論文用 LaTeX テキストから発表用 LaTeX テキストへの自動変換（変換容易性とエディタ親和性について）

寒川　光, samukawa@sic.shibaura-it.ac.jp
芝浦工業大学システム理工学部数理科学科

- 筆者は，論文やマニュアルのような解説文章を，論文発表や勉強会でプレゼンテーションに使用できるスライドに変換するスクリプト texslide.pl を自作して使用している [1].
- 原稿が**1センテンス1レコード**で書かれていれば，スクリプトの処理は比較的単純である.
- 章や節のタイトルはすべてスライド用に移行する.
- 文章は指示文（%term）で選択されたものだけを移行し，箇条書きする.

- figure，table，equation 環境などで書かれた図，表，数式は，拡大してスライドの中央に置く．
- スライドを改ページする %newslide や，複数の数式や文章を 1 枚のスライドに並べるために %copystart と %copyend で挟むようにして，これらを発表用のスライドを眺めて試行錯誤的に作業する．
- この方法によると，論文用と発表用の 2 通りの原稿を 1 つのソーステキストで管理でき（シングルソース），改訂はそのファイルに対してだけ行えば両方に反映されるので，原稿の管理が楽になる．

- 大学で講義に使用する場合，講義ノートは大部分の文章を表示することが多いので，いったん全文章に %term 指定をして，不要なものを落とすほうが作業が楽なことに気付いた.
- そこで指示行を自動的に挿入するスクリプトを作成して，変換スクリプトの前に実行することで，自動変換に近づきつつある[a].
- 複数の数式が，1つないし2つの文章を挟んで連続する文脈は，それらを集めて並べるための指示行 %copystart と %copyend を挿入する（texinscopy.pl）.
- その後，全ステートメントに対し %term を入れ（texinsterm.pl），その出力を texslide.pl で変換している.

[a] 全部を表示するのであれば，講義ノートを GSview の拡大表示で使用することもできるが，式変形の過程の中に改ページが入る場合があり，スライドを起こしたほうが具合がよい.

- この簡便な方法では完全な発表用スライドを作ることはできないので，さらに1行だけのスライドを前後のスライドに含める texbrushup.pl なども作成した[a].
- このように対症療法的に，用途に合わせて必要な Perl スクリプトを作成している.
- 自作のスクリプトは，自分の原稿の書き方に強く依存しており，他の人に有用とは思われない.
- 文書処理プログラムの特徴はこのあたりにあるように思われる.

[a] これらを makefile で実行する．make で論文用，make slide でスライドができる.

- 例えば，別行立ての数式は \[と \] で囲む書き方もあるが，変換プログラムで扱いにくくなるので，必ず equation などの環境を使用している（parsability）.
- 多くの書き方の中から何を選ぶかは，テキストエディタとの相性によるところが大きいように思われる.
- 筆者は vim を使用している.
- 1 センテンス 1 レコード原則も，別行立ての数式をすべて数式環境で書くのも，vim を使用すると苦にならない[a].

[a] テキストエディタの機能が低い場合は，タイプ数を減らす目的でマクロをたくさん定義する方向に向かいやすいが，反対にマクロを使わず，文字数の多いコマンドも vim の機能を駆使して原稿の他の箇所からコピーすることが多い.

- このような書き方の傾向を，エディタとの親和性（affinity）と考えると，原稿の書き方の規制は難かしいように思われる．

変換の概要

seminar クラスを対象にデザインした指示行と変換スクリプト

- 変換対象となる環境を keyenv と記述する.
- 図（figure）[a]，表（table）[b]，数式（equation, displaymath, eqnarray, align, gather），箇条書き（itemize，enumerate，description），参考文献（thebibliography）を対象とし，他の環境をその内側に使うことの多い breakbox や，他の環境の中で使わせることの多い verbatim は除外してある.

[a] picture，pspicture は含まない.
[b] tabular は含まない.

- ソーステキストにこれらの環境を見つけると，\begin{slide} と \end{slide} とに囲んでそのまま移行（コピー）する．
- 左側が入力，右側が出力ファイルのレコードになる．

```
\begin{keyenv}
 XXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXX
\end{keyenv}
```

```
\begin{slide}
\begin{keyenv}
 XXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXX
\end{keyenv}
\end{slide}
```

- 指示行 %copystart と%copyend に囲まれたテキストも，\begin{slide} と \end{slide} で囲んでコピーする.

```
%copystart                  \begin{slide}
 XXXXXXXXXXXXXX              XXXXXXXXXXXXXX
 XXXXXXXXXXXXXX              XXXXXXXXXXXXXX
%copyend                    \end{slide}
```

```
\section{AAA}
bbbbbbbbbbbbbbb
%term
 XXXXXXXXXXXXXX
%term
 YYYYYYYYYYYYYY
%newslide
ccccccccccccccc
```

```
\begin{slide}
\begin{center} \Huge AAA \en
 \hrulefill
\begin{itemize}
 \item XXXXXXXXXXXXXX
 \item YYYYYYYYYYYYYY
\end{itemize}
\end{slide}
```

- Perl スクリプトはソーステキストの各レコードを読み込んだら，現在いる状態（環境の中か外かなど）に従って，読み込んだレコードを,「コピーする」「コピーしない」「\item を付けてコピーする」の 3 通りに処理する.

- 例えば keyenv 環境の中を読み込んでいるときは「コピーモード」にあり，この状態から \end{keyenv} を読むと，\end{slide} を入れて，次の keyenv 環境や%newslide 指示行，%copystart 指示行，\section などの章の始まりを探す「サーチモード」に状態遷移する.

- 「サーチモード」では読み込んだレコードに従って次の処理を行う.

- %term 指示のないレコード（上の例では bbbb や cccc），あるいは keyenv に指定されていない環境の始まりを読み込んでも，何もしないで次のレコードの処理に進む.
- %newslide 指示行を読み込むと，蓄積したキューを出力して，新たなスライドの処理に進む.
- \section などの章立てコマンドを読み込むと，蓄積したキューを出力してから，上の例のように，タイトルを表示するためにタイトルとフォントサイズを変数 $title と $titlefont に格納する.
- %term 指示行を読み込むと，「enforce モード」に状態遷移する.「enforce モード」では，読み込んだレコードをキューに蓄積して，「サーチモード」に戻る（%term の次のレコードが出力される）.
- で %copystart 指示行を読み込むと，蓄積したキューを出力して，「copy モード」に状態遷移する.「copy モード」では読み込んだレコードを出力するが，%copyend を読み込むと「サーチモード」に戻る.

- この処理では，それまでに読み込んだ情報（ソーステキストの文脈）で，そのレコードの処理を決定できるので，処理プログラムの形態は「フィルター的」で，古典的な awk でも記述できる（と考えられる）.
- フィルターは図 1 に示す入力行を加工して出力行とする処理が繰り返される.

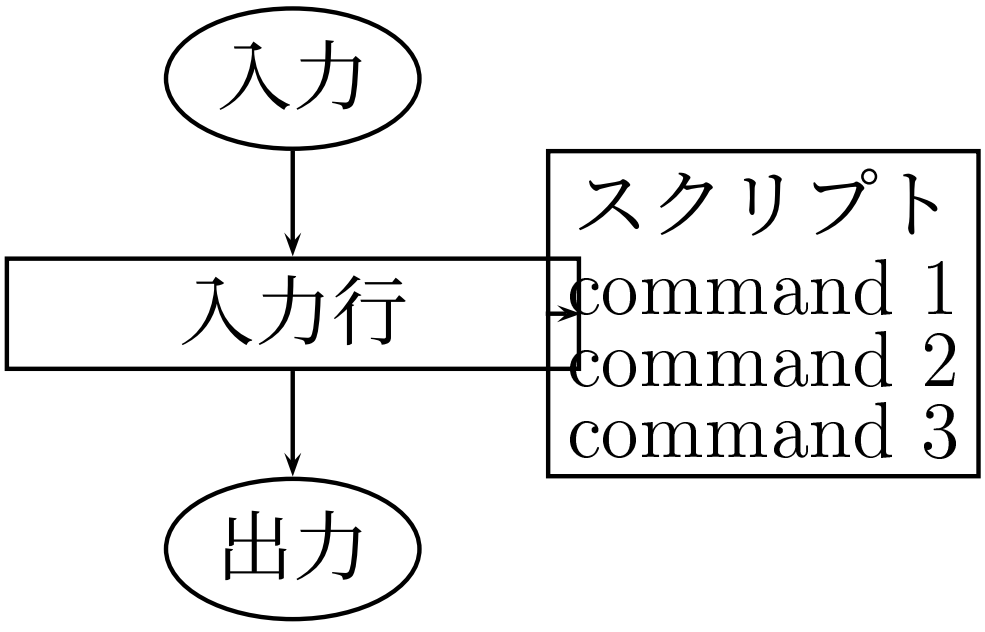

Figure 1: 入力行を加工して出力行とする処理（スクリプト言語の起点）

- しかし，スクリプトを最初に作成するとき，スライドの空きスペースを考慮して，%term 指定されたレコードをそのスライドに書くか，次のスライドに移るかを判定したかったので，出力レコードをキューに蓄積して，次のレコードを読み込む方式を採っている．

- キューに蓄積されたレコードは，%newslide 指示行，%copystart 指示行，\section などを読み込んだときで，上記の箇条書きでは下線で示した.
- 出力するとき，$title 変数にタイトルが保存されていれば，先にタイトルを出力してから蓄積されたレコードに \item を付けて出力する.

- このほか，\begin{document} までのプリアンブル部のコピーや，\includeonly で入力ファイルを切り替える処理なども含んでいるので，やや複雑なスクリプト（240 行）になっている．

指示行の自動的な挿入

- 環境のすべてのレコードに %term を指定するスクリプト **texinsterm.pl** を書くのは難しくない.
- しかしそのような処理だけでは，複数の数式環境が，1 レコードあるいは 2 レコードを挟んで連続する文脈では，数式，文章，数式，文章とスライドが細切れになる.
- 次の例は数値解析の数値積分の説明の一部である.

- これをそのまま数式環境と文章を独立したスライドに移行すると，数式，文章，数式，文章，数式，文章，数式，文章と 8 枚のスライドに分かれ，説明不能になる[a].

[a]数式をはじめから \[と \] の別行立てで書くと，この式を他から参照するラベルを付けられない.

これは

$$F(a) = F'(a) = \ldots = F^{(n-1)}(a) = 0$$

$$F(b) = F'(b) = \ldots = F^{(n-1)}(b) = 0$$

ならば満たされる．したがって $2n$ 次の多項式

$$F(x) = (x-a)^n (x-b)^n \tag{1}$$

はこの条件に適する．したがって C を任意の定数として

$$P_n(x) = C \frac{d^n}{dx^n} (x-a)^n (x-b)^n \tag{2}$$

が求める多項式である．区間が $[-1, 1]$ のとき

$$P_n(x) = \frac{1}{2^n \cdot n!} \frac{d^n}{dx^n} (x^2 - 1)^n \tag{3}$$

がルジャンドルの球関数である．この関数は，球面調和関数の構成要素で，球面上の関数を展開するとき使われる．

- このような文脈では，はじめの数式環境の前に %copystart，最後の文章の後に%copyend を挿入するスクリプト texinscopy.pl を作成し，texinsterm.pl では%copystart と %copyend に挟まれた部分は除外する．

- **texinscopy.pl** では，数式環境が終わってから次の数式環境が始まるまでに何レコードあるかを数えなくてはならない（数行先のレコードまで読んでから処理を決める必要がある）.
- そこで，読み込んだレコードをキューに蓄積しながら，数式と次の数式の間のレコードを数えて，2行以下で次の数式が来たら，キューに貯める操作を継続し，3レコード以上になったり，別の環境や章（\section など）が来たらスライドを閉じる[a].

[a] このように，単純に1レコードを読んでは処理するフィルター的なスクリプトではない（読み込んでいるレコードと，出力するレコードとの間にズレが生している）.

- このレコード数だけで制御する方法は，長い数式展開に対して 1 枚に収まりきらないスライドを生成してしまう.
- 適当な位置に %newslide を挿入するのと同等の処理を前処理で間接的に行うために，元のテキストに %だけが 10 行以上続いた場合は，そこで%copyend を挿入する処理を加えた.
- テキストエディタ vim を使用していると，同じ行を繰返すのはキー操作が簡単で，入力ミスがない.

- この対処によっても，1 行だけのスライドができることもあり，これに対処する方法が見つからなかったので，**texbrushup.pl** を作成して，1 行だけのスライドを前のスライドに含めるようにした.
- texbrushup.pl では，1 回目のパスで全レコードを読み込み，各レコードがコメント行かそれ以外（Rcd）かを調べる.
- コメント行は，%newslide，%term，%copystart，%copyend の指示行ならそれぞれ Pns，Ptm，Pcs，Pce を，それ以外なら other をキューに記録し，メインループではこのキューを参照しながら数行先を調べて処理を進める 2 パス方式とした.

- 結局，texinscopy.pl，texinsterm.pl，texbrushup.pl の 3 段階の前処理を行ってから texslide.pl による変換を行っている．
- 前処理用の 3 つのスクリプトは texslide.pl に比較すると簡単なスクリプトで済む．

powerdot への変換

- 今回の発表を機会に，LaTeX の標準のディストリビューションに含まれるようになった powerdot クラスも試したので紹介する.
- powerdot では，パワーポイントのような，箇条書きを \pause コマンドにより \item ごとに表示したり，しおりを使用して，離れたスライドに飛ぶ機能が魅力的である.

- また，複数の背景が用意されていて，写真撮影に適した暗い背景色を選ぶこともできる．
- seminar クラスと比較すると，原稿が 1 つのソーステキストで閉じている場合は，このようなプレゼンテーション効果の違いにしか過ぎないが，プレゼンテーション中にクリックによって別の資料に移る操作ができるので，動画を表示したり，Web ページを表示したりする場合は便利である．

しおりの利用

- section，subsection，subsubsection の引数（章，節，項のタイトル）を powerdot が，目次のように扱うことで，スライドの左の柱に並べて，そこをクリックすると，そのスライドに飛ぶ機能がついた.
- これは pdf についてサポートされる.

- 変換スクリプト **texslidot.pl** はほとんど texslide.pl と同じであるが，section などの処理を次のように変更している．

```
\section{AAA}
%term
 XXXXXXXXXXXXXX
%term
 YYYYYYYYYYYYYY
%newslide
```

```
\begin{slide}{AAA}
\begin{itemize}
 \item XXXXXXXXXXXXXX
 \item YYYYYYYYYYYYYY
\end{itemize}
\end{slide}
```

- ただし，スライド中に verbatim 環境があるときは，\begin{slide}{タイトル} に，オプション引数 “[method=direct]” を加える．

```
\section{AAA}
\begin{verbatim}
```

```
\begin{slide}[method=direct]{AA
\begin{verbatim}
```

- この処理のために，texslidot.pl が完了してから，その出力ファイルを読み，何枚目のスライドに verbatim 環境があるかを 1 回目のパスで調べ，メインループではその情報を参照しながら，\begin{slide}{ タイトル } を，\begin{slide}[method=direct]{ タイトル } に変換するポスト処理 **texslidotpost.pl** を作成した[a].

[a]powerdot は LaTeX の verbatim 環境とは異なり，行末にゆくと折り返したり，フォントサイズによっては verbatim にならないなどの奇妙な現象が見受けられた.

wideslide の利用

- 省略時の背景は，スライドの左側にしおりの欄があり，全画面を図や式の表示に使用できない[a].

[a]図がスライドからはみ出す現象を抑えるために，texslide.pl と texslidot.pl では，指示行 %SlideReplace によって，pspicture 環境中の unit を変更できる.

- そこで texslidotpost.pl に，スライド中に align 環境や figure 環境（1 カラム目から \begin{figure}）があるときは，\begin{slide} を \begin{wideslide} にする機能を追加した.

pauseの利用

- 箇条書き（itemize，enumerate，description 環境）に，指示行を続けて \begin{itemize}%Pause のように記述されたときは，\item レコード \pause を付加して出力する（texslidot.pl）.
- この指定により，箇条書きのアイテムを，1つずつ表示しながら説明できる.

makefile

- 現行では，seminar クラスと powerdot クラスの併用を考えている．

- したがって次の makefile を使用して，“make slide” で seminar 用の，“make pod” で powerdot 用のスライドを起こせるようにした.
- powerdot 用には，seminar 用の texslide.pl に変更を加えた texslidot.pl を作成し，\includegraphics のように powerdot ではエラーになる機能を除外する **texskipslide.pl** を前処理として加え[a]，後処理として texslidotpost.pl を加えた 6 回の変換を行っている.
- 2004 年に最初の texslide.pl を作ってから，パッチワークの積み重ねでこのような姿になった.
- 一度リフレッシュしたいと思うが，文書処理技術と LaTeX 2ε 自身が進化を続ける以上，そのタイミングもなかなか訪れなかった[b].

---

[a] %SKIPslidemaking と %SKIPendslidemaking 指示行で挟まれた部分は除外する（%copystart と %copyend の逆の処理）.

[b] Perl も直交性の低いプログラミング言語で，自分で書いたコードでも，あまり読みたいと思わない，という理由もある.

```
TC = platex
DC = dvipsk
BIN = /cygdrive/c/usr/bin/

text: UnixCommand.dvi
	$(DC) UnixCommand.dvi
UnixCommand.dvi: UnixCommand.tex
	$(TC) UnixCommand.tex
slide:
	perl $(BIN)texinscopy.pl UnixCommand.tex >
	perl $(BIN)texinsterm.pl UnixCommandCp.tex
	perl $(BIN)texbrushup.pl UnixCommandTm.tex
	perl $(BIN)texslide.pl UnixCommandBp.tex >
	$(TC) slUnixCommand.tex
	$(DC) -t a3 slUnixCommand.dvi
pod:
	perl $(BIN)texskipslide.pl UnixCommand.tex
	perl $(BIN)texinscopy.pl UnixCommandSk.tex
	perl $(BIN)texinsterm.pl UnixCommandCp.tex
	perl $(BIN)texbrushup.pl UnixCommandTm.tex
	perl $(BIN)texslidot.pl UnixCommandBp.tex >
	perl $(BIN)texslidotpost.pl spUnixCommand.t
	$(TC) sqUnixCommand.tex
	$(DC) sqUnixCommand.dvi
```

- vim では ":%s/UnixCommand/PlotBasic11/g" のコマンドで処理するファイル名を PlotBasic11 に変更ができる.

文書処理プログラムの特徴

- 変換プログラムとそれを使用する文書作製の相互依存はたいへん強いものになっている.
- その事情は次のとおり.

- 書き方の自由度が多すぎて，すべてのケースに対応する変換プログラムを書くことは（筆者には）不可能である．
- 変換プログラムを書くことは，自分用の限られた文書を対象とするので比較的簡単で，**仕様を自分で決める楽しさ**も味わえる．
- 限定された変換機能でも，その場しのぎの「逃げ方」（durty trick）が見つかることが多い．
- 変換プログラムの作製中も，そのような「逃げ方」を思いつき，それによって変換プログラムの作成を簡単にできる．
- 変換プログラムの機能を知っているので，変換のアルゴリズムを意識した文書を書いている．

- 「逃げ方」の例として，複数の環境がネストした場合，keyenv 環境のスキャンは，1 カラム目から \begin{keyenv} が始まる方式で訊いているので，itemize 環境の中に itemize 環境がある場合は，内側の itemize を 1 カラム字下げして書くようにしている．

- したがって，keyenv の環境でも，変換対象から外したい場合は，字下げする．
- また，講義ノートにプログラム例を載せる場合は，breakbox，quote，verbatim の 3 つの環境をネストさせているが，keyenv には quote だけを入れている．
- こうすることで，スライドには quote 環境とその内側の verbatim 環境で記述されたプログラムが取りだされ，breakbox の枠はスライドでは落とされる（他の文字がないので，枠は必要ない）．

- 1 センテンス 1 レコード原則の例外として，脚注が \footnote{ 文章 1．文章 2．} のように複数のセンテンスがあっても，1 レコードにしておかないと，変換後の LaTeX がエラーになる[a].

[a] ルールを作れば，それに対する例外ルールが必要になる．「青信号なら交差点を進行してよい」が，「緊急自動車がサイレンを鳴らして来たら止まれ」．それを無視して進行した人間には「罰則」が課せられる．

規制の勧め

- 前章で紹介した，スライドの自動生成は，変換用スクリプトプログラムが複雑な処理を行うことで達成されたものではなく，簡便な変換プログラムに適合するソーステキストの書き方を守ることで実現されたと考えるべきである．
- ここで，ソーステキストを1センテンス1レコード原則で書くようになった周辺の事情について紹介し，LaTeX 2$_\varepsilon$を汎用的に使いまわすために，規制した使用法を提案する．

ソーステキストの書き方の自由度

- 1センテンス1レコード原則でない例が普通と思われる.

- ソーステキストの可読性のために，読点で行替えしたり，行頭にタブを入れるプログラマが多い．

- 数式に至っては，大変自由度が多い.
- 次の表示を得るソーステキストの例を示す.

被積分関数 $f(x) = \dfrac{1}{\sqrt{1-x^2}}$ を -1 から 1 まで，台形公式，シンプソン公式，ガウス・ルシャンドル公式で積分して精度を比較する．このとき $x = -1, 1$ が特異点となっているので工夫して取り除く．

$I = \displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\, dx$ の様に端点に特異性のある解析関数の積分を効率よく数値積分するために京都大学数理解析研究所によって1974年に提案された，二重指数関数型変換を用いる．

$x = \tanh\left(\frac{\pi}{2}\sinh t\right)$ として，変数変換すると

x	-1	$\longrightarrow$	1
t	$-\infty$	$\longrightarrow$	∞

$$dx = \frac{\dfrac{\pi}{2}\cosh t}{\cosh^2\left(\dfrac{\pi}{2}\sinh t\right)}\, dt$$

- 数式モードに移行する $ を最初に入れて，数式の中で日本語で文章を書くことができる.
- ソーステキストの可読性はよいかもしれないが，文章が英語の場合には使えない.
- seminar クラスでは，数式モードではダブルバイト文字は使えない.

```
$被積分関数f(x)=\dfrac{1}{\sqrt{1-x^2}}を-1から1まで,台形公式,シンプソン公式,ガウス・ルシャンドル公式で積分して精度を比較する.このときx=-1,1が特異点となっているので
工夫して取り除く.$\[I=\int_{-1}^1\frac{1}{\sqrt{1-x
この様に端点に特異性のある解析関数の積分を効率よく数値積分するために京都大学数理解析研究所によって1974年に提案
された,二重指数関数型変換を用いる.$\[x=\tanh\left(\fr
として,変数変換すると$
\begin{center}
\begin{tabular}{c|ccc}
$x$&$-1$&$\longrightarrow$&$1$\\\hline
$t$&$-\infty$&$\longrightarrow$&$\infty$
\end{tabular}
$\hspace{0.5cm}dx=\dfrac{\dfrac{\pi}{2}\cosh t}{\c
\end{center}
```

- 個性的なソーステキストを貰ったとき，これを自分用に変更するのには大変手間がかかる．

パーサビリティを意識した書き方

- 1 センテンス 1 レコード原則に従い，Perl スクリプトにとって変換しやすい形にする．
- 可読性を改行と 1 カラム目の%で保つ．
- vim を使用していると，longrightarrow のような長いスペリングの制御綴りが複数回出てきてもマクロに置き換えないことが多い．
- 前の例を書き換えた例である．

被積分関数 $f(x)=\dfrac{1}{\sqrt{1-x^2}}$ を -1 から 1 まで，台形公式，シンプソン公式，ガウス・ルシャンドル公式で積分して精度を比較する．
このとき $x=-1,1$ が特異点となっているので工夫して取り除く．

$\DS I=\int_{-1}^1\frac{1}{\sqrt{1-x^2}}\ dx$ の様に端点に特異性のある解析関数の積分を効率よく数値積分するために京都大学数理解析研究所によって1974年に提案された，二重指数関数型変換を用いる．
$x=\tanh\left(\frac{\pi}{2}\sinh t\right)$ として，変数変換すると
%
\begin{center}
\begin{tabular}{c|ccc}
x&-1&$\longrightarrow$&1\\\hline
t&$-\infty$&$\longrightarrow$&∞
\end{tabular}
$\hspace{0.5cm}dx=\dfrac{\dfrac{\pi}{2}\cosh t}{\c
\end{center}

- 変換スクリプトの例にも見られるように，書き方を規制したり，使用する環境を制限することで，プログラムによる処理が容易になる．
- 変換スクリプトは，索引の生成や，他の用途でも使うことがある [1].

- また，現在ではソーステキストをすべてキー入力する機会は少なくなっている．
- テキストエディタの機能も充実しているので，凝ったマクロを使うよりも，ポータビリティの良いソーステキストが好まれるようである．
- 情報処理学会の論文投稿用のスタイルファイルも，そのような方向で更新された[a]．
- TeX が開発された頃は，まだ Perl のようなグルー言語（glue language）は存在しなかった．

[a]例えば “\Figref{hoge}” で “図 1” のように表示されるマクロが用意されていて，“図 \ref{hoge}” と記述する必要をなくしてくれていたが，このように書いたテキストを，通常の jarticle スタイルに変更すると \Figref でエラーになった．

- 現在では Perl よりも普通の言語に近い Ruby のようなスクリプト言語もよく使われている.
- このような事情から，ソーステキストの書き方の標準を，スクリプトプログラムからのパーサビリティ（parsability）を軸に考え，TEX のコミュニティでそれを広めてゆくことを重要と考えている[a].

[a] parsability よりも scannability と呼んだほうが正確かもしれないが.

おわりに

- 本資料 LaTeXpresenSP.pdf と LaTeXpresenSP.tex を “make slide” で発表用とした slLaTeXpresenSP.pdf を添える.

References

[1] 寒川光. LaTeX & PostScript スーパーユーザのテクニック. 共立出版, 2006.