Mentor Talk:
WL Document Processing

for Wolfram Summer Program 2024/Jack Heseltine (Mentor)

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
- Website: https://heseltime.github.io
- LinkedIn: https://www.linkedin.com/in/heselt-in-e/

Happy to stay in touch!

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/work-flows 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 in the a target format with the same content.

Concepts (& Code)

To understand this document processing topic in Wolfram Language (WL), we need just a bit of **conceptual background** that can be looked up as needed.

- **Propositional Logic**
  - Theorema leans heavily on this category of logic in how it expresses itself.
- **LaTeX**
  - Used as an intermediate language to compile the PDF-document from.

Code

Other than this, the focus is WL/Mathematica documents and engineering a project/pipeline in this context, with **code samples** that might help you with what you want to do in your own project.

Unless indicated otherwise, code will be available at this ***GitHub repo*** as well: https://github.com/heseltime/Tma2TeX

(Feel free to hold me to it if something is missing!)

The Project: Tma2TeX (Theorema)

Theorema: Automated Theorem Prover

- https://github.com/windsteiger/Theorema

  "A System for Automated Reasoning (Theorem Proving) and Automated Theory Exploration **based on Mathematica**"

Institutional Context: Johannes Kepler University in Linz (Hagenberg), Research Institute Symbolic Computation

What this Project Comes Down to: Theorema Notebooks are Mathematica Notebooks are Wolfram Language Expressions

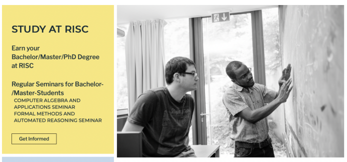
Project Motivation: While Theorema and Mathematica is fine as a programming environment, the institute need LaTeX and PDF for publication purposes mainly.

Project Goal: A fairly automated system that extends Theorema with transformation functionality, or a prototype thereof.

Project Overview Link

- https://risc.jku.at/th/theorema-project-document-processing/

BTW: For anyone interested in study abroad in Austria ...



• Theorema & Tma2TeX Demo

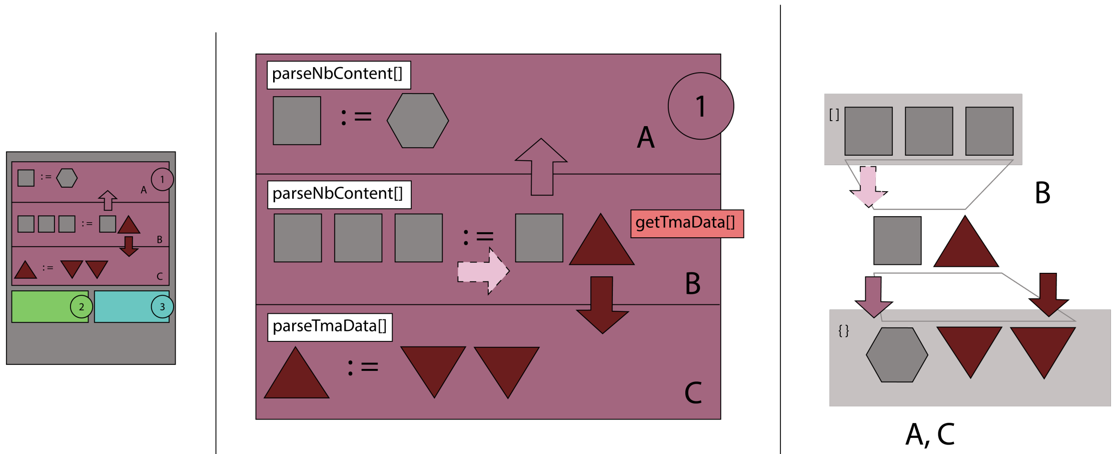
Note on the IDE used: Eclipse with Wolfram Workbench

The Main Approach: Recursive Descent

*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

Recursion/Transformation Order
(Top to Bottom)

Should we look at some Code?

As of June 13th, 2024 (repo link):

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[]

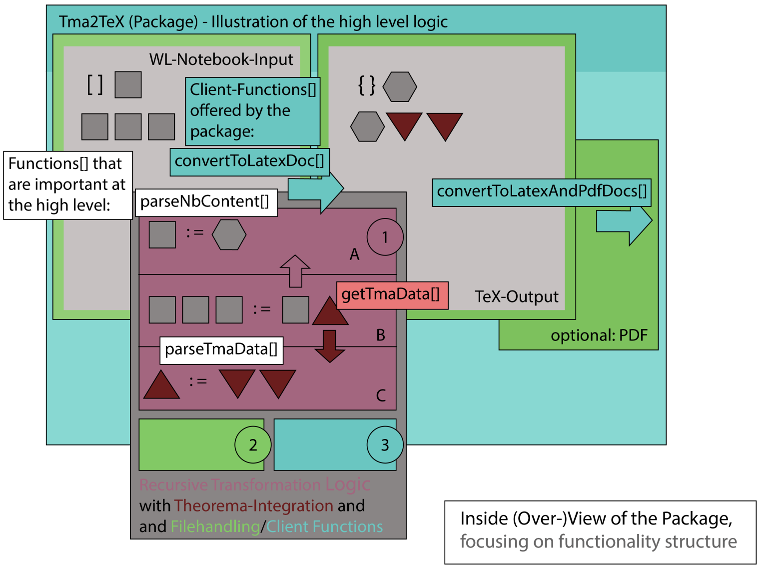
In[7]:=
```
(*--Part 1.C.0,
Recursive Pattern Matching:getTmaData[] selects the relevant part in
   Theorema`Common`FML$ in preperation for a second recursive descent,
see 1.B.2--*)getTmaData[id_Integer] :=
 Module[{assoc, cleanStringKeysAssoc, numericKeysAssoc},
  assoc = Association[Cases[$tmaData, Theorema`Common`FML$[
       {idFormula_, _}, expr_, no_] :→ (idFormula → expr), {1}]];
  cleanStringKeysAssoc =
   Association[StringReplace[#, "ID:" → ""] → assoc[#] & /@ Keys[assoc]];
  numericKeysAssoc = Association[
    ToExpression[#] → cleanStringKeysAssoc[#] & /@ Keys[cleanStringKeysAssoc]];
  numericKeysAssoc[id]]
```

parseTmaData[]

In[8]:=
```
(*--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]
```

Wrapping It Up: High-Level WL, File-Handling & LATEX/Templating



More Code: Main Client Functions

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

In[4]:=
```
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}];]
```

Thanks!

- SW: WL as Computational Language, something to knit documents together with perhaps?
- Bruno Buchberger (Research Institute Symbolic Computation): on Rewriting (in a math context, originally) -

> Wolfram's pattern matching is essentially the natural concept of conditional rewriting.
>
> Mathematica as a Rewrite Language, Bruno Buchberger 1996