February 24, 2000
RT0346
Computer Science 10 pages

Research Report

The Boolean Operation and The Minimalization for XML Schema Based on Tree Automaton Theory

Kazuyo Yagishita and Hiroshi Maruyama

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Tree Automaton Theoryに基づく XML SchemaのBoolean OperationとMinimalization

八木下 和代, 丸山 宏

平成12 年 2 月 24 日

概 要

XML(Extensible Markup Language) の普及の背景には, ネットワーク上に散在する複数の文書を連携させ, アプリケーションを用いて処理したいという要求がある. よって我々はXMLの構造を規定するスキーマを操作し, union などの boolean operation や最小化が可能となれば, 様々な場面で活用できると考える.

また, 木の集合を表現するためのオートマトンとして木オートマトン (Tree Automaton) が知られている. 木オートマトンは, boolean operation に関して閉じていることが証明されているT-regular set のクラスに属する言語 (T-regular language) を受理し, boolean operation の処理方法や最小化のための手法が提案されている. よって, 要素の木構造であるXML 文書を表現しているXML のスキーマを, 木オートマトンによって記述することができれば, 木オートマトン理論に基づいてそれらの boolean operation や最小化が可能となる.

本論文では, まずT-regular set の定義と木オートマトンによる記述方法について述べた後, 木オートマトンの boolean operation や最小化のための手法を説明する. 次にT-regular set とXML のスキーマとの関連に着目し, 現在W3C で制定が勧められているXML Schema(W3C Working Draft 17 December 1999) を用いて, その木オートマトンによる記述方法を提案する. さらに簡単な例を用いて, 木オートマトン理論に基づくXML Schema の union の求め方および最小化の方法を具体的に示す.

keywords: XML Schema, T-regular, 木オートマトン (Tree Automaton), boolean operation, minimalization

1. はじめに

XML(Extensible Markup Language) の普及の背景には, ネットワーク上に散在する多くの文書をアプリケーションを用いて処理したいという要求がある. その要求は以下の 3 つのタイプに分類される [1].

1. 複数の文書データベースを Web によって連携させたい.
2. クライアントで文書情報に対する処理を行いたい.
3. WWW エージェントによる文書情報の選択を可能にしたい.

ここでは特に 1 に注目する. 例えば A 社が B, C 社と同様の取引を行っている場合, 必要書類を共通のスキーマに従ったXML を用いて記述・交換していれば, A 社はそのスキーマを処理するためのアプリケーションを 1 つ用意するだけで, これらの文書を処理することが可能となる. しかしB, C 社がそれぞれ独自のスキーマを使用している場合, A 社は各スキーマに応じた処理を行うために, 複数のアプリケーションプログラムを用意する必要が生じる. しかしこのような場合でも, A-B, A-C 間でやり取りされるXML 文書は類似した内容と構造を持つため, 類似したスキーマで表現されていると考えられる. よって, A 社がB, C 社の使用しているスキーマのunion を求めることが可能となれば, 結果として得られたスキーマを処理するためのアプリケーションを 1 つ用意すれば良いこととなる. ここで, 複数のスキーマのunion を取るとは, 各スキーマによって受理されるXML 文書すべてを受理し, かつそれ以外のXML 文書は受理しないような

スキーマを求めることを意味する. これは一例に過ぎないが, 我々は union を含め, スキーマ間の boolean operation(union, intersection など) の処理が可能となれば, 様々な場面で活用できると考える.

また, 通常の文字列を扱うオートマトンを拡張し, 木の集合を表現するためのオートマトンとして木オートマトン (Tree Automaton) が知られている [2]. ここで, 木オートマトンが受理できる言語 (木の集合) は T-regular set のクラスに属する言語 (T-regular language) である. XML 文書は要素の木構造であるため, そのスキーマは木の集合を表現している. よってスキーマによって受理される XML 文書が T-regular set のクラスに属していれば, スキーマを木オートマトンを用いて表現することが可能となる. 木オートマトンはその union や intersection の構成方法が提案されているため [2], スキーマを木オートマトンを用いて表現することができれば, 上記の演算処理が可能となる. また木オートマトンを最小化するための方法も提案されているため [3], これを応用することでスキーマの最小化も可能となると考えられる.

以下第 2 章で, T-regular set の定義と特徴について説明し, 次いで第 3 章で, 木オートマトンの定義や処理方法について述べる. その後第 4 章で, XML 文書のスキーマとして現在制定が進められている XML Schema(W3C Working Draft 17 December 1999) を用いて, 木オートマトンによる記述方法を提案し, また XML Schema の union の求め方や最小化の仕方を具体例を用いて説明する. 最後に第 5 章で, まとめと今後の課題について述べる.

2. T-regular set[4]

2.1. Ranked Alphabet と Tree

有限集合 A に対して, $\gamma_A \subseteq A \times \{\ 0,\ 1,\ 2,\ \cdots\ \}$ を満たす relation γ_A を A の rank と言い, このとき A を ranked alphabet と呼ぶ. 任意の k (≥ 0) に対して, $\{\ a \in A \mid (a,\ k) \in \gamma_A\ \}$ を $A^{(k)}$ で示す. 例えば $A = \{\ a,\ b,\ c\ \}$ とし, 各要素は $\gamma_A = \{\ (a,0),\ (a,2),\ (b,1),\ (c,1)\ \}$ によってランク付けされているとする. このとき $A^{(k)}$ ($k = 0,\ 1,\ 2$) は以下のようになる.

$$A^{(0)} = \{\ a\ \},\quad A^{(1)} = \{\ b,\ c\ \},\quad A^{(2)} = \{\ a\ \}$$

ranked alphabet A に対して, 以下の定義を満たす最小の木集合を T_A とする.

1. if $a \in A^{(0)}$, then $a <>= a < \epsilon >\in T_A$,
2. if $k > 0 \wedge a \in A^{(k)} \wedge (t_1,\ t_2,\ \cdots,\ t_k \in T_A)$, then $a < t_1 t_2 \cdots t_k > \in T_A$.

すなわち ranked alphabet の要素の rank は, その要素の子供要素の数を表わしている. また $t \in T_A$ に対して, $Branch(t)$ を以下のように定義する.

1. $\{a\}$, if $t = a <>= a \in A^{(0)}$,
2. $\{a < Root(t_1)Root(t_2) \cdots Root(t_k)\} \cup \{\cup_{1 \leq j \leq k} Branch(t_j)\}$,
   if $t = a < t_1 t_2 \cdots t_k >\ \ (k > 0 \wedge a \in A^{(k)} \wedge (t_1, t_2, \cdots, t_k \in T_A))$.

ここで $Root(t)$ は t のルートを表わす. $t = a < b < ac > aa >$ とすると $Branch(t) = \{\ a < baa >,\ b < ac >,\ a,\ c\ \}$ となる.

2.2. T-grammar と T-local set

ranked alphabet K, $P \subseteq Branch(T_K)$, $I \subseteq K$ の 3 つ組からなら構成される $G =< K,\ P,\ I >$ を T-grammar と呼ぶ. ここで P は以下の条件を満たすものとする.

- $\exists X \in K$ に対して, 集合 $P(X) = \{\ x \in K^* \mid X < x >\in P\ \}$ は K 上の正規集合 $(\subseteq K\ *)$ である.

そして木集合 $\{\ t \in T_K \mid Root(t) \in I,\ Branch(t) \subseteq P\ \}$ を G によって生成される T-local set と呼ぶ.

2.3. Projection

A, B を ranked alphabet, $h : A \to B$ を $\forall k \geq 0$ に対して $h(A^{(k)}) \subseteq B^{(k)}$ となるような関数とする. この時, 関数 h を以下のように拡張し, $h' : T_A \to T_B$ を定義する.

1. if $t = a \in A^{(0)}$, then $h'(t) = h(t) \in B^{(0)}$,
2. if $t = a < t_1 t_2 \cdots t_k >$ $(k > 0 \wedge a \in A^{(k)} \wedge (t_1, t_2, \cdots, t_k \in T_A))$,
   then $h'(t) = h(a) < h'(t_1) h'(t_2) \cdots h'(t_k) > \in T_B$.

関数 $h' : T_A \to T_B$ を木に関する projection と呼ぶ. 定義より projection は, 木の枝や葉の数は変化させず, ノード記号のみを変化させるような関数である.

2.4. T-regular set

ある T-local set $T \subseteq T_K$ と projection $h' : T_K \to T_A$ が 存在するとき, $h'(T) = \{ h'(t) \mid t \in T \}$ を T-regular set と定義する. この T-regular set は boolean operation (union, intersection 等) の下で閉じていることが知られている [4].

3. Tree Automaton

3.1. 定義

3.1.1. 決定性木オートマトン (Deterministic Tree Automaton, DTA)

決定性木オートマトン $M =< Q, \Sigma, \alpha, F >$ は以下のように定義される.

Q : 有限状態集合
Σ : *alphabet*
$\alpha : \Sigma \times Q^*$ から Q への遷移関数
　ただし $\forall x \in Q,\ a \in \Sigma,\ u \in Q^*$ に対して, $\{ u \mid \alpha(a, u) = x \}$ は正規集合である.
F : 受理状態集合 ($F \subseteq Q$)

3.1.2. 非決定性木オートマトン (Non-Deterministic Tree Automaton, NDTA)

非決定性木オートマトン $M =< Q, \Sigma, \alpha, F >$ は以下のように定義される.

Q, Σ, F : 決定性木オートマトンの定義と同じ
$\alpha : \Sigma \times Q^*$ から Q^* への関数
　ただし $\forall X \in Q^*,\ a \in \Sigma,\ u \in Q^*$ に対して, $\{ u \mid \alpha(a, u) = X \}$ は正規集合である.

3.2. 木オートマトンと T-regular set

3.2.1. DTA と NDTA との等価性

決定性木オートマトンによって受理されるような言語を T-regular language と呼ぶ. ここで, 決定性および非決定性の木オートマトンの各定義より明らかに, 決定性木オートマトンならば非決定性木オートマトンである. また, ある言語が非決定性木オートマトンによって受理されるならば, その言語は T-regular であり, かつ T-regular に属する言語のみが非決定性木オートマトンによって受理されることが知られている. すなわち決定性木オートマトンと非決定性木オートマトンとは等価である [2]. 以下に非決定性木オートマトンを決定性木オートマトンに変換する手順を示す.

非決定性木オートマトン, および決定性木オートマトンを各々以下のように定義する.

非決定性木オートマトン $M =< Q,\ \Sigma,\ \alpha,\ F >$
決定性木オートマトン $M' =< Q',\ \Sigma,\ \beta,\ F' >$

- $Q' \subseteq 2^Q$
- $F' \subseteq Q'$
- $X \in F' \Leftrightarrow \exists x \in X$ に対して $x \in F$

ここで, 変換後の決定性木オートマトンの各要素は以下のようにして求められる.

1. 関数 α から以下の条件を満たすような関数 β' を求める.

   $x \in Q,\ a \in \Sigma$ に対して, $x \in \beta'(a, \delta(\in L(\_u))) \Leftrightarrow x \in \alpha(a, v(\in L(u)))$

   - $\_u \Leftrightarrow$ 正規表現 u 中の任意の要素 $y(\in Q)$ を $\_y$ と置き換えることで得られる正規表現
   - $\_y = y_1 \mid y_2 \mid \cdots \mid y_n$ (ただし各 $i(= 1, 2, \cdots, n(= 2^{|Q|-1}))$ に対して $y_i \in 2^Q \wedge y \in y_i$)

2. 次に上記 1 で求めた関数 β' から以下の条件を満たすような関数 β を求める.

   $X \in 2^Q$ に対して, $X = \beta(a, \gamma) \Leftrightarrow \gamma \in (\cap_{x \in X} L(\gamma(x))) \cap (\cap_{x \in \overline{X}} \overline{L(\gamma(x))})$

   - $\gamma(x) \Leftrightarrow x \in \beta'(a, \delta(\in L(\gamma(x))))$ を満たす 2^Q 上の正規表現
   - ただし関数 β' 中に上記のような定義がない場合, 下記のような定義が存在するものとする.

     $x \in \beta'(a, \delta(\in \overline{(2^Q)^*}))$

3. 非到達状態 $X\ (\in 2^Q)$ を除去する.

   (a) 上記 2 で求めた関数 β の γ 条件が ϵ を含む状態集合 $X\ (\in 2^Q)$ にチェックを入れる. $i = 1$ とする.
   (b) 第 i 段階でチェックのついた状態から遷移できる状態のうち, まだチェックのついていないものすべてにチェックを入れる. $i = i + 1$ とする.
   (c) 新たにチェックがつかなくなれば終了する. チェックのついていない状態は非到達状態であるため除去する. 残った状態からなる集合が Q' である.

3.2.2. T-regular language の NDTA による記述

ある T-regular language L が T-grammar $G =< K,\ P,\ I >$, および関数 $h(x)$ ($x \in K$) から導出される projection によって記述されているとする. このときこの言語 L を受理する非決定性木オートマトン $M =< Q,\ \Sigma,\ \alpha,\ F >$ は以下のように作成される.

$Q = K$
$\Sigma = \{\ h(x) \mid x \in K\ \}$
$x \in \alpha(a,\ v) \Leftrightarrow x < u > \in P \wedge a = h(x) \wedge v \in L(u)$
$F = I$

3.3. Boolean Operation

以下に, 文献 [2] を参考にしながら決定性木オートマトンの boolean operation の求め方を示す.

- $L,\ L_1,\ L_2,\ L'$ は T-regular language
- $M(L) =< Q,\ \Sigma,\ \alpha,\ F >$
- $M(L_i) =< Q_i,\ \Sigma_i,\ \alpha_i,\ F_i >$ ($i = 1,\ 2$), $Q1 \cap Q2 = \phi$
- $M(L') =< Q',\ \Sigma',\ \alpha',\ F' >$
- $M(L),\ M(L_i)$($i = 1,\ 2$) はすべて決定性木オートマトンである.
- $M(L')$ は非決定性木オートマトンである可能性がある.

union $L' = L_1 \cup L_2$　$M(L')$ は $M(L_1)$ または $M(L_2)$ の少なくともいずれか一方で受理される木のみ, かつそのような木すべてを受理する.

$Q' = Q_1 \cup Q_2$
$\Sigma' = \Sigma_1 \cup \Sigma_2$
$x \in \alpha'(a, v) \Leftrightarrow x = \alpha_1(a, v) \vee x = \alpha_2(a, v)$
$F' = F_1 \cup F_2$

negation $L' = \overline{L}$　$M(L')$ は $M(L)$ で受理されない木のみ, かつそのような木すべてを受理する.

$Q' = Q$
$\Sigma' = \Sigma$
$x = \alpha'(a, v) \Leftrightarrow x = \alpha(a, v)$
$F' = Q - F$

intersection $L' = L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$　$M(L')$ は $M(L_1)$ と $M(L_2)$ で共に受理される木のみ, かつそのような木すべてを受理する.

difference $L' = L_1 - L_2 = L_1 \cap \overline{L_2}$　$M(L')$ は $M(L_1)$ では受理され, かつ $M(L_2)$ では受理されないような木のみ, かつそのような木すべてを受理する.

concatenation $L' = L_1 \bullet_s L_2$　$M(L')$ は L_1 中の木のノード s 以下を L_2 中の任意の木で置き換えた木のみ, かつそのような木すべてを受理する. ここで s ($\in \Sigma_1$) が L_1 中の木のノードとして出現する場合は必ず葉であり, 割り当てられる状態を q_s とする. q_s は他のノードには割り当てられないものとする.

$Q' = (\ Q_1 - \{\ q_s\ \}\ ) \cup Q_2$
$\Sigma' = (\ \Sigma_1 - \{\ s\ \}\ ) \cup \Sigma_2$
$x \in \alpha'(a, v) \Leftrightarrow x = \alpha_1(a, f(v)) \vee x = \alpha_2(a, v)$

- $f(v) \Leftrightarrow$ 文字列 v 中に現れる F_2 の要素をすべて q_s によって置き換えた文字列

$$F = \begin{cases} (F_1 - \{q_s\}) \cup F_2 & \text{if } q_s \in F_1 \\ F_1 & \text{otherwise} \end{cases}$$

3.4. DTA の最小化

決定性木オートマトンの状態間の congruence $\sim$ を見つける. 状態 x と y が congruence の関係にある ($\Leftrightarrow x \sim y$ が成り立つ) 場合, x 及び y から始まる遷移は完全に一致する. 以下に, 文献 [3] を拡張する形で決定性木オートマトン $M =< Q,\ \Sigma,\ \alpha,\ F >$ の最小化手順 (congruence 関係の求め方) を示す.

定義) $Q(i) \Leftrightarrow i$ 回目の分割によってできた状態集合 Q の要素集合の集合

1. Q の要素を受理状態とそうでない状態とに分割する. $i = 1$ とする.
2. $i = k(k \geq 1)$ とする. また $Q(i) = \{\ B_1,\ B_2,\ \cdots,\ B_m\ \}$ とする. この時 $\forall a \in \Sigma$ に対して以下のような集合 $A(a)$ を求める.

   - $A(a) = \{\ B \in Q(i) \mid \cup_{x \in B} L(u(x)) \subseteq Q^*\ \}$
     - ただし $L(u(x)) = \begin{cases} L & \text{if } x = \alpha(a, v(\in L)) \\ \overline{Q^*} & \text{otherwise} \end{cases}$

   ここで, $x \sim_i y (\Leftrightarrow B \in Q(i) \wedge x \in B \wedge y \in B \wedge x \neq y)$ とする. この時 $\forall B \in A(a)$ に対して $\cup_{x \in B} L(u(x))$ の要素中の任意の x および y を ($x \mid y$) に置き換えてできた言語を L' とする. $x \sim_{i+1} y$ ($\Leftrightarrow$ 次の分割後も x と y は同じ同値類に属する) であるための条件は以下の通りである.

   $\forall a \in \Sigma,\ \forall B \in A(a)$ に対して $\cup_{x \in B} L(u(x)) = L'$

3. $i = k + 1$ として, 上記 2 に基づいて新たに状態を分割する.
4. 2, 3 を, $Q(i) = Q(i + 1)$ が成り立つまで繰り返す.
5. 上記 4 の条件が成り立ったとき $i = n$ であったとする. このとき $Q(n)$ 中の要素を各々1 つの状態とみなして決定性木オートマトンを構成し直す. 得られた決定性木オートマトンが求めたい最小化された木オートマトンである.

4. XML Schema

4.1. XML Schema の特徴

XML Schema は, XML 文書に対する制約を XML のシンタックスを用いて表現するための仕組みである. 従来の DTD の単なる書き換えではなく, DTD では表現できなかったような新しい種類の制約を定義することが可能となっている. 例えば DTD では, ある要素の型として文字データを指定することはできるが, その文字データの型 (例えばある特定の範囲の整数など) までは制限できない. しかし XML Schema では *datatype* を用いることによって, 上記のことが可能となる. また要素宣言の際に type 属性を用いることによって, 同一タグ名であっても出現位置の異なる要素を区別することが可能である. XML Schema が有するこのような特徴のうち, 我々は特に後者に着目した. これは木オートマトンにおいて, 状態記号とノード記号とが異なることに対応していると考えられる. よって我々は, XML Schema は木オートマトンによって表現することが可能であると考える.

4.2. 木オートマトンによる記述

XML Schema の決定性木オートマトンによる記述方法を示す.

定義

- E : タグ名の集合
- R : ルートになり得るタグ名の集合
- T : type 属性の集合
- $T(e)$: $e \in E$ と関連のある type 属性の集合

XML Schema の DTA による記述

1. XML Schema は, 以下の条件を満たす T-grammar $G =< K,\ P,\ I >$ と projection $h'(T)(T \in T_K)$ からなる T-regular language として表現できる. ただし $h'(T)$ は 2.3 節の定義に従って関数 $h(x)(x \in K)$ から生成される関数とする.
   - $K = (\cup_{e \in E}(\{e\} \times T(e))) \cup \{datatypes\}$
   - $P = \{x < u > \mid x \in K, u$ は x の内容モデルに対応した K 上の正規表現 $\}$
   - $I = \cup_{e \in R}(\{e\} \times T(e))$
   - $h(x) : K \to (E \cup \{datatypes\})$
2. T-regular language を 3.2.2 節に従い非決定性木オートマトンを用いて記述する.
3. 非決定性木オートマトンを 3.2.1 節に従い決定性木オートマトンへ変換する.

4.3. XML Schema の操作例

先に述べた方法を用いて, 実際に 2 つの XML Schema の union を求め, その結果を最小化してみる. 用いる XML Schema を表 1 に示す.

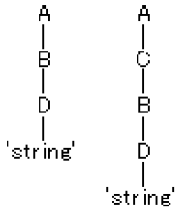
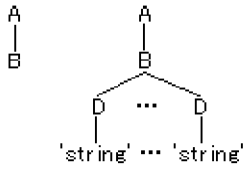
XML Schema 1	**XML Schema 2**
< *schema ...* > < *element name =′ A′* < *type* > < *group order =′ choice′* > < *element name =′ B′ type =′ typeB′/* > < *element name =′ C′ type =′ typeC′/* > < */group* > < */type* > < */element* > < *type name =′ typeB′* > < *element name =′ D′ type =′ string′/* > < */type* > < *type name =′ typeC′* > < *element name =′ B′ type =′ typeB′/* > < */type* > < */schema* >  	< *schema ...* > < *element name =′ A′* > < *type* > < *element name =′ B′ type =′ typeB′/* > < */type* > < */element* > < *type name =′ typeB′* > < *element name =′ D′ type =′ string′ maxOccurs =′ ∗′/* > < */type* > < */schema* > 

表 1: XML Schema とその木構造

4.2 節の番号	**XML Schema 1**	**XML Schema 2**
0	$E = \{A, B, C, D\}$ $R = \{A\}$ $T = \{typeB, typeC, string\}$ $T(B) = \{typeB\},\ T(C) = \{typeC\},\ T(D) = \{string\}$	$E = \{A, B, D\}$ $R = \{A\}$ $T = \{typeB, string\}$ $T(B) = \{typeB\},\ T(D) = \{string\}$
1	$K = \{A, BtypeB, CtypeC, Dstring, 'string'\}$ $P = \{A < BtypeB \mid CtypeC >, BtypeB < Dstring >,$ $\quad CtypeC < BtypeB >, Dstring <' string' >,$ $\quad 'string' < \epsilon >\}$ $I = \{A\}$ $h(A) = A,\ h(BtypeB) = B,\ h(CtypeC) = C,$ $\quad h(Dstring) = D,\ h('string') =' string'$	$K = \{A, BtypeB, Dstring, 'string'\}$ $P = \{A < BtypeB >, BtypeB < Dstring^* >,$ $\quad Dstring <' string' >,' string' < \epsilon >\}$ $I = \{A\}$ $h(A) = A,\ h(BtypeB) = B,\ h(Dstring) = D,$ $\quad h('string') =' string'$
2, 3	DTA $M_1 =< Q_1, \Sigma_1, \alpha_1, F_1 >$ $Q_1 = \{A_1, BtypeB_1, CtypeC_1, Dstring_1, 'string'_1\}$ $\Sigma_1 = \{A, B, C, D, 'string'\}$ $\alpha_1 : \Sigma_1 \times Q_1^* \to Q_1$ $A_1 = \alpha_1(A, u(\in L(BtypeB_1 \mid CtypeC_1)))$ $BtypeB_1 = \alpha_1(B, u(\in L(Dstring_1)))$ $CtypeC_1 = \alpha_1(C, u(\in L(BtypeB_1)))$ $Dstring_1 = \alpha_1(D, u(\in L('string'_1)))$ $'string'_1 = \alpha_1('string', u(\in L(\epsilon)))$ $F_1 = \{A_1\}$	DTA $M_2 =< Q_2, \Sigma_2, \alpha_2, F_2 >$ $Q_2 = \{A_2, BtypeB_2, Dstring_2, 'string'_2\}$ $\Sigma_2 = \{A, B, D, 'string'\}$ $\alpha_2 : \Sigma_2 \times Q_2^* \to Q_2$ $A_2 = \alpha_2(A, u(\in L(BtypeB_2)))$ $BtypeB_2 = \alpha_2(B, u(\in L(Dstring_2^*)))$ $Dstring_2 = \alpha_2(D, u(\in L('string'_2)))$ $'string'_2 = \alpha_2('string', u(\in L(\epsilon)))$ $F_2 = \{A_2\}$

表 2: XML Schema の DTA による表現

Step1:DTA の作成　表 1 中の XML Schema をそれぞれ決定性木オートマトンで表現した結果を表 2 に示す. 具体的な求め方は 4.2 節に基づいている.

Step2:union の計算　3.3 節を参考にしながら, 表 2 中の 2 つの XML Schema の union を求めた結果を以下に示す. なお結果は非決定性木オートマトンとなる.

NDTA $M =< Q, \Sigma, \alpha, F >$

$Q = \{A_1, BtypeB_1, CtypeC_1, Dstring_1, 'string'_1, A_2, BtypeB_2, Dstring_2, 'string'_2\}$
$\Sigma = \{A, B, C, D, 'string'\}$
$\alpha : \Sigma \times Q^* \to Q^*$
$A_1 \in \alpha(A, u(\in L(BtypeB_1 \mid CtypeC_1)))$
$BtypeB_1 \in \alpha(B, u(\in L(Dstring_1)))$
$CtypeC_1 \in \alpha(C, u(\in L(BtypeB_1)))$
$Dstring_1 \in \alpha(D, u(\in L('string'_1)))$
$'string'_1 \in \alpha('string', u(\in L(\epsilon)))$
$A_2 \in \alpha(A, u(\in L(BtypeB_2)))$
$BtypeB_2 \in \alpha(B, u(\in L(Dstring_2^*)))$
$Dstring_2 \in \alpha(D, u(\in L('string'_2)))$
$'string'_2 \in \alpha('string', u(\in L(\epsilon)))$
$F = \{A_1, A_2\}$

Step3:NDTA から DTA への変換　3.2.1 節に基づいて, Step2 で得られた非決定性木オートマトンを決定性木オートマトンに変換する.

DTA $M' =< Q', \Sigma, \alpha', F' >$

$Q' = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

3.4 節の番号	
1	$\mathbf{i = 1}$ $Q(1) = \{B_1, B_2\}$ $B_1 = \{q_1, q_2, q_3\}$ $B_2 = \{q_4, q_5, q_6, q_7, q_8\}$
2, 3, 4	$\mathbf{i = 2}$ $Q(2) = \{B_1, B_2, B_3, B_4, B_5\}$ $B_1 = \{q_1, q_2, q_3\}$ $B_2 = \{q_4\}$ $B_3 = \{q_5, q_6\}$ $B_4 = \{q_7\}$ $B_5 = \{q_8\}$ $\mathbf{i = 3}$ $Q(2) = Q(3)$
5	最小化された DTA $M_{min} =< Q_{min}, \Sigma_{min}, \alpha_{min}, F_{min} >$ $Q_{min} = \{r_1, r_2, r_3, r_4, r_5\}$ $r_1 = B_1 = \{q_1, q_2, q_3\}$ $r_2 = B_2 = \{q_4\}$ $r_3 = B_3 = \{q_5, q_6\}$ $r_4 = B_4 = \{q_7\}$ $r_5 = B_5 = \{q_8\}$ $\Sigma_{min} = \{A, B, C, D, 'string'\}$ $\alpha_{min} : \Sigma_{min} \times Q^*_{min} \to Q_{min}$ $r_1 = \alpha_{min}(A, u(\in L(r_2 \mid r_3)))$ $r_2 = \alpha_{min}(B, u(\in L(r_4)))$ $r_3 = \alpha_{min}(B, u(\in L(\epsilon \mid r_4 r_4^+))) \vee \alpha_{min}(C, u(\in L(r_2)))$ $r_4 = \alpha_{min}(D, u(\in L(r_5)))$ $r_5 = \alpha_{min}('string', u(\in L(\epsilon)))$ $F_{min} = \{r_1\}$

表 3: 決定性木オートマトンの最小化

$q_1 = \{A1\}$
$q_2 = \{A2\}$
$q_3 = \{A1, A2\}$
$q_4 = \{BtypeB1, BtypeB2\}$
$q_5 = \{BtypeB2\}$
$q_6 = \{CtypeC1\}$
$q_7 = \{Dstring1, Dstring2\}$
$q_8 = \{'string'1, 'string'2\}$
$\Sigma' = \{A, B, C, D, 'string'\}$
$\alpha' : \Sigma \times Q'^* \to Q'$
$q_1 = \alpha'(A, u(\in L(q_6)))$
$q_2 = \alpha'(A, u(\in L(q_5)))$
$q_3 = \alpha'(A, u(\in L(q_4)))$
$q_4 = \alpha'(B, u(\in L(q_7)))$
$q_5 = \alpha'(B, u(\in L(\epsilon \mid q_7 q_7^+)))$
$q_6 = \alpha'(C, u(\in L(q_4)))$
$q_7 = \alpha'(D, u(\in L(q_8)))$
$q_8 = \alpha'('string', u(\in L(\epsilon)))$
$F' = \{q_1, q_2, q_3\}$

Step4:最小化 Step3 で得られた決定性木オートマトンを 3.4 節をもとに最小化する. 結果を表 3 に示す.

5. おわりに

XML のスキーマを操作して, union などを求めたり最小化したりすることは実用上意味があることと考えられる. 現在, union などの boolean operation に関して閉じている木集合のクラスとして T-regular set が知られている. そして T-regular set のクラスに属する木集合である T-regular language は, 木オートマトンによって受理することができる. この木オートマトンは, 通常の文字列を扱うオートマトンを木集合を表現できるように拡張したものであり, その boolean operation の求め方や最小化の方法が提案されている. よって木オートマトンを用いることによって, T-regular language の boolean operation や最小化を行うことが可能である.

本論文では, まず T-regular set と木オートマトンの定義について述べた後, 木オートマトンの boolean operation や最小化の手順について詳述した. 次いで, T-regular language と XML のスキーマとの関連に着目し, 現在 W3C で制定が勧められている XML Schema を用いて, 木オートマトンによる記述方法を提案した. さらに簡単な例を用いて, 木オートマトンの理論に基づく XML Schema の操作方法について説明した.

今後の課題として以下のものが挙げられる.

1. 木オートマトンから XML Schema への変換方法の考察

   本論文では, XML Schema から非決定性木オートマトン, ひいては決定性木オートマトンへの変換方法のみを提案し, その逆の変換方法には触れていない. しかし, 木オートマトンを用いて XML Schema の union を求めたり, 最小化したりした後, 結果を再び XML Schema で表現したいという要求は存在するものと思われる. よって今後は, 木オートマトンから XML Schema への変換方法についても考察する必要がある.

2. XML Schema で定義される XML 文書に対する制約すべてに対する対応

   本論文では, XML Schema で定義されうる XML 文書に対する制約のうち, element にのみ着目し, その他の制約 (attribute など) に対する処理方法は考慮されていない. よって今後は, XML 文書に対する全ての制約に範囲を広め, 木オートマトンによる表現方法や処理方法を確立していく必要がある.

参考文献

[1] 村田真, 門馬敦仁, 荒井恭一 : XML 入門, 日本経済新聞社, 1998.

[2] Murata Makoto. : Forest-regular Languages and Tree-regular Languages, 1995.

[3] Walter S, Brainerd. : The Minimalization of Tree Automata, *Information and Control*, Vol. 13, pp. 484-491, 1968.

[4] M, Takahashi. : Generalizations of Regular Sets and Their Application to a Study of Context-Free Languages, *Information and Control*, Vol. 27, pp. 1-36, 1975.