identified a need to restructure the initial four-lecture sequence, to re-order a few lectures and to replace the two guest lectures by Linear Algebra.

In preparation for the second instance, with Patrik Jansson as lecturer, the initial lecture sequence was changed to include more Haskell introduction and less formal logic, and two new lectures on Linear Algebra were developed. In the evaluation of the second (2017) instance, the students requested more lecture notes and more weekly exercises to make it easier to get started. The evaluation also indicated that the average student had spent too few hours on the course, and the exam results suffered (42% failed, up from 32% in 2016).

At this point we decided to push for course material development and one of the student evaluators from 2017 (Daniel Heurlin) was hired part time to help out with these improvements. Patrik spent the autumn of 2017 on converting raw text notes and photos of blackboards to LaTeX-based literate Haskell lecture notes covering the full course and Daniel developed more exercises to solve. The primary focus

was on complementing the exam questions from earlier years with easier exercises to start each week with and with more material on functional programming in Haskell.

The recent evaluation of the 2018 instance was overall positive and the course saw a strong improvement in the pass rate: only 11% failed. The student evaluators suggested to "increase the pressure" on solving the exercises, to make the students better prepared for the hand-in assignments and the written exam.

5 Related work

Others have worked on using functional programming to help teach mathematics, in particular algebra, to younger (primary and secondary school) students. The Bootstrap project has successfully developed a functional programming-based curriculum that has improved students' performance in solving algebra problems [8, 9]. In [2], d'Alves et al. describe using functional programming in Elm to introduce algebraic thinking to students.

In [12] and [11], Walck describes using Haskell programming to deepen university students' understanding of physics, and in [7] Ragde describes using functional programming to introduce university students to more precise mathematical notation. We are not aware of previous literature on the use of functional programming to present mathematical analysis as we have done.

6 Conclusions and future work

During the last four years we have developed course material and worked with 150+ computer science students to improve their mathematical education through the course "Domain Specific Languages of Mathematics". We have shown how mathematical concepts like limits, derivatives and Lagrangian equations can be explored and explained using typed functional programming. (Much more about that can be read in the lecture notes [4].) We have investigated the group of students who picked DSLsofMath as an elective and we have measured positive results on later courses with mathematical content.

There are several avenues for future work: upstream and downstream curriculum changes, better tool support, and empirical evaluation.

- Upstream, we would really like to work with the teachers of the mathematics courses in the first year to see if some of the ideas from DSLsofMath could be included already at that stage. Ideally, in the long term, the DSLsofMath course material should be "absorbed" by these earlier courses.

- Downstream, it would be interesting to see how the new course could affect the way the "Transforms, signals and systems" and "Control theory" courses are taught. It seems that we could also affect the Physics course – see the BSc project "Learn You a Physics" summary in appendix B.

- When it comes to tool support it would be interesting to see how proof systems like Liquid Haskell, Agda, etc. could help the students learn. We have been cautious so far, taking it one step at a time, to avoid stressing the student with yet another language / system / tool to learn.

- Finally, we are well aware that our evaluation of the effect of the course on the students' learning is lacking the rigour of a proper empirical study. It would be interesting to work with experts on teaching and learning in higher education on such a study.

Acknowledgements

The support from Chalmers Quality Funding 2015 (Dnr C 2014-1712, based on Swedish Higher Education Authority evaluation results) is gratefully acknowledged. Thanks also to R. Johansson (as Head of Programme in CSE) and P. Ljunglöf (as Vice Head of the CSE Department for BSc and MSc education) who provided continued financial support when the national political winds changed. Thanks to D. Heurlin who provided many helpful comments during his work as a student research assistant in 2017.

This work was partially supported by the projects GRACeFUL (grant agreement No 640954) and CoeGSS (grant agreement No 676547), which have received funding from the European Union's Horizon 2020 research and innovation programme.

A DSLsofMath learning outcomes

- Knowledge and understanding
  - design and implement a DSL for a new domain
  - organize areas of mathematics in DSL terms
  - explain main concepts of analysis, algebra, and lin. alg.
- Skills and abilities
  - develop adequate notation for mathematical concepts
  - perform calculational proofs
  - use power series for solving differential equations
  - use Laplace transforms for solving differential equations
- Judgement and approach
  - discuss and compare different software implementations of mathematical concepts

`https://github.com/DSLsofMath/DSLsofMath/blob/master/Course2018.md`

B BSc project "Learn You a Physics"

The online learning material "Learn You a Physics" (by E. Sjöström, O. Lundström, J. Johansson, B. Werner) is the result of a BSc project at Chalmers (supervised by P. Jansson) where the goal is to create an introductory learning material for physics aimed at programmers with a basic understanding of Haskell. It does this by identifying key areas in physics with a well defined scope, for example dimensional analysis or single particle mechanics, and develops a domain specific language around each of these areas. The implementation of these DSLs are the meat of the learning material with accompanying text to explain every step and how it relates to the physics of that specific area. The text is written in such a way as to be as non-frightening as possible, and to only require a beginner knowledge in Haskell. Inspiration is taken from Learn You a Haskell for Great Good and the project DSLsofMath at Chalmers and University of Gothenburg. The source code and learning material is freely available online.

C Course codes

The courses used to calculate the first 3 semesters pass rate and mean grade are the ones listed above DSLsofMath (DAT326) in Table 4. Of these, the ones used to calculate the Math/physics pass rate and mean grade were TMV210, TMV216, TMV170, MVE055 and TIF085.

Course code		Name
TDA555		Introduction to functional programming
TMV210	M	Introduction to discrete mathematics
EDA452		Introduction to computer engineering
TMV216	M	Linear algebra
DAT043		Object oriented programming
TMV170	M	Calculus
EDA343		Computer communication
EDA481		Machine oriented programming
DAT290		Computer science and engineering project
MVE055	M	Mathematical statistics and discrete mathematics
DAT037		Data structures
TIF085	F	Physics for engineers
DAT326	*	Domain Specific Languages of Mathematics
SSY080		Transforms, signals and systems
ERE103		Control theory

Table 4: Course codes

References

[1] Robert Alexander Adams & Christopher Essex (2010): *Calculus: a complete course*, 7th edition. Pearson Canada.

[2] Curtis d'Alves, Tanya Bouman, Christopher Schankula, Jenell Hogg, Levin Noronha, Emily Horsman, Rumsha Siddiqui & Christopher Kumar Anand (2018): *Using Elm to Introduce Algebraic Thinking to K-8 Students*. In Simon Thompson, editor: *Proceedings Sixth Workshop on Trends in Functional Programming in Education*, *Electronic Proceedings in Theoretical Computer Science* 270, Open Publishing Association, pp. 18–36, doi:10.4204/EPTCS.270.2.

[3] Cezar Ionescu & Patrik Jansson (2016): *Domain-Specific Languages of Mathematics: Presenting Mathematical Analysis Using Functional Programming*. In Jeuring & McCarthy [5], pp. 1–15, doi:10.4204/EPTCS.230.1.

[4] Patrik Jansson & Cezar Ionescu (2018): *Domain Specific Languages of Mathematics: Lecture Notes*. Available from `https://github.com/DSLsofMath/DSLsofMath`.

[5] Johan Jeuring & Jay McCarthy, editors (2016): *Proceedings of the 4th and 5th International Workshop on Trends in Functional Programming in Education, TFPIE 2016, Sophia-Antipolis, France and University of Maryland College Park, USA, 2nd June 2015 and 7th June 2016*. *EPTCS* 230, doi:10.4204/EPTCS.230.

[6] Jacob Jonsson, Peter Ngo, Cecilia Rosvall, Filip Lindahl & Joakim Olsson (2016): *Programmering som undervisningsverktyg för Transformer, signaler och system. Utvecklingen av läromaterialet TSS med DSL*. Technical Report.

[7] Prabhakar Ragde (2013): *Mathematics Is Imprecise*. In Marco T. Morazán & Peter Achten, editors: *Proceedings First International Workshop on Trends in Functional Programming in Education*, *Electronic Proceedings in Theoretical Computer Science* 106, Open Publishing Association, pp. 40–49, doi:10.4204/EPTCS.106.3.

[8] Emmanuel Schanzer, Kathi Fisler & Shriram Krishnamurthi (2018): *Assessing Bootstrap: Algebra Students on Scaffolded and Unscaffolded Word Problems*. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, ACM, New York, NY, USA, pp. 8–13, doi:10.1145/3159450.3159498.

[9] Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi & Matthias Felleisen (2015): *Transferring Skills at Solving Word Problems from Computing to Algebra Through Bootstrap*. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, ACM, New York, NY, USA, pp. 616–621, doi:10.1145/2676723.2677238.

[10] Gerald Jay Sussman & Jack Wisdom (2013): *Functional Differential Geometry*. MIT Press.

[11] Scott N. Walck (2014): *Learn Physics by Programming in Haskell*. In James Caldwell, Philip K. F. Hölzenspies & Peter Achten, editors: *Proceedings 3rd International Workshop on Trends in Functional Programming in Education, TFPIE 2014, Soesterberg, The Netherlands, 25th May 2014.*, *EPTCS* 170, pp. 67–77, doi:10.4204/EPTCS.170.5.

[12] Scott N. Walck (2016): *Learn Quantum Mechanics with Haskell*. In Jeuring & McCarthy [5], pp. 31–46, doi:10.4204/EPTCS.230.3.

[13] Björn Werner, Erik Sjöström, Johan Johansson & Oskar Lundström (2018): *Learn You a Physics for Great Good*. Technical Report. Learning material available from `https://dslsofmath.github.io/BScProj2018/`.