E_018

法令改正文の形式化に基づく新法テキストの自動生成

Automated Consolidation of Acts based on Formalization of Amendment Sentences

外山 勝彦† 稲垣 真太郎† 小川 泰弘†
TOYAMA Katsuhiko INAGAKI Shintaro OGAWA Yasuihro

1．はじめに

近年，法律実務における情報技術の利用などにより，法令データベースの構築が望まれている．法律実務では，法令適用における不遡及の原則や法令制定・改廃に伴う経過措置があるため，改正前の法令や廃止法令といった旧法令を参照する場合がある．そのため，法令データベースには現行法令だけでなく，旧法令も蓄積されていなければならない．しかし，印刷物としての法令集や現在の法令データベース [1] は，一般に，更新時点における現行法令のみを収録しているため，旧法令の収集や復元が必要である．

ところで，わが国の法令の一部改正では逐語改正方式が用いられている [2]．この方式では，改正を施す法令中の箇所と改正内容を「改め文」と呼ばれる法令文によって規定し，それを改正前の法令に適用することにより，改正後の法令（新法令）を得る．そのような処理は統合と呼ばれるが，従来，専門家が手作業により行ってきたため，新法令が直ちに得られないなどの問題があった．

しかし，改め文の表現は一定の文法に従っている上，その意味は明確であると考えられる．そこで本稿では，改め文を正規表現によって形式化するとともに，それに基づいて自動統合処理を行うシステムを提案する．このシステムを用い，各改正時の改め文を順に解釈し，その意味に従った統合処理を新規制定時の法令に対して行えば，各改正時の新法令を順に得ることができる．

2．改め文の形式化

改め文の例を次に示す．

例1 ：第二条中「公団」を「機構」に改める。
例2 ：第四項に次のただし書を加える。
例3 ：第十三条を削り、第十四条を第十三条とする。

改め文の形式化を行うために，まず，平成 14 年に制定された法律 192 本 [3] から改め文 8,114 文を人手により抽出した．それを分析した結果，改め文の述語となる動詞（以下「改め動詞」と呼ぶ．）には，「改める」など表 1 に示す 7 種類のみが用いられていることが分かった．そこで，その事実と改め文の書式に関する特徴を利用して，法令テキストから改め文を自動抽出するツールを作成し，平成元年から同 16 年までに制定された法律 2,200 本から改め文 76,159 文を収集した．さらに，それらの文を改め動詞によって節に分割し，157,876 節を得た．この節は改正操作の基本単位であると考えられ，以下「改め節」と呼ぶ．例えば，例 3 は 2 つの改め節から構成されている．

また，それらの改め節を分析した結果，法令の一部改正は 10 種類の基本操作に分類できることが分かった．これは改め節の意味の分類に相当する．改め節中に現れる改め動詞の頻度を基本改正操作の種類別に集計した結果を表 1 に示す．この表から，基本改正操作の種類ごとに使用される改め動詞は限定され，改め節はその意味と現れる改め動詞により，12 種類に分類できることが分かる．

さらに，それらの改め節の構文パターンは，プログラミング言語 Ruby における正規表現を用いて，図 1 に示す 16 種類によって記述することができた．例えば，例 1，例 2 はそれぞれパターン 1，パターン 6 にマッチする．なお，同じ種類の基本改正操作であって，同じ改め動詞を使用していても，構文パターンが異なる場合がある．例えば，例 2 に対して，「第四項にただし書として次のように加える。」という表現の改め節が存在する．

3．法令自動統合システムの設計と実現

提案する法令自動統合システムの中核は，改め節解析部と新法生成部から構成される．

改め節解析部は，入力された改め節に対して，それがマッチする正規表現を図 1 に示す順に探した上，基本改正操作の実行に必要な情報を抽出して，図 2 に示す 11 種類の中間表現へ変換する．例えば，例 1，例 2 はそれぞれ

```
substitute_string(" 第二条" ," 公団" ," 機構" )
add_structure(" 第四項" , doc," ただし書" )
```

に変換する．ここで，**doc** は「次の」が参照している法令テキスト中の部分であり，この改め節の直後に現れる．各中間表現は前述の 12 種類の改め節に対応する．ただし，基本改正操作のうち番号の移動に対して「とする」を使用する場合は極めて少ないので，これは「繰り上げる／繰り下げる」を使用する場合と共通の中間表現に変換する．

一方，新法生成部は，中間表現を Ruby の関数と見なして直接実行することにより，改正前の法令テキスト（旧法テキスト）に対する処理を行い，改正後の法令テキスト

表 1　改め節における改め動詞の出現頻度

基本改正操作		改め動詞						計	割合 (%)
		改める	加える	削る	とする	付する	繰り上げる 繰り下げる		
文字列	置換	72,757	0	0	0	0	0	72,757	46.1
	追加	0	17,500	0	0	0	0	17,500	11.1
	削除	0	0	9,762	0	0	0	9,762	6.2
構造	置換	9,292	0	0	0	0	0	9,292	5.9
	追加	0	17,656	0	0	1,068	0	18,724	11.9
	削除	0	0	7,302	0	0	0	7,302	4.6
番号	変更	0	0	0	21,527	0	0	21,527	13.6
	付与	0	0	0	0	197	0	197	0.1
	移動	0	0	0	5	0	774	779	0.5
名前と番号の同時変更		38	0	0	0	0	0	38	0.0
計		82,087	35,156	17,064	21,532	1,265	774	157,878	100.0
割合（%）		52.0	22.3	10.8	13.6	0.8	0.5	100.0	

†名古屋大学大学院情報科学研究科，Nagoya University

```
1. / (.*?)中?(「.*」)を、?(「.*)に改め(、|る。) /
2. / (.*)の(見出し|.名)を「(.*)」に改め(、|る。) /
3. / (.*)を次のように改め(、|る。) /
4. / ((.*?中)?「(.*)」の下に、?「(.*)」を加え(、|る。)) /
5. / ((.*)に)?(.+?)として、?次の((.+)を|ように)加え(、|る。) /
6. / ((.*)に)?次の(後段|ただし書)を加え(、|る。) /
7. / ((.*)の(前|次))に次の((.+?)を|ように)加え(、|る。) /
8. / (.*?)に次の((.+?)を|ように)加え(、|る。) /
9. / ((.*?)中?「(.*)」を削(?:り、|る。)) /
10. / (.*)を削(り、|る。) /
11. / (.*?)?([ア-ン])から([ア-ン]まで)を(.*?)?([ア-ン])から[ア-ン]までと(し、|する。) /
12. / (.*)を(.*)と(し、|する。) /
13. / (.*)に見出しとして「(.*)」を付(し、|する。) /
14. / ((.*)に)?項番号を付(し、|する。) /
15. / ((.*)に)?(次の)?(.*)を付(し、|する。) /
16. / (.*?)(から|及び)(.*?)(まで)?を(.*)ずつ繰り(上げ|下げ)(、|る。) /
```

図1　改め節の正規表現

```
1. substitute_string(pos, str1, str2)
2. add_string(pos, str)
3. delete_string(pos, str)
4. substitute_structure(pos, doc)
5. add_structure(pos, doc, atr)
6. attach_structure(pos, doc, atr)
7. delete_structure(pos)
8. renumber(str1, str2)
9. attach_number(pos)
10. shift(pos, stp)
11. rename(str1, str2)
```

図2　中間表現

（新法テキスト）を生成する．それらの関数は，改め節の意味や統合処理の専門家の慣習に従って実現した．

ここでシステムの入出力とする法令テキストは，XMLによって構造化した文書である．法令は，題名，目次，本則，附則などの論理的構造を持ち，そのうち本則は，編，章，節，款，目という階層的な部分構造を持つ．章以下の構造は，さらに条，項，号などの階層的な部分構造を持つ．そこで，それらの論理的構造を記述するための文書型定義（DTD）を設計し，この DTD に基づいたタグを法令テキストに付与した．中間表現の関数は，文字列や，この論理的構造を表す木構造に対する操作として実現している．

4. 複雑な統合処理への対応

新法生成部の実現においては，複雑な改め節の処理や慣習に従った統合処理が必要である．以下，そのような例のいくつかを述べる．

例4：本則中「総理府令」を「内閣府令」に、「自治大臣」を「総務大臣」に改める。

例5：第三条及び第四条中の「青年」を「青年等」に改める。

例 4 は，同一箇所に対して同一種類であるが複数の基本改正操作を行う場合，一方，例 5 は複数箇所に対して同一種類の基本改正操作を行う場合である．それらの改め節に対しては，それぞれ内容や箇所を解析し，複数の中間表現を生成するように実現した．すなわち，例 4 については，

```
substitute_string(" 本則" ," 総理府令" ," 内閣府令" )
substitute_string(" 本則" ," 自治大臣" ," 総務大臣" )
```

の 2 つの中間表現を生成し，例 5 については，

```
substitute_string(" 第三条" ," 青年" ," 青年等" )
substitute_string(" 第四条" ," 青年" ," 青年等" )
```

の 2 つの中間表現を生成する．

例6：第三条中「並びに」を「及び」に改め、同条に次の一項を加える。

例 6 では，2 番目の改め節において，改正箇所が陽に示されていない．すなわち，「同条」が参照する条を同定する照応解析が必要であるが，そのような箇所は直前に出現したものであるという発見的規則を用いて対処した．

5. 実験

実現したシステムの動作を確認するために実験を行った．すなわち，新規制定された法律 17 本 [3] と，それらに対するすべての一部改正法 [3] から抽出した改め節 968 節を入力とし，順に統合処理を行って最終的に生成された新法テキストが現行法令 [1] と一致しているかどうかを調べた．なお，対象とした法律はそれぞれ 1〜16 回の一部改正があり，適用される改め節の合計はそれぞれ 10〜100 節程度である．また，この 968 節中で規定される基本改正操作の分布は，表 1 とほぼ等しい．

表2　統合処理失敗の原因

種類	原因箇所	箇所数
法令テキストの誤り	新規制定時（誤植）	17
	現行（誤植，統合誤り）	4
システムの不具合	改め節解析部	1
	新法生成部	1
計		23

実行した結果，1,161 個の中間表現が生成された．比較対象となる改正箇所は 4,355 箇所となったが，そのうち 4,332 箇所は正しく統合処理が行われ，23 箇所は統合処理に失敗した．その原因を表 2 に示す．そのうち 1 箇所は，設計した DTD においては 1 つの木ではないとする構造を追加しようとしたものである．一方，この改め節は，単に文字列の追加として統合処理を行おうとしていた．すなわち，文書構造のとらえ方の違いに起因するものである．また，他の 1 箇所は，文字列の置換と削除をそれぞれ別の箇所へ同時に行う必要があったものである．

6. おわりに

本稿では，改め文を法令テキストに対する編集操作の記述ととらえて形式化し，それに基づく法令自動統合システムを提案するとともに，その中核を実現した．実験の結果，基本的かつ頻出の場合には，統合は有効に行われることを確認した．表の改正，改め文の改め文への対応などを含め，システムの改良を引き続き検討するとともに，旧法令を復元し，法令データベースの構築を推進していく予定である．

謝辞　ご指導いただいた本学大学院法学研究科・松浦好治教授に感謝します．なお，本研究の一部は文部科学省科研費補助金萌芽研究（17650072）による．

参考文献

[1] 総務省：法令データ提供システム，http://law.e-gov.go.jp/
[2] 前田正道：ワークブック法制執務全訂，ぎょうせい（2003）．
[3] 衆議院：制定法律，http://www.shugiin.go.jp/