Towards computer-assisted semantic markup of mathematical documents

CICM 2024, Montreal, Canada

Luka Vrečar, Joe Wells, Fairouz Kamareddine

Heriot-Watt University

August 8, 2024

Outline

- **Introduction**
- Background – λ-calculus (our testing ground) and sTeX
- Grammar generation
- Disambiguation GUI – includes a demo
- Conclusion and future

Introduction

- Documents written in LaTeX often contain ambiguous formulas (e.g., `$P \times Q$`).
- We can disambiguate them with sTeX (e.g., `$\cart{P}{Q}$`).
    - Other advantages - interaction with computer algebra systems, interactive theorem provers, screen readers, etc.
- Semantic markup via sTeX ("sTeX-ification") is more involved, so we hope to somewhat automate the process.

Proposed approach

For a given document we wish to sTEX-ify:

1. Generate all the prerequisites
   1.1 Identify which macros are needed and define any missing ones.
   1.2 Generate a context-free grammar.
2. Produce semantic markup
   2.1 Parse all the formulas in the document with the grammar from step 1b.
   2.2 Disambiguate any ambiguous parses with a graphical user interface (GUI).
   2.3 Create a copy of the original document, with formulas replaced by their sTEX counterparts.

Outline

- Introduction
- **Background – λ-calculus (our testing ground) and sTeX**
- Grammar generation
- Disambiguation GUI – includes a demo
- Conclusion and future work

Background – λ-calculus (variables)

Let $\mathcal{V}$ be the set of variables, defined as $\mathcal{V} = \{\mathbf{v}, \mathbf{v'}, \mathbf{v''}, \ldots\}$. We will denote the *meta-variables* that range over $\mathcal{V}$ with lowercase letters (e.g., x, y, z), that can have apostrophes or subscripted index attached (e.g., x', y_1, z''_2).

Background – λ-calculus (terms)

Let Λ be the set of all λ-*terms*. We will denote the *meta-terms* that range over Λ with uppercase letters (e.g., A, B, C), that can have apostrophes or subscripted index attached (e.g., A', B_1, C_2''). We define Λ inductively as follows:

- If $x \in \mathcal{V}$, then x is in Λ.
- If $A, B \in \Lambda$, then the *application of* A *to* B, denoted by (AB), is in Λ.
- If $x \in \mathcal{V}$ and $A \in \Lambda$, then the *abstraction in* A *over* x, denoted by $(\lambda x.A)$, is in Λ.

Background – λ-calculus (notational conventions)

We employ some notational conventions when writing out λ-*terms*.
We follow the conventions from our Foundations course notes:

- We can remove the outermost parentheses in a term: we can write AB instead of (AB).
- Application is left-associative: we can write $(AB)C$ as ABC.
- The scope of an abstraction extends as far to the right as possible: $\lambda x.xy$ is equivalent to $\lambda x.(xy)$, NOT $(\lambda x.x)y$.
- Multiple consecutive abstractions can be "compressed": we can write $\lambda x.(\lambda y.(\lambda z.A))$ as $\lambda xyz.A$.

Background – sTeX

- Developed by the KWARC research group
- Semantic macros to preserve structure and meaning of formulas
  - Still human readable when compiled to PDF
- Hundreds of macros already exist for mathematics and CS

Macros for λ-*terms*

- `\symdef{var}[name=variable, args=i]{#1}`
- `\symdef{abs}[name=abstraction, args=ai]{`
  `\maincomp{\lambda}\argsep{#1}{}\comp{.}#2}`
- `\symdef{app}[name=application, args=ii]{#1 #2}`

Outline

- Introduction
- Background – λ-calculus (our testing ground) and sTeX
- **Grammar generation**
- Disambiguation GUI – includes a demo
- Conclusion and future work

Grammar generation - initial approach

1. Find sTeX macro definitions and replace argument placeholders with a special nonterminal, `arg`.
2. Create a main rule, with `arg` on the LHS and all other nonterminals on the RHS.
3. Add a simple text-recognizing regex if all else fails

Macro definition	**Grammar rule**
`\symdef{var}[args=1]{ #1}`	var → arg
`\symdef{app}[args=2]{ #1 #2}`	app → arg arg
`\symdef{abs}[args=2]{ \lambda#1.#2}`	abs → "\lambda" arg "dot" arg
Main rule	arg → var | app | abs | [a-z]+?

Grammar generation - issues with the initial approach

- The grammars would *over-generate*, i.e., they produced many non-sensical trees
- Assuming anything can be an argument to any macro does not make sense mathematically
  - For abstraction for example, the first argument should only be a variable

Grammar generation - adding types

- Some sTeX macro definitions also contain *types*
- `\symdef{natplus}[args=2, type=\funspace{\Nat, \Nat}{\Nat}]{#1 + #2}`
- This macro has type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ - it takes in two natural numbers (*input types*) and returns a natural number (*output type*)
- We can restrict grammar rules by matching output types with arguments of the correct input type for each notation rule

$$\begin{aligned}
\texttt{natplus} &\to \texttt{natArg1} \;+\; \texttt{natArg2} \\
\texttt{natArg1} &\to \texttt{natType} \\
\texttt{natArg2} &\to \texttt{natType} \\
\texttt{natType} &\to \texttt{natplus} \mid \ldots
\end{aligned}$$

Grammar generation - adding types

- Not a lot of macros actually provide types, so we need a different solution
- Possibly, we can create an interface for editing grammars where users can select which macros can be arguments to other macros
- In this way we add types to macros in a more "loose" sense

Grammar generation - adding precedence

- In sTeX, we can add precedence to macros for things like automated bracketing
- We can use them as precedences during parsing

Grammar generation - issues and improvements

- Grammars sometimes contain cycles, which our GLR parser cannot work with
  - We can address this with a different parser, like DynGenPar
- There is currently no way to generate a grammar from more than one sTEX archive at a time - addressed in future work
- Grammars must sometimes be manually edited
  - Improving the code might solve this to some extent
  - Developing an interface for creating/merging/editing grammars will also help

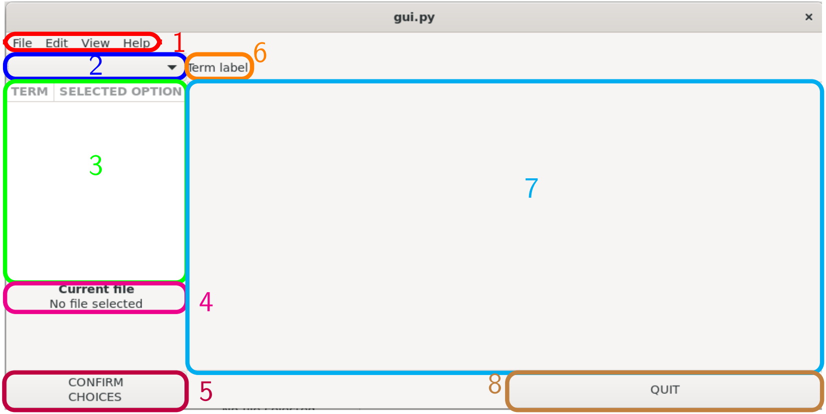
Outline

- Introduction
- Background – λ-calculus (our testing ground) and sTEX
- Grammar generation
- **Disambiguation GUI – includes a demo**
- Conclusion and future work

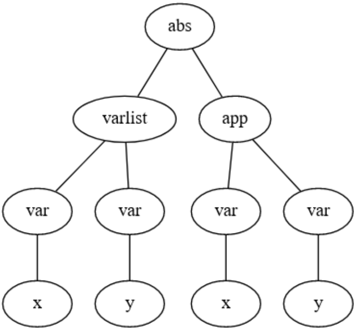
A GUI for disambiguation during parsing - motivation

- Formulas may parse ambiguously, and comparing terminal printouts is not easy
- We can visualise all parses side by side in a nicer way
- This tool can then evolve into a program for all steps of sTeX-ification, from grammar generation to producing the actual sTeX-ified documents

A GUI for disambiguation during parsing - design



A GUI for disambiguation during parsing - tree visualisation




```
abs            abs
  varlist        varlist
    var            var
      x              x
    var            var
      y              y
  app            app
    var            var
      x              x
    var            var
      y              y
```

A GUI for disambiguation during parsing - example

I will now show the GUI in practice on a small example file

A GUI for disambiguation during parsing - improvements

- Currently, it would be hard to use it with large complex formulas
  - Adding more compact visualisations
  - Joining parse trees as much as possible
- "α-equivalent" formulas must be disambiguated separately (e.g., $\lambda x.xz$ and $\lambda y.yz$)
- Context is important, but the GUI just shows formulas
  - Showing a PDF with highlighted ambiguous formulas that users can interact with to show parse trees

Outline

- Introduction
- Background – λ-calculus (our testing ground) and sTeX
- Grammar generation
- Disambiguation GUI – includes a demo
- **Conclusion and future work**

Conclusion and future work

- We showed that our approach has advantages
- There are limitations (cyclical grammars, the GUI design)
- We have been working on addressing the limitations and hope to share our results with you in the near future!
  - We have found a solution for cyclical grammars
  - Work on the new GUI is already underway