タグと CSS に着目した Web ページのスキーマ化について

木村 智洋 † 萩原 威志 ††
Tomohiro KIMURA Takeshi HAGIWARA

† 新潟大学大学院自然科学研究科
Graduate School of Science and Technology, Niigata University
†† 新潟大学工学部情報工学科
Dept. of Imformation Engineering,Faculty of Engineering, Niigata University
kimura@cs.ie.niigata-u.ac.jp, hagiwara@ie.niigata-u.ac.jp

HTML は，ページレイアウトや表示方法を指定するためのものであるため，記述内容が本来持っていたはずの意味的構造が失われている場合が多い．我々は HTML ドキュメントからこの意味的構造を表すスキーマを得ることを目的として研究しているが，今回は，CSS の属性値や表示内容を利用して，共通するノードを発見し，複数の HTML ドキュメントから共通するスキーマを抽出する手法を提案する．

1 はじめに

インターネット上には様々な情報があふれ，我々の日常生活にかかせないメディアとなった．これらの情報のほとんどは HTML 形式で発信されているわけであるが，HTML は，ページレイアウトや表示方法を指定するための規格のため，記述内容が本来持っていたはずの意味的構造が失われている場合が多い．ここで言う意味的構造とは，たとえばニュース記事の見出しや日時，記事本文の組み合わせであったり，製品スペックの一覧表のスペック名とその値との関係のようなものをさすが，このような意味的構造は，人間がページを閲覧するときにそのレイアウトや文字装飾から認識している．しかし，計算機処理を行う場合には人間が行う解釈をプログラムして情報の仕分を行う必要がある．セマンティック Web の世界になってしまえば，このようなジレンマとは無縁になると思われるが，それはまだ先の話であり，そこまでの間には既存情報の再利用（情報抽出，変換など）の方法の確立が望まれる．

そこで，我々は HTML ドキュメントから意味的構造を表すスキーマを得られるのではないかと考え，[1] において HTML 記述のタグ構造に着目し HTML ドキュメントに含まれる情報を再利用しやすいスキーマを得ることを試みた．

同じタグ名でもその内部のタグ構造が少しでも異なれば，異なる意味構造として捉え，それを RelaxNG スキーマ [2] で表現し，逆に異なるタグでも同じ意味として扱いたい場合があるため，同意化という手法で対応していた．

しかし，近年レイアウト情報もタグ指定から CSS での指定へとシフトしてきているため，タグ構造のみに着目していてはなかなか役立つ構造を得ることができないという問題が大きくなっている．そこで，今までのタグ構造の抽出に加えて CSS [5] の属性値やクラス指定を利用して意味的構造を表現するスキーマ表現を得ることを考える．具体的には，コンテンツデザイナが意図した意味的構造が CSS のクラスや HTML の attribute 指定として残ることが多いため，これを [1] で行っていたコンパクション手続きで利用する．

また，[1] では，一つの HTML ファイルから一つのスキーマ表現を得ることしか出来なかったが，CSS の属性値や表示方法の違いを利用して，共通するノードを発見し，複数の HTML ドキュメントから共通するスキーマを抽出する手法を提案する．

2 スキーマ統合の概略

複数の HTML ドキュメントから統一されたスキーマを得る概略を図 1 に示す．図 1(1) 個別のスキーマの生成では，一つの HTML ドキュメントからは一つのスキーマが生成される．生成されたスキーマは基本的に元になった HTML ドキュメントのみを正確に表現する．文献 [1] で導入した同意化の手続きの設定しだいですべての HTML ドキュメントに適用可能な一つのスキーマを生成することは可能であるが，結構な手間とトライ＆エラーを繰り返す必要がある．

そこで，図 1(2) スキーマの統合の手続きを導入する．スキーマの統合に必要なことは，共通するノードを発見することである．そのために，CSS のクラス名や attribute 指定を利用し，クラス名や ID 名，スタイルの指定が同じで，かつ，同じタグ名であるならば，それらを共通するノードとしてみなすことにする．

第四節では，個別のスキーマ生成について説明し，第五節では，スキーマの統合の詳細を説明する．

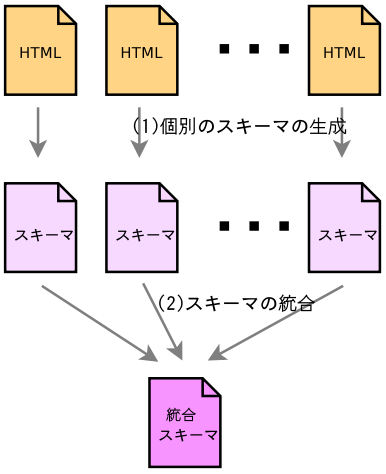

図 1: 統合スキーマの生成過程

3 属性値付き正規木文法

[1] では正規木文法 [4] をスキーマ生成過程で利用している．正規木文法は RelaxNG で XML ドキュメントを形式化する際に使用されているが，あるノードとその子要素との関係を表現し，木の深さを上手く表現できる．また属性も要素と同様に扱うことで表現できる．しかし，属性名は扱えるが属性値を扱えるわけではない．スキーマの統合では属性値を参照するので，ここでは，属性値を扱えるように正規木文法を拡張する．

定義 1 (属性値付き正規木文法) 属性値付き正規木文法は 4 つの組 $G = (N, T, S, P)$ によって定義される．ただし，

- N は非終端記号の集合，
- T は終端記号の集合，
- S は開始記号の集合 $(S \subseteq N)$，
- P は生成規則の集合で，$X \to aR_{att}R_{elem}$ の形をしている．ただし，$X \in N, a \in T$, R_{elem} は N 上の正規表現，$R_{att} \in T$,

R_{att} と R_{elem} を区別するのは，R_{att} は属性を表す生成規則の列であり，R_{elem} は要素を表す生成規則の列で，それらが混ざらないようにするためである．また，属性を表す生成規則では $(R_{att} \in T) \wedge (R_{elem} = \varepsilon$ で，R_{att} が属性値そのものを表すように記号名をつける．

4 個別のスキーマの生成

4.1 属性値付き正規木文法への変換

最初のステップでは，それぞれの HTML ドキュメントを属性値付き正規木文法へ変換する．一つのタグ a_m に対応する一つの生成規則 $X_m \to a_m R_m$ を生成する．ここで R_m はタグ a_m の子のタグに対応する生成規則の左辺である非終端記号の並びとなる．X_m は他の非終端記号と異なる限りユニークなものとなる．子がテキスト要素である場合は $X_m \to pcdata$ という生成規則となる．属性 att = "$value$" からは $Att_m \to att\ (value)$ という生成規則を生成する．$value$ をそのまま R にするのは，属性値を利用しやすくするためである．また，属性をもつタグの場合は，属性をそのタグの子タグと同様に見なし，それぞれの属性に対応する生成規則の左辺の非終端記号を子タグに対応する非終端記号より前に配置する．また開始記号は処理すべき HTML ドキュメントのルートノードに対応する生成規則の左辺の非終端記号である．

属性付き正規木文法への変換を以下の部分的な HTML ドキュメントを例として説明する。

```
<div class="class1"
     style="background-color:White;color:Black">
  <div>text1</div>
  <div>text2<b>text3</b></div>
</div>
```

これは以下のような属性付き正規木文法に変換される．

N = { $div_0, div_1, div_2, b_0, pcdata_0, pcdata_1, pcdata_2, Att_0, Att_1$}
T = { div,b,pcdata,class1, background-color:White;color:Black }
S = { div_0}
P = { div_0 → div ($Att_0, Att_1, div_1, div_2$),
div_1 → div ($pcdata_0$),
div_2 → div ($pcdata_1, b_0$),
b_0 → b ($pcdata_2$),
$pcdata_0$ → pcdata (), $pcdata_1$ → pcdata (),
$pcdata_2$ → pcdata (), Att_0 → class (class1),
Att_1 → style (background-color:White;color:Black) }

4.2 値を利用する属性の制限

スキーマの統合では CSS の属性値を利用するが，全ての属性値を厳密に使用すると共通ノードを発見できなくなってしまう．したがって扱う属性名と style 属性のプロパティを制限する．扱う属性名は class, id, style の３つである．style 属性のプロパティは color, background-color, font-size, font-wight, font-style に制限した．

4.3 異なるタグ名の同意化

本研究では，あるノード (タグ) からリーフ (テキスト要素，空要素) までの構造が少しでも異なれば，異なるノードとして考えている．しかしこれでは不都合が生じる場合がある．例えば，4.1 節の例のようにテキスト要素中にボールドがあると二階層目の二つの div タグ同士は異なる意味のタグとしてとらえられてしまう．本研究を実際にある HTML ドキュメントに適用してデータを吸い出すことを考えると，A タグでリンクを指定しているとそのデータの意味を人間が理解するのに不必要であることが多い．このようなことを防ぐために異なるタグ名の同意化という手順を導入する．

親タグ名と同意化したい子のタグ名の対を与えることでこれを設定する．親タグを $PTag$，子タグを $CTag_1, CTag_2, ..., CTag_n$ として，以下のように与える．

同意化の設定 $PTag : CTag_1 \mid CTag_2 \mid ... \mid CTag_n;$

異なるタグの同意化より前までに得られた属性付き正規木文法 $G = (N, T, S, P)$ を

N = { $X_0, X_1, ..., X_m$ }
T = { $a_0, a_1, ..., a_m$ }
S = { X_0 }
P = { $X_0 \rightarrow a_0\ R_{att0}\ R_{elem0}$,
$X_1 \rightarrow a_1\ R_{att1}\ R_{elem1}$,
...
$X_m \rightarrow a_m\ R_{attm}\ R_{elemm}$ }

とする．

手順 A 一つのタグの同意化の設定に対して，

1. $X \rightarrow PTag\ R_{att}R_{elem}$ となる生成規則を集合 P から取り出し，そのうちの R_{elem} に出現する非終端記号の集合を N_p とする．

2. 生成規則 $X \rightarrow aR_{att}R_{elem}$ ($X \in N_p$) として，a が $\{CTag_1, ..., CTag_n\}$ に含まれるような X の集合を $\{X_{c1}, ..., X_{cn}\}$ とする．

3. 生成規則 $X_b \rightarrow a_bR_{attb}R_{elemb}$ ($\in P$) として，a_b が $PTag$ で，R_{elemb} に出現する非終端記号すべてが集合 $\{X_{c1}, ..., X_{cn}\}$ に含まれるような X_b を列挙し，それを集合 N_b とする．

4. 生成規則 $X \rightarrow aR_{att}R_{elem}$ として，集合 N_b の各要素を左辺とする生成規則の R_{att} を列挙し，それを集合 N_{att} とする．

5. $N_{att} = \{A_1, ..., A_i\}$ として，新たな生成規則 $X_{a1} \rightarrow PTag(A_1, (X_{c1} \mid ... \mid X_{cn})+)$, ..., $X_{ai} \rightarrow PTag(A_i, (X_{c1} \mid ... \mid X_{cn})+)$ を属性付き正規木文法 G に追加する．ここで $X_a \notin \{X_1, ..., X_m\}$ である．

手順 A をタグの同意化の設定すべてに対して行う．

手順２までによって，集合 $\{X_{c1}, ..., X_{cn}\}$ のそれぞれの要素は PTag の直下に出現するノードでそのタグ名が $CTag_1, ..., CTag_n$ のいずれかである名前付きパターンの名前になる．同意化にまったく関係のない場所にある $CTag_1, ..., CTagn$ を同意化に含めないようにしている．

4.4 同じ構造をもつ生成規則の置換・変換

ここではどのような手順で生成規則を置換・変換するかを説明する．

開始記号から以下の手順 B を開始する．

手順 B ある生成規則 $X \rightarrow aR(R = R_{att}, R_{elem})$ について，

1. R に出現する非終端記号を左辺に持つ生成規則のうち手順 B を行っていない生成規則について手順 B を行う．

2. R において同じ非終端記号 X が連続するなら，その連続する非終端記号を $X+$ に置き換える．

3. aR と同じ右辺を持つ生成規則が存在するなら，その生成規則の左辺の非終端記号を X_n として，すべての生成規則の R に存在する X_n を X で置き換える．

4. aR で表現可能な左辺をもつ生成規則が存在するなら，その生成規則の左辺の非終端記号を X_m として，すべての生成規則の R に存在する X_m を X で置き換える．

5 スキーマの統合

ここでは，前節で述べた複数の属性付き正規木文法を一つの属性付き正規木文法に統合する方法を示す．最初に，それぞれの HTML ドキュメントから得た属性値付き正規木文法を結合する．その後，R_{att} として表現されている属性情報を利用し，タグの同意化を行った後，同じ構造をもつ生成規則の置換・変換を再び行う．以下，属性値付き正規木文法の結合と属性情報を用いたタグの同意化の手順を示す．

5.1 属性付き正規木文法の結合

属性付き正規木文法の結合は以下の手順で行う．

1. 複数の属性付き正規木文法を $G_1, ...G_n$ として，それぞれの開始記号を $html_1, ..., html_n$ とする．

2. 結合された属性付き正規木文法を $G_s = \{N_s, T_s, S_s, P_s\}$ として，まず，$S_s = \{root_1\}$ で，$root_1 \rightarrow root(html_1, ..., html_n)$ を P_s にいれる．

3. 属性付き正規木文法 $G_1, ..., G_n$ の生成規則を G_s に入れていく．その際，非終端記号が他の属性付き正規木文法と被らないように適宜変更する．

5.2 属性情報を用いたタグの同意化

生成規則 $X \rightarrow aR_{att}R_{elem}$ として，同じ a と R_{att} を持つ生成規則をそれぞれ $X_1 \rightarrow a_m\ R_{att}\ R_{elem1}, ..., X_n \rightarrow a_m\ R_{att}\ R_{elemn}$ とする．$R_{elem1}, ..., R_{elemn}$ に含まれる非終端記号を $X_{m1}, ..., X_{mm}$ として，$X_{m1}, ..., X_{mm}$ の a をそれぞれ，$a_{m1}, ..., a_{mm}$ とする．そして a_m と $a_{m1}, ..., a_{mm}$ を用いてタグの同意化の設定を生成させる．この場合の設定は，$a_m : a_{m1}|...|a_{mm};$ と生成される．このタグの同意化の設定を用いて，タグの同意化を行う．

タグの同意化のあと，同じ構造をもつ生成規則の置換・変換を再び行う．

6 適用例

適用例として，以下の３つの HTML ドキュメントの断片に対する統合されたスキーマを抽出してみる．

```
<div>                    | <div>
  <div class="class1">   |   <div class="class1">
    <a ... > text </a>   |     text
    text                 |     <a ... > text </a>
    <b> text </b>        |   </div>
  </div>                 |   <b> text </b>
  <b> text </b>          | </div>
</div>                   |
-------------------------------------------------
<div>
  <div class="class1">
    text
  </div>
  <b> text </b>
</div>
```

この HTML ドキュメントで共通のノードとなるのは一段目の div 同士であり，二段目の div 同士である．この HTML ドキュメントからはそれぞれ以下のような属性付き正規木文法の生成規則が得られる．

$P_0 =$ {
$div_0 \rightarrow$ div (div_1, b_0),
$div_1 \rightarrow$ div $(Att_0, a_0, pcdata_0, b_0)$,
$a_0 \rightarrow$ a $(pcdata_0)$,
$b_0 \rightarrow$ b $(pcdata_0)$,
$pcdata_0 \rightarrow$ pcdata(),
$Att_0 \rightarrow$ class (class1)
}

$P_1 =$ {
$div_0 \rightarrow$ div (div_1, b_0),
$div_1 \rightarrow$ div $(Att_0, pcdata_0, a_0)$,
$a_0 \rightarrow$ a $(pcdata_0)$,
$b_0 \rightarrow$ b $(pcdata_0)$,
$pcdata_0 \rightarrow$ pcdata(),
$Att_0 \rightarrow$ class (class1)
}

$P_2 =$ {
$div_0 \rightarrow$ div (div_1, b_0),
$div_1 \rightarrow$ div $(Att_0, pcdata_0)$,
$b_0 \rightarrow$ b $(pcdata_0)$,
$pcdata_0 \rightarrow$ pcdata(),
$Att_0 \rightarrow$ class (class1)
}

この三つの生成規則を一つの生成規則にまとめると以下のようになる．

$P =$ {
$root_0 \rightarrow$ root (div_0,div_1,div_2),
$div_0 \rightarrow$ div (div_1,b_0),
$div_1 \rightarrow$ div (Att_0,a_0,$pcdata_0$,b_0),
$a_0 \rightarrow$ a ($pcdata_0$),
$b_0 \rightarrow$ b ($pcdata_0$),
$pcdata_0 \rightarrow$ pcdata (),
$Att_0 \rightarrow$ class (class1),
$div_2 \rightarrow$ div (div_3,b_1),
$div_3 \rightarrow$ div (Att_1,$pcdata_1$,a_1),
$a_1 \rightarrow$ a ($pcdata_1$),
$b_1 \rightarrow$ b ($pcdata_1$),
$pcdata_1 \rightarrow$ pcdata (),
$Att_1 \rightarrow$ class (class1),
$div_4 \rightarrow$ div (div_5,b_2),
$div_5 \rightarrow$ div (Att_2,$pcdata_2$),
$b_2 \rightarrow$ b ($pcdata_2$),
$pcdata_2 \rightarrow$ pcdata (),
$Att_2 \rightarrow$ class (class1)
}

この生成規則には，$div : a|b|pcdata;$ というタグの同意化を行えばよい．結果以下の生成規則を得る．

$P =$ {
$div_0 \rightarrow$ div (div_6,b_0),
$a_0 \rightarrow$ a ($pcdata_0$),
$b_0 \rightarrow$ b ($pcdata_0$),
$pcdata_0 \rightarrow$ pcdata (),
$Att_0 \rightarrow$ class (class1),
$div_6 \rightarrow$ div (Att_0,($a_0|b_0|pcdata_0$)+)
}

これによって得られたスキーマは，一段目がタグ div で，二段目はタグ div と タグ b の並び，二段目のタグ div の内容はタグ a，タグ b，テキスト要素の複数の列を許すものである．また，スキーマが生成される元となった HTML ドキュメントのどれをもこのスキーマで表現できている．

7 おわりに

本論文では属性値を参照できるように正規木文法を拡張し，属性値に基づいて複数のスキーマに共通するノードを発見し，そのノードに関してタグの同意化を行うことで統合されたスキーマを抽出する手法を提案した．CSS の属性値がしっかりと管理されている HTML ドキュメント群に適用できるスキーマを生成することが可能となった．しかし，クラス名がタグ毎に異なるような HTML ドキュメントには適用することができない．また，構造のまったく異なるドキュメントを判別することも必要である．逆にそのようなドキュメントから何か共通する構造を抽出することができれば，面白いかもしれない．

参考文献

[1] 木村智洋，萩原威志，XHTML のスキーマ情報の抽出，日本ソフトウエア科学会第 20 回大会論文集 2B-5,2003

[2] RELAX NG Specification, Committee Specification 3 December 2001, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[3] RELAX NG Compact Syntax, Committee Specification 21 November 2002. http://www.oasis-open.org/committees/relax-ng/compact-20021121.html

[4] M. Murata and D. Lee and M. Mani, Taxonomy of XML Schema Languages using Formal Language Theory, In *Extreme Markup Languages*, Montreal, Canada, Aug. 2001.

[5] Cascading Style Sheets: http://www.w3.org/Style/CSS/