Параметрический подход к построению синтаксических деревьев для частично формализованных текстовых документов*

К. В. Чувилин[1,2]
kirill@chuvilin.pro
[1]Московский физико-технический институт, Россия, г. Долгопрудный, Институтский пер., 9
[2]ИФТИ, Россия, Московская область, г. Протвино, Заводской проезд, 6

Данная работа посвящена исследованию возможности автоматического построения логической структуры (абстрактного синтаксического дерева) для текстовых документов, формат которых не является полностью определенным стандартами или другими общими для всех документов правилами. В отличие от синтаксисов, описываемых формальными грамматиками, в таких случаях нет возможности в автоматическом режиме построить синтаксический анализатор. Типичными примерами таких форматированных документов с не полностью формализованным синтаксисом разметки являются текстовые файлы в формате LaTeX. В данной работе они используются как ресурсы для практической реализации разрабатываемых алгоритмов. Актуальность анализа именно LaTeX-документов обусловлена тем, что многие научные издательства и конференции используют систему верстки LaTeX, и это порождает важные прикладные задачи по автоматизации рубрикации, коррекции, сравнения, сбора статистики, отображения для WEB и т. п. При синтаксическом анализе документов в формате LaTeX требуется дополнительная информация о стилях: символах, командах и окружениях. В данной работе предлагается метод их описания в формате JSON, который позволяет задавать не только информацию, необходимую для синтаксического анализа, но и метаинформацию, упрощающую дальнейший интеллектуальный анализ. Такой подход использован впервые. Описываются разработанные алгоритмы построения синтаксического дерева документа в формате LaTeX, использующие такую информацию как внешний параметр. Полученные результаты успешно применены в задачах сравнения, автоматической коррекции и рубрикации научных статей. Реализация разработанных алгоритмов доступна в виде набора библиотек, распространяемых по лицензии LGPLv3. Ключевыми особенностями предлагаемого подхода являются гибкость (в рамках рассматриваемой задачи) и простота описания параметров. Предложенные подходы позволяют решить задачу синтаксического анализа документов в формате LaTeX. Но для широкого практического использования разработанных алгоритмов требуется сформировать базу описаний элементов стилей.

Ключевые слова: *абстрактное синтаксическое дерево; дерево; интеллектуальный анализ текстов; синтаксический анализ; JSON; LaTeX*

DOI: 10.21469/22233792.2.2.06

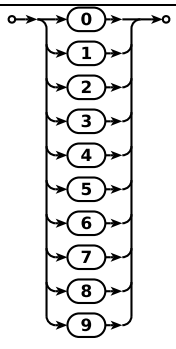
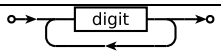
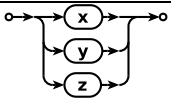
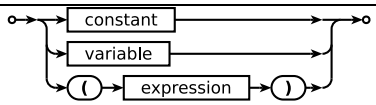
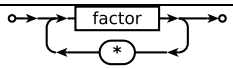
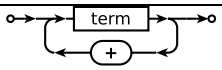
1 Введение

1.1 Способы описания и анализа форматированных текстовых документов

В современных информационных технологиях широко распространено применение текстовых файлов: для хранения и передачи данных (XML, JSON), для отображения данных (HTML, CSS, Markdown, BBCode, Textile), для обработки данных (C/C++, Python,

*Работа выполнена при финансовой поддержке РФФИ, проекты № 16-37-60049 и № 16-07-01267.

Таблица 1 Пример описания грамматики: простые арифметические выражения. Терминальные символы: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `x`, `y`, `z`, `(`, `)`, `*`, `+`

Символ	РБНФ	Синтаксическая диаграмма
`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`	

JavaScript, C# и многие другие языки программирования). Содержимое таких файлов структурировано с помощью специальной *разметки*.

Файлы каждого типа описываются собственным способом разметки. Чтобы понять, какая именно информация содержится в файлах, требуется знать правила такой разметки. Обычно подобные правила называются *форматом* или (компьютерным) *языком*. Примерами как раз и являются языки программирования, языки разметки, языки спецификаций и т. д. Формат отвечает за то, как размещаются и оформляются, за что отвечают элементы информации в текстовых документах.

Но синтаксис (или *грамматику*) языка тоже нужно как-то описать. Зачастую он допускает довольно большое количество возможных конструкций, которые, тем не менее, состоят из повторяющихся блоков. Благодаря такой специфике возможно описание устройства одних блоков через другие с допустимой рекурсией. Этот подход положен в основу формальных систем определения синтаксиса, таких как: форма Бэкуса–Наура (БНФ) [1], расширенная БНФ (РБНФ) [2], синтаксические диаграммы [3]. БНФ и РБНФ описывают грамматику с помощью текстовых конструкций, синтаксические диаграммы являются визуальной интерпретацией РБНФ.

В табл. 1 приведен пример грамматики языка, с помощью которого можно строить арифметические выражения. Есть набор терминальных (конечных) символов: цифры, знаки переменных, операции, скобки. Остальные конструкции определяются через них и через друг друга: цифра, константа, переменная, множитель, терм, выражение.

Формальные грамматики описаны для большинства популярных языков разметки и программирования. Примеры таких описаний:

- 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`.

Наличие формальной грамматики для языка позволяет не только получить стандартизацию формата файлов, но и автоматизировать процесс их анализа.

В данной работе *структурированными текстовыми документами* называются текстовые документы, для которых можно построить абстрактное синтаксическое дерево. Анализ документа подразумевает построение такого дерева. Алгоритм, который строит синтаксическое дерево по структурированному текстовому документу, называется *программой грамматического разбора* или *парсером*.

Оказывается, что наличие формальной грамматики позволяет автоматически построить парсер [4]. Известны два подхода использования формальных грамматик для построения синтаксических деревьев: нисходящий и восходящий синтаксические анализы. При нисходящем синтаксическом анализе правила формальной грамматики применяются начиная со стартового символа до тех пор, пока не будет получена требуемая последовательность. Такой подход реализуют рекурсивный нисходящий парсер и LL-анализатор. При восходящем синтаксическом анализе происходит восстановление выражений до стартового символа. Соответствующие алгоритмы: LR-анализатор и GLR-парсер.

Таким образом, задача построения синтаксического дерева для файлов, описываемых языком с формализованной грамматикой, может считаться решенной.

1.2 Проблемы при работе с документами в формате LaTeX

В общем виде проблему можно сформулировать следующим образом: для документов в формате LaTeX нет формальной грамматики. Это означает и отсутствие строгой стандартизации, и отсутствие возможности автоматически построить парсер известными методами. Источником такой проблемы являются четыре факта:

1) сигнатура команд и символы LaTeX не определены в общем виде. И количество параметров, и способы отделения параметров для разных команд могут существенно отличаться. В большинстве команд параметры выделяются фигурными скобками. Но в некоторых случаях есть необязательные параметры, которые могут быть в квадратных скобках. Кроме того, можно определить команду, в которой параметры будут разделяться, например, точкой или любым другим знаком;
2) и сигнатура, и набор команд и символов определяются стилевыми файлами. Стилевые файлы LaTeX могут содержать правила форматирования и оформления, переопределенные символы, команды и окружения. Например, есть общепринятая команда `\author{Имя автора}` для указания автора документа. Но некоторые стилевые файлы переопределяют ее так, что появляется необязательный параметр: `\author{Имя автора}[Имя для колонтитулов]`;
3) и сигнатура, и набор доступных команд и символов определяются контекстом. Например, есть общеупотребительный символ тире: `---`. Он работает независимо от используемого языка, но отображается с отбивками, не принятыми в русской типографике. Для получения правильных отбивок в публикации нужно использовать символ `''---`,

но он недоступен, если не был выбран русский язык. Также сильно разнятся наборы команд и символов внутри формул и вне;

4) TEX не подразумевает наличие синтаксического дерева. LATEX является наиболее популярным пакетом макрорасширений для TEX — системы компьютерной верстки, разработанной Дональдом Кнутом [5, 6]. TEX предоставляет средства для структуризации и оформления текстов, но только в LATEX появляются команды и окружения, совокупность которых позволяет формировать абстрактное синтаксическое дерево. В LATEX-документах допустимы фрагменты на «чистом» TEX, но такие прецеденты являются скорее исключением и при качественной верстке должны быть перемещены в стилевой файл. В рамках данного исследования подобные фрагменты исходного кода могут быть интерпретированы как отдельные терминальные вершины синтаксического дерева.

Таким образом, задача построения синтаксического дерева для документов в формате LATEX разрешима, но требует отдельных исследования и алгоритмов.

1.3 Актуальность анализа документов в формате LATEX

Многие научные издательства и конференции используют LATEX для подготовки публикаций. Как следствие, есть ряд практических задач, связанных с обработкой документов в таком формате, и на этапе подготовки документов, и для интеллектуальной обработки существующих коллекций: автоматизация коррекции, статистический анализ, извлечение информации, преобразование форматов.

Самой первой задачей из тех, с которыми работал автор, требующей синтаксического разбора документов LATEX, оказалась задача автоматизации коррекции *типографических ошибок*. Такие ошибки связаны с оформлением отступов, формул, штрифтами и т. п. При текущем уровне технологий их исправление производится корректорами вручную, что порождает проблемы, связанные с качеством и временем обработки. Процесс коррекции научных статей изображен на рис. 1. Оказалось, что эту задачу можно решить методами машинного обучения, при этом обучающая выборка составляется из пар синтаксических деревьев документов до обработки профессиональным корректором и после [7].

Опубликованные статьи, как правило, попадают в одну из систем цитирования. Для русскоязычных статей создан РИНЦ (Российский индекс научного цитирования), который управляется eLIBRARY. Это национальная библиографическая база данных научного цитирования, аккумулирующая более 9 млн публикаций российских авторов, а также информацию о цитировании этих публикаций из более чем 6000 российских журналов [8]. Процесс загрузки статей в базу данных РИНЦ изображен на рис. 2. На данный момент одним из этапов является ручная обработка в системе Articulus, которая принимает только документы в формате HTML. В данном случае качественно преобразовать код LATEX в HTML можно, только анализируя синтаксическое дерево для выделения логических бло-

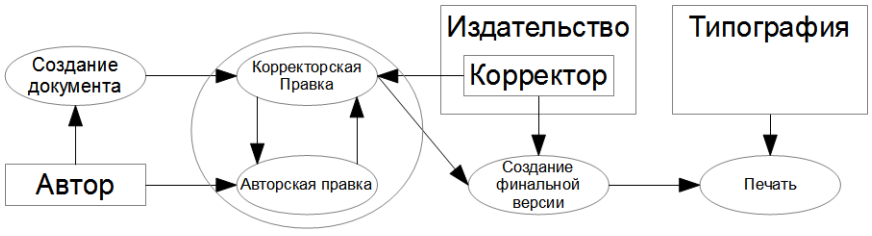


Рис. 1 Бизнес-процесс коррекции научных статей

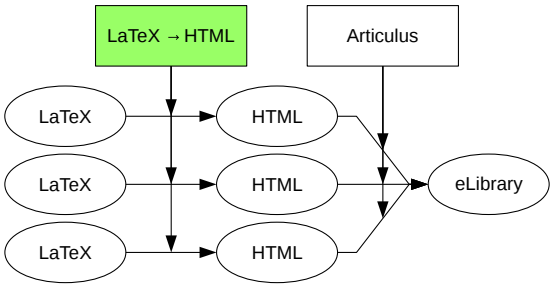


Рис. 2 Бизнес-процесс добавления статей в РИНЦ

ков. Более того, подобный анализ откроет возможность избавления от ручной обработки статей.

На самом деле, задача преобразования формата возникает не только в контексте РИНЦ. Формат HTML также удобно использовать для отображения статей на WEB-ресурсах. Бывает, что издательства требуют документ в формате DOC или DOCX, а материалы, в силу своей научной природы, готовы только в LaTeX. Для хранения и обработки структурированной информации удобно использовать XML. В каждом из случаев требуется корректно выделять логические блоки документов, чего нельзя добиться без качественного анализа синтаксической структуры, т. е. построения синтаксического дерева.

Выделение логических блоков и удобное их представление также полезны в задачах интеллектуального анализа текста, таких как сбор статистики по авторам и издательствам, каталогизация, построение выборки для тематического моделирования.

Таким образом, задача построения синтаксического дерева для документов в формате LaTeX актуальна для практического использования.

1.4 Известные реализации и их недостатки

Исследования, связанные с синтаксическим анализом документов LaTeX, уже проводились, и существует ряд реализаций, среди которых: на Python — plasTeX [9], на Perl — LaTeX::Parser [10], на Java — SnuggleTeX [11]. В силу особенностей анализа формата в каждом из них используются внешние ресурсы — идея, активно применяющаяся в данной работе. Но при этом все эти проекты игнорируют логический смысл элементов синтаксического дерева. Например, нельзя определить, является ли математический символ оператором или просто буквой, а команда в тексте — тэгом разметки или изменением состояния. Такие свойства важны не только для корректного анализа синтаксической структуры документов LaTeX, но и для последующего интеллектуального анализа синтаксического дерева. Самое простые примеры: не зная, какие символы являются буквами или цифрами, нельзя сгруппировать их, соответственно, в слова или числа.

Таким образом, имелась потребность в новом исследовании, на удовлетворение которой и нацелена данная работа.

2 Структура документа LaTeX

В этом разделе описывается то, как воспринимаются документы формата LaTeX в данной работе. Это хорошо зарекомендовавшая себя эвристика, позволяющая формализовать понятие синтаксического дерева для таких документов.

Отдельно нужно сказать: в исходном коде LaTeX допустимы комментарии — части строк, начинающиеся с неэкранированного символа %. Для упрощения описания алгоритмов будем считать, что комментарии удаляются в ходе предобработки.

Весь исходный код файла состоит из блоков, каждый из которых может быть *символом*, *командой* или *окружением*.

Символ LaTeX— это произвольный набор последовательных знаков. Символ может быть терминальным или содержать параметры. Типичными представителями терминальных символов являются цифры и буквы. Знак тире `''---` также является терминальным. В качестве примера символа с параметром выступает обозначение формулы в нотации `$$#1$$`. Где `#1` означает параметр, в данном случае это — тело формулы.

Отдельно нужно отметить символ пробела. Ему эквивалентно любое количество подряд идущих знаков пробела или табуляции и, быть может, одного знака переноса строки. Если в наборе подряд идущих пробельных знаков больше одного знака переноса строки, такой набор эквивалентен двум переносам строки и воспринимается как символ разделения абзацев. Это общеизвестная специфика верстки с издательской системой LaTeX.

Команда LaTeX — это последовательность знаков вида `\имя_команды шаблон`. Обязательное `имя_команды` должно представлять собой последовательность латинских букв, которая может заканчиваться знаком звездочки. Необязательный `шаблон` имеет тот же формат, что и символы LaTeX, и так же может содержать параметры. Пример команды, которая используется для выделения текста полужирным: `\textbf#1`.

Окружение LaTeX — это последовательность знаков вида:
`\begin{имя_окружения}шаблон_команды_начала`
`тело_окружения`
`\end{имя_окружения}шаблон_команды_конца`.
`имя_окружения` имеет тот же формат, что и `имя_команды`. Для документов формата LaTeX общепринято, что весь выводимый контент помещается внутрь окружения `document`.

Независимо от своей природы (символ, команда или окружение) каждый блок документа обладает назначением (логическим смыслом), которое в данной работе называется *типом лексемы*. Выделяемые типы лексем представлены в табл. 2.

Учет логического смысла элементов важен не только при последующем интеллектуальном анализе файлов, но и используется во время синтаксического анализа. Он позволяет определять контекст параметров для символов и команд и ограничивать перебор возможных вариантов. В рамках подобных исследований такой подход применен впервые.

Контекст также определяется состоянием анализатора. Это аналогично поведению компилятора TeX в зависимости от того, какой режим активирован. Список поддерживаемых режимов доступен в табл. 3.

Режимы могут изменяться в любом месте документа по отдельности или группами. Кроме того, может быть начата группа локального определения режимов. После завершения группы состояния режимов восстанавливаются на значения, которые были до начала группы.

3 Описание стилевых элементов LaTeX

Как было указано ранее, наборы доступных символов, команд и окружений LaTeX определяются стилевыми файлами. Это специфика, которая приводит к тому, что описания используемых команд нужно каким-то образом получать. Задача извлечения этой информации из кода стилевых файлов крайне нетривиальна и сравнима, если не превосходит, сложность реализации компилятора TeX. Поэтому в данной работе предлагается использовать внешним образом сформированные описания символов, команд и окружений, которые потом будут переданы как параметр алгоритму синтаксического анализа.

Таблица 2 Типы лексем LaTeX

Тип лексемы	Комментарий
BINARY_OPERATOR	бинарный математический оператор
BRACKETS	логические скобки
CELL_SEPARATOR	разделитель ячеек таблицы
CHAR	знак
DIGIT	цифра
DIRECTIVE	директива LaTeX
DISPLAY_EQUATION	выключенная формула
FILE_PATH	путь файловой системы
FLOATING_BOX	плавающий блок
HORIZONTAL_SKIP	горизонтальный интервал
INLINE_EQUATION	включенная формула
LABEL	идентификатор метки
LENGTH	линейное измерение
LETTER	буква
LINE_BREAK	разрыв строки
LIST_ITEM	элемент списка
LIST	список
NUMBER	последовательность цифр
PARAGRAPH_SEPARATOR	разделитель абзацев
PICTURE	картинка
POST_OPERATOR	математический постоператор
PRE_OPERATOR	математический преоператор
RAW	необрабатываемая часть исходника
SPACE	пробел
SUBSCRIPT	нижний индекс
SUPERSCRIPT	верхний индекс
TABLE	таблица
TABULAR_PARAMETERS	параметры таблицы
TAG	тэг форматирования
UNKNOWN	неизвестный элемент
VERTICAL_SKIP	вертикальный интервал
WORD	последовательность букв
WRAPPER	обертка

Таблица 3 Режимы LaTeX

Режим	Комментарий
LIST	внутри списка элементов
MATH	внутри математического выражения
PICTURE	внутри описания изображения
TABLE	внутри таблицы
TEXT	обычный текст (по умолчанию)
VERTICAL	между абзацами

Далее будут представлены основные структуры, с с помощью которых формируется информация о стилевых файлах.

Operation — операция над состоянием LaTeX:

- `directive` — директива: `BEGIN` (начать) или `END` (закончить);
- `operand` — режим LaTeX или `GROUP` (группа локального определения режимов).

Операция описывает процесс изменения режимов. `BEGIN` означает активацию режима, `END` — деактивацию.

Parameter — параметр символа или команды:

- `lexeme` — тип лексемы (логический смысл), необязательно указывается;
- `modes` — режимы, в которых определен;
- `operations` — набор операций, выполняемых перед параметром.

Symbol — символ LaTeX:

- `lexeme` — тип лексемы (логический смысл);
- `modes` — режимы, в которых определен;
- `operations` — набор операций, выполняемых после символа;
- `parameters` — описание параметров;
- `pattern` — шаблон LaTeX.

Шаблон описывает сигнатуру символа, в которой `#номер_параметра` обозначает положение параметров, а остальные знаки соответствуют знакам символа в исходном коде документа. Пример на языке JSON описания символа включенной формулы:

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

Command — команда LaTeX:

- `lexeme` — тип лексемы (логический смысл);
- `modes` — режимы, в которых определена;
- `operations` — набор операций, выполняемых после команды;
- `parameters` — описание параметров;
- `pattern` — шаблон LaTeX;
- `name` — имя команды.

От описания символа отличается только тем, что добавлено имя. Пример на языке JSON описания двух команд вставки информации об авторе документа с одним и тем же именем, но разным набором параметров:

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

Environment — окружение LaTeX:

- `lexeme` — тип лексемы (логический смысл);
- `modes` — режимы, в которых определено;
- `name` — имя окружения.

Пример на языке JSON описания окружения для горизонтального выравнивания по центру:

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

Предложенный способ описания стилевых элементов является простым для понимания: он не требует глубоких навыков программирования или знания теории формальных языков. Описания команд могут быть составлены и пользователями LaTeX, обладающими опытом верстки с помощью этой системы. Достаточно знания допустимых синтаксисов символов и команд и понимания логического смысла элементов. В то же время, этот способ достаточно гибок, поскольку позволяет описывать не только синтаксические конструкции, но и задавать метаинформацию (режимы LaTeX, типы лексем) для управления контекстом, а сам набор описаний не является фиксированным и может быть модифицирован при замене или добавлении стилевого файла.

4 Синтаксическое дерево документа LaTeX

Взаимное расположение символов, команд и окружений LaTeX образует древовидную структуру, корнем которой является окружение `document`. Узлы этой структуры называются *токенами*. При синтаксическом анализе документа в формате LaTeX элементы исходного кода, в зависимости от лексем и контекста, могут порождать токены определенных типов, список которых представлен в табл. 4.

Полученное дерево удобно использовать для преобразования документа LaTeX в другой формат, интерпретируя отдельные токены. И, поскольку каждому токену соответствует тип лексемы, такая структура оказывается дополнительно информативной для интеллектуального анализа.

Таблица 4 Типы токенов синтаксического дерева LaTeX

Тип токена	Пример исходного кода
LaTeX environment body	\begin{tabular}{c|c} height & 1.2m \end{tabular}
LaTeX command	\includegraphics [width=10cm] {../figure.eps}
LaTeX environment	\begin{tabular}{c|c} height & 1.2m \end{tabular}
Label	\ref{equation1}
Linear dimension	\textwidth=10cm
Number	height 1.2\,m
Paragraph separator	Paragraph □ New paragraph
Filesystem path	\includegraphics [width=10cm] {../figure.eps}
Space	height□1.2\,m
Symbol	height 1.2 \, m
Tabular parameters	\begin{tabular}{c|c} height & 1.2m \end{tabular}
Word	height 1.2 \,m
Raw char sequence	\verb|complex source|

5 Алгоритмы синтаксического анализа элементов документа LaTeX

В этом разделе описываются разработанные в ходе описываемого исследования алгоритмы синтаксического анализа отдельных фрагментов исходного кода документа в формате LaTeX: шаблона, символа, команды, окружения. Допускаются рекурсивные вызовы одним алгоритмом других. Это необходимо, поскольку, вообще говоря, нет ограничений на типы и глубину вложенных токенов, а предлагаемый метод разбора исходного кода подобен рекурсивному нисходящему парсеру.

Алгоритм 1 описывает процесс синтаксического анализа шаблона команды или символа LaTeX. У него два основных предназначения: выбрать, какой синтаксис применим, и рекурсивно собрать набор токенов параметров. Если алгоритм возвращает TRUE, предложенный синтаксис применим, и в $parameterTokens$ будет список полученных токенов для параметров в соответствующем порядке. Если алгоритм возвращает FALSE, в данном месте не может быть использована команда или символ с предлагаемым синтаксисом.

Тонкости, которые не вошли в описание алгоритма: если параметр имеет особенный тип лексемы, соответствующий, например, параметрам таблицы или пути в файловой системе, используется специальный алгоритм синтаксического анализа, которые возвращает токен соответствующего типа.

Алгоритм 1 Синтаксический анализ шаблона команды или символа

Вход:
W — строка для анализа, pos — текущая позиция,
W_p — шаблон LaTeX, $pos_p = 0$ — текущая позиция в шаблоне,
$style$ — описание символа или команды, чей шаблон анализируется,
$parameterTokens$ — стек токенов параметров (изначально пустой)
Выход: TRUE, если код соответствует шаблону; FALSE, в противном случае

```
1: пока pos_p не в конце W_p
2:   если W_p[pos_p] — пробел то
3:     если не удается считать пробел из W в позиции pos то
4:       return FALSE
5:     передвинуть pos к концу пробела
6:     pos_p = pos_p + 1
7:   иначе если W_p[pos_p] == # то
8:     pos_p = pos_p + 1
9:     номер параметра = W_p[pos_p]
10:    получить свойства параметра из style
11:    если не удаётся считать параметр из W в позиции pos то
12:      очистить parameterTokens
13:      return FALSE
14:    добавить считанный токен параметра в parameterTokens
15:    передвинуть pos к концу параметра
16:    pos_p = pos_p + 1
17:  иначе
18:    если W[pos]! = W_p[pos_p] то
19:      return FALSE
20:    pos = pos + 1
21:    pos_p = pos_p + 1
22: return TRUE
```

Алгоритм 2 описывает процесс синтаксического анализа символа LaTeX. По исходному коду в текущей позиции и активным режимам LaTeX выбираются символы с допустимыми шаблонами. Все они перебираются до тех пор, пока не будет найден подходящий. Если один из допустимых шаблонов оказался применим, то генерируется токен соответствующего символа, а стек полученных при анализе шаблона параметров формирует набор дочерних токенов. Если ни один из допустимых шаблонов не подошел, возвращается токен символа с неизвестным описанием. Таким образом, алгоритм всегда возвращает положительный результат.

Тонкости, которые не вошли в описание алгоритма: если полученный токен имеет тип лексемы пробела, разделителя абзацев, буквы или цифры, он преобразуется в токен соответствующего типа.

Алгоритм 2 Синтаксический анализ символа

Вход: W — строка для анализа, pos — текущая позиция
Выход: токен символа

```
1: сохранить текущее состояние
2: получить описания символов для текущего состояния, начинающиеся с W[pos]
3: для всех полученных описаний символов
4:    если W, начиная с позиции pos, соответствует шаблону то
5:       t = токен символа с текущим описанием
6:       дочерние токены t = стек токенов параметров, полученных при анализе шаблона
7:       return t
8:    восстановить сохраненное состояние
9: return токен символа с неизвестным описанием
```

Алгоритм 3 описывает процесс синтаксического анализа команды LaTeX. Он практически повторяет логику алгоритма 2 за тем исключением, что допустимые команды определяются по имени, и в случае, если исходный код в текущей позиции не содержит команду, алгоритм не возвращает токен.

Алгоритм 3 Синтаксический анализ команды

Вход: W — строка для анализа, pos — текущая позиция
Выход: токен команды или ничего, если в текущей позиции нет команды

```
1: если W начинается не с \имя_команды то
      выход
2: сохранить текущее состояние
3: получить имя команды
4: получить описания команд с полученным именем для текущего состояния
5: для всех полученных описаний команд
6:    если W, начиная с позиции pos, соответствует шаблону то
7:       t = токен команды с текущим описанием
8:       дочерние токены t = стек токенов параметров, полученных при анализе шаблона
9:       return t
10:   восстановить сохраненное состояние
11: return токен команды с неизвестным описанием
```

Алгоритм 4 описывает процесс синтаксического анализа окружения LaTeX. Если исходный код в текущей позиции не содержит начало окружения, ничего не возвращается. В противном случае возвращается токен, соответствующий окружению, описание которого извлекается по имени.

Тонкости, которые не вошли в описание алгоритма: при некорректном документе формата LaTeX конец окружения может отсутствовать (в этом случае будет автоматически сформирована команда конца при завершении исходного кода), а если описания окружения с данным именем нет, то будет сформирован токен окружения с неизвестным описанием.

Алгоритм 4 Синтаксический анализ окружения

Вход: W — строка для анализа, pos — текущая позиция
Выход: токен окружения или ничего, если в текущей позиции нет начала окружения

если W начинается не с `\begin{имя_окружения}` **то**
  выход
получить имя окружения
t = токен окружения, соответствующего имени
сдвинуть pos концу `\begin{имя_окружения}`
считать шаблон команды с именем «`имя_окружения`»
записать соответствующий токен, как токен команды начала t
пока с позиции pos в W не стоит `\end{имя_окружения}`
  считать дочерний токен t
сдвинуть pos концу `\end{имя_окружения}`
считать шаблон команды с именем «`endимя_окружения`»
записать соответствующий токен, как токен команды конца t
return t

Алгоритм 5 описывает процесс получения очередного токена. Априори не известно, какого типа токен находится в текущей позиции исходного кода. Поэтому перебираются возможные варианты: пробел, окружение, команда, символ. Токен символа всегда может быть получен, поэтому каждая итерация «поглощает» некоторый ненулевой фрагмент исходного кода. Таким образом, алгоритм выполним в любом случае.

Алгоритм 5 Синтаксический анализ кода LaTeX

Вход: W — строка для анализа, $pos = 0$ — текущая позиция
Выход: $tokens$ — последовательность считанных токенов

пока pos не в конце W
  сохранить текущее состояние
  если удалось считать считать пробел в W с позиции pos **то**
    записать токен пробела в $tokens$
    сдвинуть pos к концу пробела
    перейти к новой итерации
  восстановить сохраненное состояние
  если удалось считать считать окружение в W с позиции pos **то**
    записать токен окружения в $tokens$
    сдвинуть pos к концу окружения
    перейти к новой итерации
  восстановить сохраненное состояние
  если удалось считать считать команду в W с позиции pos **то**
    записать токен команды в $tokens$
    сдвинуть pos к концу команды
    перейти к новой итерации

восстановить сохраненное состояние
считать символ в W с позиции pos
записать токен символа в $tokens$

Описанные алгоритмы в совокупности с комментариями о тонкостях охватывают все возможные ситуации в используемой интерпретации синтаксиса LaTeX.

6 Реализация

Описанные в данной работе идеи были реализованы автором несколько раз. Впервые они успешно использовались для задач автоматической коррекции документов [12] и построения XML-описаний статей для применения в тематическом моделировании. Реализация была сделана с помощью Qt/C++, исходные коды публично не распространялись.

С 2016 г. автором запущен проект по реализации синтаксического анализатора LaTeX на JavaScript [13]. Это набор библиотек, распространяемых по лицензии LGPLv3. Основной их задачей является прозрачное внедрение инструментов анализа документов в формате LaTeX для всех окружений, поддерживающих JavaScript, в том числе для WEB.

Доступны следующие библиотеки:

- `Latex.js`. Содержит основные определения структур LaTeX, такие как лексемы, режимы и операции;
- `LatexStyle.js`. Содержит определения структур стиля LaTeX: символы, команды, окружения. Также предоставляет инструменты для работы с коллекциями таких структур в формате JSON;
- `LatexTree.js`. Содержит определения структур синтаксического дерева LaTeX: токены всех типов, само дерево;
- `LatexParser.js`. Предоставляет класс синтаксического анализатора, принимающего описания стилей и позволяющего по исходному тексту документа в формате LaTeX получать синтаксическое дерево.

На данный момент принципы работы всех библиотек соответствуют алгоритмам, приведенным в этой работе. С их использованием разрабатывается проект по отображению LaTeX-документов в браузере средствами HTML.

7 Заключение

В данной работе рассматривалась задача синтаксического анализа текстовых документов, формат которых не является полностью определенным стандартами или другими общими для всех документов правилами. В качестве образца таких файлов были выбраны документы в формате LaTeX. Было показано, что для языка их разметки нет формализованной грамматики, поэтому требуется отдельный подход, учитывающий внешние ресурсы, соответствующие подгружаемым стилевым файлам. Существующие решения не используют логические значения элементов синтаксиса, поэтому не могут быть применены для интеллектуального анализа текстов.

Предложен подход, который позволяет с помощью несложных конструкций описать стилевые файлы LaTeX. Описаны разработанные алгоритмы синтаксического анализатора, принимающего такие описания. Они реализованы в виде набора библиотек на языке JavaScript, распространяемого по лицензии LGPLv3.

Таким образом, можно считать, что задача синтаксического анализа документов LaTeX решена в рамках рассматриваемого в этой работе представления о синтаксическом дереве таких документов.

Но остается важная проблема, которая препятствует быстрому внедрению разработанного синтаксического анализатора — необходимость ручного описания элементов стиля: символов, команд и окружений LaTeX. Это несложный, но трудоемкий процесс, требующий навыков верстки от исполнителей. Продолжение исследований по рассматриваемой тематике может быть направлено на автоматизацию этого процесса.

Литература

[1] Programming systems and languages / Ed. S. Rosen. — McGraw Hill computer science ser. — New York, NY, USA: McGraw Hill, 1967. 734 p.

[2] ISO/IEC 14977:1996. Information technology — syntactic metalanguage — extended BNF. http://www.iso.org/iso/catalogue_detail?csnumber=26153.

[3] Syntax diagram. Wikipedia. https://en.wikipedia.org/wiki/Syntax_diagram.

[4] *Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д.* Компиляторы: принципы, технологии и инструментарий / Пер. с англ. — 2-е изд. — М.: Вильямс, 2008. 1184 с. (*Aho A., Lam M., Sethi R., Ullman J.* Compilers: Principles, techniques, and tools. — 2nd ed. — Prentice Hall, 2006. 1000 p.)

[5] *Lamport, L.* LaTeX: A document preparation system. — Reading, MA, USA: Addison-Wesley Professional, 1994. 273 p.

[6] *Кнут Д.* Всё про TeX / Пер. с англ. — М.: Вильямс, 2003. 560 с. (*Knuth D. E.* The TeX book. — Reading, MA: Addison-Wesley Professional, 1984. 496 p.)

[7] *Chuvilin K.* Machine learning approach to automated correction of LaTeX documents // 18th FRUCT & ISPIT Conference Proceedings. — Saint-Petersburg: Technopark of ITMO University. P. 33–40. http://fruct.org/publications/fruct18/files/Chu.pdf.

[8] eLIBRARY.RU — Российский индекс научного цитирования. http://elibrary.ru/project_risc.asp.

[9] plasTeX — a Python framework for processing LaTeX documents. http://plastex.sourceforge.net/plastex/index.html.

[10] *Heinicke S.* LaTeX-Parser-0.01. http://search.cpan.org/~svenh/LaTeX-Parser-0.01/.

[11] SnuggleTeX — overview & features. http://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html.

[12] *Чувилин К. В.* Автоматический синтез правил коррекции текстовых документов формата LaTeX. — М.: Вычислительный центр им. А. А. Дородницына Российской академии наук, 2013. Дисс. ... канд. техн. наук.

[13] texnous latex-parser — Bitbucket. https://bitbucket.org/texnous/latex-parser/.

Поступила в редакцию 1.09.2016

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

K. V. Chuvilin[1,2]
kirill@chuvilin.pro
[1]Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow, Russia
[2]ICPT, 6 Zavodskoy proezd, Protvino, Moscow Region, Russia

This article investigates the possibility of logical structure (abstract syntax tree) automatic construction for text documents, the format of which is not fully defined by standards or other rules common to all the documents. In contrast to the syntax described by formal grammars, in such cases, there is no way to build the parser automatically. Text files in LaTeX format are the typical examples of such formatted documents with not completely formalized syntax markup. They are used as the resources for the implementation of the algorithms developed in this work. The relevance of LaTeX document analysis is due to the fact that many scientific publishers and conferences use LaTeX typesetting system, and this gives rise to important applied task of automation for categorization, correction, comparison, statistics collection, rendering for WEB, etc. The parsing of documents in LaTeX format requires additional information about styles: symbols, commands, and environments. A method to describe them in JSON format is proposed in this work. It allows to specify not only the information necessary to pars, but also meta information that facilitates further data mining. This approach is used for the first time. The developed algorithms for constructing a syntax tree of a document in LaTeX format that use such information as an external parameter are described. The results are successfully applied in the tasks of comparison, autocorrection, and categorization of scientific papers. The implementation of the developed algorithms is available as a set of libraries released under the LGPLv3. The key features of the proposed approach are flexibility (within the framework of the problem) and simplicity of parameter descriptions. The proposed approach allows one to solve the problem of parsing documents in LaTeX format. But it is required to form the base of style element descriptions for widespread practical use of the developed algorithms.

Keywords: *abstract syntax tree; JSON; LaTeX; parsing; text mining; tree*

DOI: 10.21469/22233792.2.2.06

References

[1] Rosen, S., ed. 1967. *Programming systems and languages*. McGraw Hill computer science ser. New York, NY: McGraw Hill. 734 p.

[2] ISO/IEC 14977:1996. Information technology — syntactic metalanguage — extended BNF. Available at: http://www.iso.org/iso/catalogue_detail?csnumber=26153 (accessed August 31, 2016).

[3] Syntax diagram. Wikipedia. Available at: https://en.wikipedia.org/wiki/Syntax_diagram (accessed August 31, 2016).

[4] Aho, A., M. Lam, R. Sethi, and J. Ullman. 2006. *Compilers: Principles, techniques, and tools*. 2nd ed. Prentice Hall. 1000 p.

[5] Lamport, L. 1994. *LaTeX: A document preparation system: User's guide and reference*. 2nd ed. Reading, MA: Addison-Wesley Professional. 273 p.

[6] Knuth, D. E. 1984. *The TeXbook*. Reading, MA: Addison-Wesley Professional. 496 p.

*The research was supported by the Russian Foundation for Basic Research (grants 16-37-60049 and 16-07-01267).

[7] Chuvilin, K. V. Machine learning approach to automated aorrection of LaTeX documents. *18th FRUCT & ISPIT Conference Proceedings*. Saint-Petersburg: Technopark of ITMO University. 33–40. Available at: `http://fruct.org/publications/fruct18/files/Chu.pdf` (accessed August 31, 2016).

[8] eLIBRARY.RU — Rossiyskiy indeks nauchnogo tsitirovaniya [eLIBRARY.RU — Russian Science Citation Index]. Available at: `http://elibrary.ru/project_risc.asp` (accessed August 31, 2016).

[9] plasTeX — a Python framework for processing LaTeX documents. Available at: `http://plastex.sourceforge.net/plastex/index.html` (accessed August 31, 2016).

[10] Heinicke, S. LaTeX-Parser-0.01. Available at: `http://search.cpan.org/~svenh/LaTeX-Parser-0.01/` (accessed August 31, 2016).

[11] SnuggleTeX — overview & features. Available at: `http://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html` (accessed August 31, 2016).

[12] Chuvilin, K. V. 2013. Automatic synthesis of correction rules for text documents in the LaTeX format. Moscow: Dorodnicyn Computing Center of the Russian Academy of Sciences. PhD Diss. (In Russian.)

[13] texnous latex-parser — Bitbucket. Available at: `https://bitbucket.org/texnous/latex-parser/` (accessed August 31, 2016).

Received September 1, 2016