社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

構造に基づいたXML文書のスキーマ変換器自動生成

小川　順平† 　粕谷　英人†† 　大久保弘崇†† 　山本晋一郎††

† 愛知県立大学大学院 情報科学研究科　〒480–1198 愛知郡長久手町大字熊張字茨ヶ廻間 1522–3
†† 愛知県立大学 情報科学部　〒480–1198 愛知郡長久手町大字熊張字茨ヶ廻間 1522–3
E-mail: †ogawa@yamamoto.ist.aichi-pu.ac.jp, ††{kasuya,ohkubo,yamamoto}@ist.aichi-pu.ac.jp

あらまし　XML を利用するプログラムは，特定のスキーマに依存して作成される．XML 文書が表現する情報を変えずにスキーマを変換することができれば，スキーマ毎にプログラムを修正する必要はなくなる．本研究は，スキーマが異なるが本質的に同一の情報を表現する XML 文書を学習対とし，XML 文書のスキーマを変換する XSLT を自動生成することを目的とする．2 つの文書に同一のテキストノードのみが現れるという制約の下で，同一テキストの位置を表す XPath の対から，変換を行う XSLT を生成する手法を提案する．また，生成アルゴリズムの精度をあげるため，DTD から得られるスキーマ情報を利用することを考える．

キーワード　XML，スキーマ変換器，自動生成，XPath

Automatic Programming of XML Transformer between Schemas based on Correspondence of Structures

Junpei OGAWA†, Hideto KASUYA††, Hirotaka OHKUBO††, and Shinichiro YAMAMOTO††

† Aichi Prefectual University, Graduate School of Information Science
1522–3, Ibaragabasama, Kumabari, Nagakute, Aichi-gun, Aichi, 480–1198, Japan
†† Aichi Prefectual University, Faculty of Infomation Science and Technology
1522–3, Ibaragabasama, Kumabari, Nagakute, Aichi-gun, Aichi, 480–1198, Japan
E-mail: †ogawa@yamamoto.ist.aichi-pu.ac.jp, ††{kasuya,ohkubo,yamamoto}@ist.aichi-pu.ac.jp

Abstract　An XML processing program is developed depending on a specific schema. If the XML schema can be converted holding structual information on XML documents, the program do not required to be modified according to respective schema. The goal of this research is automatic generation of XSLT programs that convert the schema of the XML document, using a study pair that represents identical information but their schemata are differ. We propose a technique for generating an XSLT that converts the schema from an XPath pair that represents the position of an identical text under the condition that only an identical text node occurs in two documents. In addition, we present schema information obtained from DTD to improve the accuracy of the generation algorithm.

Key words　XML, Schema Transformer, Automatic Generation, XPath

1. はじめに

1.1 背　　景

XML は，文書やデータに意味や構造をマークアップするための言語であり，スキーマと呼ばれる文章の型定義を利用者が定めることができるという柔軟性をもつ．しかし，柔軟性は，同一分野においても複数のスキーマが乱立する，プログラムが想定するスキーマのバージョンが異なる，などの問題の原因ともなり得る．

現状では，スキーマが変更された場合，その度にプログラムを修正しなければならないが，プログラムを修正することはコストが大きい．プログラムの修正作業が起きないように，管理する情報の分野によって標準のスキーマを定めることが必要である．しかし，分野によっては主流といえるスキーマがなく，多種のスキーマが存在している場合もあり，XML の利用者は，1 つのスキーマを選択しなければならないが，それは容易ではない．また，スキーマを選択しプログラムを作成した後に，他のスキーマが主流になることもある．したがって，利用するスキーマに応じてプログラムを修正する状況は，避けられない．

この状況の改善案として，次の 2 つが考えられる．1 つめは，

スキーマ変更前後の XML 文書に対して差分を取り，スキーマの変更情報を得る方法である．この変更情報を用いて，プログラム修正のためのパッチを作成する．具体的には，プログラムの修正の支援として，XML Diff and Patch [4] というツールの利用がある．これを利用すれば，プログラムの修正コストは軽減できる．しかし，スキーマの数だけプログラムを修正する必要があることには変わりがない．2 つめは，本質的に同一の情報を管理する XML 文書を，プログラムがサポートしているスキーマへ変換する変換器を作成する方法である．この方法を用いれば，プログラムを修正せずにプログラムを再利用することができる．しかし変換させたいスキーマの数だけ変換器を作成する必要が生じるので，この方法もコストがかかる．

1.2 目　　的

本研究の目的は，XML 文書のスキーマ変換を行うために，スキーマ変換器を作成する作業にかかる労力を軽減することである．そこで本稿では，スキーマ変換器の自動生成を行う．スキーマが異なるが本質的に同一の情報を表現する XML 文書の対からスキーマ変更情報を得て，スキーマ変換器を自動生成する．スキーマ変換器を自動生成できれば，変換器を作成するコストがなくなり，既存のプログラムをスキーマに依存することなく再利用することが可能になる．

2. 自動プログラミング

プログラムを自動生成する方法は大きく 2 種類に分けられる．1 つめは演繹的生成であり，完全な仕様からプログラムを自動生成しようという試みである．2 つめは，帰納的生成であり，不完全な仕様からプログラムを自動生成する試みである．本研究は，後者の帰納的生成について言及をする．

プログラムの帰納的生成は，不完全な仕様からプログラムを生成することである．基本的な考え方は，入力とそれに対応した出力を例として与えることで，他の入力に対しても望まれた出力が得られるプログラムを生成することである．

2.1 節では，XML と同様に木構造をもつデータを対象に変換器生成する研究として，S 式の変換器の帰納的生成をあげる．

2.1 S 式変換器の自動生成

ここでは，S 式データの変換器の帰納的生成に関する文献 [1]〜[3] の特徴について述べる．これらの研究はともに，ある入力 S 式データと出力 S 式データの対の集合から有限個を例として与え，与えたい入力のすべてに対して，望まれた変換を行うプログラムを生成することを目的としている．

2.1.1 例 の 制 限

文献 [1]〜[3] では，帰納的生成に入力として許される S 式に 2 つの制限をもうけられている．この制限によって，プログラムを生成する問題をある程度大きくならないように設定している．

（1） 入力 S 式に出現するアトムは，それぞれ唯一である．

（2） 出力 S 式に出現するアトムは，入力 S 式内に出現してなければならない．

制限（1）により，入力 S 式には同じ名前のアトムが複数回出現できないしないため，S 式におけるあるアトムの出現位置が唯一に定まる．制限（2）により，入力 S 式に出現しないアトムは出力 S 式にも出現しないため，出力 S 式内のアトムは入力 S 式内のアトムのいずれかが移動しているみなすことが可能となる．

2.1.2 変換器の制限

S 式の変換器は，基本関数（car, cdr, cons），述語（atom），条件式（cond）で構成される．基本関数は，S 式のアトムの選択，組み立てを行う関数のみである．そのため，アトムの名前を変更したり，アトムを新たに生成することはできない．しかし，前述の制限（2）のおかげで，出力 S 式には新しいアトムが出現しないので，これらの関数のみで問題にはならない．述語は対象 S 式がアトムであるかのチェックを行うのみのため，条件文は S 式の構造の違いで判断し，手続きを選択するために使われる．S 式の変換器生成はこれらの手続きの組み合わせの学習としている．さらに，文献 [1], [3] では，構造について関連のある複数の例を与えることで，生成アルゴリズムに役立てている．

これらの関数で構成された変換器は，例として与えられた S 式と同じ構造の S 式や，その構造について関連のある構造をもつ S 式に関して，望まれた変換を行う．

文献 [1]〜[3] 著者らは，問題点としてアトムの生成や比較を行う関数を扱っていないことをあげている．これらの関数をアルゴリズムに適用すると，問題を大きく広げてしまうため，生成アルゴリズムに適用することは非常に難しいとしている．

2.2 S 式と XML の比較

S 式を扱った既存研究と XML 文書のスキーマ変換を比較する．XML 文書のスキーマ変換の特徴をあげて，問題の違いについて議論する．

S 式に関する既存研究では，S 式に現れるアトムの移動・組み立てを行う変換器を生成することを目的としていた．しかし，本研究では，XML 文書のスキーマ変換を目的としているため，XML 文書に出現する要素の移動・組み立てでは変換を行えないことは明らかである．なぜなら，XML のスキーマが異なれば，XML 文書中の情報の表現方法が異なるからである．

XML 文書中で情報の表現方法には，(1) テキストノードの文字列で記述，(2) タグ名，(3) タグの属性，の 3 種類がある．一般的に，スキーマ変換器を作成する際には，変換する対象となるスキーマ（変換元）と目的のスキーマ（変換先）との情報の表す方法の変更情報を，それぞれの仕様書など得ることになる．本稿では，これらの表現方法の変更情報を，スキーマ情報が記述されている仕様書や DTD に加えて，変換元のスキーマと変換先のスキーマによって実際に記述されている XML 文書から獲得することを考える．すなわち，スキーマが異なるが本質的に同一の情報を表現する 2 つの XML 文書の対からスキーマの変更情報を取得し，スキーマ変換器を自動生成する．

以下では，S 式に関する既存研究を XML に適用する際に考えられる問題と，XML 文書以外の変換器生成に役立てるための追加情報について述べる．

2.2.1 変換能力向上の必要性

同名要素の出現　XML 文書は，テキストにタグでマークアッ

プすることで，構造や多種の情報を付加したデータである．タグの名前に意味づけをし，複数の同名のタグによって構造を構成しているため，同名の要素が存在することが一般的である．さらに，それらは出現する木構造の位置によって，意味が変わることがない．したがって，同名の要素は同等の扱いをするべきであり，それぞれ区別することは有効ではない．XML 文書内に出現する要素が唯一に区別できるという制限をもうけず，同名要素を扱えるようにする必要がある．

新しい要素の出現　XML のスキーマが異なるということは，XML 文書内に扱うタグセットそのものが異なることが多い．すなわち，変換元で出現するタグと変換先で出現するタグは，本質的に同じ意味を表していたとしても，同じ名前ではないということである．そのため，新たな要素の出現を扱うことは重要である．

要素名の認識　文献 [1]〜[3] では，S 式の構造でしか入力対象を区別することができない．XML 文書は構造だけでなく，タグの名前，属性，その値，テキストを用いてデータ内容を表現しているので，構造だけで XML 文書を識別することは妥当ではない．そのため，要素比較を行う述語を扱うことが必要である．

このように，XML という特徴をもったデータを変換器の帰納的生成で扱うためには，変換能力向上の必要である．

2.2.2　追加情報の可能性

既存研究では，制限された S 式で表せるデータ形式すべてを変換対象とし，アルゴリズムの入力として与えているのは S 式の対の集合のみであった．そのため，対の集合の中から構造の規則性を見つける必要があった．しかし，XML には，スキーマ情報を記述した DTD が存在する．DTD を用いることで，XML 文書の構成要素として現れるタグや属性がどのような規則で構成されるのかを知ることができる．このスキーマ情報を変換器生成アルゴリズムに利用できれば，前述の 3 つ側面で広がった問題を小さくできる可能性がある．

3. スキーマ変換器自動生成

この章では，変換器生成を行うアルゴリズムを定義し，実際に XML 文書の対の例と生成されたスキーマ変換器の紹介する．

3.1 入力と出力

変換器生成アルゴリズムの入力と出力を定義する．

入力は，XML 文書の対（2 つの XML 文書）と DTD である．XML 文書の対は，変換元スキーマで記述された XML 文書と，変換先スキーマで記述され本質的に同一の情報を表現する XML 文書である．この XML 文書の対は 3.2 節で述べる制限を満たしているとする．もう 1 つの入力は，変換先のスキーマ情報が記述されている DTD である．

出力は，変換元のスキーマで記述された XML 文書を，適切に変換先のスキーマで記述された XML 文書へ変換する実行可能なプログラムとする．

3.2 対象とする XML 文書とそのスキーマ

2.2 節で述べたように，XML で表現できるすべてのデータに対して，スキーマ変換器を自動生成することは非常に問題が大きくなってしまう．そこで，研究の初段階の設定として対象とする XML 文書とそのスキーマに制限をもうける．

3.2.1　XML 文書のスキーマの制限

変換対象として扱う XML 文書のスキーマには，以下の制限を与える．

- どのタグ要素も属性をもたない．
- どのタグ要素も複合コンテンツをもたない．すなわち，ある要素が子要素を 2 つ以上もっていた場合，どの子要素もテキストノードではない．
- 再帰構造は現れない．

3.2.2　XML 文書の対における制限

スキーマの制限に加えて，さらに XML 文書の対は次の条件を満たすものとする．

- 1 つの XML 文書内のテキストノードの文字列は，それぞれユニークである．
- 変換先 XML 文書内のテキストノードの文字列は，変換元 XML 文書内のテキストノードに存在する．

3.2.3　XML 文書のデータ管理方法の制限

XML 文書のデータの管理方法について考える．

- XML 文書は，レコード型のデータを扱う．

あるデータを管理するために，そのデータをどのような単位で表現するかは自由に設定することができる．例えば，図書を管理する XML 文書があったとする．A 図書館では，図書ごとに著者とタイトルを管理している．B 図書館では，著者とタイトルを別々に管理をして，ID 属性などで関連づけを行って，管理することもある．このように，スキーマが異なることは XML 文書のデータ管理方法まで異なることがある．データの管理方法によらず，スキーマの変換を行うことが望ましいが，問題が大きくなる．そこで，与える XML 文書の 2 つのスキーマは，データ管理方法が同じであることに制限する．

XML 文書が，あるデータをレコード単位で管理していた場合について考える．XML 文書において，レコードを記述する順序は重要でないことが多い．なぜなら，レコードを実際に閲覧する場合，XSLT などでそのときの状況に応じたソートを施すためである．XML 文書内に現れるレコードの順序までを，与えられた例によって，変換器に学習させる必要はないと考える．そこで，アルゴリズムに与える XML 文書の対のレコードは，それぞれ同じ順序で現れるものとして扱うこととする．

3.3 変換のコンセプト

XML 文書のスキーマの変換を行うにあたって，変換に対する基本的な考え方を明示する．

XML 文書の変換を行うために，XML 文書のデータが木構造であることに着目し，XML 文書の変換を木構造データの変換と考える．木構造データの変換を行うにはノードの対応付けが必要であるが，スキーマが異なるため，ノードは一般的に対応が付かない．しかし，前節においてデータの管理方法が同じであるという前提から，それぞれの部分木がもつ情報は同じであると仮定することができる．木構造データの変換をさらに部分木の変換と問題を小さくしてみることができる．部分木の対応をとるためにもノードの対応をとる必要があるが困難であるので，本稿ではテキストノードについては制限により必ず対応

がつくため，テキストノードを用いて部分木の対応をはかることを考えた．

XML 文書の対が与えられると，最初に**パステーブル**を作成する．パステーブルとは，1 つのテキストノードの 2 種の XPath [6] をペアでもつ要素を 1 つの行で構成される対応表のことである．作成手順は，次のとおりである．変換先に存在するすべてのテキストノードを幅優先の順序で取り出す．それらのテキストノードについて，それぞれ変換元，変換先の XPath を調べる．変換器生成アルゴリズムは，このパステーブルをもとに変換器を生成する．

3.4 DTD の利用

XML 文書のスキーマ情報を記述した DTD の追加情報としての利用を考える．

1 つめは，レコード型データの内部データが出現する順序の獲得である．図書管理データベースの XML 文書をスキーマ変換する例をあげる．いま 3 冊の図書を管理する XML 文書の対がある．それぞれ XML 文書には，3 つの部分木に 3 冊の以下の情報をもっていたとする．

（1） タイトル（メイン），著者名
（2） タイトル（メイン），原作者名
（3） タイトル（メイン，サブ）

望ましい変換器は図書情報のタイトル（メイン，サブ），著者名，原作者名の出現に対応しているものである．さらに，出現順序もこの順序でスキーマで定義されているものとする．これらの図書情報をスキーマ変換するために，上記の例を利用する．最も容易なのは，1 冊の図書データの対を選択することである．しかし，上記の例では，どの図書の対を選択しても 4 つのタグ要素の出現・順序情報を得ることはできない．次に，複数の図書データを利用することを考える．1 冊めと 3 冊めの図書データを利用すれば，タイトルのメインとサブのデータ順序を獲得することができる．しかし，著者名と原作者名のデータ順序は，3 冊すべての図書データをみても判断することはできない．すべてのデータの出現順序が一意に決まる図書データセットを例にすることも考えられるが，ここでは，DTD の利用を考える．DTD に図書の内部データの出現順序がタイトル（メイン，サブ），著者名，原作者名と記述されていれば，上記 3 つの例のみで変換器を完成させることができる．また，DTD に順序が指定していないければ，例の中から出現順序を判断する必要もなく，出現する可能性があるすべての要素が出現しているかどうかをみるだけでいいことになる．

2 つめは，タグ要素の接尾演算子（?，+，*）の利用である．接尾演算子の指定がない要素は，必ずそのタグ要素が 1 つだけ出現することを表す．接尾演算子の指定がある場合は，そのタグ要素が出現しなかったり，複数回出現することがあることを表す．この場合，与えられた例の中でそのタグ要素が 1 つしか出現していなかったとしても，変換器にはタグ要素の有無や複数個の出現に対応する必要が求められる．そこで，DTD に記述されているタグ要素の接尾演算子を利用することで，対応することが可能になる．

3.5 変換器生成アルゴリズム

変換器を生成するためのアルゴリズムを付録に示す．

アルゴリズムは大きく 2 つのプロセス `MAIN`, `CREATE_PROGRAM` から構成される．`MAIN` は入力として与えられた XML 文書の対からパステーブルを作成し，XSLT を生成するための初期設定を行う．`CREATE_PROGRAM` がパステーブルから XSLT の生成を行う．`CREATE_PROGRAM` は，XSLT を生成するための 4 つの手続きからなる．(1) テキストノードを出力するための `value-of` 要素を生成する．(2) タグ構造を構成するためのタグ要素を生成する．(3) 特定の要素に対して同じ変換を行うように，`template match` 要素を作成する．(4) 部分木ごとに変換処理をわけるため，部分木ごとに `CREATE_PROGRAM` をよぶ．

アルゴリズム内では，2 つの大域変数を用意する．1 つは生成過程のプログラムを表す P である．生成されるプログラム，すなわち変換器としての解は唯一に求めるため，大域変数のプログラムに書き込みを行い，変換器を完成させる．アルゴリズムの終了時に，P の内容を実行可能な変換器としてファイルに書き込む．もう 1 つは，プログラム P の適切な記述位置を保持するスタック S である．アルゴリズム内において，プログラムを作成する関数を再帰的に呼ぶことがある．そのとき書き込んでいる場所を関数呼び出しの深さごとに，スタックに覚えおく．

パステーブルの XPath ごとにポインタを用意する．ポインタは，XPath 上を動き，‘/’ で区切られた要素を指す．それぞれの XPath の処理している場所を表す．ポインタを “進める” とは，XPath 上のポインタを今の要素の右の ‘/’ を越えた次の要素を指すことである．

3.6 実　　装

前節で述べた変換器生成アルゴリズムを Scheme 言語の 1 つである Gauche [5] にて 500 行程度で実装した．今回は，XML 文書の変換に特化している XSLT [7] を変換器のプログラミング言語として用いる．実装した変換器自動生成プログラムに対して与えた入力である XML 文書の対とそこから生成された変換器を紹介する．

図 1 は，異なるスキーマで記述された XML 文書の対である．それぞれの XML 文書では，ともに 2 冊の同じ図書を管理している．“我が輩は猫である” の図書では，タイトル，著者，出版年の情報をもっているが，“学問のススメ” の図書では，出版年の情報をもっていない．図 2 は，変換先のスキーマ情報を記述した DTD である．この XML 文書の対と変換先の DTD から，スキーマ変換器を生成する．

はじめに，XML 文書の対を用いて，パステーブルを作成する．変換先の XML 文書内にあるテキストノードをあげて，それぞれの XPath を示すと表 1 のようになる．このパステーブルと DTD により，図 3 に示される XSLT プログラムが生成される．スキーマ変換器として望ましいものは，図書データとして，タイトル，著者名，出版年のタグ名の変換や順序入れ替えが行えていることである．さらに，例では 2 冊の図書の対を与えているが，図書が何冊であっても図書データの変換を行える必要がある．図 3 の XSLT は，スキーマ変換器として，望まれる振る舞いを行うことがわかる．

```
<図書>
  <本>
    <著者>
      夏目漱石
    </著者>
    <タイトル>
      坊ちゃん
    </タイトル>
    <出版年>
      1906
    </出版年>
  </本>
  <本>
    <著者>
      福沢諭吉
    </著者>
    <タイトル>
      学問のススメ
    </タイトル>
  </本>
</図書>
```

```
<lib>
  <book>
    <data>
      <year>
        1906
      </year>
      <title>
        坊ちゃん
      </title>
      <author>
        夏目漱石
      </author>
    </data>
  </book>
  <book>
    <data>
      <title>
        学問のススメ
      </title>
      <author>
        福沢諭吉
      </author>
    </data>
  </book>
</lib>
```

図 1　XML 文書の対 （変換元, 変換先）

```
<!ELEMENT lib      (book*)>
<!ELEMENT book     (data)>
<!ELEMENT data     (year?, title, author)>
<!ELEMENT year     (#PCDATA)>
<!ELEMENT title    (#PCDATA)>
<!ELEMENT author   (#PCDATA)>
```

図 2　変換先 XML 文書の DTD

表 1　パステーブル：テキストノードとその XPath

テキストノード	変換元 XML 文書	変換先 XML 文書
1906	/図書 [1]/本 [1]/出版年 [1]	/lib[1]/book[1]/data[1]/year[1]
坊ちゃん	/図書 [1]/本 [1]/タイトル [1]	/lib[1]/book[1]/data[1]/title[1]
夏目漱石	/図書 [1]/本 [1]/著者 [1]	/lib[1]/book[1]/data[1]/author[1]
学問のススメ	/図書 [1]/本 [2]/タイトル [1]	/lib[1]/book[2]/data[1]/title[1]
福沢諭吉	/図書 [1]/本 [2]/著者 [1]	/lib[1]/book[2]/data[1]/author[1]

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transfo rm">
<xsl:template match="図書">
  <lib>
    <xsl:apply-templates select="本"/>
  </lib>
</xsl:template>
<xsl:template match="本">
  <book>
    <data>
      <xsl:apply-templates select="出版年"/>
      <title>
        <xsl:apply-templates select="./タイトル"/>
      </title>
      <author>
        <xsl:apply-templates select="./著者"/>
      </author>
    </data>
  </book>
</xsl:template>
<xsl:template match="出版年">
  <year>
    <xsl:value-of select="."/>
  </year>
</xsl:template>
</xsl:stylesheet>
```

図 3　生成された XSLT

4. 評　　価

4.1 自動プログラミングの能力

同名要素の出現　これまでの S 式の自動プログラミングでは，出現するすべてのアトムについて唯一に識別する必要があった．しかし XML では，タグ要素は同名のものが複数出現することが一般的である．本稿で提案したアルゴリズムでは，テキストノードのみ唯一に対応づける必要があるが，他のタグ要素については同名要素が出現が可能である．タグ要素の識別は，XPath を用いて木構造の位置を利用して行っている．

新しい要素の出現　S 式の既存研究では，すべてのアトムについて対応づいていなければならないので，新しい要素の出現を認めていなかった．XML では，スキーマが異なればタグセットが異なることが多いため，新しい要素の出現は，許す必要がある．本稿のアルゴリズムでは，部分木を対応づけることで，その部分木内に出現するタグ要素は，対応づいていなくてもよくなり，新しい要素の出現が可能になった．

要素名の認識　S 式に関数既存研究では，与えられた S 式の構造で場合分けを行っていた．XML では，文書自体の木構造よりもタグ名が大きな意味を持つと考えた．本稿のアルゴリズムでは，XML を構造で条件分岐するよりも要素名での分岐が重要であると考え，部分木のルート要素名で条件分岐を行うようにした．これにより，同じ要素をルートにもつ部分木に対して，同じ変換を行えるようになった．

4.2 スキーマ変換の能力

本稿で提案したアルゴリズムでは，研究の初段階として，XML 文書や変換手続きにいくつかの制限をもうけていた．その制限において，XML 文書のスキーマ変換としてテキストノードにマークアップされたタグセットの変換が行えるようになったといえる．XML 文書の表現能力を大きく制限しているが，レコード型のデータを扱った XML 文書のタグセット変換については，スキーマ変換器として有用といえる．スキーマ変換器としてより実用的にするための，改善点を 2 つあげる．

（1）　属性を扱うことができる．

（2）　データ管理方法をレコード型に制限しない．

項目（1）の属性は，タグの要素と同じように扱えば，比較的容易に対処できると考える．しかし，同じ情報を表現する属性の出現場所が異なっていたり，他方ではタグ構造でその情報を表現している可能性があるので，段階的にスキーマに制限を設けて解決していく必要がある．また，属性の値に関しては，値の文字列が同じであれば対応がつきやすいが，異なる場合もある．この場合は，どの値が同じ情報を表現するのか追加情報を与えたり，XML 文書の中から発見する必要がある．項目（2）の管理方法に関しても，スキーマ変換という目的のためには，改善したい点である．これら 2 つの点を改善するためには，セマンティックな情報表現の同値判断能力を考える必要がある．タグ名，属性，その値や XML の構造のセマンティックな情報表現から，意味の類似・同値を発見する方法を調査する必要がある．

4.3 関 連 研 究

関連研究として Microsoft XML Diff and Patch [4] がある．この研究も本研究と同様に，XML 文書の対からスキーマの変更情報の獲得を行っている．XML 文書の木構造としてのデータの Diff 情報を XML として出力し，これをスキーマの変更情

報としている点である．本研究と異なるのは，XML 文書のスキーマを変換するのではなく，出力された Diff 情報の XML をもとに，既存の XML を利用するプログラムの Patch を作成するための支援を目的としている．

Patch を作成する作業は手作業で行うため，対象スキーマが異なるスキーマごとに XML を利用するプログラムの Patch を作成しなければならない．また，Diff 情報が記述された XML の利用方法はユーザに委ねられているため，利便性がよいとはいえない．

本稿では，XML 文書のスキーマを変換するための変換器を自動生成しているで，既存の XML を利用するプログラムをそのまま再利用することが可能である．したがって，XML 利用者の作業軽減に有用である．

5. おわりに

5.1 まとめ

本稿では，スキーマに制限を加えた XML 文書のスキーマ変換器を自動生成する手法を提案した．この手法によって得られる変換器を用いれば，XML 文書のスキーマを変換する作業にかかる労力を軽減することができる．対象とするスキーマや変換器生成問題を制限したため，XML のタグ構造を変更する変換器しか生成できないが，変換元と先のスキーマを考慮しながら変換器を作成する作業を十分に支援できる．

5.2 今後の課題

本稿のアルゴリズムでは，属性を扱えない，テキストノードの対応がつかなければならないなどの制限の下で，生成アルゴリズムを考えている．属性は，XML の特徴的な概念であるので，扱えるようにしていきたい．また，今回例にあげた書籍管理をしている XML 文書のように，テキストノードが対応していることは多くはないため，テキストノードに依存しすぎない手法を考える必要がある．また，4.2 節で述べたような能力を変換器に与えるためには，セマンティックな情報を扱える手法を導入していく必要がある．

文　献

[1] Summers, P, D.: “A Methodology for LISP Program Construction from Example”, Journal of the ACM, Vol.27, No.1, 1977, pp.162–175

[2] Biermann, A. W.: “The Inference of Regular LISP Programs from Examples”, IEEE Transactions on Systems, Man, and Cybernetics, Vol.8, No.8, 1978, pp.585–600

[3] Kitzelmann, E., Schmid, U.: “Inductive Synthesis of Functional Programs: an Explanation Based Generalization Approach”, Journal of Machine Learning Research 7, 2006, pp.429–454

[4] Microsoft XML Diff and Patch, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxmlnet/html/xmldiffgui.aspx

[5] Gauche, http://www.shiro.dreamhost.com/scheme/gauche/

[6] XPath, http://www.w3.org/TR/xpath

[7] XSLT, http://www.w3.org/TR/xslt

付　録

1. アルゴリズムの定義

```
T          : パステーブル
T_i, T_o   : T の（変換元，変換先）
T_i, T_o[i] : T_i, T_o の i 行目のタグ要素
ADD        : プログラム断片の適切な記述位置

var P;
stack S;
MAIN(XML 文書の対，DTD)
BEGIN
  T を作成;
  P に XSLT のひな形を代入;
  T_i[1] の template match 要素を作成;
  T_i のすべてのポインタを進める;
  T_o[1] のタグ要素を記述;
  T_o のすべてのポインタを進める;
  PUSH(S, P の ADD);
  CREATE_PROGRAM(T);
  PRINT P;
END

CREATE_PROGRAM(T)
BEGIN
  var TMP:  template match 要素をリスト管理;
  var POS:  P の記述位置;
  POS = POP(S);
  WHILE (T の行数 != 0)
    IF (T_o[1] がテキストノードを指している)
      IF (TMP に要素がない)
        POS に value-of 要素を記述;
        T_i[1] 以下の XPath を値とする;
      ELSE
        TMP の ADD に value-of 要素を記述;
        Ti[1] 以下の XPath を select 属性値とする;
      POS を更新;
      T の 1 行目を削除;
    ELSE IF ((T_o のすべてのタグ要素が同じ要素名，番号である)
      AND (T_o[1] の 1 つ前タグ要素を DTD から探し，
           T_o[1] が (*, +, ?) のいずれも付属していない)
      AND ((1 < T の行数) AND (T_i のすべてのタグ要素が同じ要素名，番号である)))
      IF (TMP に要素がない)
        T_o[1] のタグ要素を P の POS に記述;
        POS を更新;
      ELSE
        T_o[1] のタグ要素を TMP の ADD に記述;
      T_o のすべてのポインタを進める;
    ELSE IF (T_i のすべてのタグ要素が同じ要素名である)
      T = MERGE(T);
      IF (TMP に要素がない)
        T_i[1] を select 属性値とした apply-template 要素を P の POS に記述;
        POS を更新;
      ELSE
        T_i[1] を select 属性値とした apply-template 要素を TMP の ADD に記述;
      T_i[1] を属性値とした template match 要素を TMP に追加する;
      T_o[1] のタグ要素を TMP の ADD に記述;
      T_i のすべてのポインタを進める;
      T_o のすべてのポインタを進める;
    ELSE
      TMP のすべての template match 要素を P に追加;
      FOREACH T_o の要素の名前ごとに行単位で T を分割 (T^(1,...,j,...,n))
        P の POS を S に PUSH;
        CREATE_PROGRAM(T^(j));
        POS を更新;
      END FOREACH
    T のすべての行を削除;
  END WHILE
  POP(S);
  RETURN;
END


MERGE(T)
BEGIN
  T_i のタグ要素番号で分割;
  T_i のそれぞれのタグ要素以下の XPath を要素とした集合を作成;
  その集合の要素を DTD に記述される ELEMENT の接続演算子を用いてソート;
  ポインタが指すタグ要素の番号をすべて 1 にする;
  RETURN T;
END
```