卒業論文

Wiki2LaTeX フィルターの開発

関西学院大学 理工学部 情報科学科
6644 吉井了平

2010 年 3 月

指導教員　西谷　滋人　教授

概 要

西谷研究室では，これまで卒業論文を執筆する際，アウトラインプロセッサー TAO に直接 LaTeX の命令を付加した形で原稿を作成し，それを txt 形式で書き出しを行い，LaTeX を実行して pdf ファイルを得るという方法を使ってきた．

しかし，LaTeX 初学者にとって LaTeX 命令や規約を参考書や Web 等で調べながら，本文の文章を練って執筆するのは，文章以外に LaTeX の使い方に余計な気を取られてしまい，文章作成自体のみに集中できず，作業効率が悪いという問題点があった．

そこで今回「何とか文章作成において頻出の LaTeX 命令の入力だけでも軽減できないか」という課題の元に，それを実現するフィルター開発に取り組んだ．

手法として，原稿作成段階において，主要な LaTeX 命令を Wiki の一種である PukiWiki の記法に置き換えるという手法を用いた．具体的には，PukiWiki 記法を HTML のタグに変換するための既存のフィルター pukipa.rb を雛形とし，LaTeX 命令変換用へと改良することにより，PukiWiki 記法を LaTeX 命令へと変換可能にした．

具体的な雛形フィルターの改良としては，まず雛形フィルターの構造分析を行った．その結果，HTML タグの出力部分を加工し，代わりに LaTeX 命令を出力するように改良すればうまくいくのではないかという仮説を得られたため，HTML タグと LaTeX 命令の対照表を作成し，その表を元にプログラムを改良した．

その際，雛形フィルターのデフォルト機能 (見出し，記号付きリスト，番号付きリスト，定義語) に加えて，章，段落，数式の処理，コマンドの処理，相互参照のラベル付け，コメントアウト機能を新たに追加することによって，より一層文章作成環境を充実させた．

また 0 タイプリリース後のユーザからの改良要望と作業効率の向上のため，プログラム実行時に，複数のフィルターが順次一括実行され，LaTeX の実行および pdf ファイルを得て，それを open するまでの一連の流れを一回のプログラム実行で可能にした．

この結果，文章作成のみに集中しやすくなり，また原稿作成段階において，文章全体の体裁を視覚的にも捉えやすくできたため，文章作成への集中と作業効率の向上の実現につながった．

目 次

第1章　研究の目的・背景

1.1　文章作成環境比較による Editor の選定

私たち西谷研究室では，卒業論文を執筆する際 TAO というアウトラインプロセッサーと組版ソフト LaTeX を用いている．

文章作成ソフトとしては Word が一般的であり，Word には WYSIWYG(What You See Is What You Get) という言葉で表されるように，ユーザーが入力した内容，つまり画面に表示された内容が印刷などの出力が一致するという利点がある．これはユーザにとってとてもわかりやすい仕組だが，逆に言うとユーザーがレイアウト作業のすべてを行わなければならないわけで，執筆以外にも労力が必要となる [3]．具体的な不満点としては，図や表を入れたときに番号を振り直す必要が生じることや，全体構成を変更した際，目次部分も変更しなければならないなど，編集が進むにつれて付随して生じる変更があまりにも手間である．

また，卒業論文のように何十ページにもわたる文章を書くとなると，どこに何が書いてあるのか一目で判断したり，思い出して編集するのは非常に困難であり，編集環境の面からも文章作成自体にのみ集中することが難しい．

そこで，文章を階層構造の形式で作成し，ユニットごとに展開や折り畳んだりできる編集環境に優れたアウトラインプロセッサー TAO を文章作成のエディターとして採用している．TAO を用いることにより，章や見出し単位での文章の編集・管理が可能となるため，文章作成以外のことに神経や労力を使うリスクを軽減し，文章の作成のみに集中することができる．

1.2　組版ソフト LaTeX の選択

また，本研究室では複雑な数式を扱うことが多いため，数式が美しく表示できる組版ソフトを選ぶことが必要不可欠である．そのため本研究室では，数式の扱いに長けた組版ソフト LaTeX を採用している．

以下に LaTeX による数式組版の例を示す．

$$r_i(t+h) = r_i(t) + h\frac{dr_i(t)}{dt} + \frac{1}{2!}h^2\frac{dr_i^2(t)}{dt^2} + \frac{1}{3!}h^2\frac{dr_i^3(t)}{dt^3} + \cdots$$

$$F(r) = -\frac{d}{dr}u(r) = 4\epsilon\left(12\frac{\sigma^{12}}{r^{13}} - 6\frac{\sigma^6}{r^7}\right)$$

$$\omega^2 = C\left(\left(\frac{1}{m}+\frac{1}{M}\right) \pm \sqrt{\left(\frac{1}{m}+\frac{1}{M}\right)^2 - \frac{4\sin^2 ka}{Mm}}\right)$$

このような複雑な数式を美しく組版処理できるのが LaTeX 最大の特徴であり，理系の文章作成において重宝されている理由の一つでもある．

1.3 従来の卒論執筆方法と提案方法の比較

本研究室の従来の卒業論文の執筆方法としては，エディターとして TAO を用い，そこに直接 LaTeX の命令を使って本文を書き，それを TeX 形式で書き出し，LaTeX を実行して pdf ファイルを得るという方法であった．しかし，初めて LaTeX を使う者にとって，LaTeX の命令というのは取っ付きにくく，複雑なものであり，LaTeX の命令をいちいち参考書などで調べながら本文を書いていくなど，文章作成自体に集中しづらいという不満な点もあった．

そこで今回作成したのが，Wiki2LaTeX フィルターである．従来の方法で不満のあった LaTeX 命令のうち，文章作成において頻出である命令を Pukiwiki の記法とその他に自分で設定した記号に置き換えることにより，わずらわしい LaTeX 命令の入力作業の軽減を実現した．

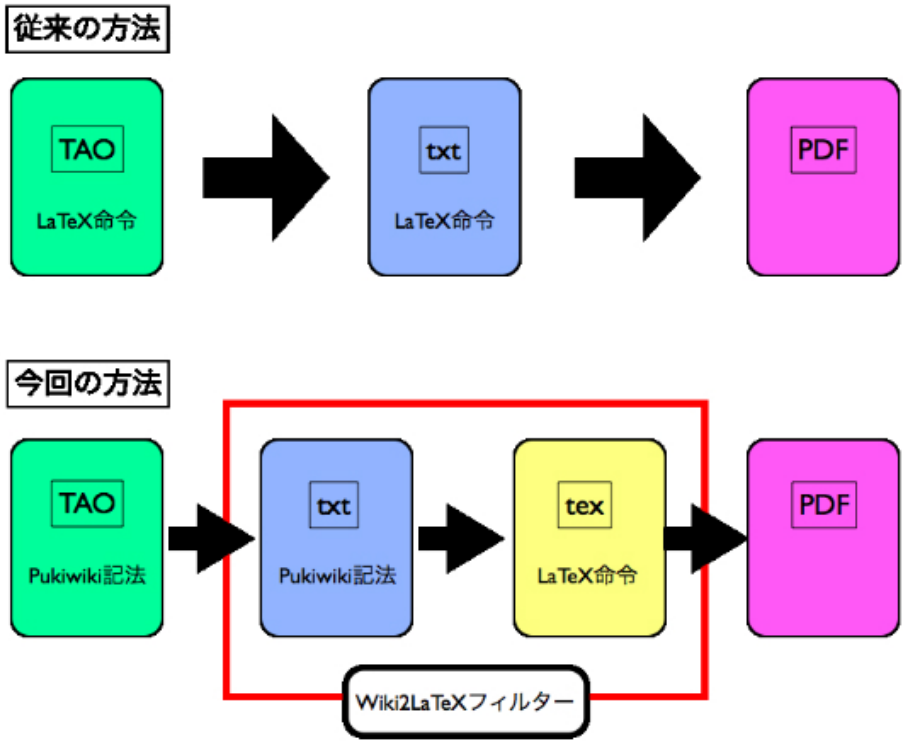

図 1.1: 従来の方法と提案方法の執筆方法比較

これにより，TAO での本文作成段階において，レイアウトも視覚的に捉えやすく，本文の作成に集中しやすい文章作成環境を実現した．以下に TAO での原稿作成段階においての，従来の方法と今回開発した新たな方法を用いて作成した原稿のサンプルをそれぞれ示す．

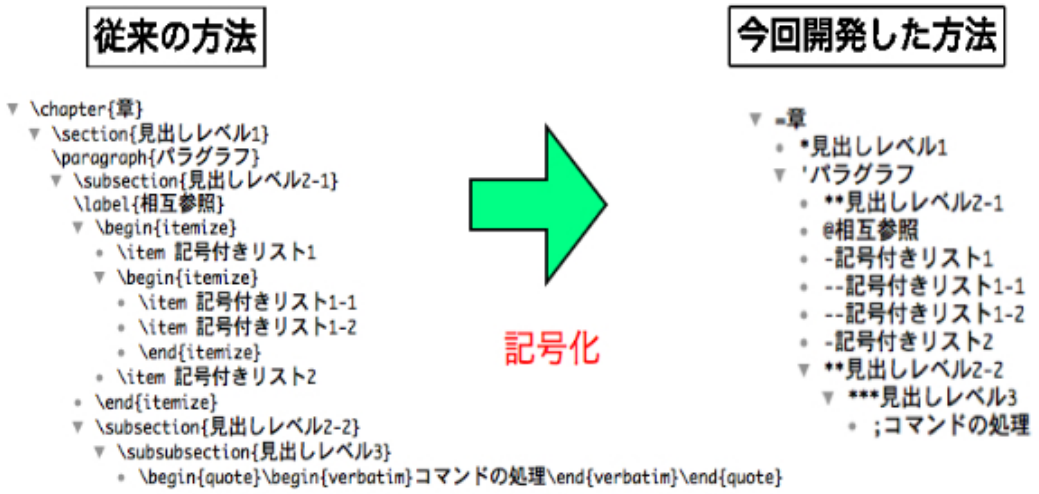

図 1.2: 従来の方法と提案方法の TAO 原稿比較

第2章　フィルターの開発手法

2.1　アジャイル開発手法による開発方針

今回フィルターを開発するにあたり，開発方針としてアジャイル開発手法を採用した．アジャイル開発 (agile development) 手法とは，適応型の計画を行い，タイムボックスを適用しながら反復型・進化型の開発を行い，段階的に出荷するものである．アジャイル開発手法のモットーは「変化を受け入れる」ことであり，戦略上重要なことは「機動性」である．

アジャイル開発手法の共通の見解として，アジャイルアライアンスの宣言と原則がある．アジャイルアライアンス宣言の具体的な例としては，

- プロセスやツールよりも，個人と相互作用
- 包括的なドキュメントよりも，動くソフトウェア
- 計画に従うよりも，変化に対応すること

などが挙げられる．これは左側の項目にも価値があるが，右側の項目の価値を重く見るということである．

またアジャイルアライアンス原則の具体例としては，

- 最も重要なことは，価値あるソフトウェアを早い段階から継続的に納品し，顧客を満足させることである．
- たとえ開発が進んだ後であっても要求の変更を歓迎する．アジャイルなプロセスとは，顧客の競争優位のために変化を生かすものである．
- 動くソフトウェアを頻繁に納品する．その間隔は２，３週間から２，３ヶ月で，短いほど良い．
- 動くソフトウェアこそが進捗を計る一番の尺度である．
- アジャイルなプロセスは，開発の持続可能性を促進するものである．スポンサー，開発者，ユーザーが一定のペースを無期限に保てなければならない．
- 簡潔さ (できる限り作業しないで済ませる術) こそ本質である．

- 最も優れたアーキテクチャ，要求，設計は，自己組織化チームから創発されるものである．
- チームは定期的に，より効果的な方法を検討し，自分たちのやり方を調整し適合させる．

などが挙げられる [7].

今回フィルターを開発するにあたって，アジャイル開発手法の方針に則り，まずある程度の実用的な機能が実装された動くプログラムが完成した段階でそれを０タイプとし，そのプログラムと使い方のマニュアルを研究室内の学生に配布し，実際の卒論作成に使用してもらう．その後使用していて，不具合や不満に感じる点をフィードバックしてもらい，それを元に改善改良をする．この流れを何度も繰り返し，よりユーザにとって使いやすいフィルターを開発していく．

2.2 使用プログラミング言語 Ruby

Ruby とは，まつもとゆきひろ氏により開発されたオブジェクト指向スクリプト言語である．一般的に，Ruby は数値計算には向いていないと言われる．実際に，Ruby は C や Fortan などの他の言語と比べ，実行速度などの面で遅い．しかし，Ruby の言語は，単純な構文で書かれており，可読性に優れている．さらにコンパイルを必要としないインプリンタ方式を採用しているため，実行の手間が少ない．加えて，高機能なスクリーンエディタとして有名な Emacs と併用すれば，構文に応じてスクリプトが色分けされるので．開発環境が非常に良いものとなる．本研究では，すべてのプログラムを Ruby を用いて作成した．

2.3 Wiki2LaTeX フィルター開発の雛形

2.3.1 PukiWiki とは

PukiWiki というのは PHP 言語で実装された Wiki の一種で，Wiki の中でも相当メジャーな部類に入る．

Pukiwiki の文法はプレーンテキストをベースに，直感的な記述ができるようになっている．見出しは大きい方から「*」「**」「***」で表現され，今回作成したフィルターでは見出しレベル３（***）までサポートしている．なぜなら対応する LaTeX の見出しレベル命令が３ (subsubsection) までしか存在しないためである．

また箇条書きは,「-」または「+」を使って記述する.「-」が記号付き箇条書き,「+」が番号付きの箇条書きである．箇条書きの一項目は必ず一行で書かなければならず，複数行にわたる項目はサポートされていない.「–」「—」のように記号の数を増やせばネスト (入れ子) した箇条書きも記述可能である．

また定義リストも記述可能となっている [1].

2.3.2 pukipa.rb

自作フィルターの雛形として pukipa.rb を参照した．pukipa.rb は，Pukiwiki の独自文法を HTML に変換する Parser である．Parser とは，入力データを受け取って，認識可能なパターンを抽出するソフトウェアである [6].

以下に pukipa.rb のソースコード上での呼び出し関係を図で表現したスタティックコールグラフを示す.

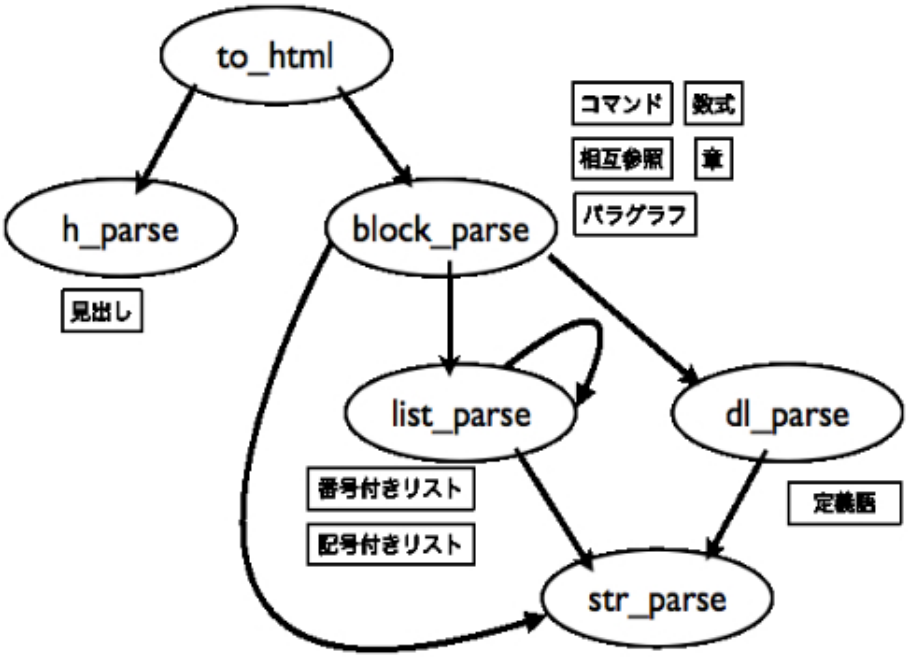

図 2.1: pukipa.rb のスタティックコールグラフ

このスタティックコールグラフより，pukipa.rb の処理方法は読み込んだソースデータを見出し部分，箇条書き部分，引用部分などそれぞれの機能ごとに正規表現で分岐させ，機能ごとに HTML へと変換する関数を作成して変換していることが分かる.

この構造は変換前の wiki と変換後の HTML の対応が視覚的に捉えやすく，今回のように HTML 以外のフォーマットに改良する際に効率的に作業を進められる構造である.

そこで，この雛形フィルター pukipa.rb の構造解析より,「プログラム全体の構造は変更せず，各々の関数部分の HTML タグを出力している部分を LaTeX 用に変更すれば良い」という方針が導かれた.

2.4 雛形フィルター pukipa.rb の改良

pukipa.rb を参考にして，Pukiwiki 記法で書かれた txt ファイルを LaTeX 形式に変換するフィルターを作成した．具体的には，まず HTML と LaTeX の主要なコマンドの対照表を作成し，それを元に pukipa.rb における HTML の出力部分をそれぞれ対応した LaTeX 命令を出力する形に変更した．

表 2.1: HTML タグと LaTeX 命令対照表

項目	HTML	Latex
見出し	 見出し 	\section 見出し
印付き 箇条書き	 その 1 その 2 	\begin{itemize} \item その 1 \item その 2 \end{itemize}
見出し付き 箇条書き	<dl> <dt>見出し</dt><dd>内容</dd> <dt>見出し</dt><dd>内容</dd> <dl>	\begin{description} \item[見出し] 内容 \item[見出し] 内容 \end{description}
引用	<blockquote> 引用文 </blockquote>	\begin{quote} 引用文 \end{quote}

テキストを
そのまま出力
	<pre>
	テキスト
	</pre>
		\begin{verbatim}
		テキスト
		\end{verbatim}

また，元の pukipa.rb ではサポートされていなかった章，パラグラフ，数式処理とコマンド処理は，元の TAO の原稿作成段階で「先頭にそれぞれに対応させた記号を付ける」という規約を設定し，正規表現を用いてその部分をマッチングさせて LaTeX 形式へと変換できるように改良し，機能を追加した．以下にそれぞれの作業過程を詳述していく．

2.4.1 記号割振りの再編

HTML タグと LaTeX 命令の対照表を作成し，pukipa.rb を改良した後，今度はそれぞれの LaTeX 命令に対応する記号割当を行った．

見出し，記号付きリスト，番号付きリスト，定義語に関しては，元の雛形とした pukipa.rb のデフォルトと同じ記号を用いたが，それ以外に新たに追加した機能については，自分で記号を割り振らなければならない場面があった．

そこで，まずは使用可能な記号を全て洗い出すことから始めた．LaTeX にはその規約によって使ってはならないとされている記号がいくつかある．具体的には#,$,%,&,_,{ },￥,^,～, ＜, ＞,|の 13 個の記号である．また，上記のように pukipa.rb のデフォルトでは*,-,+,:の 4 個の記号が既に使用されていたため，これら合計 17 個の記号以外の記号の中から記号を割り振らなければならないことを，使用可能な記号の洗い出しから得られた [8].

つまり，使用可能な記号は (!,?,,,( ),.,/,;,=,@,[ ],’), 0～9 , A～Z , a～z であった．このうち，数字やアルファベットに関しては，地のテキストや LaTeX 命令の中でも用いられるため，いくら機能的に問題がないとしても視覚的に紛らわしく，トラブルや LaTeX のエラー誘発の原因になりかねないので使用することは避けた．つまり実質使用可能な記号は最終的に (!,?,,,( ),.,/,;,=,@,[ ],’) の 13 個に絞られた．

表 2.2: 使用可能な記号一覧

Pukiwikiの主要なタグ	* + - > :
Latexにおいて使用不可な記号	# $ % & _ { } ¥ ^ ~ <> |
自作フィルタにて使用可能な記号	! ? , ( ) . / ; = @ [ ] ‘ 0~9 A~Z a~z

今回新たに追加した機能は，章，段落，数式の処理，コマンド処理，相互参照のラベル付け，コメントアウトの 6 つである．以下にデフォルトの見出し，記号付きリスト，番号付きリスト，定義語の記号とともに，記号と LaTeX 命令の対応一覧表を示す．

表 2.3: 記号と LaTeX 命令対応一覧

記号	対応するLatex命令	Latex命令の意味
*	\section{ }	見出し
-	\begin{itemize} \item \item \end{itemixe}	記号付きリスト
+	\begin{enumerate} \item \item \end{enumerate}	番号付きリスト
:	\begin{description} \item[語: 意味] \item[語:意味] \end{description}	定義語
?	\[\]	数式の処理
;	\begin{quote}\begin{verbatim} \end{verbatim}\end{quote}	中央揃えそのまま出力
=	\chapter{ }	章
@	\label{ }	相互参照ラベル付け
‘	\paragraph{ }	段落
%	%	%以下をコメントアウト

2.4.2 デフォルト機能の LaTeX 化

元の雛形フィルター pukipa.rb のデフォルト機能の見出し，記号付きリスト，番号付きリスト，定義語に関して，HTML タグの出力部分を LaTeX 命令の出力へと改良した．

具体例として，見出し部分の改良について詳述する．まず見出しの HTML タグと LaTeX 命令の対応は以下のようになっている．

表 2.4: 見出し HTML タグ・LaTeX 命令対応表

HTML タグ	対応する LaTeX 命令
見出し 1	\section{見出し 1 }
見出し 2	\subsection{見出し 2 }
見出し 3	\subsubsection{見出し 3 }

これを元に見出しの HTML タグ<h1>見出し 1 </h1>の部分は\section{見出し 1 }，<h2>見出し 2 </h2>の部分は\subsection{見出し 2 }，<h3>見出し 3 </h3>の部分は\subsubsection{見出し 3 }の出力になるように書き換えた．また補足事

項として，LaTeX では深さ 3 つまり subsubsection までの LaTeX 命令しか用意されていないため，深さ 4 以上の見出しが指定された場合には，プログラム実行時ターミナル上に「****はサポート外」というエラーメッセージが表示されるようにした．

実際のプログラム該当部の改良前と改良後を以下に示す．

改良前

```
# h(見出し) のパーサ
def h_parse(str)
  str.chomp!

  h_regexs = {
    '*'    => 'h' + @h_start_level.to_s,
    '**'   => 'h' + (@h_start_level + 1).to_s,
    '***'  => 'h' + (@h_start_level + 2).to_s,
    '****' => 'h' + (@h_start_level + 3).to_s,
  }
  h_regexs.each do |tmp_regex,prefix|
    regex = Regexp.new('^' + Regexp.escape(tmp_regex) + '([^*])')
    if str =~ regex
      str.gsub!(regex,"¥¥1").gsub!(/^¥s+/,'')
      str = "<%s>%s</%s>" % [prefix,str_parse(str),prefix]
      break
    end
  end
  @log.debug "inline:¥n%s" % str
  str
end
```

改良後

```
# h(見出し) のパーサ
def h_parse(str)
  str.chomp!
  if str=~/\*\*\*\*/ then
    $stderr.print str+"はサポート外\n"

  else
    error_message = 'No ****!'+"\n"
    h_regexs = {
      '*'    => '\section',
      '**'   => '\subsection',
      '***'  => '\subsubsection',

    }
    h_regexs.each do |tmp_regex,prefix|
      regex = Regexp.new('^' + Regexp.escape(tmp_regex) + '([^*])')
      if str =~ regex
        str.gsub!(regex,"\\1").gsub!(/^\s+/,'')

        str = "\%s{%s}" % [prefix,str_parse(str)]
        break
      end
    end
    @log.debug "inline:\n%s" % str
    str
  end
end
```

図 2.2: 見出し変換のプログラム該当部

2.4.3 新たな機能の追加

フィルターの雛形として採用した pukipa.rb でサポートされていたデフォルト機能の LaTeX 化の他に，新たに章 (chapter), 段落 (パラグラフ)，数式の処理，コマンド処理，相互参照のラベル貼り，コメントアウトの 6 つの機能を追加した．

それぞれの機能の記号の割振りとしては, 使用可能な 13 個の記号 (節 2.4.1，図 2.2 参照) の中から割当を行った．

具体的には，章には=, 段落には’, 数式の処理には?, コマンド処理には;, 相互参照ラベル貼りには@, コメントアウトには%を割り当てた．

これに伴い，プログラム内に新たな関数の部分を追加した．具体的には`block_parse`内 (スタティックコールグラフ 図 2.1 参照) に以下の部分を追加した．

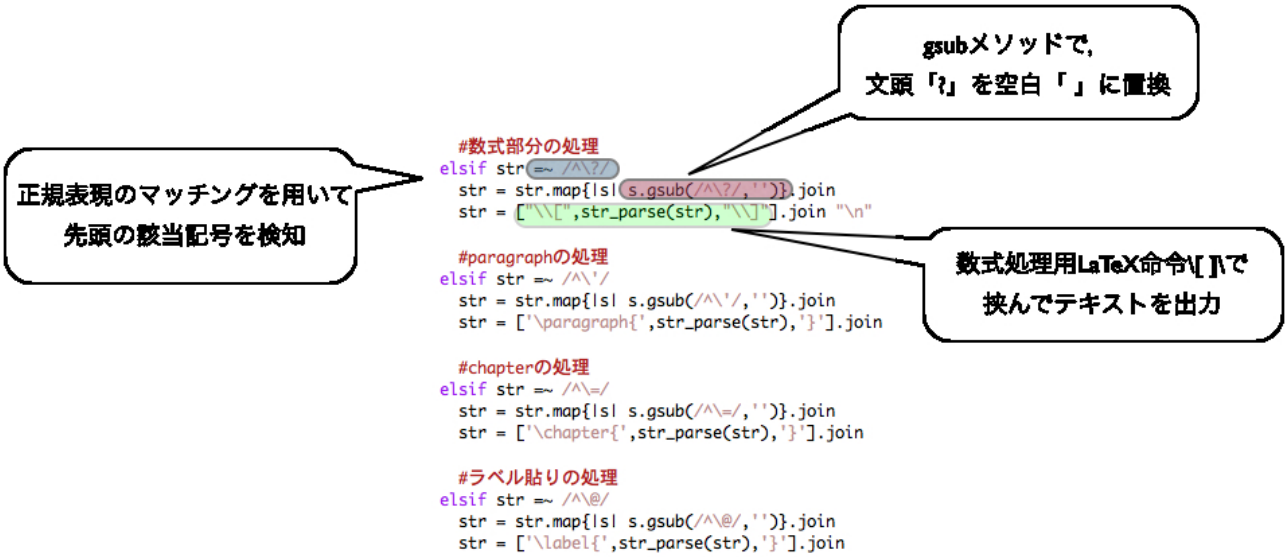

図 2.3: 追加機能のプログラム該当部

第3章　Wiki2LaTeXフィルターの使用結果および改良点

3.1　Wiki2LaTeXフィルターの使用結果

WikiLaTeXフィルターの開発により，TAOでの原稿作成段階において記号を用いるだけで，レイアウトを指定し，それをフィルターに通すことによりtexファイル形式への変換を可能にした．簡単なサンプル原稿を用いて，実行結果を変換の流れの順に従って以下に示す．

まずTAOにおいての原稿段階を示す．テキストの先頭に記号が付いていることが分かる．このファイルをtxt形式で書き出した後に，Wiki2LaTeXフィルターにかける．

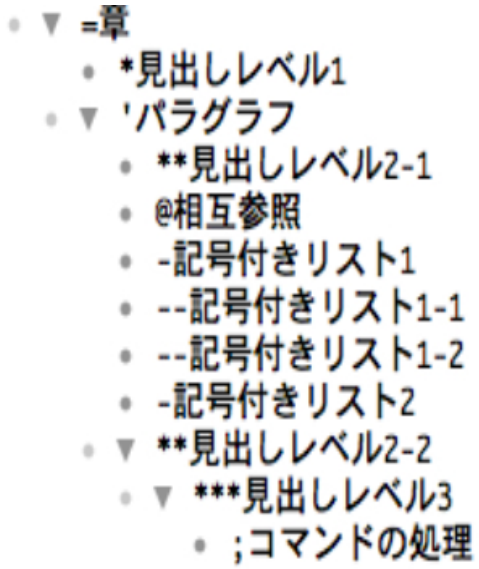

図 3.1: TAO原稿サンプル

txt形式で書き出したファイルを，Wiki2LaTeXフィルターにかけて得られたものが以下のtexファイルである．TAO段階で記述した記号が，それぞれLaTeX命

令に変換されていることが確認できる．この tex ファイルを LaTeX 実行し，PDF ファイルを得る．

```
\begin{document}
\chapter{章}
\section{見出しレベル1}
\paragraph{パラグラフ}
\subsection{見出しレベル2-1}
\label{相互参照}
\begin{itemize}
\item 記号付きリスト1
\begin{itemize}
\item 記号付きリスト1-1

\item 記号付きリスト1-2

\end{itemize}

\item 記号付きリスト2

\end{itemize}
\subsection{見出しレベル2-2}
\subsubsection{見出しレベル3}
\begin{quote}\begin{verbatim}
> コマンドの処理
\end{verbatim}\end{quote}
\end{document}
```

図 3.2: 原稿サンプル tex ファイル

tex ファイルを LaTeX 実行し得られた PDF ファイルが以下のものである．TAO の原稿段階で指定されたレイアウト通りに組版されており，美しい文書体裁が実現されている．記号を用いたことにより，TAO の原稿段階でもある程度レイアウトが視覚的に捉えやすくなったこともこの結果より確認できる．

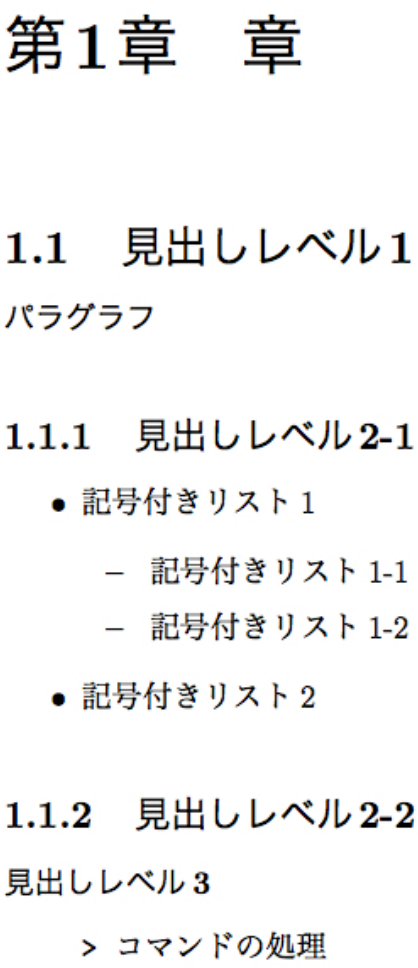

図 3.3: 原稿サンプル PDF ファイル

以上のように記号で指定した LaTeX 命令は Wiki2LaTeX フィルターによりきちんと変換されていることが確認された.

3.2　不具合点の改善

0 タイプをリリースし，ユーザからフィードバックをもらった結果，以下の不具合が発見された.

- URL が表示できない

URL が表示できないという不具合をユーザから指摘されたため，その点を改善した．プログラムを見直した結果，雛形フィルターとして採用した pukipa.rb の HTML におけるハイパーリンク機能が原因であることが分かった.

そこで，該当部のプログラムを削除することにより，フィルターを通しても URL が正しく表示できるように改善した．

3.3 ユーザの要望による改良

3.3.1 ユーザからの要望

アジャイル開発手法の方針に則り，ユーザからもらった要望を元に，フィルターを改良した．
要望としては以下の要望があった．

- 複数のフィルターの実行を，一括で処理できるようにしてほしい．
- 不要な中間ファイルがディレクトリー内に増えないようにしてほしい．
- LaTeX のヘッダーを自分で編集できるようにしてほしい．

3.3.2 組み込み関数 system による一括実行

0 タイプシステムのリリース後，ユーザからもらった要求に中に，フィルターが複数あるため実行時に順々に実行しなければならないのは，順番を覚えるなど手順に気を取られて集中しづらく，かつ面倒であるという意見があった．そこで，組み込み関数 system を用いて最終的な pdf ファイルの open までを一回の実行で全て一括で行えるように改良した．組み込み関数とは，単純にコマンドを実行する最も単純な方法で，system メソッドはコマンドを起動して，それが終了するまで待つ．実際にはプログラム中では以下のように使用する [2].

```
> system "ruby defaultparse.rb paper.txt > paper.tex"
```

その際に最終的には不要である中間ファイルが一回の実行に伴い，いくつも生成されてしまうので，プログラム実行後，自動的に削除されるように `temp_file` を用いてこの問題を解決した．なお `temp_file` については節 3.3.3 (P.18) を参照．
実際のプログラムの動きとしては，以下のことが一括で行われている．例として `paper.txt` という txt ファイルを引数とし，コマンドオプションを指定しなかった場合の実行例を以下に示す．

```
> ruby stringcodechange.rb paper.txt

> ruby enn2backslash paper.txt > temp_file
```

```
> ruby defaultparse.rb temp_file > paper.tex

> platex paper.tex

> dvipdfmx paper.dvi

> open paper.pdf
```

以上のプログラムが system.rb というプログラムを一回実行するだけで，一括で実行されるようになった．以下が一括処理を可能にした system.rb 内でのプログラム実行の流れである．

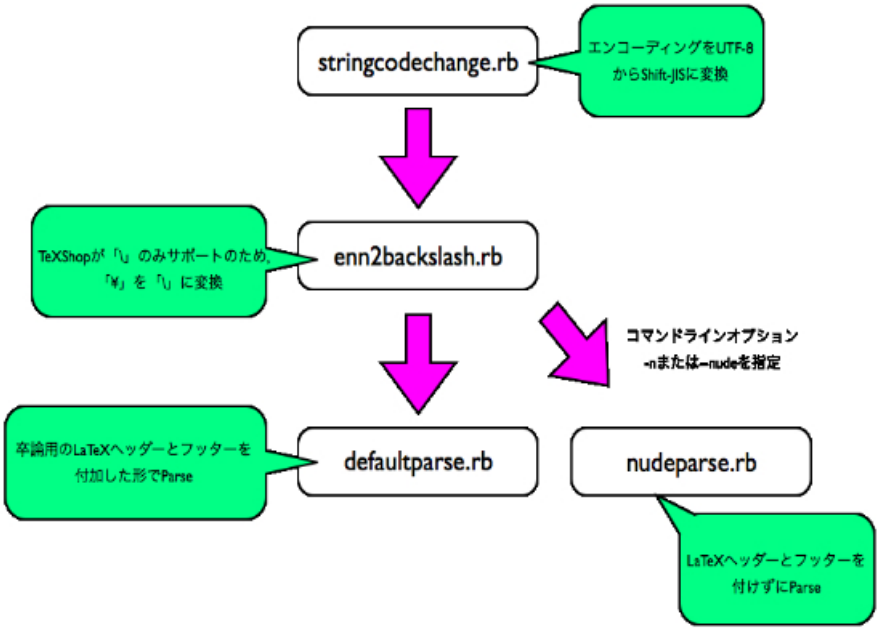

図 3.4: system.rb 内の流れ

3.3.3 一時ファイル作成による不要中間ファイル自動削除処理

一時ファイルとは，スクリプトの中で作成され，スクリプト実行終了時に自動で削除されるファイルである [2].

今回作成したフィルターを実行する中で，いくつもの中間ファイルが作成されるため，放っておくとディレクトリー内にどんどん不要なファイルが増えてしまい収集が付かなくなってしまう問題があった．

そのため，今回必要な中間ファイル以外はすべて一時ファイルとして実行終了後に自動削除されるようにした．なお具体的な必要とする中間ファイルは txt ファイル，tex ファイル，dvi ファイル，pdf ファイルである．

3.3.4 オプションパース (Option Parse) による書式柔軟性の向上

オプションパース (Option Parse) とは，様々な命令をコマンドラインで付け加えられるオプションである．今は GUI の時代ではあるといえ，コマンドラインにはコマンドラインの良さがある．本研究におけるコマンドラインオプションで言えば，実行する際に，様々なオプションが設定できることによって，ターミナル上でオプションを使えるという利点がある [4].

今回フィルターの実行時に使用できるコマンドラインオプションを作成した．system.rb というプログラムを実行時に，-n または–nude とコマンドオプションを指定し実行すると,TEX ファイルに変換する際に，ヘッダー部分とフッダー部分の LATEX 命令を出力せずにパースできる機能である．ヘッダーの具体例としては，デフォルトでは,

```
\documentclass[12pt,a4j]{jreport}
\pagestyle{plain}
\usepackage[dvipdfm]{graphicx}
\usepackage{here}
\usepackage[psamsfonts]{amssymb}
\begin{document}
```

を付加した形で TEX ファイルへと書き出すようになっているが，もしその他の書式や文字フォント，用紙のサイズを使用したい場合にいちいちプログラムを書き換えなければならないのは大変面倒である．そのためヘッダーとフッダーは付加せずに書き出すようにすることで，TAO での原稿作成段階で，柔軟に書式選択，用紙のサイズ設定，文字のフォント調節を手動でも設定可能にした．

ではなぜデフォルトの設定も必要なのか？という疑問が発生するが，今回のフィルターはあくまで卒業研究というある程度書式等があらかじめ決められた書物の執筆であることと，LATEX 初学者でも気軽に使ってもらえるシステムの実現にはデフォルトで設定なしに使えるものを提供する必要があると考えるためである．

第4章　総括

本研究室では, 卒業論文を執筆する際, アウトラインプロセッサー TAO に LaTeX 命令を直接埋め込み, それを txt 形式で書き出した後にコンパイルして PDF ファイルを得るというのが従来の手法であった. しかし, この方法は LaTeX の使い方, つまり LaTeX 命令や規約に気を取られてしまい, 文章構成や図, 表など肝心な文章の推敲のみに集中できないという不満点があった.

そこで今回「文章作成において頻出の LaTeX 命令の入力だけでも何とか軽減できないか」という課題の元に, それを実現するフィルター開発を行った. 手法として, 従来の方法で TAO に直接 LaTeX を埋め込んでいた代わりに Pukiwiki 記法を用いることによって文章の体裁を指定した. また Pukiwiki 記法を HTML のタグに変換する既存のフィルター pukipa.rb を雛形とし, Pukiwiki 記法を LaTeX 命令に変換するフィルターに改良した. これが今回開発した Wiki2LaTeX フィルターである. 今回開発した方法では,TAO に埋め込んだ Pukiwiki 記法を txt 形式で書き出し,Wiki2LaTeX フィルターに通してコンパイルして PDF ファイルを得た.

具体的な作業としては雛形フィルター pukipa.rb を解析し, その結果から，それぞれの機能の変換は各々関数が作られており，そこでそれぞれ個別に処理されていることが分かったため，pukipa.rb の全体の構造はそのままに,HTML タグの出力部分のみを LaTeX 命令を出力するように改良するという方針を決めた. その方針に基づき,HTML タグと LaTeX 命令の対照表を作成し, その表を元にプログラムを改良した. その結果 Wiki2LaTeX フィルターを使用することにより, TAO での原稿作成段階で使用した記号を LaTeX 命令へと変換可能にした.

また, 雛形フィルターのデフォルト機能 (見出し, 記 号付きリスト, 番号付きリスト, 定義語) に加えて, 章, 段落, 数式の処理, コマンドの処理, 相互参照のラベル 付け, コメントアウト機能を `block_parse` 関数部分に新たに追加することによって, より一層文章作成環境を充実させた.

その結果，今回開発した Wiki2LaTeX フィルターを使用することで,LaTeX 命令の記述にかかる手間が大幅に軽減され, 文章作成のみ集中しやすい環境を実現した. また原稿作成段階においても不要な LaTeX 表記を省略して表示されることから, 文章全体の体裁を視覚的にも捉えやすい文章作成環境が実現された.

現状では LaTeX 命令の全てを網羅するには到底及びはしないが，文章作成において頻出の LaTeX 命令の入力を簡略化できたことは,LaTeX 初学者のユーザーにとっては非常にストレスレスな環境の実現であると言える.

また Wiki2LaTeX フィルターの開発以外にも，一回のプログラム実行で，全ての

フィルター実行と LaTeX 実行および PDF ファイルの取得から open までを一括で行えるようになったことは，作業効率の向上という面において，非常に有用であったと言えるのではないだろうか．

しかし，理想を言うならば不満点もまだまだ存在することは否めない．現在使用している自作フィルターのプログラムは，広義には TAO を LaTeX に変換し，最終的に PDF ファイルを得ているが，実際には TAO に PukiWiki という道具を用いて文章の体裁を予め指定して書き出ししているのが現状である．しかし，最も理想的な形は，PukiWiki も使わず TAO の元々の階層構造を活かしたまま，LaTeX の命令に対応させて変換する形である．

現段階で挙げられる方法としては，TAO を txt 形式で書き出すのではなく，opml 形式で書き出し，XMLparser を改良し，TAO の階層構造を活かしたままでの LaTeX へ変換するという方法であるが，現時点では文章の体裁面 (見出し等以外の箇条書きや定義語，数式など) をいかに PukiWiki での指定なしに変換するのかという課題があるのが現状である．

また現在使用しているプログラムは，特に Class 等を使っておらず，プログラムとしては美しいプログラムとは言い難い．今回のように，元々HTML 用であった雛形プログラムを LaTeX 用に書き換えるという作業面においては，現状の整理されていないプログラムの方が改良しやすいというメリットはあるが，将来的にはデザインパターンの Observer パターンのような美しいプログラムへ改良していくことが今後の課題である．

参考文献

[1] 青木峰郎 著 「Rubyist Magazine 出張版 正しい Ruby コードの書き方講座」(株式会社 毎日コミュニケーションズ　2007).

[2] 青木峰郎, 後藤裕蔵, 高橋征義 著 「Ruby レシピブック 第 2 版 268 の技」(ソフトバンククリエイティブ株式会社 2007).

[3] 北浦訓行 著 「LyX 入門」(株式会社 技術評論社 2009).

[4] るびきち 著 「Ruby 逆引きハンドブック」(株式会社 シーアンドアール研究所 2009).

[5] デイビット・クロス 著 「Perl データマイジング データ加工のテクニック集」(株式会社 ピアソン・エデュケーション 2003).

[6] ラス・オルセン 著 「Ruby によるデザインパターン」(株式会社 ピアソン・エデュケーション 2009).

[7] クレーヴ・ラーマン 著 「初めてのアジャイル開発」(日経 BP 社 2004).

[8] 奥村晴彦 著 「LaTeX 2_ε 美文書作成入門」(株式会社　技術評論社 2004).

謝辞

本研究を遂行するにあたり，終始多大なる有益なご指導，及び丁寧な助言を頂いた西谷滋人教授に深い感謝の意を表します.

また，本研究の進行に伴い，西谷研究員の皆様にも様々な知識の提供，ご協力を頂き，本研究を大成することができました．最後になりましたが，この場を借りて心から深く御礼申し上げます.

付 録A　Wiki2LaTeX フィルター利用手引き

A.1　規約一覧

- 記号は必ず文頭に付ける
- 数式，コマンドを連続で記述する場合は，間に空ユニットを 1 行挿入する
- 見出しは深さ 3 (subsubsection)，リストは深さ 2 までのサポート
- 改行は return キーでの同ユニット内での改行ではなく，兄弟のユニットを使っての改行とする
  - 記号が先頭と見なされなくなってしまうため

A.2　TAO ファイルの書き出し

保存したい TAO のファイル上部のタブから，[ファイル] → [書き出し] を選択

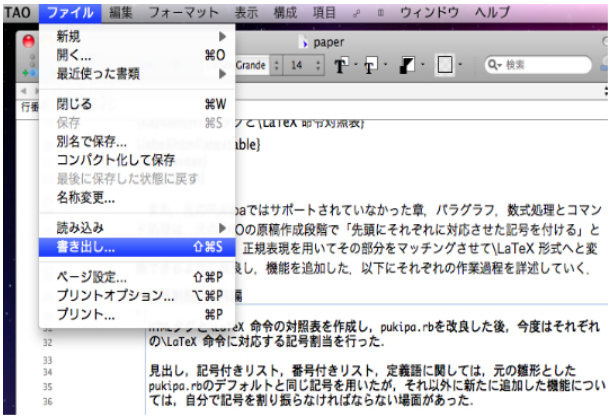

図 A.1: [ファイル] → [書き出し] の選択

保存先をフィルターを保存したディレクトリに指定し，形式を [テキスト形式], エンコーディングを [UTF-8] に指定して書き出しを実行する．この際，チェックボックスは何も選択していない状態になっていることも確認する．

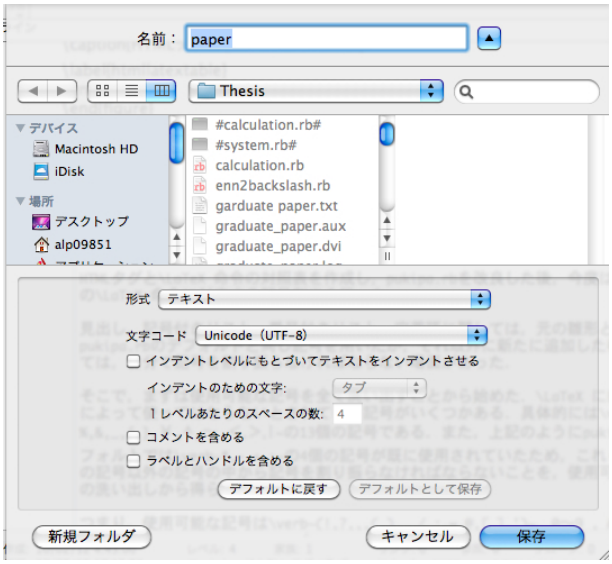

図 A.2: 形式，エンコーディングの選択

これにより，ファイル名と同じ名前の txt ファイルが得られる．

A.3 Wiki2LaTeX フィルターの使用手順

A.3.1 オプション

またオプションとして，以下のように-n または--nude を付けて実行すると LaTeX のヘッダーとフッターが付加されていない tex ファイルが得られる．

```
> ruby system.rb -n how_to_use_tao2latex.txt
```

```
> ruby system.rb --nude how_to_use_tao2latex.txt
```

またオプションの内容が知りたい場合は-h を指定して実行すれば説明を見ることができる．

```
> ruby system.rb -h how_to_use_tao2latex.txt
```

A.4　TAO での原稿の書き方

A.4.1　対応表

表 A.1: 記号と LaTeX 命令対応一覧

記号	意味
*	見出し
-	記号付きリスト
+	番号付きリスト
:	定義語
?	数式の処理
;	コマンド
=	章
@	相互参照ラベル付け
’	段落
%	コメントアウト

A.4.2　章

記号「=」を先頭に付け，続けてテキストを記述する．
これは本来の LaTeX 命令での\chapter{ }に相当する．

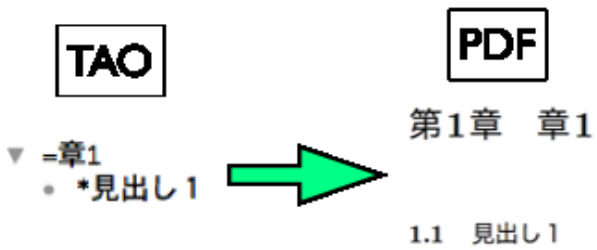

図 A.3: 章 (chapter) の変換例

A.4.3　パラグラフ

記号「’」を先頭に付け，続けてテキストを記述する．
これは本来の LaTeX 命令での\paragraph{ }に相当する．

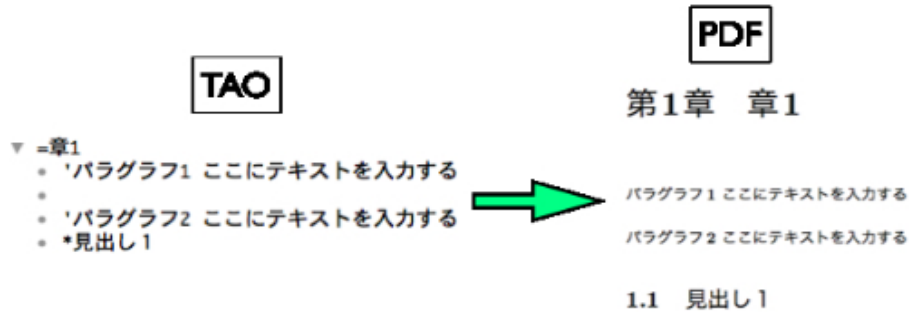

図 A.4: 段落 (paragraph) の変換例

A.4.4 見出し

記号「*」を先頭に付け，続けてテキストを記述する．
これは本来の LaTeX 命令での`\section{ }`に相当する．

表 A.2: 見出し対応表

記号	対応する LaTeX 命令
*	`\section{}`
**	`\subsection{}`
***	`\subsubsection{}`

図 A.5: 見出しの変換例

A.4.5 記号付きリスト

記号「-」を先頭に付け，続けてテキストを記述する．
これは本来の LaTeX 命令の`\begin{itemize} \item \item \end{itemize}`に相当する．

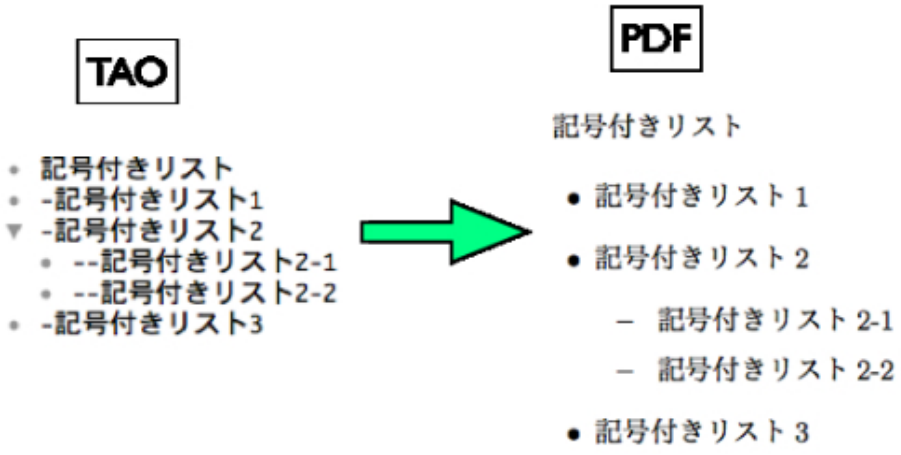

図 A.6: 記号付きリストの変換例

A.4.6　番号付きリスト

記号「+」を先頭に付け，続けてテキストを記述する．
これは本来の LaTeX 命令の`\begin{enumerate} \item \item \end{enumerate}`
に相当する．

図 A.7: 番号付きリストの変換例

A.4.7　コマンド

記号「;」を先頭に付け，続けてテキストを記述する
これは本来の LaTeX 命令の`\begin{quote}\begin{verbatim} \end{verbatim}\end{quote}`
に相当する．

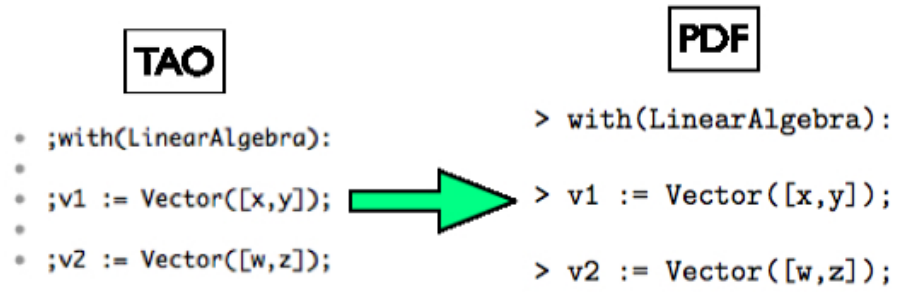

図 A.8: コマンド処理の変換例

A.4.8　数式

記号「?」を先頭に付け，続けてテキストを記述する
これは本来の LaTeX 命令の\[\] に相当する．

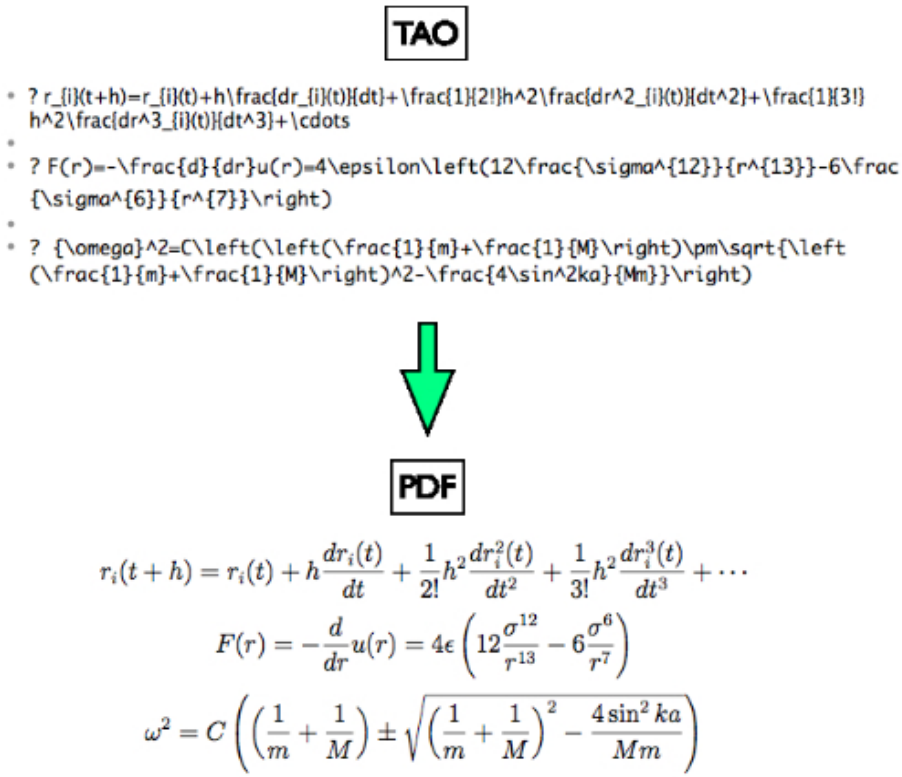

図 A.9: 数式の処理の変換例

A.4.9　相互参照

記号「@」を先頭に付け，続けてテキストを記述する
これは本来の LaTeX 命令の\label{ }に相当する．
相互参照とは,「第 5.3 節を参照」や「結果は 123 ページの図 10.5 のようになった」のように，ページ，章，節，図などの番号を入れることである．
参照したい部分は，出力する命令の直後にラベル (名札) を貼っておき，例えば節番号を出力したいところでは\ref{}と入力し，ページ番号を出力したいところでは\pageref{}と入力し参照する．[8]

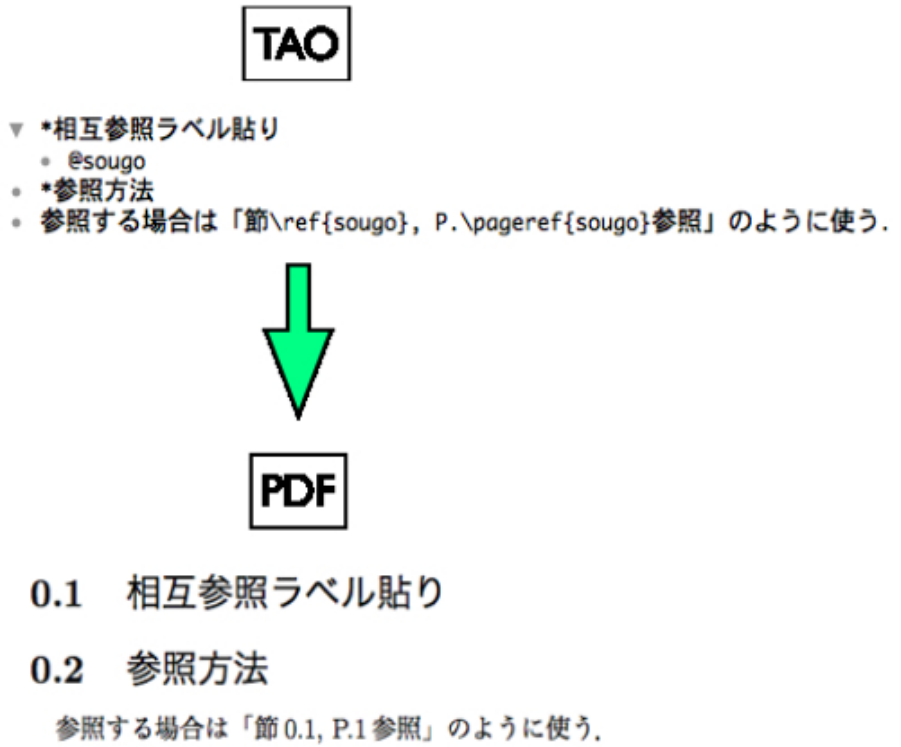

図 A.10: 相互参照ラベル貼りの変換例

A.4.10 定義語

記号「:」を先頭に付け，続けてテキストを記述する
これは本来の LaTeX 命令の`\begin{description} \item[語:意味]\item[語:意味] \end{description}`に相当する.

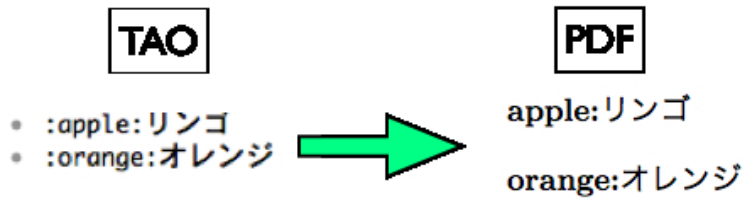

図 A.11: 定義語の変換例

A.4.11　コメントアウト

記号「%」を先頭に付け，続けてテキストを記述する