Parametric approach to the construction of syntax trees for partially formalized text documents

Kirill Chuvilin

MIPT, ICPT

Agenda

1. Methods for the description and analysis of structured text documents
2. Problems with documents in the LaTeX format
3. Motivation
4. Known implementations and their limitations
5. LaTeX document structure
6. Description of LaTeX style elements
7. Syntax tree of a LaTeX document
8. Parsing algorithms for LaTeX document elements
9. Implementation

Examples of structured text documents

Data storage and transmission

XML, JSON

Data viewing

HTML, CSS, Markdown, BBCode, Textile

Data processing

C/C++, Python, JavaScript, C# and many other programming languages

What is a structured text document?

A syntax tree could be constructed

- Definitely describes the content and structure of the document
- Contains all the necessary information to process a document

Parser

The algorithm, which generates a syntax tree for a document.

Formalizing of the text structure

1. The **content** is defined by the **markup**
2. The **markup** is defined by the **format** or the (computer) **language**
3. The **language** is defined by the **syntax** or the **grammar**
4. The **grammar** is defined by…

The ways to define a grammar

Text notations

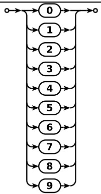
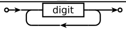
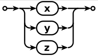
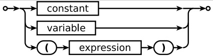
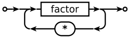
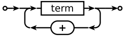
- Backus–Naur Form (BNF)
- Extended Backus–Naur Form (EBNF), ISO/IEC 14977:1996(E)
  `http://www.iso.ch/cate/d26153.html`
- Augmented Backus–Naur Form (ABNF), RFC 5234
  `https://tools.ietf.org/html/rfc5234`

Visual

- Syntax diagrams

Example of grammar

Terminal chars: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, x, y, z, (,), *, +

Symbol	EBNF	Syntax diagram
digit	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"	
constant	digit , {digit}	
variable	"x" | "y" | "z"	
factor	constant | variable | "(", expression, ")"	
term	factor | term, "*" , factor	
expression	term | expression, "+" , term	

Examples of formal grammar descriptions

- **C++** (ISO/IEC 14882:1998(E)): `http://www.externsoft.ch/download/cpp-iso.html`
- **C#** 1.0/2.0/3.0/4.0: `http://www.externsoft.ch/download/csharp.html`
- **ECMAScript (JavaScript)**: `antlr3.org/grammar/1153976512034/ecmascriptA3.g`
- **JSON**: `http://rfc7159.net/rfc7159`
- **XML**: `https://www.w3.org/TR/REC-xml/#sec-notation`
- **HTML 5**: `https://gist.github.com/tkqubo/2842772`

Applications of formal grammars

- Standardization of description
- Generation of parsers:
  - Top-down parsing
    recursively descending parser and LL parser
  - Bottom-up parsing
    LR parser and GLR parser.

  Spirit Parser Framework, Coco/R, The SLK Parser Generator, etc.

General problem

If there is no a formal grammar

- No standardization
- No automatic synthesis of parsers

Example: LaTeX documents

Problems with LaTeX parsing

1. Signature isn't defined in general terms
2. Signature and available items are defined in style files
3. Signature and available items are determined by context
4. TeX does not imply that any syntax tree exists

Examples

`\author{author name}` is a usual command

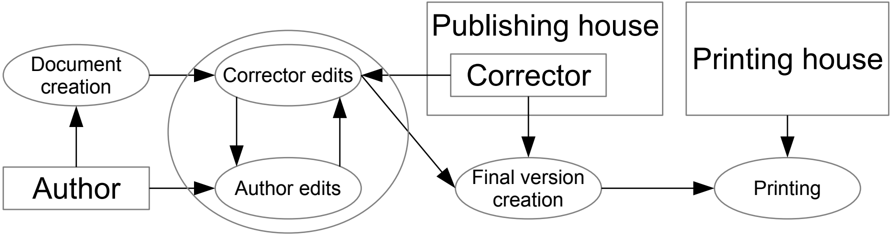
`\author{author name}[name for headers]` only for some styles

`---` and `-` in all languages

`’’---` and `’’=` only for Russian

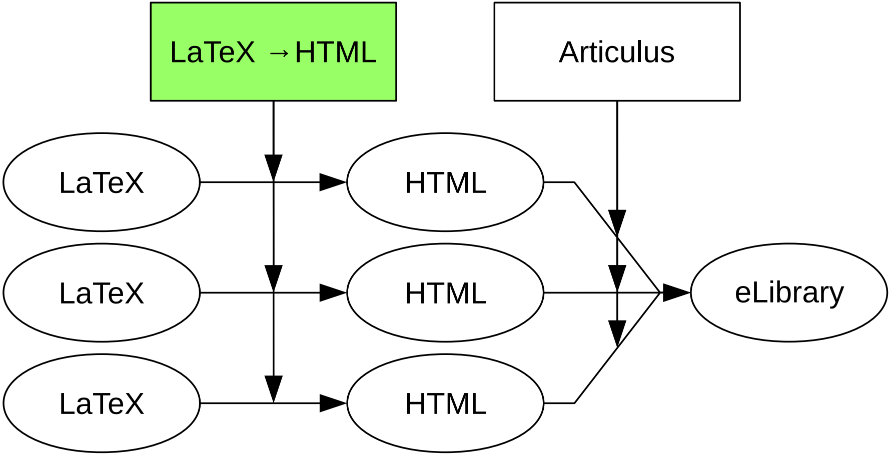
Motivation: automation of typographical error correction

- LaTeX is used for the scientific publications (IEEE, IDP, MMRO, FizmatLit, etc.)
- The corrections are made manually by editors



K. V. Chuvilin, *Automatic synthesis of correction rules for text documents in the LaTeX format*. PhD dissertation. Dorodnicyn Computing Centre of the Russian Academy of Sciences, Moscow, 2013. (in Russian)

Motivation: eLibrary



Motivation: statistical analysis of texts

- Topic modeling
- Catalogization
- Collecting statistics on authors and publishers

Motivation: format conversion

- **XML** is for storage of structured information
- **HTML** is for WEB publishing
- It happens that publishers require **DOC/DOCX**

Known implementations

- plasTeX (Python):
  `http://plastex.sourceforge.net/plastex/index.html`
- LaTeX::Parser (Perl):
  `http://search.cpan.org/~svenh/LaTeX-Parser-0.01/`
- SnuggleTeX (Java):
  `http://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html`

The logical sense of the elements isn't taken into account

- Not possible to allocate the logical blocks accurately
- Difficulties for data mining

LaTeX source structure

- Symbols:
  letters, digits, punctuation marks, `{#1}`, `$#1$`, etc.
- Commands:
  for example, `\author#1`
- Environments:
  for example, `\begin{document}...\end{document}`

The proposed approach

Use external information for parsing

1. Formalize descriptions of symbols, commands and environments,
2. Prepare descriptions according to the LaTeX style files and classes
3. Include the relevant descriptions to parse a document

LaTeX lexeme types

Lexeme type	Comment
BINARY_OPERATOR	binary mathematical operator
BRACKETS	logical brackets
CELL_SEPARATOR	tabular cell separator
CHAR	single char
DIGIT	digit
DIRECTIVE	LaTeX directive
DISPLAY_EQUATION	separate equation
FILE_PATH	file system path
FLOATING_BOX	floating unit
HORIZONTAL_SKIP	horizontal interval
INLINE_EQUATION	inline equation
LABEL	label ID
LENGTH	linear dimension
LETTER	letter
LINE_BREAK	line break
LIST_ITEM	list item
LIST	list of items

Lexeme type	Comment
NUMBER	sequence of digits
PARAGRAPH_SEPARATOR	paragraph spacer
PICTURE	picture
POST_OPERATOR	mathematical postoperator
PRE_OPERATOR	mathematical preoperator
RAW	not processing part of the source
SPACE	space or analog
SUBSCRIPT	subscript
SUPERSCRIPT	superscript
TABLE	tabular
TABULAR_PARAMETERS	tabular arguments
TAG	formatting tag
UNKNOWN	unknown element
VERTICAL_SKIP	vertical spacing
WORD	sequence of letters
WRAPPER	wrapper

LaTeX modes

States of the parser

Mode	**Comment**
`LIST`	in a list of items
`MATH`	in a mathematical expression
`PICTURE`	in the description of an image
`TABLE`	in a tabular
`TEXT`	plain text (default)
`VERTICAL`	between paragraphs

Operation on LaTeX state

Structure

- `directive` is the action directive: `BEGIN` or `END`
- `operand` is LaTeX mode or `GROUP` (group of local mode definitions)

Description of symbol or a command parameter

Structure

- `lexeme` is the lexeme type (logical sense), optional
- `modes` are the modes where the parameter is defined
- `operations` are the operations performed before the parameter

Description of LaTeX symbol

Structure

- `lexeme`
  is the lexeme type (logical sense)
- `modes` are the modes where
  the symbol is defined
- `operations` are the operations
  performed after the symbol
- `parameters`
  are the parameter descriptions
- `pattern` is the LaTeX pattern

Example

```
{
    lexeme: INLINE_EQUATION,
    modes: [TEXT],
    operations: [{
        directive: END,
        operand: MATH
    }],
    parameters: [{
        operations: [{
            directive: BEGIN,
            operand: MATH
        }]
    }],
    pattern: ’’$#1$’’
}
```

Description of LaTeX command

Structure

- `lexeme`
  is the lexeme type (logical sense)
- `modes` are the modes where the command is defined
- `operations` are the operations performed after the command
- `parameters`
  are the parameter descriptions
- `pattern` is the LaTeX pattern
- `name` is the name of the command

Example

```
{
    lexeme: TAG,
    modes: [TEXT],
    parameters: [{ }, { }],
    pattern: ’’[#1]#2’’,
    name: ’’author’’
}
{
    lexeme: TAG,
    modes: [TEXT],
    parameters: [{ }],
    pattern: ’’#1’’,
    name: ’’author’’
}
```

Description of LaTeX environment

Structure

- `lexeme`
  is the lexeme type (logical sense)
- `modes` are the modes where the environment is defined
- `name` is the name of the environment

Example

```
{
    lexeme: WRAPPER,
    modes: [TEXT],
    name: ’’center’’
}
```

Token types

Token type	Source example

LaTeX environment body	\begin{tabular}{c|c}
	$\boxed{\texttt{height \& 1.2 m}}$
	\end{tabular}

LaTeX command	$\boxed{\texttt{\textbackslash includegraphics}}$ $\boxed{\texttt{[}}$width=10cm$\boxed{\texttt{]}}$ $\boxed{\texttt{\{}}$../figure.eps$\boxed{\texttt{\}}}$
LaTeX environment	$\boxed{\texttt{\textbackslash begin\{tabular\}}}${c|c} height & 1.2 m $\boxed{\texttt{\textbackslash end\{tabular\}}}$
Label	\ref{$\boxed{\texttt{equation1}}$}
Linear dimension	\textwidth=$\boxed{\texttt{10cm}}$
Number	height $\boxed{\texttt{1.2}}$ \,m

Token type	Source example
Paragraph separator	Paragrapg □ New paragraph
Filesystem path	\includegraphics [width=10cm] {$\boxed{\texttt{../figure.eps}}$}
Space	height□1.2\,m
Symbol	height 1.2 $\boxed{\texttt{\textbackslash ,}}$ m

Tabular parameters	\begin{tabular}{$\boxed{\texttt{c\vert c}}$}
	height & 1.2 m
	\end{tabular}

Word	$\boxed{\texttt{height}}$ 1.2 \,m
Raw char sequence	\verb|$\boxed{\texttt{complex source}}$|

Parsing of a symbol or command pattern

```
Require: $W$ is the source string to parse,
    $pos$ is the current position in the source,
    $W_p$ is the LaTeX pattern,
    $pos_p = 0$ is the current position in the pattern,
    $style$ is the description of the symbol of the
    command that owns the pattern,
    $parameterTokens = []$ is the stack of the parameter
    tokens.
Ensure: TRUE, if the source corresponds to the pattern;
    FALSE otherwise.

while $pos_p$ not ins the end of $W_p$ do
    if $W_p[pos_p]$ is a space then
        if cannot read a space from $W$ at $pos$ then
            return FALSE
        end if
        move $pos$ the the space end
        $pos_p = pos_p + 1$
    else if $W_p[pos_p] == \#$ then
        $pos_p = pos_p + 1$
        parameter index $= W_p[pos_p]$
        get the parameter properties from $style$
        if cannot read the parameter from $W$ at $pos$
        then
            clear $parameterTokens$
            return FALSE
        end if
        push the readed parameter token in
        $parameterTokens$
        move $pos$ to the parameter end
        $pos_p = pos_p + 1$
    else
        if $W[pos]! = W_p[pos_p]$ then
            return FALSE
        end if
        $pos = pos + 1$; $pos_p = pos_p + 1$
    end if
end while
return TRUE
```

Parsing of a symbol

Require:
W is the source string to parse,
pos is the current position in the source.
Ensure: a symbol token.

```
backup the current state
get the descriptions of the symbols starting with $W[pos]$ for the current state
for all the obtained symbol descriptions do
    if $W$ staring from $pos$ corresponds to the symbol pattern then
        $t =$ token of the symbol with the current description
        child tokens of $t=$ the parameter tokens stack obtained by the pattern parsing
        return $t$
    end if
    restore the backuped state
end for
return a symbol token with undefined description
```

Parsing of a command

Require:
W is the source string to parse,
pos is the current position in the source.

Ensure: a command token or nothing if there is no command at the current position.

```
if $W$ at $pos$ doesn't start with \command_name then
    exit
end if
backup the current state
get the command name
get the descriptions of the commands with the obtained name for the current state
for all the obtained command descriptions do
    if $W$ staring from $pos$ corresponds to the command pattern then
        $t$ = token of the command with the current description
        child tokens of $t$= the parameter tokens stack obtained by the pattern parsing
        return $t$
    end if
    restore the backuped state
end for
return a command token with undefined description
```

Parsing of an environment

Require:

W is the source string to parse,
pos is the current position in the source.

Ensure: an environment token or nothing if there is no environment at the current position.

```
if $W$ at $pos$ doesn't start with \begin{environment_name} then
    exit
end if
get the environment name
$t$ = the environment token that corresponds to the name at the current state
move $pos$ to the end of \begin{environment_name}
parse the pattern of the command with the name "environment_name"
store the corresponding token as the token of $t$ begin command
while $W$ at $pos$ doesn't correspond to \end{environment_name} do
    parse a child token of $t$
end while
move $pos$ to the end of \end{endenvironment_name}
parse the pattern of the command with the name "endenvironment_name"
store the corresponding token as the token of $t$ end command
return $t$
```

Parsing of a LaTeX source code

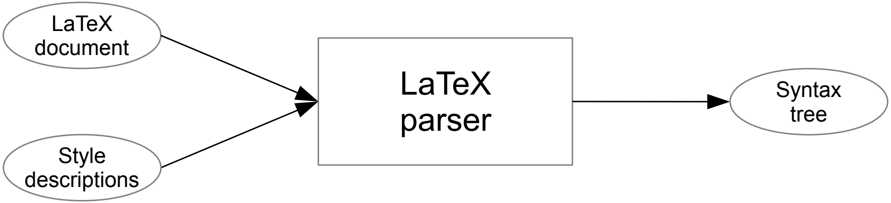
```
Require:
    $W$ is the source string to parse,
    $pos = 0$ is the current position in the source.
Ensure: $tokens$ is the sequence of the parsed tokens.

while $pos$ not in the end of $W$ do
    backup the current state
    if can parse a space from $W$ at $pos$ then
        push the space token to $tokens$
        move $pos$ to the space end
        go to a new iteration
    end if
    restore the backuped state
    if can parse an environment from $W$ at $pos$ then
        push the environment token to $tokens$
        move $pos$ to the environment end
        go to a new iteration
    end if
    restore the backuped state
    if can parse a command from $W$ at $pos$ then
        push the command token to $tokens$
        move $pos$ to the environment end
        go to a new iteration
    end if
    restore the backuped state
    parse a symbol from $W$ at $pos$
    push the symbol token to $tokens$
end while
```

Implementation

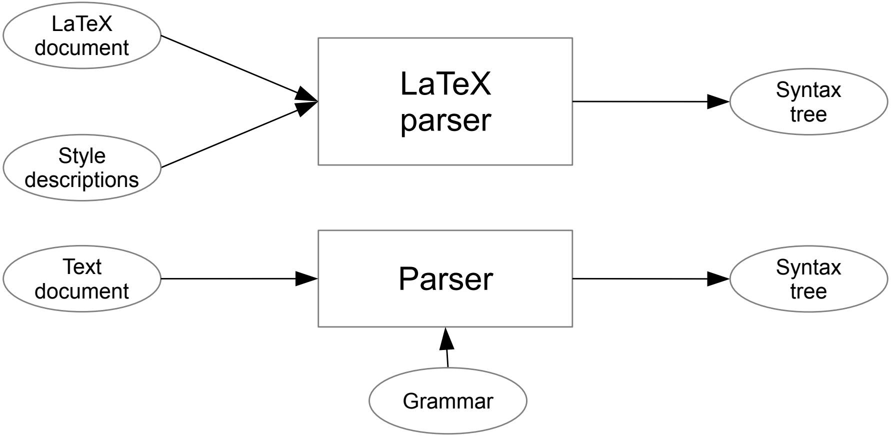
Set of LGPLv3 libraries: `https://bitbucket.org/texnous/latex-parser/`

- `Latex.js`
  basic LaTeX structures
- `LatexStyle.js`
  LaTeX style structures and collection methods
- `LatexTree.js`
  LaTeX syntax tree structures
- `LatexParser.js`
  parser class

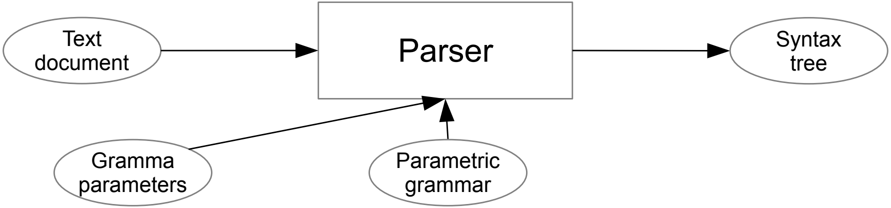
How does the process look like



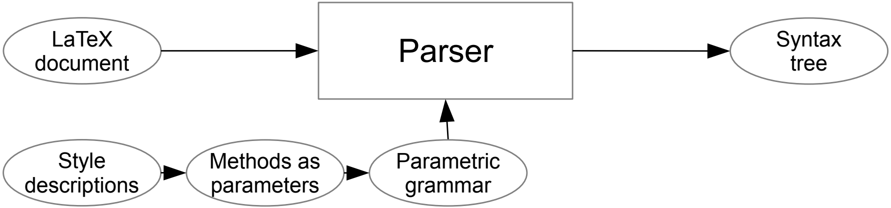
How does the process look like



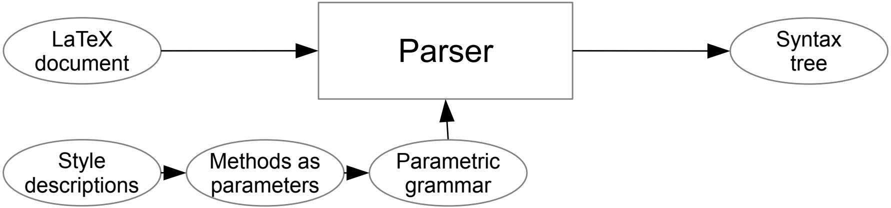
How should the process look like



How really does the process look like



How really does the process look like



LaTeX grammar terms

- Space
- Comment
- Pattern parsing method
- Symbol parsing method
- Command parsing method
- Environment parsing method

Summary

Results

- A formal way of parsing without a static formal grammar
- The logical structure of elements is taken into account

Work in progress

- Preparing of the element description collection
- LaTeX to HTML converter

Further research

- Automatic synthesis of element descriptions

Contacts

Kirill Chuvilin
MIPT, ICPT

kirill@chuvilin.pro

@kirill_chuvilin

@chuvilin

kirill_chuvilin