Towards computer-assisted semantic markup of mathematical documents

Year 1 progression talk

Luka Vrečar

Heriot-Watt University

June 13, 2023

Introduction

- The language of mathematics (LoM) is the language used when producing mathematical documents
- Many attempts at *computerising* it
- Full of ambiguities!
- We usually typeset $P \times Q$ using `$P \times Q$`
  - Is this a matrix product or a Cartesian product?
  - We can use sTEX — a LaTeX macro package for semantic markup — instead
  - `$\cart{P, Q}$` vs. `$\matrixtimes[x]{P, Q}$`
- I hope to develop ways of automatically adding sTEX
  - This can improve learning materials as well

The language of mathematics (LoM)

- The language used in mathematical documents
- In depth analysis by Ganesalingam and Iancu
- I will focus on the symbolic part — it has a clearer structure, and precise definitions so I'm hoping it will be easier to work with

Attempts at computerising the language of mathematics

Representations of LoM

- Controlled natural languages (CNLs) — MathNat, ForTheL
- Discourse representation structures (DRS) — used in Ganesalingam's work

Software making use of these representations

- Proof assistants — Isabelle, Mizar, Coq, Lean, ...
- Proof checkers — Naproche, Naproche-SAD, ...

Machine learning (ML)

- Could be used for processing text, lots of recent progress in natural language processing
- There have been attempts at using ML for working with LoM
  - Müller and Kaliszyk tried to disambiguate symbolic formulas and using sTeX to represent output
  - Pagel and Schubotz tried to infer the meaning of symbols in a formula using surrounding text
  - Hutterer tried generating suggestions for what sTeX macros to use when typesetting formulas
- I lack experience with ML and would need to spend time learning it
- ML is probability-based so unsuitable on its own (even educated guesses aren't enough for mathematical truths)
- Could be useful for generating suggestions for the user

Semantic markup

- Encoding the meaning of objects into the objects themselves
- Recall $P \times Q$ — `$P \times Q$`
- Meaning encoded within the formula — `$\cart{P, Q}$`

In LaTeX we can do this using sTeX

sTeX — about

- A package for defining semantic macros in LaTeX
- First released in 2008, but it was hard to use
  - Difficult setup
  - Complex internal dependency structure
- Major rework in 2022 to fix these issues
- Includes the Semantic Multilingual Glossary of Mathematics (SMGloM) — a library of macros for many areas of mathematics

sTEX — usage

Defining new macros

- We can write $a \times b$ as `$a \times b$`, but this can be ambiguous!
- Instead we can define an sTEX macro — `\symdef{realmult}[args=2]{#1 \times #2}`
- Now we can write $a \times b$ as `$\realmult{a}{b}$`

However...

sTeX — usage

This is not practical!

- What if we wish to write $a \times b \times c \times d$? Instead of using many `\realmult` calls, we use *flexary arguments*
- Redefine the macro to
  `\symdef{realmult}[args=a]{\argsep{#1}{\times}}`
- Now we can write $a \times b \times c \times d$ as `$\realmult{a,b,c,d}$`

Other features

- Defining new notations —
  `\notation{realmult}[dot]{\argsep{#1}{\cdot}}`
  Using `$\realmult[dot]{a,b,c,d}$` produces $a \cdot b \cdot c \cdot d$

The λ-calculus — introduction

We use it as a testing ground for implementations

- Developed by Alonzo Church in the 1930s
- Turing-complete model of computation
- Relatively simple — only 3 “building blocks”
- This will not be a formal introduction

The λ-calculus — basic definition

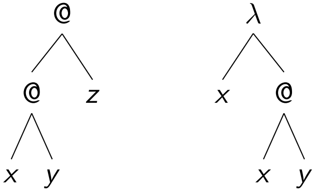
- We use x, y, z to range over $\mathcal{V} ::= v \mid x'$
- $\mathcal{V}$ is countably infinite
- We use A, B, C to range over $\Lambda ::= x \mid AB \mid \lambda x.A$
- Each term is either a variable, application, or abstraction
- $(\lambda x.(xy))$, z, and $((xy)z)$ are all terms
- We use brackets to disambiguate the structure, but that can be difficult to read

The λ-calculus — notational conventions

Notational conventions

- Parentheses around application can be dropped — $(AB) = AB$
- Application is left-associative — $((AB)C) = ABC$
- The scope of an abstraction extends as far to the right as possible
- Multiple “nested” abstractions can be written with just one λ and dot — $(\lambda x.(\lambda y.(\lambda z.(A)))) = (\lambda xyz.(A))$

The terms xyz and $\lambda x.xy$ as trees



The λ-calculus — operations

We define two operations here (there are more)

- Substitution — $A[x := B]$ replaces all free occurrences of x in A by B
  - Example: $(\lambda x.(xy))[y := z]$ gives $(\lambda x.(xz))$
- β-reduction — represents "computation"
- $(\lambda x.(A))B \to A[x := B]$
  - Example: $(\lambda x.(xy))z \to (xy)[x := z]$ which gives (zy)

An sTeX module for the λ-calculus

Motivation

- sTeX macros for the λ-calculus did not exist before
- Potentially useful for the Foundations 1 course (a course exclusively about the λ-calculus)
- Learning experience for using sTeX

An sTeX module for the λ-calculus — implementation

Implementation

- I implemented 3 macros: `\app`, `\abs`, `\var`
- Needed support for notational conventions
- I added 9 notations — some of them are used in tandem

`\app[nb]{A}{B}`	AB
`\abs[nb]{x}{A}`	$\lambda x.A$
`\abs[nob]{x}{A}`	$\lambda x.(A)$
`\abs[nib]{x}{A}`	$(\lambda x.A)$
`\abs[nb-nodot]{x}{A}`	λxA
`\abs[nib-nodot]{x}{A}`	(λxA)
`\abs[nested]{x}{A}`	$x.A$
`\abs[nested-nodot]{x}{A}`	xA
`\abs[full]{x}{A}`	$(\lambda x.(A))$

`\abs[nb-nodot]{x}{\abs[nested-nodot]{y}{\abs[nested]{z}{A}}}`
to typeset $\lambda xyz.A$

An sTEX module for the λ-calculus — redesign

There were issues

- It was impractical — many notations, needed 3 different ones for typesetting $\lambda xyz.A$
- `\abs[nested-nodot]{x}{y}` and `\app[nb]{x}{y}` look identical when typeset — could be ambiguous

I started redesigning the macros for better usability

- `\abs` now uses flexary arguments — `\abs{x, y, z}{A}`
- Unresolved issues with balanced brackets when using the "full notation" e.g., $(\lambda x.(\lambda y.(\lambda z.(A))))$
- Unfinished, but plan to finalize the changes soon

A motivating example

Suppose you're writing a document on the λ-calculus

- Automatic removing of brackets, showing steps in substitutions and β-reductions
- Having it done automatically with macros inside LaTeX would be convenient

Example

`\removeBrackets{`$(\lambda x.(xy))$`}`	$\lambda x.xy$
`\betaReduce{`$(\lambda yz.z(yz))(\lambda x.x)$`}`	$(\lambda yz.z)(yz)(\lambda x.x) \equiv_\beta$ $\lambda z.z((\lambda x.x)z) \equiv_\beta$ $\lambda z.z(\lambda x.x)z \equiv_\beta$ $\lambda z.zz$

I created something similar — `lambda-calculus.lua`

`lambda-calculus.lua` — implementation

Implemented in Lua — easy interface with LaTeX via macros

Current functionality

- Converting sTeX macros to trees
  - Operating on trees is easier than on strings of macros
- Performing substitution and β-reduction on the trees
  - Substitution returns just the end result of the computation
  - β-reductions returns a list of all the steps
- Applying notational conventions
  - This always tries to minimize the number of brackets that are needed and "compresses" nested abstractions

Note: it needs to be updated due to the λ-calculus macro redesign and sTeX updates

`lambda-calculus.lua` — discussion

- Requires the use of sTeX macros
- Showcases advantage of using semantic macros in documents
- Potential application in the Foundations 1 course
- Substitution and notational conventions aren't applied step by step (just the end result is returned)
- Upgrades — more β-reduction strategies, de Bruijn indices

A grammar for parsing human-written λ-calculus

Motivation

- sTeX is required to use `lambda-calculus.lua`
- Manual conversion of formulas to sTeX macros takes time
- Automated parsing might be a solution

Found `pyparsing`, a parsing library for Python

- Comes with its own grammar syntax and parser
- I'm profficient in Python — easier to build stuff with `pyparsing`

A grammar for parsing human-written λ-calculus

- Created a grammar that can parse human-written λ-terms
- Proof of concept, only works in a controlled environment
- Used it on small documents — created modified copies with "sTeX-ified" formulas

$$
\begin{aligned}
\texttt{lexp} &\rightarrow \texttt{app} \mid \texttt{term} \\
\texttt{app} &\rightarrow \texttt{term}\ \texttt{term} \\
\texttt{term} &\rightarrow \texttt{var} \mid \texttt{abs} \mid \texttt{parexp} \\
\texttt{abs} &\rightarrow \texttt{lam}\ \texttt{binder}\ \texttt{.}\ \texttt{lexp} \\
\texttt{lam} &\rightarrow \lambda \mid \texttt{\textbackslash lambda} \\
\texttt{binder} &\rightarrow \texttt{binder}\ \texttt{var} \mid \texttt{var} \\
\texttt{var} &\rightarrow \texttt{[a-z]}\,\texttt{[0-9']}^{*} \\
\texttt{parexp} &\rightarrow (\ \texttt{lexp}\ )
\end{aligned}
$$

A grammar for parsing human-written λ-calculus

Discussion

- Creating the grammar took a long time
- It didn't accurately reflect the typesetting of a term, only its structure
- It only worked in a controlled environment (no subscripts in variable names, or handling of typesetting-only LaTeX code)

Not used in future work, also due to `pyparsing` drawbacks

Switch from `pyparsing`

`pyparsing` proved inadequate for several reasons

- Not exhaustive — not good for ambiguous grammars
- No backtracking — parsing success depends on the order in which rules are applied

I searched for another Python library so I could reuse some code

- Focus on GLR-based libraries — exhaustive parsing
- Tried `GLRParser`
  - No extensive documentation
  - Terminals could not be capitalised (which is needed in mathematics)
- Found `parglare`

parglare

- Supports LR and GLR parsing
- Comes with a grammar syntax with potentially useful extra functionality such as rule priorities and attaching meta-data
- Ignores whitespace in input sentences — like TEX's math mode
- Extensive documentation and still updated

Made the switch when working on automatic grammar generation

Automatic grammar generation

Motivation

- Manual creation takes lots of time
- Lots of sTeX macros already exist to represent output

Current idea

- Create a grammar from sTeX macros
- Parse documents with it, produce sTeX equivalents of formulas

Neither of these ideas are final and might change

Automatic grammar generation — implementation

Supporting infrastructure

- Extracting formulas from `.tex` files
  - Currently only $s, will add support for other delimiters in the future
- Finding macro definitions in an sTeX module
  - Currently focused on sTeX definitions, support for user-defined macros is planned for the future
- `Definition` objects to store information
  - Used for storing information about definitions extracted from macros
  - Keeps track of the name, any notations, stores the LaTeX source code and the grammar rules generated from it

Automatic grammar generation — implementation

Rule generation

- Each argument placeholder is replaced by `arg`
- Each `\argsep` is turned into a separate rule of the form
  `exlist` $\rightarrow$ `ex exlist` $|$ `ex`

Example:

```
\symdef{abs}[args=ai]{\lambda \argsep{#1}{} . #2}
```

generates the following rules:

```
abs → "\lambda" abs1list "." arg
abs1list → abs1 abs1list | abs1
```

Automatic grammar generation — implementation

Grammar generation and parsing

- Rules are converted to `parglare` syntax
- A rule is added for `arg` — each non-terminal can be on the right side (e.g., `arg` $\rightarrow$ `abs`)
- Rules with just one `arg` are converted into regular expressions instead
- Parsing small example sentences and finding all parses

Automatic grammar generation — discussion

Drawbacks

- The grammars overgenerate — *xyzw* generates 8 parse trees, but only 1 is correct
- The `arg` rule is too broad — not everything can be an argument to everything else
- Regular expressions are temporary and unsuitable for arbitrary LaTeX
- Potentially we could use more information provided by macro definitions (types, precedence, associativity)

In the future we will likely need to work with a TeX parser to handle user-defined macros and other math-mode delimiters

Future work

- Finishing the rework of the sTeX module for the λ-calculus
- Updating `lambda-calculus.lua` to work with the new sTeX module
- Creating a demonstration for interested users
- Figuring out improvements for all the mentioned drawbacks

Questions