自然言語処理 85－5
（1991. 9. 19）

文書短縮のための文字列置換アルゴリズム

津田和彦[†] 中村雅巳[††] 青江順一[†]

[†]徳島大学工学部 [††]住友金属工業株式会社

新聞・雑誌などの記事文書の校正作業では，あらかじめ決められた制限文字数・行数内に文書を納めるための文書短縮作業が必要となる．文書短縮機能を実現するためには，文字数削減が可能な表記を形態素単位の制約条件により表した短縮規則を定義することと，文書中より短縮可能な表記を検出し短縮された文字列に置換する文字列置換アルゴリズムを構築する必要がある．本稿では，短縮規則の一例を示した後，我々が提案する文字列置換アルゴリズムについて述べる．本アルゴリズムは，Ahoらの提案したストリングパターンマッチングマシンを検索対象文書中の文字列が置換された場合でも効率的に検索できるように拡張したものである．本アルゴリズムを用い様々な文書に対して文書短縮処理を行った結果，短縮可能な部分の検索に要する文書の走査回数は，Ahoらのアルゴリズムを用いた場合の約１／３にすることができた．また，本稿に示した短縮規則のみを用いた場合でも全文書の約４～８％の行数を短縮できることがわかった．

An Efficient String Exchange Algorithm for Text Reduction.

Kazuhiko TSUDA [†] Masami NAKAMURA [††] Jun-ichi AOE [†]

[†] Faculty of Engineering, University of Tokushima.
[††] Sumitomo Metal Industries, LTD.

This paper presents a method for string exchange algorithm for text reduction. This algorithm is using morpheme replacements which are supported by rules for shortening morphemes, and are defined as the combination of a pattern of categories and restrictions on morphemes. In order to obtain a fast detection of the pattern, the string pattern matching algorithm of Aho et al. is extended by applying a matching algorithm which replaces the input morpheme. It is shown, from the simulation results on several texts, that the text obtained can be reduced between 4% and 8% from the original texts, and a number of morphemes scanning can be reduced to 1/3 from the algorithm of Aho et al.

1．はじめに

近年，ワードプロセッサ，デスク・トップ・パブリッシング・システム（ＤＴＰ）などが大量に普及し，日本語文書を計算機上で作成することが日常的になってきた．これに伴い，作成した文書をよりよい文書にするための校正作業を支援するソフトウェアの開発が望まれており，様々な研究が報告されている[11][12]．

新聞・雑誌記事などの文書は，用字・用語の統一や誤字・脱字の訂正はもちろんのこと，予め決められた紙面に文書を納めなければならないなど様々な制約下で作成されている．しかし，文書作成者が文書作成時にこれらの制約をすべて満たした文書を作成するのは困難であり，現実には，作成した文書の校正，編集作業を行うことによって作成した文書が制約を満たすように書き換えている．

文書の校正作業の内，表記の統一作業は全文書を注意深く読み返す必要があり多大な人手と時間を要する．また，文書を決められた紙面に納める作業は，文書中の文を削除することで行われる．この作業もまた削除した文と前後文書と意味的な関係を考慮せねばならず，熟練した校正者が多くの時間を割り当て行っている．

これら校正作業の効率を向上させるため，我々は日本語文書校正支援システムを開発している．我々が開発している日本語文書校正支援システムは，用字・用語の統一や誤字・脱字の訂正作業を支援するだけでなく，予め決められた紙面に納める作業を支援するための文書短縮機能を持つという特徴がある．

文書短縮作業を支援するソフトウェアを実現するには，まず，短縮可能な表記を短縮規則としてまとめた辞書を構築する事と，校正対象文書中より短縮規則にマッチする表記を高速かつ漏れなくマッチングし短縮した表記に置換するアルゴリズムを確立することが必要となる．

しかし，この短縮規則には，出版社独自の表記規則や各校正者の経験などを取り入れねばならず，使用する人毎に様々な規則が存在するようになる．そのため，文書短縮を実現するためのアルゴリズムは，上記の制約の他に様々な短縮規則に適用できなければならない．

そこで，本稿では，上記問題点を解決するため我々が提案した文書短縮のための文字列置換アルゴリズムについて報告する．本アルゴリズムは，校正対象文書の１回の走査で短縮規則にマッチする表記をマッチングでき，置換処理を行った場合にも効率的なマッチングができ，かつ短縮規則の追加・削除・変更が容易にできるという特徴を持つものである．

2．短縮規則

前章で述べたように，短縮規則は出版社独自の表記規則や各校正者の経験などを取り入れる必要があるため，使用する人毎に様々な規則が存在するようになる．ここでは，その一例を記載する．

本文書短縮機能は，文書短縮処理は形態素解析を用いた日本語文書校正支援システム[16]の１機能として開発するため，形態素単位の置換・削除操作として実現される．

形態素とは，言語学的に意味がある最小言語単位のことであるが，ここでは形態素辞書に登録されている表記を表すものとする．この置換・削除を定義する規則を短縮規則と呼び，下記に示すように形態素列$x_1x_2 \cdots x_n$を別の形態素列$y_1y_2\cdots y_m$（n, m≧1）への書き換え規則として定義し，制約条件を規則下［］内に付記する．

短縮規則ｐ　$x_1x_2\cdots x_n \rightarrow y_1y_2\cdots y_m$
　［x_i：x_i($1 \leq i \leq n$)に関する条件］
　［y_j：y_j($1 \leq j \leq m$)に関する条件］

以後，断りのない限り，この規則をp番目の規則として参照する．また，以下の関数を準備しておく．

cat(x)　：形態素xの品詞を返す関数．
concept(x)：形態素xの概念．但し，この概念は形態素の品詞をより詳細に分類するためのもの．
synonymy(C)：概念Ｃの類義語の集合．
replace(C)：概念Ｃに対する類義語の集合中で長さが最小の形態素．

また，以下で用いる品詞は，形態素解析の結果，確定された品詞を表す．例えば，“名詞”はサ変名詞が名詞として解析された場合を含む．

2．1　文脈に依存しない置換規則

本節では，１つの形態素だけを置換し，隣接形態素が変更されない規則を定義する．

短縮規則①　$x_1 \rightarrow y_1$
　［x_1:cat(x_1)＝接続詞，助述表現，関係表現など］
　［x_1:replace(concept(x_1))が定義されている］
　［y_1:y_1＝replace(concept(x_1))］
　［y_1:cat(y_1)＝cat(x_1)］

（例１）首藤[10]による背反関係を表す接続詞（形態素ｘ＝“にもかかわらず”）より次が得られる．

cat(x)＝接続詞
concept（ｘ）＝接続－背反
synonymy（接続－背反）＝｛しかし，けれども，が，ところが，なのに，でも，にもかかわらず，とはいえ，さりとて，されど，しかるに｝
replace（接続－背反）＝が

以上より，次の短縮が可能となる．但し，α，βは形態素列を表し，／は形態素の区切りを表す．

α／にもかかわらず／β
$\rightarrow \alpha$／replace（concept（にもかかわらず））／β
$\rightarrow \alpha$／が／β

本短縮規則は，名詞,形容詞の類義語[5]に対しても同様に定義できる．

2．2　文脈に依存しない削除規則

本節では，対象形態素列だけを削除し，隣接形態素列が変更されない場合の短縮規則を提案する．

2．2．1 並列複合語における接頭語，接尾語の削除

ここでは並列複合語“RISC／プロセッサ／と／CISC／プロセッサ”中の共通接尾語“プロッセサ”を削除して“RISC／と／CISC／プロセッサ”に短縮する規則を定義する．以下に，その規則を示す．

短縮規則②　$x_1x_2x_3x_4x_5 \rightarrow x_1x_2x_3x_5$
[x_1：cat(x_1)＝接頭語または名詞]
[x_2：cat(x_2)＝名詞]
[x_3：cat(x_3)＝接続詞（カンマを含む）]
[x_4：cat(x_4)＝cat(x_1)かつx_4＝x_1]
[x_5：cat(x_5)＝名詞]

短縮規則③　$x_1x_2x_3x_4x_5 \rightarrow x_1x_3x_4x_5$
[x_1：cat(x_1)＝名詞]
[x_2：cat(x_2)＝接尾語または名詞]
[x_3：cat(x_3)＝接続詞（カンマを含む）]
[x_4：cat(x_4)＝名詞]
[x_5：cat(x_5)＝cat(x_2)かつx_5＝x_2]

（例２）　次に短縮規則③を利用した例を示す．

α／レベル／１／と／レベル／２／β
→α／レベル／１／と／２／β

2．2．2　複合語への短縮

複合語への短縮規則は，宮崎の係り受け規則[6]，島津ら[8]の名詞句の意味関係より得られるが，次にその一部として宮崎の係り受け規則(その２)のNo.12を制約条件とした規則を示す．

短縮規則④　$x_1x_2x_3 \rightarrow x_1x_3$
[x_1：cat(x_1)＝サ変名詞]
[x_2：cat(x_2)＝助述表現
x_2∈{する，するための，することによって}]
[x_3：cat(x_3)＝名詞]

（例３）　次に短縮規則④を利用した例を示す．

α／受験／するための／資格／β
→α／受験／資格／β

2．2．3　属性表現の短縮

“変数temp”の表現のように英数字や記号で与えられた変数の属性名は，一度出現すると以後省略できる．次がその短縮規則である．

短縮規則⑤　$x_1x_2 \rightarrow x_2$
$x_2x_1 \rightarrow x_2$
[x_1：cat(x_1)＝名詞]
[x_2：cat(x_2)＝名詞]
[x_2：x_2が英，数，特殊記号の文字列である]
[x_1，x_2：x_1x_2かx_2x_1の組が文書中で既出現]

この最後の制約条件を判定するために，属性名とその文字列の組を解析中に記憶しておく必要があるが，この処理は容易であるので，詳細は省略する．

（例４）　次に短縮規則⑤を利用した例を示す．

α／変数／temp／β／…／α'／変数／temp／β'
→α／変数／temp／β／…／α'／temp／β'

2．2．4　指示代名詞に関する名詞句の削除

指示代名詞で，前文の名詞句を指示する場合，例えば”このxは”などの表現は冗長となるので以下の短縮規則で削除する．但し，空記号列をεで表す．

短縮規則⑥　$x_1x_2 \rightarrow \varepsilon$
[x_1：cat(x_1)＝指示代名詞（この，その，…）]
[x_2：cat(x_2)＝名詞]

（例５）　次に短縮規則⑥を満足する例を示す．

α／分析／ソフト／β／…／α'／この／ソフト／は／費用／など…
→α／分析／ソフト／β／…／α'／費用／など／…

2．3　文脈に影響を与える場合の短縮

本節では，対象形態素列が置換されるが，隣接した形態素列も変化する場合の規則を提案する．

2．3．1　関係・助述表現の短縮

次の規則は，短縮規則①において関係・助述表現x_2を置換する場合，その前の形態素x_1がy_1に変化される場合を表す．

短縮規則⑦　$x_1x_2 \rightarrow y_1y_2$
[x_1：cat(x_1)＝動詞]
[x_2：cat(x_2)＝助述表現，関係表現]
[x_2：replace(concept(x_2))が定義されている]
[y_1：cat(y_1)＝cat(x_1)，x_1が語尾変化した形態素]
[y_2：y_2＝replace(concept(x_2))]
[y_2：cat(y_2)＝cat(x_2)]

ここで，x_1からy_1へ語尾変化する情報は，y_2=replace(concept(x_2))と共に定義されているものとする．

（例６）首藤による助述表現（x＝はじめようとしている）から次が得られる．

concept（x）＝起動直前
synonymy（起動直前）＝{ところだ，つつある，はじめようとしている，はじめつつある，かけている，うとしている}
replace（起動直前）＝ところだ

以上より，次の変換が可能となる．

α／作り／はじめようとしている
→α／作り／replace（concept（はじめようとしている））
→α／作る／ところだ

2．3．2　指示代名詞に関係する名詞句の短縮

短縮規則⑥に対して，指示代名詞で構成された名詞句は，指示代名詞を変更することで短縮することが可能である．

短縮規則⑧　$x_1x_2 \rightarrow y_1$
[x_1：cat(x_1)＝指示代名詞（この，その，…）]
[x_2：cat(x_2)＝名詞]
[y_1：cat(y_1)＝指示代名詞（これ，それ，…）]

（例７）　次に短縮規則⑧を満足する例を示す．

α／分析／ソフト／β／…／α'／この／ソフト／は／費用／など…
→α／分析／ソフト／β／…／α'／これ／は／費用／など／…

３．マッチングアルゴリズム

短縮規則をマッチングする場合，形態素自身のマッチングは，文字単位の比較になり，膨大な処理時間が必要となるので，形態素を品詞に抽象化し，この品詞列より短縮規則の候補を高速に検出する．そして，この候補に品詞以外の制約条件を比較し，適用可能な規則を決定することで高速化を行う．

文書中より特定のキーワードを効率的に検出する手法としてストリング・パターン・マッチング・マシン（ＳＰＭ）がよく知られている．その中でもahoら[1]の提案したパターン・マッチング・マシン（マシンM）は，１回の走査で複数のキーワードを効率的に検出できるという特徴を持つ．本稿で提案する短縮規則のマッチングアルゴリズムは，マシンＭにより品詞列マッチングを行う．

しかし，マシンＭは検索対象の置換処理に対する走査回数の軽減については考慮していないので，本章ではマシンＭの適応法を説明した後，置換処理に対する走査回数の軽減を実現するための拡張法を提案する．

３．１　パターンマッチングマシンM

マシンＭは，テキストストリングの１回の走査で複数のキーワードを発見できるという特徴を持つ．本論文で提案する文書短縮法では，短縮規則左辺の品詞列を検出するため，マシンＭを次のように定義する．

$$M = (S,\ I,\ g,\ s_I)$$

Sは状態の有限集合であり，以後正数値の番号で表す．Iは本来入力記号の有限集合であるが，本論文では前述のように形態素の品詞集合とし，品詞は<>付きで表す．s_Iは初期状態であり，以後番号０で表す．gは状態遷移関数であって，goto関数と呼ぶ．

関数gは$S\times I$から$S\cup\{fail\}$への写像を定義し，状態sから状態tへの遷移が品詞<h>に対して定義されていればg(s,<h>)=t，そうでなければg(s,<h>)=failと書く．

状態sでマッチングが失敗したとき，次に進むべき状態をFAIL(s)で表し，failure関数と呼ぶ．状態sで短縮規則pの左辺の品詞列がマッチするならば$p\in RULE(s)$を定義し，RULE(s)をoutput関数と呼ぶ．

図１のマシンMはAhoら[1]の例の入力記号を品詞に変更したもので，1:<n><v>，2:<r><n><v>，3:<n><o><r>，4:<n><v><a><r>なる規則番号と品詞列パターンより構成されたものである．図１(a)の状態遷移図はgoto関数を表し，^{<n>,<r>}は<n>，<r>以外の記号を表す．

図２に入力品詞列"<p><r><n><v><a><r>"に対する状態遷移を示す．この遷移の一部を簡単に説明する．状態４で入力品詞<v>の場合を仮定すると，g(4,<v>)=5より状態５に進む．ここでRULE(5)≠emptyなのでRULE(5)の規則番号２，１が出力される．次に状態５で品詞<a>のマッチングするがg(5,<a>)=fail，FAIL(5)=2なので状態２を経てg(2,<a>)=8なる状態８に進む．

３．２　置換処理への拡張

本節では，提案するマシンＭの拡張（マシンＭ'と呼ぶ）の概要を説明する．マシンＭ'は，形態素解析の結果を（形態素，品詞）の組として格納する入力スタックinと，その組を処理結果として格納する出力スタックoutを利用する．但し，以後矛盾のない限り，上記の組は品詞のみを議論の対象とする．

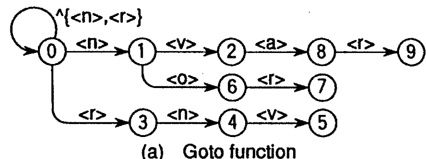
ここで，次の短縮規則

短縮規則p　<n><v><a><r>➡<n><o><r><n>

短縮規則q　<n><o><r>➡<n><r>

に対して，短縮規則pがマッチングしたとき（スタックの内容は図3(a)参照）を仮定する．ここで，マシンの現在の入力品詞はoutのトップに対応し，inのトップは次の入力品詞に対応する．従って，短縮規則pの左辺の品詞列は全てoutに存在する．

さて，この状況に対してAhoら[1]のマシンＭをそのまま使う場合を考えると，outから左辺の品詞列<n><v><a><r>ポップアップして右辺の品詞列<n><o><r><n>を左側がトップ方向になるようにinにプッシュすることになる（図3(b)）．しかし，outの上部に存在する品詞列が他の短縮規則の左辺の接頭辞に対応する必要があるので，次の何れかの再スタート処理を必要とする．



(a)　Goto function

s	1	2	3	4	5	6	7	8	9
FAIL(s)	0	0	0	1	2	0	3	0	3

(b)　Failure function

s	RULE(s)
2	{1}
5	{2，1}
7	{3}
9	{4}

(c)　Output function

図１　パターンマッチングマシンMの関数

入力品詞		<p>	<r>	<n>	<v>	<a>	<r>
状態遷移	0	0	3	4	5 2	8	9

図２　状態の遷移

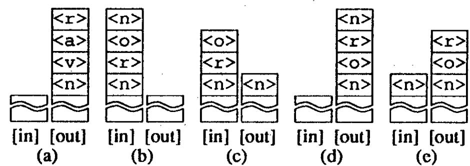


図３　スタック内容の操作

再スタート(イ)　outの上部より，マシンMを定義する短縮規則の左辺の最大の長さに対する品詞列をポップし，それをinにプッシュした後，状態0よりマッチングを再スタートする．

再スタート(ロ)　outの各要素にその品詞が遷移してきた状態番号を加えて格納し，図3(b)の後outのトップの要素に格納された状態番号からマッチングを再スタートする．

これらは入力のバックトラックや余分な情報の格納処理が必要となり，能率的な処理は行えない．

これらの問題点を考慮して，マシンM'への拡張を議論する．まず簡単な拡張であるが，規則pのように左辺と右辺で共通の品詞列（ここでは<n>）がある場合，その品詞列は残しておいた方が，スタックへのデータ入出力数の軽減につながる（図3(c)参照）．従って，各規則に対するoutからのポップアップ数を次のように定義する．

［定義１］　$x_1x_2\cdots x_n \rightarrow y_1y_2\cdots y_m$なる短縮規則pに対して，$x_i = y_i, 1 \leq i \leq n, m$なる最大のiを求め，その値をkとするとき，$pop(p) = n - k$を定義する．

上記の規則pの例では，pop(p)=3となる．

より効果的な拡張は，置換した品詞列（規則の右辺）をinではなく可能な限りout上において，そのまま処理を続行することである．このためには，状態sでマッチする規則pの右辺に対する状態遷移をマシンM上で前もって計算し，置換後に進む状態を関数SUCCESS(s,p)で定義することである．この関数をマシンMより決定する手順を，次の２つの場合で説明する．

(i)　短縮規則pのみ存在する場合．

規則pがマッチしている場合，図１では状態９まで遷移が進んでいるので，まずpop(p)=3だけ状態遷移をバックする（状態１となる）．次に，状態１より左辺の共通接頭辞の品詞<n>を除く品詞列<o><r><n>に対する遷移をマシンM上で進め，<o><r>に対する遷移の後状態７に達するが，g(7,<n>)=failなので，FAIL(7)=３なる状態３より<n>を遷移して状態４に達する．従って，SUCCESS(9,p)=4が決定される．このSUCCESS(9,p)=4を利用すると図3(d)のスタック状況で再スタートが可能となる．

(ii)　短縮規則pとqの双方が存在する場合．

状態７までの遷移は(i)の場合と同様であるが，この場合は状態７で別の短縮規則qにマッチするので，これ以上遷移を進めることはできず，SUCCESS(9,p)=7が決定される．しかも，この場合規則pの右辺の最後の品詞<n>はマッチされずに残ってしまうので，結局品詞<n>はinにプッシュする必要がある（図3(e)参照）．

(ii)の場合のinに入れる品詞を規則の右辺から計算するために，SUCCESS(s,p)が規則pの右辺の何個の品詞列を辿った先の状態を与えているのかを表すNEXT(s,p)を定義する必要がある．上記の(i)では，NEXT(9,p)=4，(ii)ではNEXT(9,p)=3となる．

以上の形式的定義を次に与える．

［定義２］　$x_1x_2\cdots x_n \rightarrow y_1y_2\cdots y_m$なる短縮規則pが適用可能な状態sのSUCCESS(s,p)とNEXT(s,p)を次のように定義する．

短縮規則の左辺の品詞列パターンより構成されるAhoらに基づくマシンMを仮定する．

$g(t, cat(x_1)cat(x_2)\cdots cat(x_n)) = s$

$g(t, cat(y_1)cat(y_2)\cdots cat(y_k)) = s'$, $k = n - pop(p)$

なる品詞列$cat(y_1)cat(y_2)\cdots cat(y_k)$に続く$cat(y_{k+1})cat(y_{k+2})\cdots cat(y_m)$なる入力品詞列に対して，アルゴリズム１により状態s'より品詞列$cat(y_{k+1})cat(y_{k+2})\cdots cat(y_m)$が遷移する状態列を$s_1, s_2, \cdots, s_r$とするとき，

①　$1 \leq h \leq r$なるすべてのs_hに対して$RULE(s_h) = empty$であって，$RULE(s_{h+1}) \neq empty$なるs_{h+1}が存在するとき，$SUCCESS(s,p) = s_{h+1}$，また，状態sまでに遷移を完了した入力品詞列を$cat(yk+1)\cdots cat(yj)$，$k+1 \leq j \leq m$とするときNEXT(s,p)をjとする．

②　①のs_{h+1}が存在しない場合は$SUCCESS(s,p) = s_r$，NEXT(s,p)=mとする．

３．３　関数SUCCESSの構成アルゴリズム

図５に関数SUCCESSの構成アルゴリズムを示す．但し，このアルゴリズムはAhoら[1]のマシンMのfailure関数とoutput関数を求める手続きに，関数SUCCESSを求める手続きSUCCESS_CONSを加えたもであり，Methodの本体とFAIL_CONSはAhoらのアルゴリズムと同様なので説明は省略する．

$x_1x_2\cdots x_n \rightarrow y_1y_2\cdots y_m$をp番目の規則とするとき，[1]で求まる変数kは$x_i = y_i$（$1 \leq i \leq n, m$）が成り立つiの最大値であり，[2]では状態sよりpop(p)だけ遷移を戻った状態番号を返す関数back(s,pop(p))を用いて，状態tを$cat(x_k)$が遷移した状態まで戻す処理を行う．次のfor-loopでは，状態tより$cat(y_{k+1})$以降の遷移を辿る．while-loopは遷移が未定義の場合のFAIL(t)による状態の遷移を行う．但し，途中で$RULE(s) \neq empty$なる状態sに達した場合は[3]でループを終了する．そして，[4]と[5]でSUCCESS(s,p)とNEXT(s,p)をそれぞれ決定する．

以下，上記手法が正しいことを証明する．状態sにおいて短縮規則pがマッチした場合，pop(p)の定義１より

$g(t, cat(x_1)cat(x_2)\cdots cat(x_n)) = s$

$g(t, cat(y_1)cat(y_2)\cdots cat(y_k)) = s'$, $k = n - pop(p)$

なる品詞列$cat(y_1)cat(y_2)\cdots cat(y_k)$（k=0の場合はε）が必ず存在する．

故に，状態s'より残りの品詞列$cat(y_{k+1})cat(y_{k+2})\cdots cat(y_m)$の遷移について考える．まず$cat(y_{k+1})$に対して$g(s', cat(y_{k+1})) = s''$の場合は，状態の遷移は明かである．また，$g(s', cat(y_{k+1})) = fail$の場合は

$$\left.\begin{array}{l} FAIL(s_h) = s_{h+1},\ 0 \leq h < r \\ g(s_h, cat(y_{k+1})) = s'' \end{array}\right\} \quad (1)$$

なるfailure, goto関数が必ず存在する．なぜなら，初期状態0以外の全ての状態にfailure関数は必ず定義されており，かつ初期状態0において$g(0, cat(y_{k+1}))$= failにはならない．故にアルゴリズム2の手続きSUCCESS_CONSにより品詞列$cat(y_{k+1})cat(y_{k+2})\cdots cat(y_m)$の遷移は正しく進められる．

次に，マッチング途中で出力状態が存在する場合について考える．上記の式（1）のfailure関数による遷移において，$RULE(s_0)$ = emptyならば$RULE(s_h) \neq$ emptyにはならない．なぜなら，$RULE(s_h)$は短縮規則番号の集合であり，$RULE(s_0)$の要素は$RULE(s_0)$から$RULE(s_h)$までの集合の要素の和集合なので，$RULE(s_0)$=emptyならば，必ず$RULE(s_h)$はemptyとなる（詳細は文献(1)の補題2参照）．故に，failure関数による遷移で短縮規則とマッチすることはない．

```
Algorithm1:Construction of SUCCESS  function.
Input:Goto function and output function.
Output:Functions FAIL,RULE and SUCCESS.
Method
 begin
  queue←empty;
  for each c such that g(0,c)=s≠0 do
   begin
    queue←queue∪{s};
    f(s)←0;
   end
  while queue≠empty do
   begin
    let r be the next state in queue;
    queue←queue-{r};
    for each c such that g(r,c)=s≠fail do
     begin
      queue←queue∪{s};
      FAIL_CONS(s,r,c);
      for each p in RULE(s) do
        SUCCESS_CONS(s,p);
     end
   end
 end

procedure FAIL_CONS(s,r,c)
 begin
  t←FAIL(r);
  while g(t,c)=fail do t←FAIL(t);
  FAIL(s)←g(t,c);
  RULE(s)←RULE(s)∪RULE(FAIL(s));
 end

 procedure SUCCESS_CONS(s,p)
  begin
   let x1x2…xn⇒y1y2…ym be the p-th rule;
[1]k←n-pop(p);
[2]t←back(s,pop(p));
   for i←k+1 until m do
    begin
     while g(t,cat(yi))=fail do t←FAIL(t);
     t←g(t,cat(yi));
[3]  if RULE(t)≠empty then break;
    end
[4]SUCCESS(s,p)←t;
[5]NEXT(s,p)←i
  end
```

図4　SUCCESS関数の構成アルゴリズム

次に，goto関数による遷移で

$$\left.\begin{array}{l} g(s_h, cat(y_{i+1}))=s'', \quad k \leq i < m \\ RULE(s'') \neq empty \end{array}\right\} \quad (2)$$

の場合，手続きSUCCESS_CONSの[L3]で必ず処理は停止し，SUCCESS(s, p)をs''に設定する．上記の式（2）においてs''が存在しない場合は，$g(s_h, cat(y_m))$=s''なる状態s''をSUCCESS(s, p)にセットする．故に，補題の成立は明かである．　（証明終）

また，NETX(s, p)が正しく構成されることについても補題1より同様に説明できるので，証明は省略する．

4．文書短縮アルゴリズム

文書短縮作業は，用字・用語を統一や誤字・脱字の訂正を行う作業のように，必ずしも校正対象文書の全文に対して実行する必要はない．たとえば，1記事の文書に対して短縮したい行数がほんの数行であった場合，全文に対して短縮処理を実行すると行数を減らしすぎて空欄が生じるなどの問題が起こる可能性もある．

文書の段落をTで表すとき，短縮容易度priority(T)は段落最終行の文字数remain(T)が少ないほど高く，段落の総文字数$row(T_1)$が多いほど高いので，priority(T) = row(T)／remain(T)と定義する．

文書短縮アルゴリズム2を図5に示す．アルゴリズム2は，短縮処理を行う文書と目標短縮行数TOTALを入力し，短縮された文書はoutに格納される．入力文書の段落を$T_1, T_2, \cdots, T_d$, $1 \leq d$で表し，短縮された行数の累計をグローバル変数countで表す．また，段落Tが処理済みかどうかを判定するために，use(T)を設定する．Methodの本体の最初のfor-loopは初期設定であり，次のfor-loopでは$priority(T_1)$の高い順に短縮処理関数$matcher(T_1, remain(T_1))$を呼んで短縮処理を進める．目標とする短縮行数が少ない場合，短縮が容易な段落についてのみ処理を実行することで処理時間を短縮できる．

関数matcher(T, r)では，段落Tの形態素に対応する品詞列はinに格納されているものとする．そしてrepeat-loopでは順次入力品詞をinより取り出し，マシンM'を用いて短縮処理候補RULE(s)を検出する．次に呼ばれるcheck(RULE(s))は，RULE(s)の各短縮規則に対して品詞以外の制約条件を判定して，マッチすれば規則番号を，マッチしなければ零を返す関数である．ただし，RULE(s)の候補の中で複数の規則が適用可能となった場合は，短縮の効果の高い（文字数が短くなる）ものを優先する．次に呼ばれるreduce(p)は，規則pによる短縮文字数を返す関数である．次のif文では段落最終行の文字数rよりreduce(p)を減算したrが負または零の場合，行数が減るので次のif文でcountが目標行数TOTALに達したかを判定し，目標値に達した場合は短縮処理を終了する．目標値に達していない場合はrを1行の最大文字数line_maxとrを加えた値にセットする．短縮処

理後，マシンM'の制御をSUCCESS(s, p)の状態に戻す．このため，規則pの左辺と右辺の共通の最大接頭辞の品詞列の長さk=n-pop(p)と（nは規則pの左辺の長さ），SUCCESS(s, p)まで遷移する規則pに対する右辺の品詞列長さh=NEXT(s, p)に対して，pop(p)だけoutよりポップアップした後，$y_{k+1}\cdots y_h$と$y_m\cdots y_{h+1}$に対応する品詞列をoutとinにそれぞれプッシュしておく．なおRULE(s)≠emptyの判定が，while-loop文で行われているのは，SUCCESS(s, p)で進んだ状態sもまたRULE(s)≠emptyになる場合があるからである（3．2節の規則pとqの双方が存在する場合の説明を参照）．

```
Algorithm2:Method of reducing texts
Input:A text,a number TOTAL of lines to be
expected.
Output:The reduced text.
Method
 begin
  count←0;
  for all Ti for 1≦i≦d do
   begin
    use(Ti)←false;
    priority(Ti)←row(Ti)/remain(Ti);
   end
  for Ti such that use(Ti)=false and
        priority(Ti) is the maximum do
   begin
    matcher(Ti,remain(Ti));
    use(Ti)←true;
    if all use(Ti) are true then
      begin print count; exit; end
   end
 end

procedure matcher(T,r)
 begin
  s←0;
  repeat
   Pop up the top element c from in;
   Push c on out;
   while g(s,c)=fail do s←FAIL(s);
   s←g(s,c);
   while RULE(s)≠empty do
    begin
     p←check(RULE(s));
     if p≠0 then
      begin
       r←r-reduce(p);
       if r≦0 then
        begin
         count←count+1;
         if count≧total then
          begin
           print"reducing is complete";
           exit;
          end
         r←line_max+r;
        end
       s←SUCCESS(s,p);
       k←n-pop(p);
       h←NEXT(s,p);
       Pop up pop(p) elements from out;
       Push cat(yk+1)…cat(yh) on out;
       Push cat(ym)…cat(yh+1) on in;
      end
    end
  until in becomes empty;
 end
```

図5　文書短縮アルゴリズム

2章の短縮規則に対する状態遷移図を図6に示す．図6を用いて，次の入力文に対する短縮処理を簡単に説明する．

変数　／　x　／　と　／　変数　／　y
名詞　　名詞　　接続詞　　名詞　　名詞

まず，／変数／x／の二つの名詞を辿ると状態5(RULE(5)={5}≠empty）に達するので，短縮規則⑤の残りの制約条件が成立するか否かを判定する．ここで，"変数x"は既に出てきた属性名と記号xの組と仮定すると，この規則の制約条件が満足され，"x"に短縮される．そして，状態はSUCCESS(5, 5)=4に移る．次に，g(4, 接続詞)＝failより状態はFAIL(4)=0に移り，状態0から／変数／y／に対しても同様に処理すると入力文は"xとy"に短縮される．以上の"変数x"，"変数y"に対して短縮規則⑤が成立しない場合は，状態8まで進み，短縮規則②が成立し，入力文は"変数xとy"に短縮される．

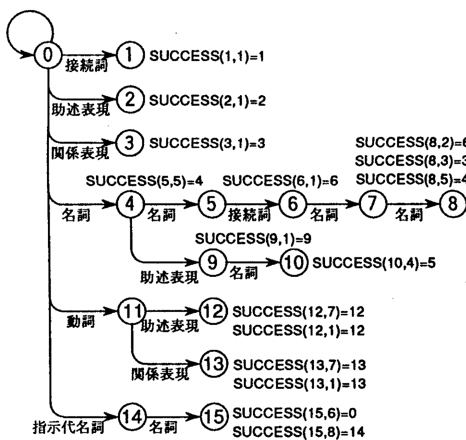


図6　短縮規則の状態遷移図

5．評価

本論文で提案した文書短縮のための文字列置換アルゴリズムの有効性を確認するため，新聞・雑誌および論文の全文書に対し短縮処理を行った実験結果を表1に示す．

本アルゴリズムを用いることにより，2章に示した8つの短縮規則の置換処理により生じた規則の右辺の品詞列を走査した回数は，マシンM（3．2節の再スタート(ロ)の場合）の走査回数と比較して約1／3に減少することが分かった．また，目標短縮行数が5行程度と少ない場合，全段落の約14%～19%の処理で目標が達成されるので，短縮容易度の導入もこのような少ない行数の短縮には有効である．

また，文書短縮率については，本稿に示したわずか8つの短縮規則を適用しただけでも，雑誌記事の編集

者が実用するために必要な短縮率という5％程度という数字を満足できた．表1に示す短縮率は，既に人手によって編集された後の文であり，編集前の文書に対して適用した場合は更に短縮率が向上すると思われる．

6．むすび

以上，文書短縮のための文字列置換アルゴリズムを提案し，種々の文書に対して適用することで，その有効性を確認した．本稿で提案した文字列置換アルゴリズムの効率は，短縮規則の追加に対しても低下しないので，校正者が培ってきた経験をより多くの短縮規則として反映させるても十分実用可能である．今後は，本アルゴリズムを用いて英語用文書短縮機能などを実現していく予定である．

謝辞

本研究を行う機会を与えて下さいました住友金属工業株式会社システムエンジニアリング事業本部システム研究開発部部長横井玉雄氏に感謝いたします．

表1　文書短縮処理の実験結果

文書名	文書1	文書2	文書3	文書4
文書情報				
全形態素数	2,995	2,530	3,903	4,682
段落数	38	33	42	36
全行数	466	299	333	341
1行文字数	12	16	25	24
短縮結果				
短縮行数	21	29	21	24
短縮率(%)	4.5	9.7	6.3	7.0
適用回数				
短縮規則①	46	67	39	61
短縮規則②	0	1	13	7
短縮規則③	1	5	2	13
短縮規則④	78	80	44	98
短縮規則⑤	0	0	13	7
短縮規則⑥	1	5	5	3
短縮規則⑦	2	23	16	11
短縮規則⑧	6	9	15	4
走査回数				
マシンM	384	519	397	637
マシンM'	134	190	147	204

文書1：朝日,毎日,読売各社の 1991/3/25 朝刊 "選抜高校野球" 記事および "兵庫県農高答案改ざん" 記事
文書2：日経パソコン　1991/2/4 pp.89,164-168　1991/2/18 pp.159
文書3：青江 "ダブル配列にによる高速ディジタル検索アルゴリズム" 共立出版 bit Vol.21 , No.6 pp.776-784
文書4：中村ら "ニューラルネットによる英単語品詞列予測モデル" 信学論D-II No.1 1991 pp.1-7

文　献

（1）Alfred V.Aho and Margaret J.Corasick："Efficient string matching : An aid to bibliographic search"，**Commun ACM**,Vol18, Num6.pp.333-340 (Jun.1975).

（2）J.Aoe Y.Yamamoto and R.Shimada:"A Method for Improving String Pattern Matching Machines," IEEE Trans. Software Eng. ＳＥ－１０，１，pp.116-120 (Jan.1984).

（3）青江順一："ﾀﾞﾌﾞﾙ配列による高速ﾃﾞｨｼﾞﾀﾙ検索ｱﾙｺﾞﾘｽﾞﾑ,"信学論（Ｄ），Ｊ７４－Ｄ，９，pp.1592-1600（１９８８－０９）．

（4）Jun-ichi Aoe: "An Efficient Digital Search Algorithm by Using A Double-Array Structure", IEEE Trans. Soft ware Eng. ＳＥ－１５，９，pp.1066-1077 (Sep.1989).

（5）藤原与一，磯貝英夫，室山敏昭："表現類語辞典"，東京堂出版，（１９８５－０３）．

（6）宮崎正弘："係り受け解析を用いた複合語の自動分割法"，情処学論，２５，６ pp.970-979（１９８４－１１）

（7）小野顕司，浮田輝彦，天野真家："文脈構造の分析"，情処学ＮＬ究報，70-2(1989-01)

（8）島津明，内藤昭三，野村浩郷："日本語文意味構造の分類－名詞句構造を中心に－"，情処学ＮＬ研報，４７－４（１９８５－０１）

（9）首藤公昭："文節構造モデルによる日本語の機械処理に関する研究"，福岡大学研報４５自然科学編，６，pp.88-119（１９８０－０３）

（10）首藤公昭，吉村賢治，津田健蔵："日本語技術文における並列構造"，情処学論，２７，２，pp.183-190（１９８６－０２）

（11）菅沼明，倉田昌典，牛島和夫："日本語文書推敲支援ツール『推敲』における否定表現の抽出法"，情処学論，３１，６，pp.792-800（１９９０－０６）

（12）鈴木恵美子，武田浩一："日本語文書校正支援システムの設計と評価"，情処学論，３０，１１，pp.1402-1412（１９８９－１１）

（13）Masayuki Takeda："Advances in Software Science and Technology 2,1990"，Iwanami Shoten,Publishers, pp.131-151 (1990-02).

（14）高木朗，伊東幸宏："情報処理実用シリーズ 10自然言語の処理"，丸善（１９８７－０７）

（15）田中章夫："研究資料日本文法第４巻４章「接続詞の諸問題－その成立と機能"，明治書院（１９８４－０９）

（16）津田和彦，番野邦彦，中村雅巳，青江順一："日本語文書校正支援システムにおける文書短縮法"，信学技報，NLC91-2,(1991-05)