DeduKt

A Mathematical Framework for Symbolic Reasoning

Independent Society of Knowledge

January 31, 2024

Abstract

This proposal aims to provide the necessary information regarding a symbolic computation / proof assitant called DeduKt. We will Introduce our project, discuss the goals, recall some previously design frameworks, explain the functionality and usage of our program, and also talk about the development plan for the DeduKt.

Contents

1 Justifying DeduKt

DeduKt is an open-source mathematical framework, that assists with mathematical work through symbolic computation, interpretive and intelligent proof assitance. The main goals of this project is to design a framework for any kind of mathematical practices, extending it to a point where one would be able to design a new mathematical structure from scratch.

We aim to provide real-time assitance in both proof and symbolic computations using different models and artificial intelligence. Our goal is to design a all-in-one toolkit for mathematical practices in computers, but our initial goal is to develope a symbolic reasoning system for mathematical proof and mathematical computations.

1.1 Mathematica as an Alternative

There are many alternatives in this field, namely, Mathematica, MatheSage, Scipy etc for symbolic computation and Coq, Isabelle, Agda, Arenda etc for proof assistance, however these are not in anyway connected to each other for the benefit of both worlds, Mathematica (as a pioneer in symbolic computation) offers no proof assistance during the works in its notebooks, its main focus is to provide functions and simplification methods for current mathematical task. This in my opinion is a downside for mathematica, beside this the age of this system and since it was build in a totally different era might give us an advantage on starting a new one on our own.

Although they are trying to connect this system with Artificial Intelligence, Building such integration for a system that didn't take it into account in the first place might have some troubles. DeduKt on the other hand is being built with Ai in mind which would make it perfect for integrations of such kind.

1.2 Why Open-Source?

One of the main aspects of this project is to be able to ground it on collaborative code. Open-source projects are normally backed by a grant or some sort of resource for people to be able to spend time on it. For the starting point for DeduKt, we got no grant, but if the project gets a good reputation in scientific society then we would certainly apply for a grant.

An open source software model supports collaboration on Dedukt from the worldwide mathematical and programming communities. Opening elements of the development process, including source code itself, to review and contribution has numerous benefits. This model fosters innovation by mathematicians and developers as expertise from diverse

backgrounds invariably leads to faster progress. Improvements can be contributed by those that specialize in relevant areas, while the community as a whole rapidly surfaces any issues for swift resolution. The freely accessible and adaptable nature of open source code will make Dedukt more flexible and customizable to the evolving needs of mathematical researchers as well. Libraries and knowledge systems often must take on functions unforeseen by original creators. Dedukt's open design and extensible architecture will help avoid obsolescence or limitations imposed by proprietary restrictions.

Lastly, open source provides an avenue to maximize awareness while ensuring this mathematical intelligence system remains freely available for future generations. Visibility of capabilities and code quality for inspection builds trust and interest in the mathematical community critical to Dedukt's success. Licensing it permissively guarantees unfettered ongoing access for mathematicians and developers seeking to utilize leading-edge tools, or push them further via open enhancements.

A community focused on openness and scientific advancement will prove key to rapid advancement and sustained viability moving forward as computational mathematics grows ever more capable and vital for 21st century research breakthroughs. Embracing collaborative open source development principles positions this project to achieve its ambitious goals of crafting the preeminent intelligent software to empower mathematical practitioners.

1.3 What we mean By intelligent proof assiting

Proof assistants such as Coq or Isabelle have demonstrated immense value in constructing reliable formal proofs. However, these tools function primarily as verification systems - able to check proofs supplied to them rather than actively participate in proof generation and discovery. The mathematician or programmer must determine relevancy of other proven lemmas and construct an entire skeleton of reasoning before a proof assistant can approve it as logically valid.

Dedukt aims to take computational mathematical collaboration to a new level with built-in intelligence to aid users more extensively throughout the proof development lifecycle. Going beyond passive checking, Dedukt seeks to guide the user interactively. When a proof goal gets troublesome or stalled, Dedukt will suggest relevant lemmas and theorems to consider based on contextual understanding of the target domain.

By actively participating alongside the human to overcome sticking points with automated recommendations, Dedukt can lessen formal proof burdens. The proposed semantic enrichment and inference techniques strive to enable more flexibility for mathematicians while retaining the rigor of computer-verified results - getting them unstuck faster while avoiding logical errors through co-reasoning with this intelligent assistant. This would also come in handy with simplification and symbolic computations, suppose that one writes a formula or an equation which is already known by the system, then based on few hints the system would provide some later steps that the researcher/user would use to get different results.

2 Current Frameworks

Currently, robust proof assistants and symbolic computation systems occupy largely separate domains with little synergy between capabilities. Tools like Coq excel at formal verification of mathematical correctness, while systems like Mathematica or Maple focus on algebraic transformations, solutions, and visualizations without regard for logical soundness. As far as we are aware, there are presently no programs that function simultaneously for both computer-aided proof development and symbolic manipulation. This bifurcation is needless and hampers more advanced computational mathematics that tightly integrates logic and calculation.

The vision for Dedukt is to bridge this gap by supplying an intelligent framework adept in both formal reasoning and symbolic computation realms. Having both capabilities deeply integrated and callable from a single usable interface offers mathematicians immense power within one unified toolkit. Dedukt intends to pioneer this unification - opening the door to computer-assisted derivation of complex results that leverage interchangeably: manual proof steps, automated proof recommendations, algebraic substitutions, calculus manipulations, visual plotting of functions, and more. Tight integration facilitates using the right technique at the right time while maintaining soundness and mathematical intuitiveness. Breaking down barriers between isolated islands of specialized computational mathematics, Dedukt aims to set a new standard for proof and symbolism synergy.

2.1 Mathematica and Wolfram Language

Mathematica is a computational software program used for mathematical computing. It can handle numerical, symbolic, and graphical computations. Mathematica is widely used across science, engineering, mathematics, finance, and other quantitative fields. This in my opinion would be the main opponent to DeduKt, since other applications and softwares are not as userfriendly and as powerful as this one.

The Wolfram Language allowed Stephen Wolfram and his team to build up Mathematica's vast functional capabilities over decades. The language makes it simpler to represent complex mathematical concepts and relations compared to general-purpose programming languages. Mathematica leverages the language to implement its advanced computation abilities.

Some key advantages are:

1. Vast built-in capabilities covering numerous scientific/quantitative domains
2. Powerful symbolic and numerical math engine out-of-the-box
3. High-level Wolfram Language allows easy representation of mathematical concepts.
4. Interoperability with other systems and file formats.

Some disadvantages are:

1. Closed source, proprietary system controlled by Wolfram Research.

2. Wolfram Language has a steep learning curve

3. Computation speed lacks at times compared to lower-level programs.

4. Expensive licensing model

2.2 Coq, Agda and other proof assistants

Coq and Agda are powerful interactive theorem provers that can assist with developing mathematical proofs. Both Coq and Agda allow formally specifying mathematical definitions and properties in their own functional programming languages. These are based on intuitive logic and type theory foundations. They assist proving theorems about those formal specifications by interactively applying reduction rules and inference principles to construct proof terms. The system tracks the proof state to ensure correctness.

However, Coq and Agda have some key differences from what we aim to build: Firstly, They operate by compiling entire proofs - stepping through the full formal deductions from axioms to final results. Our project focuses more on interpreting and providing hints in real-time during proof writing. Their proof assistance is limited to checking logical validity of constructs. We want to incorporate broader mathematical intuition - e.g. providing relevant analogies when stuck on a step. The formal representations can have a steep learning curve. We want to support a more natural mathematical workflow - parsing and understanding proofs written in usual mathematical style.

Our project focuses more on assisting human intuition compared to verifying computer programs. The real-time deductions aim to be collaborative rather than authoritative.

3 Development Plan

3.1 Symbolic Computation and LaTeX

One barrier to the broader adoption of advanced computational mathematics has been usability issues stemming from reliance on esoteric programming languages and interfaces. For mathematicians uninitiated in software development, steep learning curves have posed obstacles in leveraging symbolic systems effectively. Dedukt aims to close this expertise gap through an intuitive LaTeX-based input language. This allows encoding expressions and equations in familiar mathematical notation, rather than requiring strict language syntax proficiency.

Such plug-and-play accessibility opens the door for more mathematical practitioners to benefit from dedicated symbolic capabilities planned for Dedukt. Integrated computer algebra functionality such as algebraic substitutions, calculus manipulations, equation solving and visual plotting can be accessed by students and researchers alike through the accessible LaTeX input format. Understanding and translating needs into the appropriate algorithms behind-the-scenes will be handled by the software, eliminating coding bottlenecks. Output and visualizations of worked solutions will likewise render programmatically in LaTeX for natural interpretation.

By meeting mathematicians in a comfortable user experience paradigm and handling the programming complexity internally, Dedukt makes its power more accessible to the target community. Built-in symbolic systems enrich the toolkit for interactive proof development and exploration without requiring extensive retraining. The planned capabilities aim to consistently balance mathematical rigor with usability - putting innovative tools in the hands of those needing them regardless of programming proficiency. This theme of inclusion and empowerment though LaTeX integration underpins the symbolic computation vision for Dedukt.

3.2 Computer Algebra

Computer algebra is the study and development of algorithms for manipulating mathematical expressions using computers. It is rather different than scientific computing in the sense that the latter mostly considers numerically methods and derivations for given expressions.

One of the most important part of such softwares is to provide simplification, which one can argue is everything that these softwares does. Using rewriting rules, the software provide an alternative expression (usually simpler than the initial expression) to the user. If we use the term simplifying for only its real meaning then our software needs to distinguish between simplification and everything else (computations). For example a derivative of the form:

$$\frac{d}{dx}\left(x^2+\ln x\right) \tag{1}$$

can be regarded simple by the user, or he/she perhaps think of:

$$2x+\frac{1}{x} \tag{2}$$

as simpler than the first one since the derivation is applied. We go even further and write:

$$\frac{2x^2+1}{x} \tag{3}$$

as the simplest alternative expression.

In developing these kinds of functions (simplification) we have to use an algorithm which would do the task, (1) efficiently, and (2) halts at a state for our expression.

What I would consider for such functionality is to apply a sort of scoring system, based on number of terms and operations (derivation, integration, power, etc), each simplification would be consist of a graph of alternative expressions, where going from one node to the other would add or subtract a number from your initial score, Thus higher score would count as better simplification. Note that this would give us a function that might take a path which initially gets a lower score and later becomes the best result.

As an example let us look at the following expression:

$$\frac{1}{3+2i} \tag{4}$$

where i is the imaginary number. If we (the human) would be asked to simple such a task we would consider to write the imaginary number in the natural form of imaginary numbers:

$$a+ib \tag{5}$$

to do so we have to multiply the expression by a term which would look like:

$$\frac{1}{3+i2} \times \frac{3-2i}{3-2i} \tag{6}$$

If we ask the algorithm to score this expression with respect to the first one, we would expect to get a worse score than we initially had. But if the alternative expression goes further:

$$\frac{3-2i}{5} = \frac{(3-i2)}{5} \tag{7}$$

Since the expression this expression is more acceptable than the previous one (the imaginary number is in its most common form) therefore, we have a higher score at the end.

3.3 Markov Chain

A Markov chain or Markov process is a statistical model that describes a sequence of possible state changes where the probability of each state transition depends only on the current state, not on the sequence of states that preceded it. This "memoryless" property is described by the phrase "the future is conditionally independent of the past given the present."

Markov chains model real-world processes that jump from one state to another in a random yet structured way over a sequence of discrete time steps (discrete-time Markov chain) or continuously over time (continuous-time Markov chain). They are named after Russian mathematician Andrey Markov.

Some applications of Markov chains include modeling vehicle cruise control systems, customer queues, exchange rates, animal populations, and more. They serve as the foundation for Monte Carlo simulation methods used widely across fields like statistics, physics, chemistry, economics, and signal processing for sampling from complex distributions.

Each mathematical expression forms a node in a directed graph. The expressions are connected by edges to alternative semantically equivalent expressions. The edges are weighted with a simplicity score between -1 and 1, indicating how much simpler or more complex the connected expression is compared to the original.

When a simplification process is initiated, the software traces possible paths through the graph by following edges to expressions with higher simplicity scores. The process halts when it reaches expressions with maximal simplicity scores - i.e. the simplest semantically equivalent expressions.

To handle cases where the graph contains cycles (expressions linking back to themselves) or infinite recursion, the software tracks all unique expressions explored within a defined computational budget. If a definitive simplest form is not found within the budget, the software returns all the simplest unique expressions discovered so the user can select the most appropriate.

The Markov chain graph allows formally encoding the space of mathematical expressions and relationships between them. By defining simplicity metrics on edges, the graph can be efficiently traversed to find optimized, simplified expressions. Software can automate simplification by leveraging the structure while allowing for user guidance in difficult cases involving recursion or ambiguity.

A Markov chain graph of mathematical expressions could also be leveraged to identify efficient computation paths:

In addition to a simplicity score, the edges between expression nodes can be assigned a computational efficiency score. This score reflects attributes such as the number of arithmetic operations, use of intermediate variables, parallelizability, etc. required to compute one expression from another.

When faced with a complex computation, the software could then traverse the graph to find paths leading to expressions which are efficient to evaluate in code. Much like with simplification, the search through the Markov graph can halt when expressions with maximal efficiency scores are found.

The structure allows encoding domain-specific computational knowledge so the software can learn the most performant ways of carrying out math operations. For example, the graph could represent how a triple integral can be decomposed into iterated integrals in multiple valid sequences, each with different complexities. Exploring the edges would reveal the most numerically stable and parallelizable way to transcribe the integral into code.

By combining both simplicity and efficiency scoring, the Markov chain graph creates a knowledge structure allowing software to uncover elegant and performant mathematical implementations tailored to the problem context. It can serve as a bridge between mathematical theory and computational practice. The software learns to align symbolic expressions with real-world computing constraints.