3. 再三のコンパイルと自動化

ここまで tex ファイルから文書出力を得ることを曖昧に “処理” と表現してきた．この “処理” とは実際にはどのような “処理” を行っているのかについて説明していく．具体的には，pdfTEX 系なのか否かや相互参照・文献情報の有無によって必要な処理の種類・順番や回数が変わる．最近では latexmk というコマンドのおかげで，このことを意識しなくてもいいことが多いが，この処理について知っておくことは latexmk 丸投げよりも効率がいい処理も可能で，デバッグの役にも立つので，知っておいて損はないと思う．なお，latexmk という名前からも分かる通り，今後の話は LATEX まわりの話である．

3.1. tex ファイルから pdf ファイルまで

tex ファイルから pdf ファイルを出力するまでの処理の流れについて簡単に説明する[LaTeXcompile_Yamamoto]．なお，以下では作成した tex ファイルの名前を hoge.tex とする．ユーザは hoge.tex の中身に従い，適切なマクロ体系を選択する．というよりも，日本語を使うのか，Unicode を使うのか，好みのフォントを選びたいのか，といった目的に応じて選択すべきマクロ体系があり，それに合わせて hoge.tex を作成する．本節では，例として LuaLATEX+upBIBTEX を用いることにする．

(1) はじめてのらてふこんぱいる

選択したマクロ体系で処理をする:

```
lualatex hoge.tex
```

これにより，次のファイルが生成される:

- 出力ファイル
  - hoge.dvi: LATEX, (u)pLATEX などを用いた場合
  - hoge.pdf: pdfLATEX, XELATEX, LuaLATEX などを用いた場合
- 補助ファイル hoge.aux: 相互参照・ラベルの情報を保存する．
- ログファイル hoge.log: コンパイル時の記録．エラーや警告も保存される．

このとき，もともと相互参照に関する hoge.aux を読み込まずに hoge.tex のコンパイルを行ったため，今回の出力で適切な相互参照はできていない．相互参照するべきところ（\ref）では “**??**” が表示され，hoge.log には参照が見つからなかった旨の警告*が記録される．これは一度の処理を高速・省メモリで終えるために，1 回あたりのコンパイルでは tex ファイルを最初から最後まで順に読み込みながら出力を行っているからである．このコンパイルの最中に補助ファイル（hoge.aux）にラベルを前から順に保存していく．

*LaTeX Warning: There were undefined references.

次のステップは文書の中に相互参照が含まれるかどうかによって以下のように場合分けされる:

	次の処理
参照したい文献`\cite`がある場合	(2) 参考文献処理
参考文献以外の相互参照`\ref`のみある場合	(3) 相互参照を処理する LaTeX コンパイル
相互参照も参考文献もない場合	(5) 文書の pdf 出力を得る

(2) 参考文献処理

所望の形に整形された参考文献の一覧を作る:

```
upbibtex hoge.aux
```

具体的には，(1) はじめてのらてふこんぱいる によって`hoge.aux`ファイル内に保存された文献リストの雛形から所望の形の文献リストが`hoge.bbl`に生成される[参考文献_星野]．`hoge.bbl`の中身は`thebibliography`環境であり，BibTeX 等を使わずに手動で`\bibitem`などとちまちま書いていくアレである．つまり，`hoge.bbl`にはそれぞれの情報を含めた文献項目（`bibitem`とか`entry`）が所望の形式で出力できるように収められている．

次のステップは (3) 文献リストを出力する LaTeX コンパイル．

(3) 文献リストを出力する LaTeX コンパイル

`hoge.bbl`を使って，文献リストを出力しつつ，補助ファイル`hoge.aux`に文献に関わる相互参照情報を保存する．

```
lualatex hoge.tex
```

ちなみに，(1) で作られた相互参照は今回の LaTeX コンパイル時に読み込まれた`hoge.aux`を使って，解決される．なぜならこのコンパイル時にはすでに 1 回目のコンパイルで作成された`hoge.aux`があり，そこに文献参照以外の必要な相互参照の情報が保存されているからである．

次のステップは (4) 相互参照を処理する LaTeX コンパイル．

(4) 相互参照を処理する LaTeX コンパイル

`hoge.aux`の相互参照を解決し，`hoge.dvi`か`hoge.pdf`を出力する:

```
lualatex hoge.tex
```

補助`aux`ファイルの中に前回保存したラベルがあるため，その番号を採用・出力していく．御存知の通り，出力が`dvi`ファイルになるか`pdf`ファイルになるかはマクロ体系として何を使うかに依存する．直接`pdf`ファイルが出力される系統を用いているのであれば，ここで文書は完成である．

マクロ体系	出力	次の処理
LaTeX, (u)pLaTeX	`hoge.dvi`	(5) 文書の pdf 出力を得る
pdfLaTeX, XeLaTeX, LuaLaTeX	`hoge.pdf`	-

(5) 文書の pdf 出力を得る

すでに述べたが，`dvi` ファイルとは DeVice-Independent file のことで，文書の見た目のレイアウトを画像形式・表示デバイス・プリンタなどにまったく依存しない形で記録している[DVI_Wikipedia]．これは人間が読むように設計されておらず，DVI ドライバと呼ばれる別のプログラムに入力することで，画像として表示したり，文書形式に変換したりできる．主な DVI ドライバとしては次のようなものがある:

- `dvipdfmx`: `dvi`→`pdf`．日本語文書作成においては最もよく使われている．そもそも海外では pdfLaTeX とその周辺が主流であり，`dvi` ファイルは生成されないのである．
- `dvips`: `dvi`→`ps` (PostScript). この `ps` ファイルとは `pdf` ファイルの基礎となったものである[PostScript_Wikipedia].
- `dviout`: (u)pTeX 対応の DVI プレビューア．最近では推奨されない．

無限ループ

文書作成の具体的な流れはここまでで説明したとおりである．基本的な相互参照にしろ，文献の参照にしろ，ラベルを適切に補助 `aux` ファイルに保存できれば，その後もう一回コンパイルすることですべての参照がきちんと対応して出力される．つまり`\label`，`\ref` があって，文献参照が必要ない（もしくは `bbl` ファイルに変更がない）場合には 2 回のコンパイルによって出力が得られることになる．これは基本的に間違ってはいないが，実際には 2 回では済まない場合がある．具体的には，参照したいラベル先が不明でとりあえず “**??**” と出力されていた部分に，ラベルがついている数字や文献を表す記号などが正しく代入されることで改ページの場所や図表の位置が変わってしまった場合，ラベルと番号の対応付けが変わってしまい，参照番号が正しく振られていないことになってしまうことがある*．そこで次のような収束しないおもちゃを作ることができる[loop_Qiita]:

```
\documentclass{article}
\pagenumbering{roman}
\setlength{\textwidth}{5em}
\setlength{\textheight}{2\baselineskip}
\begin{document}
  \setcounter{page}{999}
  \noindent The Page Number \pageref{foo} is\label{foo} shaking up and down.
\end{document}
```

この TeX 文書は，999 ページ始まりで，横 5 文字強，縦 2 行の本文スペースに文章を書き連ねている．`\label` は “is” の場所を示し，そのページ数を`\pageref` でローマ数字で表記している．実際にコンパイルすると，

*LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.

i. はじめてのこんぱいるでは，補助 aux ファイルが存在しないため，\pageref で参照すべきものがなく，“**??**” が出力される．その後\label で “is” のページ番号 999 が補助ファイルに保存される．
ii. 参照が解決していないので，2 度目のコンパイルを行う．今度は先程作った補助ファイルがあるので，\pageref 部分で適切に “is” の位置（999）がローマ数字 cmxcix で表示される．その結果，“is” は次の 1000 ページ目に移動してしまい，\label でこの 1000 ページの情報が再び補助 aux ファイルに保存される．
iii. “**??**” はないが，参照先と参照元があってないので，3 度目のコンパイルを行う．今度は\pageref では先程保存した 1000 ページの情報をローマ数字 m で表示する．今度は “is” が 999 ページ目に戻ってしまい，補助ファイルには再び 999 ページの情報が保存される．
ということで以下 ii. と iii. 繰り返し．

この TEX 文書をコンパイルした結果（のうち最初のページ，つまり 999 ページ）を Fig. 3.1 に示す．$n \in \mathbb{Z}_+$ 回コンパイルしたあとの出力において，ラベルが付けられる “is” のページ番号とそれを参照しているはずの \pageref{foo}の出力は必ず一致しない．このおもちゃに面白い以上の意味は見いだせないが，相互参照のために aux ファイルが必要で，同様に複数回のコンパイルが必要なこともわかるであろう．

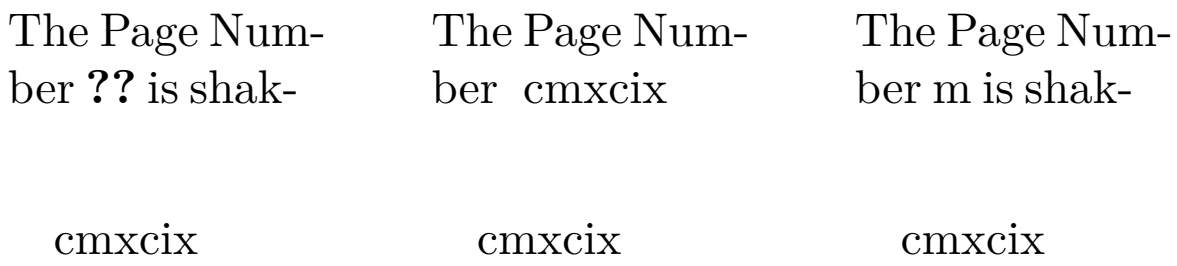

Fig. 3.1. The example of not converging cross-referencing. Left: former page (page 999) of first compilation results. Center: That of even numbered compilations. Right: that of odd numbered compilations.

3.2. 自動化

ここまででわかるように，文献を含む相互参照をきちんと機能させるためには，補助ファイル hoge.aux の存在も，それを使って複数回コンパイルすることも必要不可欠である．一方で，そのコンパイル回数はある程度目処がつくとは言え，正確に予測することは困難である．このような状況で，変更のたびに参照が失敗している箇所がないか確認しながら自分の手でコンパイルしていくのは大変面倒である．

この「複数回コンパイル等に起因する煩雑さ」を解決・軽減する方法として有名なのはもちろん

- latexmk: 現在最も有名な自動化ツール．複数回の実行に加え，BIBTEX 等の関連ツールの実行も自動で行える．latexmk は aux ファイルの内容が変化したときにコンパイルし直すことになっている．例えば aux ファ

イルに時刻の情報を含むようなコードを書けば，コンパイルのたびに aux ファイルが変化することになる*.

であるが，自動化ツールも何種類か存在する．いくつかを下記に示すが「煩雑だな」と感じるのは同じでもそれを解消するための設計指針や思想は異なっているものも多い．

- arara: 手順を書いたファイルを事前に用意して，ビルドする．有名どころ（pdflatex や biber などなど）は事前に用意されている．
- ptex2pdf: (u)platex と dvipdfmx をまとめて実行する．
- llmk: Light Latex MaKe.

*latexmk では上限回数が設定されており，これにより無限の彼方に飛ばされることはない TeX あれこれ_雑記帳.