SGML文書の論理構造変換手法

酒井 乃里子[†,☆] 高須 淳宏[††] 安達 淳[††]

電子化文書の普及にともない，SGML（Standard Generalized Markup Language）による科学技術論文の記述が一般化しているが，これらの文書の論理構造は雑誌などごとに定められる．論理構造の多様性は文書の表現力や個性化のためには欠かせないが，これらの文書を統合して活用する際には検索などでの効率的な処理やユーザが把握しやすい操作環境あるいは誌面の提供に困難が生じる．このような課題の解決として，本研究では，システム側が独自の論理構造を持ち，必要に応じて各文書をその論理構造に変換することを提案する．これにより，同一の論理構造として検索し，また統一されたインタフェースや画面で，検索条件入力や検索結果の提示を行い，文書を閲覧するデータベースが実現できる．本手法では，変換後の各論理要素に対応する，変換前の論理要素の指定を変換の仕様として与える．このとき，変換前後の要素の対応が1対多である場合や，出現可能回数の定義が異なる場合などを考慮し，またデータの整形の指定も取り入れる．本稿では，このような論理構造の変換手法について，変換仕様の記述法を中心に処理の過程も含めて述べる．

Transformation Method of Logical Structures between SGML Documents

NORIKO SAKAI,[†,☆] ATSUHIRO TAKASU[††] and JUN ADACHI[††]

More and more scientific papers are described in SGML (Standard Generalized Markup Language), but they have various logical structures. Though diversity is necessary for expressiveness of documents, it leads to difficulties for effective processing of a database and realizing user friendly interface. This study assumes a database that has its own logical structure and stores logical structures of documents with dynamic transformation capability between logical document structures for retrieval and browsing. We propose a transformation method for this purpose, and this method uses a specification which defines correspondence between original and transformed logical elements. Furthermore, the proposed specification can also handle repetition or combination of elements or changing data format. In this paper, we explain the proposed transformation method focusing on the description of the specification.

1. はじめに

電子的に文書を作成すると，編集・共有・再利用・検索など様々な利便が得られることから，多くの学会や出版社が原稿を電子的に蓄積する方向を検討している．このとき，論理構造を明示的に記述すればその効果は一層大きくなる．SGML（Standard Generalized Markup Language; ISO8879）[1]はこの目的で制定された国際規格であり，論理構造に基づく検索や文書の目次の抽出など，より高度な処理が実現できる．

SGMLが規定するのは論理構造の定義方法であり，個々の雑誌の論理構造は学会が定める．これにより個性的で表現力に富む文書を作成できるが，結果的に論理構造が多種多様になり，それらを大量に集めた全文データベースを検索などする際に，ユーザの求める情報が論理構造のどの部分に入っているのかという判断が難しくなる．そこで効率的な処理やユーザ・フレンドリなインタフェースのために，差異を吸収する機構が必要となる．

本研究ではSGMLで記述された科学技術論文を対象として，多様な論理構造を持つ論文を統合する文書データベースのための，異なる論理構造に文書を変換する手法を検討している．本稿では論理構造の多様性を検討し，論理構造の変換仕様を比較的容易に記述する方法を定義し，その仕様を用いた文書の処理アルゴリズムを提案する．

† 東京大学大学院工学系研究科
Division of Engineering, University of Tokyo
†† 学術情報センター研究開発部
R & D Department, NACSIS (National Center for Science Information Systems)
☆ 現在，日本電気株式会社
Presently with NEC Corporation

以下本稿では，まず2章でSGMLを簡単に解説し，想定するデータベースの概要および論理構造の多様性と変換の困難さを示したうえで，3章で論理構造の変換の関連研究に言及する．続いて4章では本手法の全体像を示し，5章で本研究での文書の捉え方と用語等を定義した後，6章で論理構造変換仕様の要素の割当て，また7章でデータ整形の記述を述べる．さらに8章で，実際の変換処理の動作をサンプルデータを用いた処理例とともに述べる．

2. SGMLと論理構造の変換

2.1 SGML

SGMLは論理構造を明記して文書を記述する規格である．最近ではWWW（World Wide Web）の普及にともないHTML（HyperText Markup Language）を扱う機会も多いが，HTMLはSGMLの応用例である．

SGMLによる文書は図**1**のように表される．論理構造はDTD（Document Type Definition）として定義される．DTDでは，各行に1つの要素の名前とその下位に来る要素名・出現順と回数などが記述される．定義された論理構造に基づく個々の文書インスタンスでは，データを論理要素の名前を用いたタグで前後を挟むという形式を再帰的に繰り返して論理構造の木を表現する．

図1は以降での変換前の文書例とする．

SGMLでは具体的な論理構造の作成は著者や出版社などに委ねられている．同じ分野の論文であればある程度論理構造に共通点が期待できるものの，やはり多様性は生じる．一方共通のDTDを制定するという方向性も考えられるが，内容に応じた表現形式や個々の誌面の個性も必要であり，既定の1つの論理構造で多くの論文誌などを構成するという方法は受け入れられ難い．

2.2 前提となる文書データベース

本研究ではSGMLで記述された科学技術論文を扱う図**2**のようなデータベースを想定している．このデータベースは二次情報や全文に対するキーワード検索に加えて，論理構造を条件に取り入れる検索や論理構造を操作する閲覧など高度な機能が期待できる．このシステムの動作は次のとおりである．

(1) 各論理構造とシステムの論理構造との間の変換仕様を定める．

(2) システムの論理構造を統一的なユーザインタフェースとして提示して，各文書に対する検索や閲覧の要求を受け付ける．

(3) 各文書をシステムの論理構造に変換して検索などを処理する．

(4) 検索結果や閲覧画面をシステムの論理構造に基づく形式で統一的に整形してユーザに提示する．

このデータベースモデルの特徴は2点あげられる．

- システムも独自の論理構造を持つ — 統一的な操作環境を，独自の個性や内容に応じた形式でユーザに示せる．この目的にはHTMLやLaTeXが多く用いられるが，前者は1個の固定的な論理構造なので柔軟性に欠け，後者はむしろレイアウト指向の言語なので十分とはいえない．
- 文書をもとの形式のまま蓄積する — もとの処理系も含めて検索・印刷出力などの多様な処理系に対応できる．このためにはもとの論理構造のまま蓄積したうえでの動的な変換が，記憶容量と処理量の点から望ましい．

本研究ではSGMLによる文書の，異なる論理構造間での変換手法を提案する．この手法は上述のような文書データベースには欠かせない基礎的な技法だが，一般的なSGML文書の構造変換にも適用できる．

2.3 論理構造の差異

ここで論理構造の差異について述べ，本研究の課題を明確化する．図**3**の左は図1の文書の木表現，右は左の文書を論理構造変換した結果を同様に木で表したものである．以下では変換前後の文書やそれらに関するものをそれぞれ「旧」「新」と呼ぶ．図中で要素

```
<!ELEMENT Recordlist O O (record)+>
<!ELEMENT Record O O Title, (Author)+, Ref>
<!ELEMENT Title O O #PCDATA>

<Recordlist><Record><Title>Doc1</Title><Author><Surname>
Smith</Surname><Firstname>Bob</Firstname><Middlename>
Thomas</Middlename><Affiliation>NACSIS</Affiliation></Author>
<Author><Surname>Jones</Surname><Surname>Nelson</Surname>
<Firstname>Mary</Firstname><Affiliation>NACSIS</Affiliation>
</Author><Ref>Reference1</Ref><Ref>Reference2</Ref></Record>
<Record><Title>Doc2</Title><Author><Firstname>Michael
</Firstname><Surname>Brown</Surname><Affiliation>Univ. of Tokyo
</Affiliation</Author></Record></Recordlist>
```

図1 DTDとインスタンスの例
Fig. 1 Examples of DTD and SGML Instance.

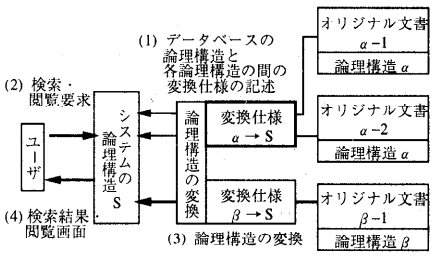

図2 想定するデータベースの機能
Fig. 2 Functions of an assumued database.

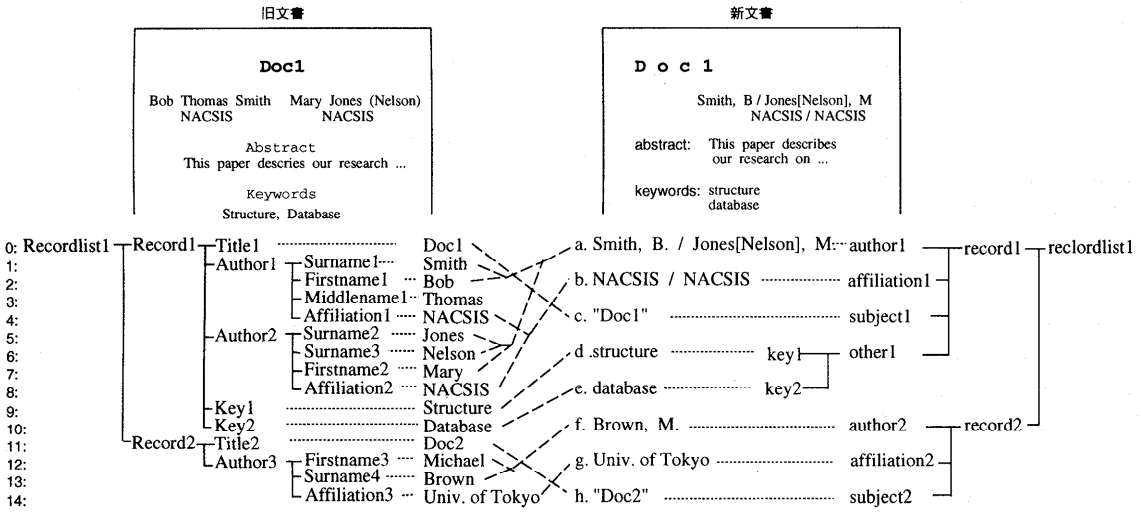

図 3 論理構造の木表現，論理構造間の差異
Fig. 3 Differences between logical structures in tree expressions.

名に添えて記した数字は同じ名前の要素の中で各々を識別する番号であり，要素名には含まれない．また例では旧要素の名前を大文字から，新要素は小文字から始まる文字列として明確に区別している．また図中で新旧それぞれの文書の左の数字・英記号は，説明のために文書の構成要素につけた番号である．

論理構造の差異は以下のように分類できる．

(1) 要素の取捨 — 旧文書の含む情報が新文書では必要なく，該当する要素がない．たとえば旧文書ではミドルネームの要素 Middlename（図 3 の *3*）があるが，新文書では扱わない．

(2) 要素名の違い — 内容が対応する新旧要素の名が異なる．たとえば旧要素 Title(*0*) と新要素 subject(*c*) はともに文書の題名の要素だが，違う名前を持つ．

(3) 論理構造の深さの違い — 対応する新旧要素が論理構造上で異なる深さに位置する．たとえば旧要素 Affiliation(*4*) は根から 3 つめにあり，新要素 affiliation(*b*) の 2 つめよりも深い．このような場合，新旧要素それぞれの中間ノードの関連づけが必要である．

(4) 1 つの旧文書中で，同じ位置にある同名の要素が複数回出現 — 次の 2 通りを区別する．

- 1 つの新要素にまとめる：新要素の出現回数が 1 回と定義されている場合など．たとえば旧要素 Affiliation(*4* と *8*) は 1 つの新要素 affiliation(*b*) にする．このような場合を**繰返し**と呼ぶ．
- 各々別の新要素にする：旧要素 Key(*9* と *10*) は別々の新要素 key(*d* と *e*) にする．

(5) 新旧要素の 1 対多の対応 — 名前の異なる複数個の旧要素が 1 つの新要素を構成する．たとえば，旧要素 Firstname(*12*) と Surname(*13*) を新要素 author(*f*) にする．このような場合を**組合せ**と呼ぶ．

また組合せの場合を含めてデータ形式の変換がともなう．たとえば旧要素 Key から新要素 key への変換ではデータが小文字に統一されている．

提案する手法ではこれらの差異を吸収して論理構造を変換する．

3. 論理構造の変換の関連研究

論理構造の取扱いは編集や検索など多様なテーマで研究が行われているが，ここでは論理構造の変換についての研究を取り上げて本論文のアプローチと対比する．

3.1 文法的手法に基づく処理

文書の論理構造は文脈自由文法と見なせる範囲が多いので，文法的な手法を適用した論理構造の操作手法が多く提案されてきた．

Blake ら[2]はマークアップされた Oxford English Dictionary（OED）を機械的に編集する手法を提案した．この手法は辞書が高度に構造化されていることを活かして文書を木構造に表し，木や部分木または葉に対するオペレータを用意して処理する．そのオペレータとは，ある中間ノードの下位にあるものを抽出

する “choose elements which are children of X（Xは中間ノード）” などである．

Warmer ら[3]はタグに対してアクションを定義する変換を提案し，たとえばSGML文書から電子メールの形式を得る目的に適用できる．文書を “<Mail><Sender>Smith</Sender><Recipient>Jones</Recipient><Body>Hello</Body></Mail>” として，文字列データはそのまま出力，「“<Sender>” タグの出現時に “From:”，“</Sender>” の出現時には改行（制御文字）を出力する」と定義する．

文書インスタンス中の要素やタグの出現に動作を定義するという同様の手法は Prices ら[4]ほかにも見られるが，この方式では要素の上下関係が反映させられない．

Warmer ら[5]はまたDTD中にアクションを書き込む手法で論理構造の変換を行った．ここで処理はC言語で記述された関数で，起動する関数名をDTDでのある要素の下位を記述するリスト部に記述する．この方法では処理を上下関係など論理構造と密接に関連して定義できるが，DTDに基づく厳密な文書処理はコストが高いことが知られている[6]．

これらの文法に即した手法は各種の処理が体系化されており望ましい一方，文書が文脈自由文法に納まらない構造，たとえば文中に任意に出現する脚注や参照などを持つ場合や変換において新旧の要素が1対多に対応するような場合を扱いきれないか，または変換仕様書の記述が複雑になり，実用的ではない．

3.2 変換仕様記述言語

高橋ら[7]は論理構造の変換仕様を記述する言語を提案している．この言語は論理構造木を探索するパーサに組み込んで，部分木などを特定する論理構造の変換を記述する．この手法によればプログラミング言語と同様高度な変換などが実現できるが，習得にあたってプログラム言語と同様に訓練が必要であり，担当者の負荷は大きい．

本研究ではこれらを踏まえて，前者と比較して文脈自由文法の範囲にとどまらない文書の論理構造を柔軟に扱え，また1対多などの変換を表現できて，しかも後者と比較して変換仕様の記述が平易であるような論理構造の変換手法を提案する．

4. 提案する変換の処理の流れ

4.1 変換処理の段階構成

提案する論理構造の変換手法は次のような段階で構成される．

(1) 論理構造の変換規則を記述する．
(2) 旧文書を解析して新文書に使われる旧要素を抽出する．
(3) 抽出した旧要素を繰返しや組合せを考慮して整理する．
(4) 旧要素に相当する新要素を設定して新文書として受理する．

4.2 要素の表現

本手法では文書をリストと捉え，そのリストを構成する各々を要素と呼び，要素が文字列などの値を持っているものとする．また要素の論理構造上の位置を表すために位置をパスで表し，パスは論理構造の中間ノードのリストで構成する．このとき中間ノードを**コンポーネント**と呼び，区切り子 “/” を挟んでコンポーネント名を連ねてパスを表す．コンポーネントは同一名のものの中で一意なIDにより識別され，IDは必要に応じてコンポーネント名に添字として表記される．

たとえば，ある要素は値が “Doc1”，パスが “/recordlist/record/subject”，その中のコンポーネント “record” はIDが1，などという．

4.3 変換規則の内容

変換規則は以下の3つの指定を含む．

(1) 各新要素を構成する1つまたは複数の旧要素
(2) 旧要素の繰返しの判断と新要素の設定のためのパスの評価法
(3) データの形式

まず(2)を5章で述べ，続いて(1)，(3)を述べる．

5. コンポーネントの優位性

コンポーネントは名前やIDを持つが，処理の過程でそれら以上のパスの評価が必要な局面が2つある．この評価のために「優位なコンポーネント」を定義し，それらの局面での優位性の役割を述べる．

5.1 繰返しの評価

同一旧文書中でコンポーネントのIDを除いて同じパスを持つ要素群は，1つの新要素にまとめるか別々にするか判断が必要である．たとえば同じデータレコードの著者の所属はまとめるが，違うデータレコードのものはまとめないという要求である．図3の例で，Author以下のIDのみが異なる旧要素 *4*，*8* は同じ新要素にするが，RecordのIDが異なる *14* は別の新要素にすることに対応する．この区別でRecordのIDは判断の鍵となるが，AuthorのIDは決め手ではないのでRecordは優位であると定義する．一般に優位なコンポーネントの方が多いためこれをデフォルトとし，変換規則では優位でないコンポーネントに “!” を付与して “!/Author” という形式で示すことにする．

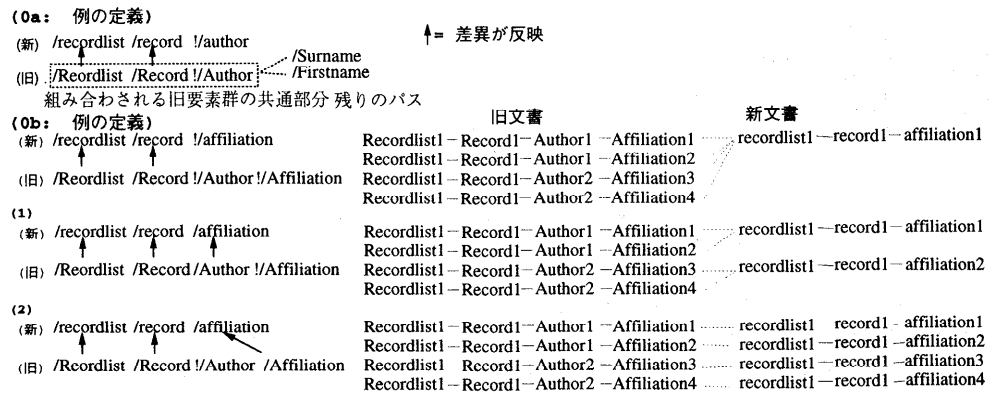

図 4 例の優位性の定義と，優位性の様々な定義の効果

Fig. 4 A definition of superiority in an example and effect of various definitions.

5.2 新要素の設定

優位性のもう 1 つの役割は，対応する新旧要素のパスの長さすなわちパス中のコンポーネント数が異なる可能性により生じる．つまりこの場合，旧要素の各コンポーネントの異同を新要素のどのコンポーネントに反映するかを指定する必要がある．例で旧要素 *4*, *8* と *14* は Record の ID が違うので別の新要素にしたが，この Record の違いを record の違いとして，それぞれ新要素 *b.* と *g.* にすることに対応する．

この指定のために Record と record はともに優位と定義する．さらに優位なコンポーネント中でのこれらの関連を表すために，それぞれのパスで根から葉へ数えてそれらが同じ順番の優位なコンポーネントであるようにする．したがって優位なコンポーネントは，対応する新旧要素のパス中で同数である．また上位のコンポーネントの ID の変更にともなって下位コンポーネントの ID も変更する．

図 3 の例でのコンポーネントの対応関係を図 **4** の *(0)* に示す．優位性は，組み合わせる旧要素群の共通な部分のパスに適用される．

また図 4 では優位なコンポーネントの様々な指定の効果も示した．*(1)* では *(0b)* に加えて Author の違いが affiliation に反映されるので，2 つずつに別れて新要素が作られる．

一方 *(2)* では *(0b)* に加えて旧文書での Affiliation の違いが新文書で affiliation の違いとして表される．4 つの旧要素は Affiliation の ID がすべて異なるので，affiliation の ID が互いに異なる 4 つの新要素になる．

この優位性により繰返しや新旧要素のパスの関連を柔軟に記述できる．

6. 変換仕様の記述

本手法での用語などを定義し，引き続いて変換規則の記述について述べる．

6.1 処理に用いる属性

本手法で要素は 2 通りに使われる．出現回数などが未定で論理構造の骨格を表し，また変換規則を記述するものと，回数などが確定しており具体的な文書を構成するものである．それぞれを（**狭義の**）**要素**と**要素インスタンス**と呼んで区別し，“*A*”“*D*” と表す．*A* は次のような属性を持つ．

- パス（*A.path* と表す．以下同様）：要素の論理構造上の位置を表す．たとえば/Recordlist/Record/Author/Surname.
- 組合せのある変換規則かどうか（*A.comb*）：組合せがある場合とない場合とでそれぞれ *COMB*, *NOT* の値を持つフラグである．

D は以下のような属性を持つ．

- パス（*D.path*）：要素インスタンスの論理構造上の位置を表す．たとえば/Recordlist1/Record1/Author1/Surname1.
- データ値（*D.val*）：要素インスタンスのデータ値．たとえば `Smith`.
- *D* が関係する変換規則の番号（*D.ass*）：その旧要素インスタンスを処理する変換規則の番号．

論理構造の中間ノードであり，パスを構成するコンポーネントは *C*，特に根にあたるものを C_0 と表す．*C* は次のような属性を持つ．

- 名前（*C.name*）：種別を表す．たとえば Recordlist.
- 識別番号（*C.id*）：各々の種別の中で一意な番号．

```
#1. 組み合わせのない場合
#   新要素 ; 旧要素 ; データ形式 ; 繰り返す時のデータ形式
0. /recordlist /record !/subject ; /Recordlist /Record !/Title ; "$1" ; $$ ($1)
1. /recordlist /record !/affiliation ; /Recordlist /Record !/Author !/Affiliation ; $1 ; $$ / $1
2. /recordlist /record !/other /ref ; /Recordlist /Record /Ref ; &lowercase($1) ; $$ , $1
#2. 組み合わせのある場合
# 新要素 ; 旧要素群のパスの共通部分 ; {各旧要素のパスの残りの部分}+ ; {各要素のデータ形式}+ ; {繰り返す場合のデータ形式}+ ;
                                                    組み合わせのデータ形式 ; 組み合わせたものの繰り返す場合のデータ形式
3. /recordlist /record !/author ; /Recordlist /Record !/Author/ : Surname : Firstname ; $1 : &initial($2). ; $$ [$1] : $$ [$2] ; $1, $2 ; $$ / $1
```

図5 変換仕様の記述例（左端の数字は，説明のための番号）
Fig. 5 An example of specification of transformation.

- 優位性（$C.sup$☆）：組合せがない場合パス全体に，組合せがある場合新要素のパス全体と旧要素群の共通な部分パスに適用する．$C.sup$ は3種類の値をとる．
  - 優位なコンポーネント：$C.sup = SUP$
  - 優位でないコンポーネント：$C.sup = NOT$
  - 優位性が適用されない，つまり組み合わされる旧要素群の共通でない部分のパスのコンポーネント：$C.sup = COMB$

パスあるいは部分パスはコンポーネントのリストであり，優位性を特に考慮しない場合や不定のときには P と表記するほか，以下の2通りがある．

P_{SUNO}☆☆：$C.sup = SUP$ or NOT

P_{COMB}：$C.sup = COMB$

以降で2つのパス（その名前やID）が同じであるとは，パス全体を通してコンポーネントが根から順に名前やIDが同じであることを意味するものとする．また旧要素に関するものは記号に "′" を付して，"P'" のように表現する．

6.2 変換仕様の記述

データベースを運用する準備作業として，著者あるいはデータベース管理者は論理構造の変換仕様を記述する．変換仕様は新旧要素間の割当てと，出力データの形式指定の2つの部分からなる．本稿の例として変換規則全体を**図5**に示す．図で左端の数字は説明用の行番号，"#" で始まる行はコメント行である．

まず割当てについて述べ，形式指定は続いて7章で述べる．

1つの旧要素が1つの新要素を構成する最も簡単な場合，割当て規則は新要素と旧要素のパス表現を区切り子 ";" で挟んで記述する．たとえば図5の *1* は，新要素/recordlist/record/affiliation が旧要素/Recordlist/Record/Author/Affiliation で構成されることを示す．つまり旧要素/Recordlist/Record/Author/Affiliation ... NACSIS が新要素/recordlist/record/affiliation ... NACSIS になる．

1つの新要素が複数の旧要素から構成される組合せがある場合の割当て規則は，同様に新要素と旧要素に関してパスを ";" を挟んで記述したものであり，新要素は組合せのない場合と同じである．一方旧要素は旧要素群が共通とする部分パスと，組み合わせる要素の数だけの残りの部分パスを ":" を挟んで記述する．たとえば図5の *3* は新要素/recordlist/record/author を構成する旧要素群はパスの共通部分が/Recordlist/Record/Author で残りの部分が Surname と Firstname であるもの，つまり/Recordlist/Record/Author/Surname と/Recordlist/Record/Author/Firstname であることを示している．これにより旧要素/Recordlist/Record/Author/Surname ... Smith と/Recordlist/Record/Author/Firstname ... Bob が新要素/recordlist/record/author ... Smith, B になる．

一般に i 番めの割当て規則は次のように表される．割当て規則は1行につき1つの新要素が記述され，各行には新要素自身のパス $A_i.path$ とそれを構成する旧要素のパス $A'_i.path$ を ";" を挟んで記述する．

$A_i.path; A'_i.path$

ここで新要素の部分 A_i は次のように表される．

$A_i.path : P_{SUNO}$

旧要素の部分 A'_i は2通りある：

$$A'_i.path : \begin{cases} P'_{SUNO} & \text{（組合せのない場合）} \\ P'_{SUNO} : P'_{COMB} : \cdots : P'_{COMB} & \text{（組合せのある場合）} \end{cases}$$

組合せのある場合の P'_{SUNO} は組み合わせる旧要素群の共通部分パスであり，P'_{COMB} はそれぞれの残りのパスである．

7. データ形式の指定法

変換仕様では新旧要素の割当てに続いて変換にともなうデータの形式を指定する．本手法ではデータの内容を自然言語的に判断するような処理は行わず，文字列置換などの形式的な処理のみを対象とし，旧要素を順次整形して新要素の形式を得る．ここで指定の記述

☆ superiority.
☆☆ superior or not.

法を述べる．

7.1 データ形式の種類

論理構造の変換での指定すべきデータ形式は3つに分類できる．

- 個々の旧要素の形式：たとえばデータを二重引用符で挟む（図3の *0* から *c*）．
- 組合せの場合の旧要素群の組み合わせ方：たとえば著者名について，もとの文書では姓と名とが別の要素だがこれを1つの要素として，“姓，名前の頭文字”という形式に（*12*，*13* から *f*）．
- 旧要素の繰返し：たとえば同じレコードの複数の著者所属は“/”を挟んで並べる（*4*，*8* から *b*）．繰返しは組合せのある場合の，組み合わせたもの全体にも起こる．
  一方，組み合わせる各々の旧要素にも起こる．たとえば2つの旧要素/Recordlist/Record/Author/Surname と/Recordlist/Record/Author/Firstname は組み合わせるが，このとき同じ著者の姓はまとめる（*5* と *6*）．

以上組合せなど処理の順序を考慮して整理すると，必要な形式の指定は次のとおりになる．

- 組合せのない場合
  (1) 個々の旧要素の形式
  (2) 繰返しの場合の形式
- 組合せのある場合
  (1) 個々の旧要素の形式
  (2) 個々の旧要素の繰返しの場合の形式
  (3) 組み合わせる形式
  (4) 組み合わせたものの繰返しの場合の形式

7.2 データ形式の記述と動作

形式の指定ではデータを表す変数に yacc（yet another compiler compiler）[8] と同様の記法を採用する．つまりそれまでに作られたデータを“$$”，新しいデータを割当て部分にパスが記された順に“$1，$2”などと表す．またあらかじめ用意した関数を呼び出しての処理も取り入れ，“&”に続けて関数名と引数として渡す変数名を記す．たとえば“&initial($1)”は用意された initial() 関数にデータ $1 を引数として渡し，戻り値で置き換えることを意味する．例ではこのほか，文字列を小文字に揃える lowercase() 関数を使用している．

記述法はまず組合せのない場合は2つの項目を“;”で区切って順に記述する．たとえば図5の *0* では，

(1) 個々の旧要素のデータは二重引用符で挟み
(2) 繰り返す場合には，新しいデータを“()”に入れてつなげる

という指定を表す．

組合せがある場合は4つの項目を“;”で区切って順に記述し，さらに各項目中で個々の旧要素について“:”で区切って記述する．このときの記述順も割当ての旧要素に関する部分の出現順に従う．たとえば *3.* は，

(1) 個々の旧要素は，割当て部分で1つめの Surname のデータはそのまま，2つめの Firstname は関数 initial() にデータを与えた結果とし
(2) 個々の旧要素が繰り返す場合には，ともに新たに出現したものの各々の整形結果を“[]”で括ってつなげ
(3) 組合せ方は“Surname のデータ，Firstname のデータ”という形式にし
(4) 組み合わせたものが繰り返す場合には“/”で区切って並べる

という指定を表している．

以上の記述に従う整形処理は，“$”で始まる部分は該当するデータに置き換え，“&”から始まる部分は該当する関数で処理した結果に置き換え，それ以外は文字列置換でデータを作成する．

8. 変換処理

ここまでに述べた仕様を用いる変換処理について述べる．

8.1 旧文書からの要素の抽出

割当て規則に記述されている旧要素は新文書に必要なので旧文書から抽出する．抽出の条件は，旧要素インスタンスが割当て規則の旧要素とパスの名前が一致することである．

処理 1.1：旧文書の解析

SGML パーザと同様の動作で旧文書を深さ優先で解析する．このとき論理構造の葉に達すると，現在解析している論理構造上の位置を表すパス（カレントパス）を D' として処理 1.2 で評価する．

（処理 1.1 終）

処理 1.2：旧要素の必要性の評価

D' を評価する．どれか1つの A'_i に対して，

- $A'_i.comb = NOT$ のとき（$A'_i.path = P'_{0SUNO}$ とする）
  P'_{0SUNO} と $D'.path$ とが名前が同じ．
- $A'_i.comb = COMB$ のとき（$A'_i.path = P'_{0SUNO} : P'_{1COMB} : \cdots : P'_{nCOMB}$ とする）
  どれか1つの P'_{kCOMB} $(1 \leq k \leq n)$ について，部分パス P'_{0SUNO} と P'_{kCOMB} をつなげたもの，つまり $P'_{0SUNO}P'_{kCOMB}$ というパスと $D'.path$ とが名前が同じ．

上の条件を満たすとき D' は新文書でも必要であり，

次段への入力に加える．

（処理 1.2 終）

例での抽出結果を図 **6** (1) に示す．この図では抽出された各旧要素インスタンスを 1 行ごとに記し，各行の内容は必要と判断した変換規則の番号とパス，値を記述している．パスは各コンポーネントごとに，名前と ID と優位性を記した．

8.2 抽出した旧要素インスタンスの整理

続いて抽出した旧要素インスタンスの繰返しや組合せを処理する．7 章と同様，繰返しと組合せの再帰的な出現を考慮して，以下の 3 段階の処理を行う．

8.2.1 組み合わせる旧要素インスタンスの繰返しの処理

複数の旧要素インスタンスを組み合わせる場合，その各々が繰り返すことがある．これは組合せ処理に先だってまとめる．

この条件は，組合せのある変換規則の 1 つに記述されている旧要素の 1 つとパス全体の名前が同じ旧要素群であり，共通部分では ID も同じことである．たとえば 2 つの旧要素インスタンス，図 3 の *5*，*6* は全体の名前が図 5 の変換規則 *3* の 1 つめの旧要素と同じであるほか，共通部分/Recordlist/Record/Author の ID も同じであるので繰返しである．一方 *1* は共通な部分パスの Author の ID が異なるのでまとめない．

```
                                        1=SUP
   D'.ass  {/C'.name C'.id: C'.sup}+  C.sup: 0=NOT      D'.val
                                          -1=COMB
 1. 0: /Recordlist1:1/Record1:1 /Title1:0                    : Doc1
 2. 3: /Recordlist1:1/Record1:1 /Author1:0/Surname1:-1       : Smith
 3. 3: /Recordlist1:1/Record1:1 /Author1:0/Firstname1:-1     : B.
 4. 1: /Recordlist1:1/Record1:1 /Author1:0/Affiliation1:0    : NACSIS
 5. 3: /Recordlist1:1/Record1:1 /Author2:0/Surname2:-1       : Jones
 6. 3: /Recordlist1:1/Record1:1 /Author2:0/Surname3:-1       : Nelson
 7. 3: /Recordlist1:1/Record1:1 /Author2:0/Firstname2:-1     : M.
 8. 1: /Recordlist1:1/Record1:1 /Author2:0/Affiliation2:0    : NACSIS
 9. 2: /Recordlist1:1/Record1:1 /Ref1:1                      : reference1
10. 2: /Recordlist1:1/Record1:1 /Ref2:1                      : reference2
11. 0: /Recordlist1:1/Record2:1 /Title2:0                    : Doc2
12. 3: /Recordlist1:1/Record2:1 /Author3:0/Firstname3:-1     : M.
13. 3: /Recordlist1:1/Record2:1 /Author3:0/Surname4:-1       : Brown
14. 1: /Recordlist1:1/Record2:1 /Author3:0/Affiliation3:0    : Univ. of Tokyo
  (1) 旧文書からの抽出結果

 1. 3: /Recordlist1:1/Record1:1 /Author2:0/Surname2:-1 : Jones[Nelson]
 2. 3: /Recordlist1:1/Record1:1 /Author2:0/Firstname2:-1 : M.
  (2) 組み合わせる要素の繰り返しの処理結果

 1. 1: /Recordlist1:1/Record1:1 /Author:1:0/Affiliation1:0 NACSIS
 2. 3: /Recordlist1:1/Record1:1 /Author:2:0   : Jones[Nelson], M
 3. 1: /Recordlist1:1/Record1:1 /Author:2:0/Affiliation2:0 NACSIS
  (3) 組み合わせの処理結果

 1. 1: /Recordlist1:1/Record1:1 /Author:1:0/Affiliation1:0 NACSIS / NACSIS
  (4) 繰り返しの処理結果

D.ass   {C.name C.id}+              D.val
 1. 0: /recordlist0/record0 /subject0    : Doc1
 2. 3: /recordlist0/record0 /author0     : Smith, B / Jones[Nelson], M.
 3. 1: /recordlist0/record0 /affiliation0: NACSIS / NACSIS
 4. 2: /recordlist0/record0 /other0/ref0: reference1
 5. 2: /recordlist0/record0 /other0/ref1: reference2
 6. 0: /recordlist0/record1 /subject1    : Doc2
 7. 3: /recordlist0/record1 /author1     : Brown, M
 8. 1: /recordlist0/record1 /affiliation1: Univ. of Tokyo
  (5) 新要素の設定結果
```

図 6 旧文書からの要素インスタンスの抽出結果（左端は説明用行番号）

Fig. 6 Result of extraction from an original document.

処理 2.1：組み合わせる要素インスタンスの繰返しの判断

A'_i が $A'_i.comb = COMB$ で，D'_j と D'_k☆ が $D'_j.ass = D'_k.ass = i$ のとき，

- $A'_i.path = P'_{SUNO} : P'_{1COMB} : P'_{2COMB}$ として，そのうち 1 つの P'_{hCOMB} $(h = 1or2)$ について，$D'_j.path$ と $D'_k.path$ の名前がともに $P'_{SUNO}P'_{hCOMB}$ と同じ
- D'_j と D'_k は P'_{SUNO} の ID も同じ

を満たせば，組み合わせる要素インスタンスの繰返しである．

D'_j と D'_k の値を整形して D'_j に統合，D'_k は削除する．

（処理 2.1 終）

例の処理結果を図 6 (2) に示す．図 6 (1) の *5*，*6* がまとめられて，図 6 (2) の *1* になった．

8.2.2 組合せの処理

組合せの条件は，組合せのある割当て規則のそれぞれとパスの名前が同じ旧要素インスタンス群で，共通な部分の ID も同じことである．たとえば 2 つの旧要素，図 6 (2) の *1*，*2* は共通部分/Recordlist/Record/Author の名前と ID が同じで残りが割当て規則図 6 の *3* のそれぞれ Surname と Firstname なので組み合わせる．

処理 2.2：組合せの判断

A'_i が $A'_i.comb = COMB$ で D'_j と D'_k が $D'_j.ass = D'_k.ass = i$ のとき，

- $A'_i.path = P'_{SUNO} : P'_{1COMB1} : P'_{2COMB}$ として，名前について $D'_j.path = P'_{SUNO}P'_{1COMB}$ かつ $D'_k.path = P'_{SUNO}P'_{2COMB}$
- D'_j と D'_k は P'_{SUNO} の ID が同じ

を満たせば組み合わせる要素群である．

D'_j と D'_k の値を整形し，$D'_j.path = P_{SUNO}$ とした D'_j に統合する．D'_k は削除する．

（処理 2.2 終）

例の処理結果を図 6 (3) に示す．図 6 (2) の *1*，*2* がまとめられて図 6 (3) の *2* になった．

8.2.3 繰返しの処理

繰返しの条件は，パス全体の名前が同じで優位なコンポーネントの ID も同じことである．たとえば旧要素インスタンス，図 6 (3) の *1*，*3* はパス全体の名前が

☆ ここでは 2 つの要素インスタンスを扱って述べているが，さらに多くの要素インスタンスが関与する場合も手法は同様に適用される．以降同様．

同じであるほか，優位なコンポーネント Recordlist と Record の ID が同じなので繰返しである．

処理 2.3：繰返しの判断

$D'_j.ass = D'_k.ass$ である D'_j と D'_k がパス全体で名前が同じであり，さらに，

- 優位なコンポーネントは ID も同じ

を満たすとき，D'_j と D'_k は繰返しである．

D'_j と D'_k は値を整形して D'_j とし，D'_k は削除する．

（処理 2.3 終）

例の処理結果を図 6 (4) に示す．図 6 (3) の *1*，*3* がまとめられて図 6 (4) の *1* になった．

8.3 新要素の設定と新文書の受理

ここまでの処理により 1 つの旧要素インスタンスは 1 つの新要素インスタンスになるよう整理されている．また新要素インスタンスのパスの名前は，関連する変換規則に記述された新要素のパスと同じに決定できる．ID の決定にはコンポーネントの優位性を利用する．

旧文書中で論理構造のノードを共有する要素インスタンス群は論理構造上近くにあったので，特に指定のない限り新文書中でも近くにあることが望ましい．このノードを共有するという状態は，コンポーネントの名前と ID が同じという条件で表される．変換仕様では，同異を反映したいコンポーネントを優位と定義している．

そこで新要素インスタンスの作成では，処理の終わっている旧要素インスタンスの中から今処理する旧要素インスタンスと最も多くのノードを共有している，つまり名前と ID が同じコンポーネントが最も多いものを見つけ出し，この旧要素インスタンスから作られた新要素インスタンスに倣って新要素インスタンスの ID を決定する．

処理 3：新要素インスタンスの設定

(1) 今処理する D'_j が $D'_j.ass = i$ のとき，設定する D_j のパスは $A_i.path$ と名前を同じに定める．

(2) $D_j.path$ の ID を決める．すでに処理済みの旧要素インスタンスの中から次のような D'_k を探す．

- パスを根から調べて，優位なコンポーネントの名前と ID が最も多く D'_j と同じ．

(3) 以下では優位なコンポーネントのみを考慮して，$D'_k.path = C'_0C'_1\cdots C'_nC'_{n+1}\cdots C'_{n+m}$，$D'_j.path = C'_0C'_1\cdots C'_nC'_{n+1}\cdots C'_{n+m'}$ とする．つまり $C'_0\cdots C'_n$ は名前と ID が 2 要素インスタンスで共通であり，残りはそうでない．また $D_k.path$ の ID は $D'_k.path$ によって決まり，$D_k.path = C_0C_1\cdots C_nC_{n+1}\cdots C_{n+m}$ とする．$C_0\cdots C_n$ は $C'_0C'_1\cdots C'_n$ により，$C_{n+1}\cdots C_{n+m}$ は $C'_{n+1}\cdots C'_{n+m}$ により決定された．

(4) $D_j.path$ の ID は，D'_j と D'_k の名前と ID が共通な部分に対応している $C_0C_1\cdots C_n$ は $D_k.path$ と同じに，それ以外の $C_{n+1}\cdots C_{n+m'}$ は新しい ID を割り振る．

（処理 3 終）

例で作成した新要素インスタンスを図 6 (5) に示す．

設定された新要素インスタンスは，パス表現から木構造を再構成しタグを付与されたマークアップ文書として出力される．

9. 評　　価

以上に述べた処理をプロトタイプとして実装した．サンプルデータとして図 3 に示す論理構造を含む論文データを作成して，変換して新文書を出力した．その処理過程は特徴的な部分を各段階の処理例として前章までに図示した．これにより 2.3 節で述べた課題，つまり単純な要素名の変換をはじめ，もとは複数に分かれていたものを組み合わせる変換，データの形式の変更，などといった差異が吸収されたことが示された．

3 章で触れたとおり文書の論理構造は文脈自由文法に近いため，基本的な論理構造に対しては yacc などを用いた処理も可能である．しかし SGML が対象とするより実際的な文書では，要素の出現順が柔軟であったり文書中の一定の範囲で任意の場所に出現可能な浮動要素が存在するという点で，論理構造自体が BNF（Backus-Naur Form）による記述が難しい水準の文法になっている．また論理構造の変換では離れて位置する旧要素群の組合せなどの可能性もあり，複雑なケースでは変換規則相互の関係など文書全体への効果を考慮した変換仕様の作成が必要になるなど，不可能ではないまでも負荷は大きい．

提案した手法によれば個々の変換規則は独立であるので，要素の出現順や他の規則との影響を離れて，新要素がどの旧要素により構成されるかだけに着目すればよく，したがって旧文書中の要素群の組合せも記述が容易である．

浮動要素については，変換仕様でパスを記述する際に相対的なパス表現を用いることで対処できる．ここでいう相対的パス表現とは，ツリー構造の根から葉までの道筋を省略なく記述する「絶対的」表現に対して，一部を省略する曖昧性を残した記述をいう．相対的なパス表現を用いることにより，浮動要素と同様の定義

を変換仕様で表現することが可能であり，さらに浮動要素だけでなくより柔軟な割当て規則の記述を可能とする．この場合，省略した新要素のパスの補完が課題となるが，もとの文書での要素間の関係を極力保持するという方針で実現することができる[9]．

データベース全体の機能としては，論理構造の変換と検索などのタイミングや論理構造を用いた検索とは何かなどまだ多方面での検討が必要であろう．しかし論理構造の変換の仕組みが実現されれば，本論文で述べたデータベースのみならず，文書から二次情報を抜き出すフィルタ，あるいは目次情報を抽出して文書の概要把握を可能にする閲覧機能など，様々な文書の活用システムの実現に寄与できると考えている．

10. おわりに

本稿では，SGMLによる科学技術論文を対象とし文書の論理構造の多様性を保持しつつ，一方でそれらの文書を統合して必要に応じて統一的に扱うデータベースを提案し，その核となる論理構造の変換手法について述べた．この手法を採用することにより文書データベースは学会や出版社の独自性や個性，文書の表現力は尊重する一方で，画一的な処理によって効率を上げることができる．同時に論理構造を用いた検索など高度な利用においてもユーザが操作しやすい環境を提供し，さらに同じ形式にレイアウトした文書閲覧画面により，ユーザの文書把握をより用意にするシステムの実現につながると考えている．

参考文献

1) van Herwijnen, E.：実践SGML，日本規格協会 (1992).
2) Blake, G.E., Bray, T. and Tompa, F.W.M.: Shortening the OED: Experience with a Grammar-Defined Database, *ACM Trans. Inf. Syst.*, Vol.10, No.3, pp.213–232 (1992).
3) Warmer, J. and van Egmond, S.: The Implementation of the Amsterdam SGML Parser, *Electronic Publishing*, Vol.2, No.2, pp.65–90 (1989).
4) Price, L.A. and Schneider, J.: Evolution of an SGML Application Generator, *ACM Conf. on Document Processing Syst.*, pp.51–60 (1988).
5) Warmer, J. and van Vliet, H.: Processing SGML Documents, *Electronic Publishing*, Vol.4, No.1. pp.3–26 (1991).
6) Hearth, J. and Welsh, L.: Difficulties in Parsing SGML, *ACM Conference on Document Processing Systens* (Dec. 1988).
7) 高橋　亨，東野純一：SGML文書の変換・再利用のための言語“AESOP”，情報知識学会第3回研究報告会講演論文集 (1995).
8) Johnson, S.C.: YACC – yet another compiler compiler, Computing Science Technical Report 32, AT&T Bell Laboratories (1975).
9) 酒井乃里子，高須淳宏，安達　淳：SGML文書データベースの論理構造変換における浮動要素の処理手法，電子情報通信学会第8回データ工学ワークショップ DEWS97 (Mar. 1997).

(平成9年 1 月13日受付)
(平成9年11月 5 日採録)

酒井乃里子

1992年東京大学工学部電気工学科卒業．1997年同大学院工学系研究科博士課程修了．博士（工学）．全文データベースシステムに関する研究に従事．1995，6年度日本学術振興会特別研究員．現在日本電気（株）に勤務．電子情報通信学会会員．

高須　淳宏（正会員）

1984年東京大学工学部航空学科卒業．1989年同大学院工学系研究科博士課程修了．工学博士．同年，学術情報センター研究開発部助手．1993年より同センター助教授．データベースシステム，文書画像理解，機械学習の研究に従事．電子情報通信学会，人工知能学会，ACM各会員．

安達　淳（正会員）

1976年東京大学工学部電気工学科卒業．1981年同大学院工学系研究科博士課程修了．工学博士．同年東京大学大型計算機センター助手．1983年同大学文献情報センター講師および助教授．1986年より学術情報センター研究開発部助教授をへて，現在教授．また，1987年より東京大学大学院工学系研究科助教授および教授を併任．オンライン情報システム，情報検索，分散処理システム，電子図書館システムなどの開発研究に従事．電子情報通信学会，IEEE，各会員．