University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science
Raul-Martin Rebane
Computer Science
3rd year BSc

UNIVERSITY OF TARTU
Institute of Computer Science
1632

Study IT in .ee
sponsored by Skype™

LaTeXEE

Parsing mathematical formulas in LaTeX

https://github.com/raulmrebane/LaTeXEE

When writing mathematical formulas in LaTeX, only the presentation information about the formulas is kept. LaTeX itself does not hold semantic information about the mathematical operations used in the documents.

LaTeXEE is created to fix that - it is a command-line application that takes in a LaTeX document as an input and outputs machine-readable mathematical formulas which also contain the mathematical meaning behind the operations.

LaTeXEE was written as a base for other applications. Using the output formulas it is easy to create programs that perform type or style-checking, find free variables, perform logical equivalency checking and so forth.

LaTeXEE was also written with the idea that no math operation should be hardcoded. Instead, the user provides the operations beforehand using a specific declare macro. An example document is to the right with the macros highlighted.

```
\documentclass[10pt,a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}

\usepackage[define,symbolindex,ignore={meaning,syntax}]{declmath}

\begin{document}
\declare{macro=\tuple, meaning=ecc.Tuple, argspec=[2], code={#1, \ldots, #2}}

\declare{syntax={infix,101,"_",l},meaning={list2.list_selector(#_2, #_1)}}

note how the popcorn usage allows us to flip the operands
this is an example of text that will be ignored during parsing
$\tuple{a_1}{a_n}$

\declare{syntax={infix,102,"+",l},meaning={#_1 + #_2}}
$2+2$

\end{document}
```

The program then uses fuzzy parsing to eliminate any unnecessary text and keep only the key components.

```
— ParsedStatement
   ├── DeclareStatement: {macro=\tuple, meaning=ecc.Tuple, argspec=[2], code={#1, \ldots, #2}}
   ├── DeclareStatement: {syntax={infix,101,"_",l},meaning={list2.list_selector(#_2, #_1)}}
   ├── FormulaStatement: \tuple{a_1}{a_n}
   ├── DeclareStatement: {syntax={infix,102,"+",l},meaning={#_1 + #_2}}
   └── FormulaStatement: 2+2
```

Then, for each formula, a context-free grammar is created. This is used to generate an ANTLR parser which is then compiled and loaded back into the program.

```
grammar RuntimeGrammar;
highestLevel : highestNumber #DEFAULT0;
highestNumber : level100 #DEFAULT1;
level100 : level100 level101 #INVISIBLETIMES
|level101 #DEFAULT2;
level101 : level101'_'level102 #Op1
|level102 #DEFAULT3;
level102: level102'+'level103 #Op2
|level103 #DEFAULT4;
level103 : lowestLevel #DEFAULT5;
lowestLevel : '{' highestLevel '}' #BRACES
|'(' highestLevel ')' #PARENS
|'\\tuple''{'highestLevel'}''{'highestLevel'}' #MACRO0
|LEXERRULE #DEFAULT6;
LEXERRULE : [0-9]+ | [a-z];
```

The formula string is then parsed using the generated parser. The resulting parse tree is translated into an OpenMath tree. OpenMath is one of the most common standards of mathematical formula representation. The result is then output as either Popcorn or XML.

```
(ecc.Tuple(list3.list_selector(1, $a), list2.list_selector($n, $a))){LaTeXEE.nonsemantic->code.'{#1,
\ldots, #2}'}
2 + 2

<OMOBJ>
    <OMATTR>
        <OMATP>
            <OMS cd="LaTeXEE" name="nonsemantic"/>
            <OMS cd="code" name="{#1, \ldots, #2}"/>
        </OMATP>
        <OMA>
            <OMS cd="ecc" name="Tuple"/>
            <OMA>
                <OMS cd="list2" name="list_selector"/>
                <OMI>1</OMI>
                <OMV name="a"/>
            </OMA>
            <OMA>
                <OMS cd="list2" name="list_selector"/>
                <OMV name="n"/>
                <OMV name="a"/>
            </OMA>
        </OMA>
    </OMATTR>
</OMOBJ>
<OMOBJ>
    <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMI>2</OMI>
        <OMI>2</OMI>
    </OMA>
</OMOBJ>
```