Seminar Talk:
Theorema Document Processing

Jack Heseltine, December 3rd 2024
for RISC, JKU Linz/Seminar Formal Methods WS 2024
(University of Applied Sciences, Campus Hagenberg - Software Engineering)

Plan for Today

- Personal Introduction/Background/Documents
- This Work/Its **Context**
- The **Project** Itself (Overview)
- **Demo** Document Transformation & **Code** Analysis
- Testing & Closing Remarks
- If Time Permits: Source Code Deep Dive (WL Focus)
- If Time Permits: Building Applications with the Wolfram Cloud (Excursus, see next slide)
    - *current topic: Appearance vs AppearanceElements vs AppearanceRules as a question about style/presentation of a (Cloud-)notebook (or any CloudObject, as a matter of fact), which is to say document representation format and transformation questions come up all the time in this technical ecosystem, as well as the format (or level) of the design: Expression (vs Option) vs StyleSheet in Wolfram ecosystem generally, Wolfram or Theorema Language (Expression) vs LATEX in the present work.*
- Q&A

Background

Me

I am a Consultant **Software Engineer for Wolfram Research (Cloud Project)** and work from Austria, where I am also completing a Masters in AI, at the Machine Learning Institute of Johannes Kepler University in Linz.

- GitHub: https://github.com/heseltime
    - **Project for Discussion Today**: https://github.com/heseltime/Tma2TeX
- Website: https://heseltime.github.io
- LinkedIn: https://www.linkedin.com/in/heselt-in-e/

• If Time Permits: Building Applications with the Wolfram Cloud (Excursus)

Documents

I first started working with documents in a software development context while building an **Enterprise Content (read: Documents) Management (ECM)** system in a team for the Red Cross, during COVID. One of the remarkable things about good ECM is how knowledge becomes accessible and processes/workflows are enabled, making for more productive (non-profit, in my case) organizations: it comes down to appropriate, readable documents, often.

My AI Masters Thesis project is also about documents, specifically how to use LLM tooling to make **PDF-documents** accessible for people using screen-readers, in a fully automated fashion.

In this talk, we will look at **Mathematica Notebooks as a type of document.**

Of interest is document transformation, i.e. turning a source document format into a target format with the same content.

This Work (Its Context)

A Software Engineering Thesis

This work happened as a collaboration between RISC (JKU) and University of Applied Sciences Upper Austria, Campus Hagenberg, as a capstone to the Bachelor's course of study in Software Engineering.

Theorema Project: **Document Processing**

The task in this thesis is to setup an environment for preparing entire (big) mathematical documents in Theorema 2.0. This comprises the design of appropriate Mathematica stylesheets and a **mechanism** for translating Mathematica notebooks into nicely formatted LaTeX documents.

Note on **Scope**: the basic Theorema formulas are the transformation object, proofs are not included in the transformation at the moment.

Mathematica/Wolfram Language (WL)

This was the **main prerequisite** in terms of conceptual background, apart from basic familiarity with First Order Predicate Logic (FOPL) and LATEX on the technical side.

- **Motivation: The Mathematical Document Problem** *(Chapter 1)*
  - While LATEX documents look good on paper, they are not live, in the sense that they do not allow for evaluation of mathematical statements (in a world where systems for doing so exist)
  - The Theorema Project and in a broader sense, the WL ecosystem today (see links in this presentation and *Appendix D* in thesis), address this problem at the root by representing event the document itself as an expression that can be worked with

Mathematica and WL as a Programming Language

- Everything is an Expression: **Compound Expression** (from Buchberger, Mathematica as a Rewrite Language) idea
  - If F is an expression that is not a variable and *T1, ..., Tn* are expressions or sequence variables (specifying one, no or multiple occurrences) then
    F[T1, ..., Tn]
    is also an expression
  - **Rewrite Rules**: expressions can also make up conditional rewrite rules of the form
    lhsExpr := rhsExpr(/; condition)
  - "Mathematica programs are just finite sequences of such (conditional)‡ rewrite rules separated by semicolons."
- We will look at code in just a moment
- Remark: Compound Expressions as a suitable data structure for FOPL (these are WL examples, extends to Theorema Language, see below)
  - Logical Connectives

```
In[14]:= And[p,q]

Out[14]=
p && q
```

  - Predicates

```
In[11]:= EvenQ[n]

Out[11]=
False
```

  - Quantifiers

```
In[12]:= ForAll[x, x > 0, x^2 > 0]

Out[12]=
```

$\forall_{x,x>0}\, x^2 > 0$

- Theorema Language Expression **Example**

```
Iff$TM[And$TM[Forall$TM[RNG$[SIMPRNG$[VAR$[Theorema`Knowledge`VAR$x$TM]]],
   True, Or$TM[Theorema`Knowledge`P$TM[VAR$[Theorema`Knowledge`VAR$x$TM]],
    Theorema`Knowledge`Q$TM[VAR$[Theorema`Knowledge`VAR$x$TM]]]],
  Forall$TM[RNG$[SIMPRNG$[VAR$[Theorema`Knowledge`VAR$y$TM]]], True,
   Implies$TM[Theorema`Knowledge`P$TM[VAR$[Theorema`Knowledge`VAR$y$TM]],
    Theorema`Knowledge`Q$TM[VAR$[Theorema`Knowledge`VAR$y$TM]]]]],
 Forall$TM[RNG$[SIMPRNG$[VAR$[Theorema`Knowledge`VAR$x$TM]]], True,
  Theorema`Knowledge`Q$TM[VAR$[Theorema`Knowledge`VAR$x$TM]]]]
```

(“Theorema`Language`” context-paths removed.)

Comparison to Object-Oriented Programming (OOP)

- Mathematica Book (1996) example

“Overloading” specific definitions and “tagging” specific objects (here as quat-objects):

```
In[15]:= quat[x_] + quat[y_] ^:= quat[x + y]
```

This is also called an Upvalue Definition

```
In[16]:= quat[1] + quat[2]
Out[16]=
         quat[3]

In[17]:= quat[1] + quat[2] + quat[3]
Out[17]=
         quat[6]
```

When you define an upvalue for quat with respect to an operation like Plus, what you are effectively doing is to extend the domain of the Plus operation to include quat objects.

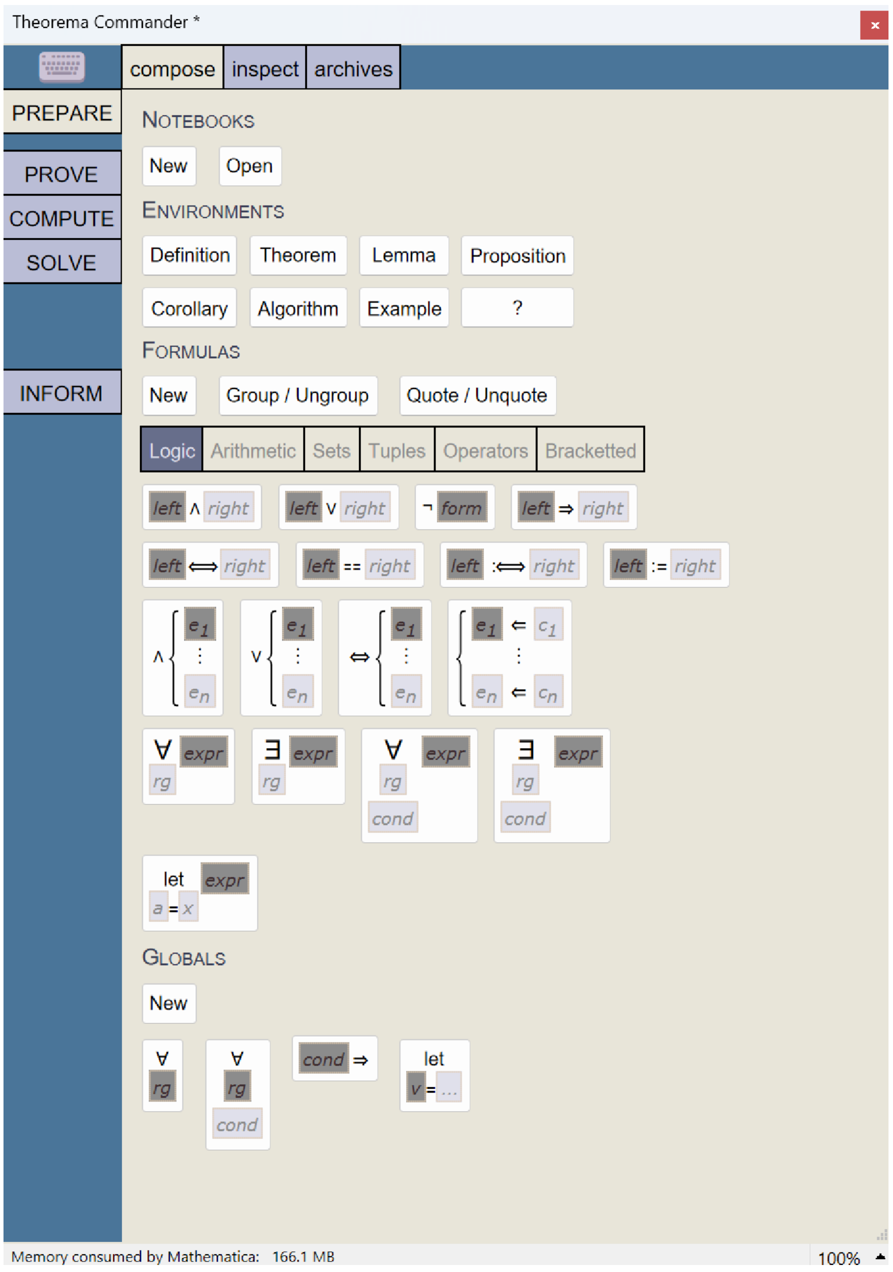
Also to Buchberger this ‘mechanism for associating rules with identifiers opens an immediate possibility of realizing “object oriented programming” in Mathematica.’ (Rewriting Paper)

Comparison to Existing Functionality in Mathematica

- Save as PDF/save as LaTeX: either way, there is a disconnect between the optics and the semantics that should be retained in an export functionality for Theorema.

Vision/Connection to Theorema

- This project is a prototype for functionality that could ultimately be exposed directly to the Theorema user in the Theorema Commander, for instance



The Theorema notebook itself will be shown in the Demo.

The Project: Tma2TeX (Theorema)

Theorema: Automated Theorem Prover

Overview

- https://github.com/windsteiger/Theorema
  "A System for Automated Reasoning (Theorem Proving) and Automated Theory Exploration **based on Mathematica**"

Large Systems with WL

General good practice also recommended by Wolfram Research

- Divide the System into Components (**Packages**)
- Write and Use Unit Tests (we will get to this later)
- Think of Architecture, Not the Code (cf. "High-Level Programming")
- Use Source Control
- Write Documentation

(These last two points do not have any WL-specific implementation.)

Programming Paradigms

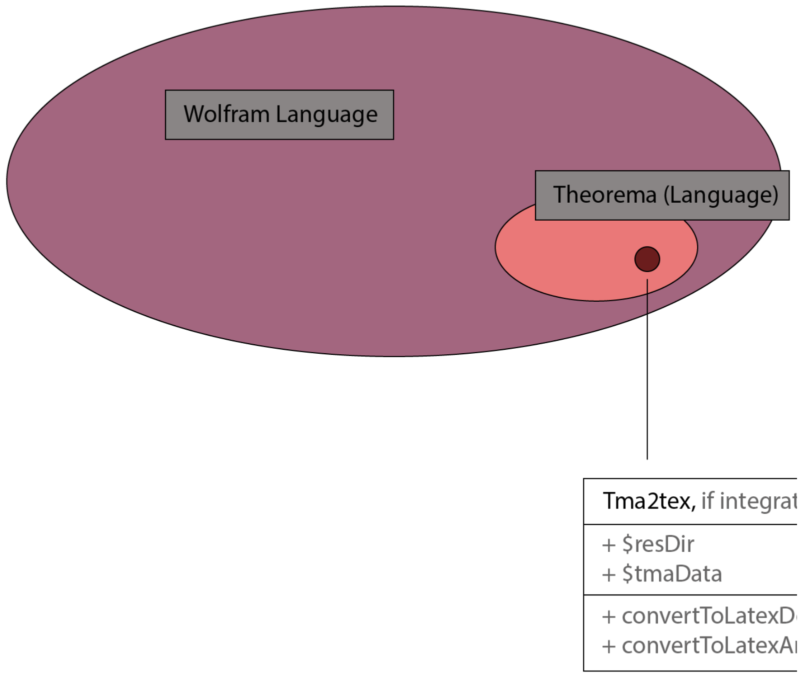
While not a pure OOP language, WL can mimic OOP concepts: that said, why limit ourselves?

- Functional vs Procedural
- "High-Level Programming"
- Rule-Based Programming, Pattern Matching (Connection to Rewriting)

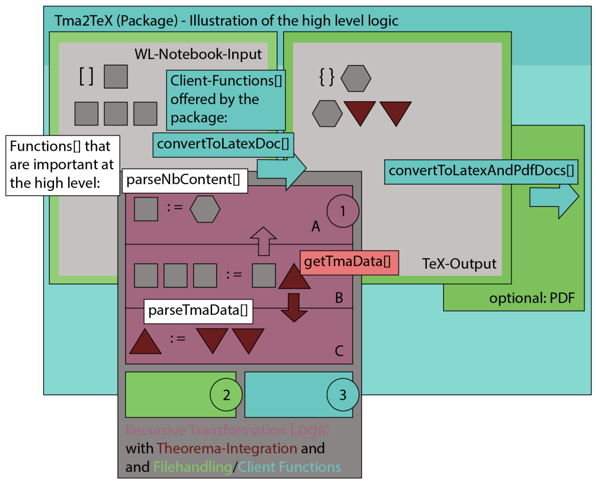
Concept (Tma2TeX)

WL-Native Approach

For direct integration with Theorema: Main idea is to make use of suitable programming paradigms and the natural Compound Expression data structure for FOPL.



Package Specification


Tma2TeX (Package) - Illustration of the high level logic
WL-Notebook-Input
[ ]
Client-Functions[] offered by the package:
convertToLatexDoc[]
{ }
Functions[] that are important at the high level:
parseNbContent[]
convertToLatexAndPdfDocs[]
:=
1
A
getTmaData[]
TeX-Output
:=
B
parseTmaData[]
optional: PDF
:=
C
2
3
Recursive Transformation Logic
with Theorema-Integration and
and Filehandling/Client Functions

Core Idea: No Layout-Information in $\LaTeX$ (Output Format)

● Theorema & Tma2TeX Demo

Note on the IDE used: Eclipse with Wolfram Workbench

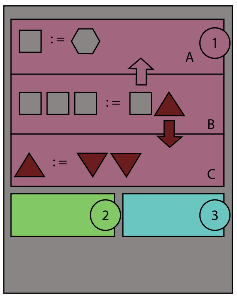
The Main Approach: Recursive Descent

Code Analysis (Overview)

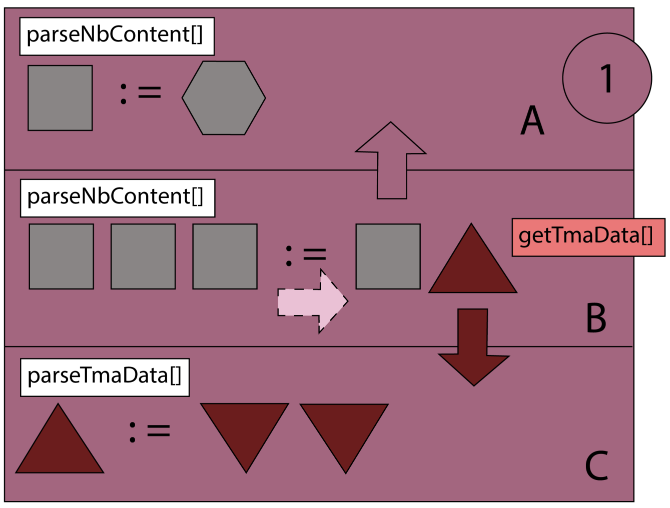
*We are now talking about WL-code in the **tma2tex.wl (package)**:*

Two recursions, **parseNbContent**[] and **parseTmaData**[], through the notebook generally and then the Theorema expressions specifically: the latter are tagged and indexed, a helper function **getTmaData**[] establishes the connection to the Theorema-internal representation via an ID.

Tma2TeX Recursion Flow



Overview Program Logic



Detail Program Logic with relevant high-level Functions[]
- Arrows indicate Recursion Flow, coloring type of expression

Code Analysis: Implementing the Double-Recursive Descent and Connecting to Theorema

parseNbContent[]

```
(*--Part 1.A,Recursive Pattern
 Matching:parseNbContent[] with a focus on (mathematical) symbol-
   level transformations--*)(*--Part 1.A.0-- Structural
 Expressions: \light{}-TeX Command available in Frontend,
to demarcate structural text output from content*)
(*parseNbContent[Notebook[l_List,___]]:="NB reached "<>parseNbContent/@l*)
```

```
(*Careful with Map:Goes to parseNbContent[c_Cell]*)
parseNbContent[Notebook[l_List, ___]] :=
 "\\light{NB reached} " <> parseNbContent[l]
(*goes to parseNbContent[l_List],this our entry point to parsing*)

parseNbContent[c_Cell] := "\\light{Cell reached} " (*matches Cells that
 are not further specified (as relevant WL or TMA cells) below*)

parseNbContent[l_List] := "\\light{List reached} "
parseNbContent[l_List] /; MemberQ[l, _Cell] :=
 StringJoin["\\light{List of cells reached} ", ToString /@ parseNbContent /@ l]


parseNbContent[Cell[CellGroupData[l_List, ___], ___]] :=
 "\\light{CellGroupData reached} " <> parseNbContent[l]


(*--Part 1.A.1-- Text Expressions (at the Cell Level)*)

parseNbContent[Cell[text_String, "Text", ___]] :=
 "\\begingroup \\section*{} " <> text <> "\\endgroup \n\n"

parseNbContent[Cell[text_String, "Section", ___]] :=
 "\\section{" <> text <> "}\n\n"


(*--Part 1.A.2-- Text/Math/Symbols at the String Level*)

(*Operators*)
parseNbContent["<"] := "\\textless"

parseNbContent[">"] := "\\textgreater"

(*Greek Letters*)
parseNbContent["Δ"] := "\\Delta"


(*--Part 1.A.3-- Boxes*)

parseNbContent[
  Cell[BoxData[FormBox[content_, TraditionalForm]], "DisplayFormula", ___]] :=
 StringJoin["\\begin{center}", parseNbContent[content], "\\end{center}\n"]

(*This particular rule does a lot of the parsing through the Tma-Env.*)
parseNbContent[RowBox[list_List]] := StringJoin[parseNbContent /@ list]

(*Underscriptboxes*)
```

```
parseNbContent[UnderscriptBox[base_, script_]] := StringJoin[
  "\\underset{", parseNbContent[script], "}{", parseNbContent[base], "}"]

parseNbContent[UnderscriptBox["∃", cond_]] :=
 "\\underset{" <> parseNbContent[cond] <> "}{\\exists}"
parseNbContent[UnderscriptBox["∀", cond_]] :=
 "\\underset{" <> parseNbContent[cond] <> "}{\\forall}"

(*--Part 1.A.4-- Symbols Dependent on Boxes ...*)
```

... and so on – here we already see output LATEX: we talk about the surrounding file-handling in the next section.

getTmaData[]

```
In[◦]:= (*--Part 1.C.0,
Recursive Pattern Matching:getTmaData[] selects the relevant part in
   Theorema`Common`FML$ in preperation for a second recursive descent,
see 1.B.2--*)getTmaData[id_Integer] :=
 Module[{assoc, cleanStringKeysAssoc, numericKeysAssoc},
  assoc = Association[Cases[$tmaData, Theorema`Common`FML$[
       {idFormula_, _}, expr_, no_] :→ (idFormula → expr), {1}]];
  cleanStringKeysAssoc =
   Association[StringReplace[#, "ID:" → ""] → assoc[#] & /@ Keys[assoc]];
  numericKeysAssoc = Association[
    ToExpression[#] → cleanStringKeysAssoc[#] & /@ Keys[cleanStringKeysAssoc]];
  numericKeysAssoc[id]]
```

parseTmaData[]

```
In[ ]:= (*--Part 1.C.1,
Recursive Pattern Matching:second recursive descent more generalized--*)
(*Generalized parsing function*)
parseTmaData[op_[args___]] := (*always seems to have list length 1*)
 Module[{nextOp, argList, parsedArgs}, nextOp = tmaToInputOperator[op];
  argList = {args};
  parsedArgs = Switch[Length[argList], (*expected to be 1*)1,
    parseTmaData[argList[[1]]], _, "unexpected number of arguments"];
  " " <> ToString[nextOp] (*TODO:LaTeX Conversion*) <> parsedArgs]

(*Parsing function for expressions with standard operators*)
(*parseTmaData[(op_?isStandardOperatorName)[args___]]:=
 With[{nextOp=tmaToInputOperator[op]},
  ToString[nextOp]<>" "<>StringJoin[parseTmaData/@{args},", "]]*)
parseTmaData[(op_?isStandardOperatorName)[args___]] :=
 Module[{nextOp, argList, parsedArgs}, nextOp = tmaToInputOperator[op];
  argList = {args};
  parsedArgs = Switch[Length[argList], 1,
    parseTmaData[argList[[1]]], 2, parseTmaData[argList[[1]]] <>
     (*--interjection--<>*)parseTmaData[argList[[2]]],
    3, parseTmaData[argList[[1]]] <> (*True/False discarded<>*)
     parseTmaData[argList[[3]]], _, "unexpected number of arguments"];
  " " <> ToString[nextOp] (*TODO:LaTeX Conversion*) <> parsedArgs]
```

Code Analysis: Main Client Functions

These are the functions offered to the user of the package.

```
In[ ]:= convertToLatexDoc[notebookPath_] :=
         Module[{nb, content, latexPath, latexTemplatePath, resourceDir = $resDir,
           texResult, sownData, filledContent}, If[Length[$tmaData] == 0,
           (*Issue message if Theorema-Formula-Data not provisioned*)Message[
            tmaDataImport::empty, "The Theorema-Formula-Datastructure is empty.
                Did you evaluate a Theorema notebook before loading
              the package and calling the conversion function?"];
           (*Additional handling for empty data can be added here*)
           Return[$Failed]];
          nb = NotebookOpen[notebookPath, Visible → False];
          content = NotebookGet[nb];
          NotebookEvaluate[content];
          (*on content:important,so that Tma env.variables are
           available in any case*)latexPath = getLatexPath[notebookPath];
          latexTemplatePath = getLatexTemplatePath[notebookPath];
          (*filledContent=
            fillLatexTemplate[resourceDir,<|"nbName"→FileBaseName[notebookPath]|>];*)
          {texResult, sownData} = Reap[parseNbContent[content],
            {"title", "author", "date"}];
          filledContent = fillLatexTemplate[
            resourceDir, <|"nbContent" → texResult, "nbTitle" → First[sownData⟦1, 1⟧],
             "nbAuthor" → First[sownData⟦2, 1⟧], "nbDate" → First[sownData⟦3, 1⟧]|>];
          Export[latexPath, filledContent, "Text"];
          (*Print[Theorema`Common`$tmaEnv];*)]

        convertToLatexAndPdfDocs[notebookPath_] :=
         Module[{latexPath, pdfPath, compileCmd, conversionResult},
          conversionResult = convertToLatexDoc[notebookPath];
          If[conversionResult === $Failed, Return[$Failed]];
          (*Compile LaTeX to PDF using pdflatex*)
          latexPath = getLatexPath[notebookPath];
          pdfPath = StringReplace[latexPath, ".tex" → ".pdf"];
          compileCmd = "pdflatex -interaction=nonstopmode -output-directory=" <>
            DirectoryName[latexPath] <> " " <> latexPath;
          RunProcess[{"cmd", "/c", compileCmd}];]
```

Testing

Messages, Failures and Testing in WL

(Since this is a Software Engineering thesis, after all.)

Closing Remarks and Potential Future Work

WL as a Software Engineering Tool

If Time Permits: Source Code Deep Dive

What would it look like to contain the main syntax/output in WL file, rather than $\LaTeX$? For this, we can take a look at the WL-native MakeTeX/TeXForm implementation, **if time**.

```
In[13]:= TeXForm[And[x, y]]
Out[13]//TeXForm=
         x\land y
```

Speaking of Modern WL

- If Time Permits: Building Applications with the Wolfram Cloud (Excursus)