XPath 充足可能性を判定する多項式時間アルゴリズムの実装と評価

Implementation and evaluation of a polynomial-time algorithm
for deciding the XPath satisfiability problem

杉村憲司 † 石原靖哲 † 藤原融 †
Kenji Sugimura Yasunori Ishihara Toru Fujiwara

1 まえがき

近年，構造化データを記述可能なマークアップ言語として XML(Extensible Markup Language) が盛んに利用されており，XML 文書の特定の要素を指定する問い合わせ言語として XPath が広く用いられている．XPath は，XML 文書を木構造に見立て，その根頂点からの経路を記述することで，XML 文書の特定の要素を指定する問い合わせ言語である．XPath はそれ自体が問合せ言語として用いられるだけでなく，XQuery や XSLT などの実用的な問合せ言語の一部としても利用されている．また，DTD(Document Type Definition) とは XML 文書のデータ構造を定義するスキーマ言語である．

与えられた XPath 式 p に対して DTD D に従うような XML 文書 T が存在するとき，p は D のもとで充足可能であるという．XPath 充足可能性判定 [1] は問合せの最適化に有用であるが，一般には NP 困難であることが知られている．

この XPath 充足可能性の判定方法について，大きく分けて 2 通りの手法が存在している．一つは，文献 [2][3][4] のように高速なソルバを用いて XPath の充足可能性判定を行う手法である．DTD と XPath 式を既存のソルバで判定可能な論理式に変換することによって，充足可能性判定を行う．この手法は，実験により，多くの場合に実用的な時間内で動作することが確認されているが，多項式時間で動作することは保証されていない．一方，DTD と XPath のクラスを制限することで，多項式時間で充足可能性判定を行う手法が提案されている．しかし，これらの手法の実際の時間での実行時間については結果が知られていない．そこで，本稿では，文献 [5][6][7] で我々が提案したいくつかの多項式時間判定アルゴリズムを実装し，その実行時間を計測することにより，これらの多項式時間判定アルゴリズムの有用性を評価する．更に，その設計及び実装したシステムを用いた評価実験を通じ，実装上の改善点を提案・実装し，さらなる処理の効率化・高速化を図ることを目的とする．

本稿では，ベンチマークとして，DTD には XMark[8] を，XPath には XPathMark[9] を用いた．これらは XML のベンチマークとして一般的に用いられているものである．実装したアルゴリズムの実行時間を計測した結果，一般的な PC 上で数十ミリ秒で動作した．このことより，多項式時間で判定可能な XPath 充足可能性判定アルゴリズムの有用性を確認した．

以降，2 節では本稿で実装する充足可能性判定法で使用する XML 文書, DTD, および, XPath について述べる．3 節では，先行研究で提案されたされた多項式時間判定アルゴリズムについて述べる．4 節では，実装したシステムと，その効率化の手法について述べる．5 節では，実装したシステムを用いて多項式時間判定アルゴリズムの評価を行った結果について述べる．また最適化の手法による効果についても述べる．6 節では，まとめと今後の課題について述べる．

2 諸定義

本節では, 本稿で行う XPath 充足可能性判定に用いる XML, DTD，および，XPath の定義について述べる．また，DTD のクラスについても述べる．

2.1 XML 文書

本稿では，XML 文書はラベルつき順序木とみなす．ノード v のラベルを $\lambda(v)$ で表す．関数 λ はノード列に対し拡張可能である．ノード列 $v_1, v_2, ..., v_n$ について $\lambda(v_1, v_2, ..., v_n) = \lambda(v_1)\lambda(v_2)...\lambda(v_n)$ とする．また，ノード v の属性 $@a$ のデータ値は $\rho_{@a}(v)$ で表す．

2.2 DTD

定義 2.1 DTD D は 5 つ組 (Σ, A, r, P, R) で表現する．Σ はラベルの有限集合，A は属性名の有限集合，$r \in \Sigma$ は木の根ノードのラベル，P は Σ から Σ 上の正規表現への写像，R は Σ から 2^A への写像である．P は各ラベルから生成される子ラベル列の集合を表しており，$P(l)$ をラベル l の内容モデル [11] と呼ぶ．また，$R(l)$ はラベル l のついたノードがもつ属性集合を表す．

P における正規表現は，定数として ϵ(空語) と Σ に含まれる記号，演算子として $\cdot$(連接，通常表記では省略)，|(選言)，$*$(0 回以上の繰り返し)，+(1 回以上の繰り返し)，?(高々 1 回の出現) で構成される．

定義 2.2 ラベル付き順序木 T が DTD $D = (\Sigma, A, r, P, R)$ に従うとは，以下の 3 つの条件を満たすことであると定義する．また，D に従う全ての木の集合を $TL(D)$ と表す．

- T の根ノードが r である．
- T の各ノード v とその子ノード $v_1, v_2, ..., v_n$ について，$P(\lambda(v))$ から $\lambda(v_1, v_2, ..., v_n)$ を生成できる．
- T の各ノード v について，$\rho_{@a}(v)$ に値が定義されている場合，かつその場合に限り，$@a \in R(\lambda(v))$ である．

2.2.1 DC-DTD(Disjunction-capsuled DTD)

DC-DTD[5] とは，内容モデルにおいて，全ての | が $*$ によって囲われているような DTD である．

例 2.1 $(a|bc)^*d(ad^*)^*$ は DC-DTD であるが，$a^*|(bc)^*$ は，| 演算子が $*$ 演算子によって囲われていないので，DC-DTD で

† 大阪大学, Osaka University

はない．

2.2.2 DC$^{?+}$-DTD

DC$^{?+}$-DTD[6] とは，以下の処理を行うと，DC-DTD になるような DTD のクラスである．

- 内容モデル中の ? 演算子を消去する．
- 内容モデル中の + 演算子を $*$ 演算子に変換する．

DC-DTD は DC$^{?+}$-DTD の部分クラスである．

例 2.2 $a^?(b|c)^+$ は DC$^{?+}$-DTD である．? 演算子を除去し，+ 演算子を $*$ 演算子に置き換えると，$a(b|c)^*$ となり DC-DTD になるためである．

また，$(a|b)^?c^+$ は DC$^{?+}$-DTD でない．? 演算子を除去し，+ 演算子を $*$ 演算子に置き換えた $(a|b)c^*$ は DC-DTD でないためである．

2.2.3 MDC-DTD

MDC-DTD[7] とは，各内容モデルにおいて，各要素は $*$ 内に囲まれているか，一度しか出現しないような DC-DTD のことである．MDC-DTD は DC-DTD の部分クラスである．

例 2.3 a^*bca^* は MDC-DTD であるが，$abca$ は $*$ に囲まれていない要素 a が二度出現しているため，MDC-DTD でない．

2.3 XPath

定義 2.3 XPath 式の構文は W3C XPath [10] をもとに，以下のように定義する．

$$
\begin{aligned}
&p ::= \chi :: l \mid \chi :: l(\alpha) \mid p/p \mid p \cup p \mid p[q],\\
&\chi ::= \cdot \mid \downarrow \mid \uparrow \mid \downarrow^* \mid \uparrow^* \mid \rightarrow^+ \mid \leftarrow^+,\\
&\alpha ::= @a = n \mid \alpha, \alpha,\\
&q ::= p \mid q \wedge q \mid q \vee q.
\end{aligned}
$$

ただし，$l \in \Sigma, @a \in A, n \in \mathbb{Z}$ とする．

XPath 式の評価において，木 T のノード v がもつ属性 $@a$ の値が n であるということを表すため，ノードから属性名と属性値の組の集合へのマッピング M_c を用いて $M_c(v) = (@a, n)$ と表記する．ここで，M_c を T の属性条件マッピングと呼ぶ．T が M_c を満たすとは，M_c が定義される各ノード v について $\rho_{@a}(v) = n$ ならば $M_c(v) \ni (@a, n)$ となっていることである．なお，$M_c(v)$ が未定義であるとは，$M_c(v) = \emptyset$ ということである．また，M_c が矛盾するとは，定義されているすべての $M_c(v)$ のなかに，v,$@a$ が同一であるが n が異なるような組が存在していることである．

定義 2.4 XPath 式 p を木 T 上の 2 つの根からの経路と属性条件マッピングの組 (w, M_c), (w', M_c') を引数とする述語とみなし，T における p の意味論を T のノード v, v' を用いて以下のように定義する．

- $w' = w$ でかつ $\lambda(w'$の最終ノード$) = l$, かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\cdot :: l)((w, M_c), (w', M_c'))$;
- 経路 wv' が T に存在し $\lambda(v') = l$，かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\downarrow :: l)((w, M_c), (wv', M_c'))$;
- 経路 wv が T に存在し $\lambda(w$ の最終ノード$) = l$, かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\uparrow :: l)((wv, M_c), (w, M_c'))$;
- 経路 ww' が T に存在し $\lambda(w$ の最終ノード$) = l$, かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\downarrow :: *)((ww', M_c), (w, M_c'))$．ただし，$w'$ は空であるかもしれない T のノード列とする;
- 経路 wv, wv' が T に存在し，v' が v に後続する兄弟でかつ $\lambda(v') = l$，かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\rightarrow^+ :: l)((wv, M_c), (wv', M_c'))$;
- 経路 wv, wv' が T に存在し，v' が v に先行する兄弟でかつ $\lambda(v') = l$，かつ $M_c = M_c'$ で矛盾しないとき，$T \models (\leftarrow^+ :: l)((wv, M_c), (wv', M_c'))$;
- $\chi \in \{\cdot \mid \downarrow \mid \uparrow \mid \downarrow^* \mid \uparrow^* \mid \rightarrow^+ \mid \leftarrow^+\}$ について，$\lambda(w'$の最終ノード$) = l$ でかつ $M_c' = M_c \cup \{v' \mapsto \{(@a, n)\}\}$ でかつ M_c, M_c' ともに矛盾しないとき，$T \models (\chi :: l(\alpha))((w, M_c), (w', M_C'))$;
- $\chi \in \{\cdot \mid \downarrow \mid \uparrow \mid \downarrow^* \mid \uparrow^* \mid \rightarrow^+ \mid \leftarrow^+\}$ について，$T \models (\chi :: l(\alpha))((w, M_c), (w', M_c'))$ でかつ $T \models (\chi :: l(\alpha'))((w, M_c), (w', M_c''))$ であるとき，$T \models (\chi :: l(\alpha, \alpha'))((w, M_c), (w', M_c' \cup M_c''))$;
- $T \models p((w, M_c), (w'', M_c''))$ かつ $T \models p'((w'', M_c''), (w', M_c'))$ であるような経路と属性条件マッピングの組 (w'', M_c'') が存在するとき，$T \models (p/p')((w, M_c), (w', M_c'))$;
- $T \models p((w, M_c), (w', M_c))$ または $T \models p'((w, M_c), (w', M_c'))$ であるとき，$T \models (p \cup p')((w, M_c), (w', M_c'))$;
- $T \models p((w, M_c), (w', M_c'))$ かつ $T \models q(w', M_c')$ のとき，$T \models (p[q])((w, M_c), (w', M_c'))$;
- ある経路 w において $T \models p((w, M_c), (w', M_c'))$ であるとき，$T \models p(w, M_c)$;
- $T \models q(w, M_c)$ かつ $T \models q'(w, M_c)$ であるとき，$T \models (q \wedge q')(w, M_c)$;
- $T \models q(w, M_c)$ または $T \models q'(w, M_c)$ であるとき，$T \models (q \vee q')(w, M_c)$.

定義 2.5 v_0 を根頂点とする木 T に対し，$T \models p((v_0, M_{c\perp}), (v, M_c))$ となるような T のノード v と T が満たす属性条件マッピング M_c の組が存在するとき，木 T は XPath 式 p を充足するという．ここで，$M_{c\perp}$ はすべてが未定義のマッピングである．

定義より，$T \models p((v_0, \{v_0 \mapsto \emptyset\}), (v, M_c))$ であれば，M_c は矛盾していない．

定義 2.6 DTD D について，ある木 $T \in TL(D)$ が XPath 式 p を充足するとき，p は D において充足可能という．

3 XPath 充足可能性判定多項式時間アルゴリズム

本節では，文献 [5][6][7] により提案され，実際に実装した多項式時間判定アルゴリズムと，そのアルゴリズムが対応する DTD のクラス，および XPath のクラスについて述べる．提案

されたアルゴリズムに従い，スキーマグラフを導入する．

定義 3.1　DTD $D = (\Sigma, r, P)$ の スキーマグラフ $G_D = (U, E)$ は以下のような有向グラフとして定義する．

- ノード $u \in U$ は，以下のどちらかである．
  - $(\perp, 1, -, r)$ ．ただし，$\perp \notin \Sigma$ である．
  - (a, i, ω, b)．ただし，$a, b \in \Sigma, 1 \leq i \leq len(P(a))$ であり，$P(a)$ の i 番目の部分式 e_i に現れる要素は b である．また，単体の要素であれば，$\omega =$ “$-$”，そうでなければ，$\omega =$ “$*$” とする．

  ノード u の 1 つ目の要素を $\lambda_{par}(u)$，2 つ目の要素を $pos(u)$，3 つ目の要素を $\omega(u)$，そして 4 つ目の要素を $\lambda(u)$ とし，特に，$\lambda(u)$ を u のラベルと呼ぶ．
- E における u から u' の有向辺が存在．$\Leftrightarrow \lambda(u) = \lambda_{par}(u')$.

例 3.1　DC-DTD $D = (\{r, a, b, c\}, r, P)$ を $P(r) = (a|b)^*ca^*, P(a) = \epsilon, P(b) = r^*, P(c) = \epsilon$ とすると，D のスキーマグラフは図 1 のようになる．

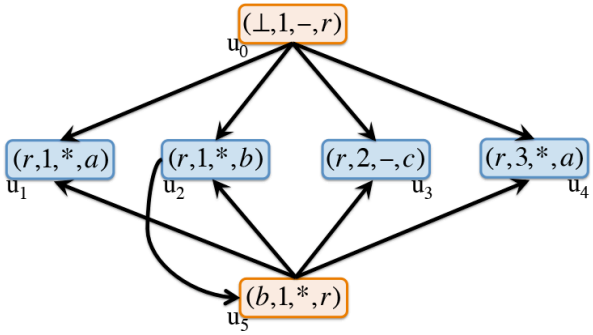

図 1　スキーマグラフの例

3.1　属性値を考慮しない DC-DTD に対する多項式時間アルゴリズム

文献 [5] より，DC-DTD において，スキーマグラフにおける XPath 式の充足性は，DTD における XPath 式の充足可能性に一致し，$p \in \chi(\cdot,\downarrow,\downarrow^*,\rightarrow^+,\leftarrow^+,\cup,[])$ または，$p \in \chi(\cdot,\downarrow,\uparrow,\rightarrow^+,\leftarrow^+)$ を満たす XPath 式 p に対しては，多項式時間で判定可能である．

定義 3.2　上向き軸を含まない XPath 式に対しては，スキーマグラフ G と XPath 式 $p \in \chi(\cdot,\downarrow,\downarrow^*,\rightarrow^+,\leftarrow^+,\cup,[])$ についての充足関係は次のように定義する．

- G においてノード u, u' が存在し，u=u' かつ $\lambda(u') = l$ ならば，$G \models (\cdot :: l)(u, u')$;
- G において u から u' への経路が存在し，かつ $\lambda(u') = l$ ならば，$G \models (\downarrow :: l)(u, u')$;
- G において u から u' へ到達可能で，かつ $\lambda(u') = l$ ならば，$G \models (\downarrow^* :: l)(u, u')$;
- $\lambda_{par}(u) = \lambda(u')_{par}$ かつ $\lambda(u') = l$ であり，「$\omega(u) =$ “$-$” かつ $pos(u) < pos(u')$, または, $\omega(u) =$ “$*$” かつ $pos(u) \leq pos(u')$」ならば，$G \models (\rightarrow^+ :: l)(u, u')$;
- $\lambda_{par}(u) = \lambda(u')_{par}$ かつ $\lambda(u') = l$ であり，「$\omega(u) =$ “$-$” かつ, $pos(u) > pos(u')$, または, $\omega(u) =$ “$*$” かつ $pos(u) \geq pos(u')$」ならば，$G \models (\leftarrow^+ :: l)(u, u')$;
- $G \models p(u, u'')$ かつ，$G \models p'(u'', u')$ であるような u'' が存在するとき，$G \models (p/p')(u, u')$;
- $G \models p(u, u')$ または，$G \models p'(u, u')$ ならば，$(p \cup p')(u, u')$;
- $G \models p(u, u')$ かつ，$G \models q(u')$ ならば，$G \models (p[q])(u, u')$;
- $G \models p(u, u')$ ならば，$p(u)$;
- $G \models q(u)$ かつ，$G \models q'(u)$ ならば，$G \models (q \wedge q')(u)$;
- $G \models q(u)$ または，$G \models q'(u)$ ならば，$G \models (q \vee q')(u)$.

定義 3.3　述語を含まない XPath 式に対しては，スキーマグラフ G と XPath 式 $p \in \chi(\cdot,\downarrow,\uparrow,\rightarrow^+,\leftarrow^+,)$ についての充足関係 $\models$ を，G のノード u, u' と G の $(\perp, 1, -, r)$ から始まる空でないノード列 s, s', s'' を用いて次のように定義する．

- $G \models (\cdot :: l)(su, su')$: 経路 $su = su'$ が G に存在し，$\lambda(u') = l$ である．
- $G \models (\downarrow :: l)(s, su')$: 経路 su' が G に存在し，$\lambda(u') = l$ である．
- $G \models (\uparrow :: l)(su, s)$: 経路 su が G に存在し，$\lambda(s$ の最終ノード$) = l$ である．
- $G \models (\rightarrow^+ :: l)(su, su')$: 経路 su, su' が G に存在し，$\lambda(u') = l$ であり，$\omega(u) =$“$-$” であれば $pos(u) < pos(u')$ となっており，$\omega(u) =$“$*$” であれば，$pos(u) \leq pos(u')$ となっている．
- $G \models (\leftarrow^+ :: l)(su, su')$: 経路 su, su' が G に存在し，$\lambda(u') = l$ であり，$\omega(u') =$“$-$” であれば $pos(u') < pos(u)$ となっており，$\omega(u') =$“$*$” であれば，$pos(u') \leq pos(u)$ となっている．
- $G \models p(s, s'')$ かつ，$G \models p'(s'', s')$ であるような s'' が存在するとき，$G \models (p/p')(s, s')$

3.1.1　上向き軸を含まない XPath 式に対する多項式時間アルゴリズム

DC-DTD のもとで XPath 式 $p \in \chi(\cdot,\downarrow,\downarrow^*,\rightarrow^+,\leftarrow^+,\cup,[])$ に対する多項式時間アルゴリズムを述べる．この場合，文献 [5] より，XPath 式 p からスキーマグラフのノードの組を列挙し，ボトムアップに解析していくことで多項式時間で充足可能性を判定ができる．

例 3.2　DC-DTD $D = (\{r, a, b, c\}, r, P)$ を $P(r) = (a|b)^*ca^*, P(a) = \epsilon, P(b) = r^*, P(c) = \epsilon$ とし，XPath 式 $p = (\downarrow :: b/\ \downarrow^* :: r)/(\downarrow :: b[\downarrow^* :: r]/\ \rightarrow^+ :: a)$ についての例を考える．

まず D についてのスキーマグラフを図 1 のように構築する．次に XPath 式 p を原子式ごとに分割し，原子式を満たすスキーマグラフのノードの組を表 1 のように列挙する．この表から $\downarrow :: b/\ \downarrow^* :: r$ をみたすノードの組を求める．$\downarrow :: b$ を満たすノードの組の後ろの要素が $\downarrow^* :: r$ を満たすノードの組の前の要素に一致するものを全て列挙する．今，$\downarrow :: b$ を満たすノードの組の後ろの要素は全て u_2 である．$\downarrow^* :: r$ を満たすノードの組のうち，前の要素が u_2 から始まるのは (u_2, u_5) のみである．よって，$\downarrow :: b/\ \downarrow^* :: r$ を満たすノードの組は,$(u_0, u_5), (u_5, u_5)$ となる．以下，同様にボトムアップに繰り返すと，XPath 式 p を

満たすノードの組 (u_0, u_4) が存在し，充足可能であると判定できる．

表 1　各原子式を充足するノードの組

原子式	原子式を満たすノードの組
$\downarrow :: b$	$(u_0,u_2),(u_5,u_2)$
$\downarrow^* :: r$	$(u_0,u_0),(u_5,u_5),(u_2,u_5),(u_5,u_5)$
$\rightarrow^+ :: a$	$(u_1,u_4),(u_2,u_4),(u_3,u_4),(u_4,u_4),(u_1,u_1)$

3.1.2 述語を含まない XPath 式に対する多項式時間アルゴリズム

DC-DTD のもとで XPath 式 $p \in \chi(\cdot,\downarrow,\uparrow,\rightarrow^+,\leftarrow^+)$ に対する多項式時間判定アルゴリズムを述べる．スキーマグラフ $G=(U,E)$ における空でない経路を s とする．XPath 式 p をトップダウンに解析することで，$G \models_{DC} p((\perp,1,-,r),s)$ を満たす s が存在するかどうかを判定，つまり，充足可能性を判定することができる．アルゴリズム $eval_0$ の仕様は以下のようになる．

$$eval_0(p,s)=\begin{cases}\{s' \mid \text{各 } s\in S \text{ について} \\ \quad G \models_{DC} p(s,s')\}(p \text{ が原子式}) \\ eval_0(p_2, eval_0(p_1,S))(p=p_1/p_2 \text{のとき})\end{cases}$$

G のノード集合 U を考えると，経路 $s=u_0u_1\cdots u_m$ の集合をノード集合の列 $U_0U_1\cdots U_m$ で表すことが可能である．ここで $U_0=\{(\perp,1,-,r)\}$ であり，$1\le i\le m$ について $u_i\in U_i$ である．以上を踏まえて，s の集合を $U_0\cdots U_m$ と表記すると，アルゴリズムの詳細仕様は以下のようになる ($\leftarrow^+$ は $\rightarrow^+$ と同様なので省略する)．

- $p=\cdot :: l$ のとき：$U_0\cdots U_m$ を返す．
- $p=\downarrow :: l$ のとき：$U_0\cdots U_mU_{m+1}$ を返す．ただし，U_{m+1} は U_m 中のあるノードの子ノードの集合である．
- $p=\uparrow :: l$ のとき：$U_0\cdots U_{m-1}$ を返す．
- $p=\rightarrow^+ :: l$ のとき：$U_0\cdots U_{m-1}U'_m$ を返す．ただし，U'_m は U_m 中のあるノードの右にあるノードの集合である．
- $p=p_1/p_2$ のとき：$eval_0(p_2, eval_0(p_1, U_0\cdots U_m))$ を返す．

3.2 属性値を考慮しない DC$^{?+}$-DTD に対する多項式時間アルゴリズム

DC$^{?+}$-DTD における XPath 充足可能性は，DC$^{?+}$-DTD を以下の処理によって変換した DC-DTD のスキーマグラフの充足性に一致することが文献 [6] によって示されている．よって，変換した DC-DTD について 3.1 節で述べたアルゴリズムを用いて処理することで，DC$^{?+}$-DTD の XPath 充足可能性判定を行うことができる．

- 内容モデル中の ? 演算子を消去する．
- 内容モデル中の + 演算子を $*$ 演算子に変換する．

3.3 属性値を考慮した場合の MDC-DTD に対する多項式時間アルゴリズム

このアルゴリズムでは属性値を考慮するため，以下のものを定義する．

定義 3.4　$G=(U,E)$ 上の $(\perp,1,-,r)$ を始点とする空でない経路 s から，(属性名 α，値 d) の組の集合 Att への部分マッピング δ を属性値マッピングと呼び，以下のように定義する．

s において δ が定義されるとき，s は属性値を特定できるノードを示す経路，(α,d) の集合はその特定できる属性値の集合を表す．

任意の s について $\delta(s)\supseteq\delta'(s)$ または $\delta'(s)$ が未定義であるならば $\delta\sqsupseteq\delta'$ と表記し，δ と δ' の $\sqsupseteq$ に関する最小上界を $\delta\sqcup\delta'$ と表記する．

$\Sigma=\{r,b,c\}, P=\{r\to b^*, b\to c\}$ の場合，b の部分木中のノードを見ている場合に限り b,c の属性値は特定可能だが，r ノードを見ているときには b が何度も現れるため，b,c を特定することはできない．つまり，XPath 式で b から r へと通過する場合，b,c で定義されている δ を未定義にする必要がある．よって以下のような表記を導入する．

$$\delta|_{Single,s}(s')=\begin{cases}\delta(s') : s' \text{が } s \text{ の接頭語に 0 個以上の単体} \\ \quad \text{ノードをつなげた経路である場合．} \\ \text{未定義 : そうでない場合．}\end{cases}$$

文献 [7] より，MDC-DTD のもとで XPath 式 $p\in\chi(\cdot,\downarrow,\uparrow,\rightarrow^+,\leftarrow^+,=)$ であれば，トップダウンで解析していくことで多項式時間で $G\models_{MDC}(p)(((\perp,1,-,r),\delta_\perp),(s',\delta'))$ となる (s',δ') があるかどうかを判定，つまり充足可能性を判定することができる．B を G の経路と属性値マッピングの組 (s,δ) の集合とすると，アルゴリズム $eval_1$ の仕様は以下のようになる

$$eval_1(p,B)=\begin{cases}\{(s',\delta') \mid \text{各 } (s,\delta)\in B \text{ について} \\ \quad G\models_{MDC} p((s,\delta),(s',\delta'))\}(p \text{ が原子式}) \\ eval_1(p_2, eval_1(p_1,B))(p=p_1/p_2 \text{のとき})\end{cases}$$

一方，G のノードの集合 U を考えると，経路 $s=u_0u_1\cdots u_m$ の集合を，ノード集合の列 $U_0U_1\cdots U_m$ で表すことが可能である．ここで $U_0=\{(\perp,1,-,r)\}$ であり，$1\le i\le m$ について $u_i\in U_i$ である．以上を踏まえ，s の集合を $U_0\cdots U_m$ と表記すると，アルゴリズムの詳細仕様は以下のようになる ($\leftarrow^+$ は $\rightarrow^+$ と同様なので省略する)．

- $p=\cdot :: l$ のとき:$(U_0\cdots U_m, \delta/_\lambda \sqcup \{s\mapsto\emptyset\}/_\lambda)$ を返す．
- $p=\downarrow :: l$ のとき:$(U_0\cdots U_mU_{m+1}, \delta/_\lambda \sqcup \{su_{m+1}\mapsto\emptyset\}/_\lambda)$ を返す．
- $p=\uparrow :: l$ のとき:$(U_0\cdots U_{m-1}, \delta/_\lambda|_{Single,\lambda(s)})$ を返す．
- $p=\rightarrow^+ :: l$ のとき:$(U_0\cdots U_{m-1}U'_m, \delta/_\lambda|_{Single,\lambda(s)} \sqcup \{su'\mapsto\emptyset\}/_\lambda)$ を返す．
- $p=\cdot :: l(@a=n)$ のとき:$(U_0\cdots U_m, \delta/_\lambda \sqcup \{s\mapsto\{(@a,n)\}\}/_\lambda)$ を返す．
- $p=\downarrow :: l(@a=n)$ のとき:$(U_0\cdots U_mU_{m+1}, \delta/_\lambda \sqcup \{su_{m+1}\mapsto\{(@a,n)\}\}/_\lambda)$ を返す．
- $p=\uparrow :: l(@a=n)$ のとき:$(U_0\cdots U_{m-1}, \delta/_\lambda|_{Single,\lambda(s_{m-1})} \sqcup \{s_{m-1}\mapsto\{(@a,n)\}\}/_\lambda)$ を返す．
- $p=\rightarrow^+ :: l(@a=n)$ のとき:$(U_0\cdots U_{m-1}U'_m, \delta/_\lambda|_{Single,\lambda(s)} \sqcup \{su'\mapsto\{(@a,n)\}\}/_\lambda)$ を返す．
- $p=p_1/p_2$ のとき: $eval_1(p_2, eval_1(p_1,(U_0\cdots U_m,\delta/_\lambda)))$ を返す．

4 実装

4.1 実装目的

文献 [5][6][7] において提案されている多項式時間判定アルゴリズムは実時間の動作について検証されておらず，その実用性が確認されていない．そこで，提案された多項式時間判定アルゴリズムの実装，その設計及び実装したシステムを用いた評価実験を通じ，実装上の改善点を提案・実装し，さらなる処理の効率化・高速化を図ることを目的とする．

4.2 開発環境

表 2 のような開発環境にて開発を行った．

表 2　開発環境

言語	Python 2.7.5
使用ライブラリ	matplotlib 1.3.1
	networkx 1.7
OS	Darwin Kernel Version 13.0.0
バージョン管理	git

4.3 主要モデルのデータ構造

4.3.1 スキーマグラフ

スキーマグラフの構成要素はノードと有向辺である．

ノード

- ノード $u \in U$ は，以下のどちらかである．
  - $(\perp, 1, -, r)$，ただし，$\perp \notin \Sigma$ または，
  - (a, i, ω, b)，ただし，$P(a)$ の i 番目の部分式 e_i に現れる b であり，かつ，もし Σ 上の単体の要素であれば，$\omega =$ “$-$” そうでなければ，$\omega =$ “$*$” であるような $a, b \in \Sigma, 1 \leq i \leq len(P(a))$ である．

  ノード u の 1 つ目の要素を $\lambda_{par}(u)$，2 つ目の要素を $pos(u)$，3 つ目の要素を $\omega(u)$，そして 4 つ目の要素を $\lambda(u)$ とし，特に，$\lambda(u)$ を u のラベルと呼ぶ．

有向辺 E における u から u' の有向辺が存在する．$\Leftrightarrow \lambda(u) = \lambda_{par}(u')$ である．

4.3.2 原子式

与えられた XPath 式は原子式に分割される．原子式は要素として，軸方向，名前空間，属性，述語をもつ．

軸方向原子式の軸の記述は XML 文書において，方向を指定する．本稿で扱った軸方向を記す．

- $\cdot$：コンテクストノード自身．
- $\downarrow$：コンテクストノードの子ノード．
- $\downarrow^*$：コンテクストノードとその子孫ノード．
- $\uparrow$：コンテクストノードの親ノード．
- $\rightarrow^+$：コンテクストノードとその兄弟ノードのうち後方のノード．
- $\leftarrow^+$：コンテクストノードとその兄弟ノードのうち前方のノード．

名前空間名前空間は Σ の要素であり，ノードのラベルである．

属性木 T のノード v が属性名 $@a$ が属性値 n であることを表すため，ノードから属性名と属性値の組の集合へのマッピング M_c を用いて $M_c(v) = \{(@a, n)\}$ とする．

述語軸方向と名前空間によって絞り込んだノード集合を更に述語を用いることで絞り込むことができる．本稿では述語に XPath 式，$\wedge$ と $\vee$ 演算子を用いることができる．

4.4 システムの入出力

4.4.1 入力

システムへの入力は XPath 式 p と DTD D の 2 種類である．入力はテキスト形式で，拡張子 xpl で与えられる XPath 式のデータと，拡張子 dtd で与えられる DTD データを与える．

例 4.1　プログラムの実行は次のようになる．

```
python xmlsat.py dtd_file xpath_file
```

XPath **式**与えられた XPath 式は，軸方向 :: ラベル (@属性名 = 属性値) または，軸方向 :: ラベル [述語] の形の原子式を / 演算子でつないだものを入力で受け付ける．このとき，軸方向は表 3 のように記述する．また，全てのラベルを指定するワイルドカードは $*$ と表記する．XPath の省略構文も入力可能である．省略構文と完全な構文の対応は表 4 のようになっている．

表 3　システムにおける軸方向の表記法

軸方向	表記法
.	self
$\uparrow$	parent
$\downarrow$	child
$\rightarrow^+$	following-sibling
$\leftarrow^+$	preceding-sibling
$\downarrow^*$	descendant-or-self

表 4　省略構文と完全な構文の対応関係

完全な構文	省略構文
child::	(省略して何も書かない)
/descendant-or-self::node()/	//
self::node()	.
parent::node()	..

DTD 入力として受け付ける DTD は ENTITY を含まないもので，テキスト形式のものである．このとき，DC-DTD, $DC^{?+}$-DTD，MDC-DTD のいずれかのクラスに分類されるような DTD でなければ正しくアルゴリズムを実行することはできない．

4.4.2 出力

与えられた入力から XPath 充足可能性判定を行い，充足可能か否かの判定結果を返す．この時，述語を含まない上向き軸を含まない DC-DTD または $DC^{?+}$-DTD に対しては，各原子式が充足するスキーマグラフのノードのタプルを出力し，述語を含まない上向き軸を含む DC-DTD または $DC^{?+}$-DTD に対してはトップダウンに解析し，その解析経過を出力する．属性

値を含む XPath と MDC-DTD に対しては，述語を含まない上向き軸を含む DC-DTD または DC$^{?+}$-DTD と同様にトップダウンな解析経過とともに，3.3 節で記したアルゴリズム $eval_1$ の経過も出力する．

4.5 アルゴリズムの効率化の手法

上向き軸を含まない DC-DTD 及び DC$^{?+}$-DTD に対して次の 2 つの効率化を実装した．

4.5.1 逐次処理の並列化

上向き軸を含まない XPath 式と DC-DTD 及び DC$^{?+}$-DTD に対する多項式時間アルゴリズムでは，XPath を原子式に分割し，その原子式が充足するスキーマグラフのノードの組を求めている．この処理は独立的に行われているので，プロセスベースで並列的に処理することによってアルゴリズムの効率化を図った．まず，システムが実行されている PC の CPU のコア数に合わせてプロセスを生成する．そして生成された各プロセスに対して，XPath 式を分割して得られた原子式とスキーマグラフを引数に与え，これらを充足するようなノードのタプルの集合を得る．分割した全ての原子式を処理し終えるまで，プロセスが処理を完了するたびに，原子式とスキーマグラフを与え，並列的に処理を行うように改良した．擬似コードで示すと次のようになる．

上向き軸を含まない XPath 式と DC-DTD 及び DC$^{?+}$-DTD 逐次処理

```
1  sg;            // スキーマグラフ
2  XPath;         // XPath 式
3  tuple_array;   // 原子式が充足するノードの配列
4
5  atomic_expression_array = split(XPath);
6  for (atomic_exp in XPath) {
7      tuple_array.append(function(sg,atomic_exp));
8  }
```

上向き軸を含まない XPath 式と DC-DTD 及び DC$^{?+}$-DTD の並列処理

```
1  sg;            // スキーマグラフ
2  XPath;         // XPath 式
3  tuple_array;   // 原子式が充足するノードの配列
4
5  process = Pool(); //プロセスの生成
6  atomic_expression_array = split(XPath);
7  parm_array;    // プロセスに与える引数
8  for (atomic_exp in atomic_expression_array){
9      parm_array.append((sg, atomic_exp));
10 }
11 tuple.array = process.map(parm_array);
```

4.5.2 探索結果のキャッシュによるスキーマグラフの探索の効率化

上向き軸を含まない XPath 式と DC-DTD 及び DC$^{?+}$-DTD に対する多項式時間アルゴリズムでは, /descendant-or-self::∗/ に対するスキーマグラフの探索を行った場合, スキーマグラフのノード数を n とすると，全てのスキーマグラフのノードに対し，その子孫ノードまで再帰的に探索するので，$\mathcal{O}(n^2)$ の実行時間を要する．よって，初めて子孫軸に対して探索したとき，その探索結果をメモリ上にキャッシュしておき，2 回目以降，子孫軸についての探索を行うときは，メモリ上の探索結果を参照することで，高速に処理を行えるように改善した．

5 評価

多項式時間判定アルゴリズムを実装したプログラムと，それに対して実装した効率化の手法についてそれぞれ実行時間を計測し，評価を行った．評価は表 5 のような実験環境で行った．

ベンチマークとして，DTD には XMark[8]，XPath には XPathMark[9] を用いた．これらは XML のベンチマークとして一般的に用いられているものである．また，XPathMark だけでは，実装したアルゴリズムを全て実行することはできなかったので，新たに実行する XPath 式を追加した．実行結果は，5 回実行した実行時間の平均の値をとった．

表 5　実行環境

CPU	2.3 GHz Intel Core i7
OS	Darwin Kernel Version 13.0.0
RAM	8GB
言語	Python 2.7.5

5.1 実行結果

5.1.1 上向き軸を含まない XPath 式と DC$^{?+}$-DTD に対する多項式時間アルゴリズムの実行結果

実行結果は，表 6 のようになった．実行した XPath 式は次のとおりである．表 6 の結果から，およそ 20 ミリ秒前後で判定可能であることが分かった．また，XPath 式 A2, A3 のみが 20 ミリ秒以上かかっているが，このことから子孫軸を含む XPath 式の処理に時間を要することが分かる．

- A1：/site/closed_auctions/closed_auction/annotation/description/text/keyword
- A2：//closed_auction//keyword
- A3：/site/closed_auctions/closed_auction//keyword
- A4：/site/closed_auctions/closed_auction[annotation/description/text/keyword]/date
- A5：/site/closed_auctions/closed_auction[descendant::keyword]/date
- A6：/site/people/person[profile/gender and profile/age]/name
- A7：/site/people/person[phone or homepage]/name
- A8：/site/people/person[address and (phone or homepage) and (creditcard or profile)]/name

5.1.2 述語を含まない XPath 式と DC$^{?+}$-DTD に対する多項式時間アルゴリズムの実行結果

上向き軸を含むような適切な問合せが XPathMark には存在しなかったため，XPathMark に含まれる問合せに上向き軸を追加し，以下のような XPath 式を用意した．実行結果は，表 7

表 6　上向き軸を含まない $DC^{?+}$-DTD についての実行結果

XPath 式	実行時間 [ms]
A1	16.278
A2	25.959
A3	22.681
A4	15.876
A5	18.961
A6	16.290
A7	15.734
A8	16.450

のようになった．表 7 の結果から，十数ミリ秒で実行可能であることが読み取れる．

- B1：/site/regions/*/item/parent::samerica//name
- B2：//keyword/parent::*/parent::listitem/text /keyword
- B3：/site/open_auctions/open_auction/bidder/..// bidder
- B4：/site/open_auctions/open_auction/bidder/../../../ closed_auctions/closed_auction

表 7　述語を含まない XPath 式と $DC^{?+}$-DTD についての実行結果

XPath 式	実行時間 [ms]
B1	16.092
B2	18.110
B3	16.726
B4	16.310

5.1.3　属性値を考慮した MDC-DTD についての実行結果

属性値を考慮した MDC-DTD についての実行結果は，表 8 のようになった．DTD のベンチマークとして用いた XMark は，? 演算子や + 演算子を含んでいるため，MDC-DTD でない．そのため，XMark の内容モデル中の，? 演算子を除去し，+ 演算子を ∗ 演算子に置換した DTD をベンチマークとして用いた．また実行した XPath 式は，適切な問合せが XPathMark には存在しなかったため，以下のような XPath 式を用意した．

- C1：/site/people/person/child::profile(@income=100)/../child::profile(@income=100)
- C2：/site/closed_auctions/closed_auction/seller (@person=12345)
- C3：/site/people/person/watches/child::watch (@open_auction = 'auction01')
- C4：/site/people/person/watches/watch (@open_auction='auction94')/../../../../open_auctions /open_auction/itemref(@item='bag')

表 8　属性値を含む MDC-DTD についての実行結果

XPath 式	実行時間 [ms]
C1	16.092
C2	16.487
C3	16.952
C4	17.869

5.1.4　実装したアルゴリズムについての評価

5.1.1 節，5.1.2 節，5.1.3 節において，実装したアルゴリズムの実行時間を計測した．その結果，一般的な PC でも数十ミリ秒で実行可能であることが判明した．従って，これらの XPath 充足可能性判定多項式時間アルゴリズムは，実用性があると考えられる．

5.2　アルゴリズムの効率化による効果

4.5 節で述べたとおり，本システムは二通りのアルゴリズムの効率化を行なった．その工夫による効果を評価する．

5.2.1　逐次処理の並列化

上向き軸を含まない $DC^{?+}$-DTD に対するアルゴリズムにおけるスキーマグラフの探索処理は，入力として与えた XPath 式を分割した原子式について独立しているため，並列化による高速化が期待できる．しかし，並列化の有無による実行時間は表 9 のようになり，並列化を行なった方が遅くなるという実験結果がでた．その理由はベンチマークとして与えた DTD が比較的小さく，スキーマグラフの探索処理に要する時間に比べ，並列化のためにプロセスを生成するオーバーヘッドの方が大きためである．生成されるプロセスの個数は常に一定であり，入力に依存しないため，プロセスの生成に要するオーバーヘッドは常に等しい．また，プロセスの生成に要する時間を求めた結果，表 10 のようになり，この表からもプロセスの生成には常に 16[ms] 程度要していて，入力に依存していないことが分かる．したがって，スキーマグラフの探索処理に時間を要するようなデータサイズの大きい DTD では，並列化の効果が期待できると考えられる．

表 9　並列化の有無による実行時間の比較

XPath 式	並列化あり [ms]	並列化なし [ms]
A1	39.864	16.278
A2	40.713	25.959
A3	38.751	22.681
A4	36.766	15.876
A5	36.499	18.961

5.2.2　データの再利用による処理の効率化

データの再利用による処理の効率化の前後では，実行時間は表 11 のようになり，データの再利用を行なったほうが高速化ができることが判明した．また，再利用できるデータが多いほど，高速化が顕著になることが読み取れる．データの再利用を検証するために次のような XPath 式を用いた．

- A2：//closed auction//keyword

表 10　プロセスの生成に要する時間

XPath 式	実行時間 [ms]
A1	16.278
A2	15.055
A3	16.518
A4	15.876
A5	15.659

- A3：/site/closed auctions/closed auction//keyword
- D1：/site//regions//samerica//item//description//parlist//listitem

表 11　データの再利用の有無による実行時間の比較

XPath 式	データの再利用あり [ms]	データの再利用なし [ms]
A2	25.646	32.012
A3	22.131	22.547
D1	19.742	73.132

6　あとがき

本稿では文献 [5][6][7] で提案された多項式時間判定アルゴリズムを実装した．実装の結果，一般的に用いられる XML のベンチマークに対して, 一般的な PC 上で数十ミリ秒で動作したことにより，提案された多項式時間判定アルゴリズムの実用性を確認した．また，実装後に，さらなる効率性改善の方法について提案し，システムの再実装を行った．その結果，処理の高速化を達成することができた．

今後は，更に広い DTD のクラスである MRW-DTD や RW-DTD[12][13] に対して属性値を考慮した XPath 式の充足可能性判定を行うことができるような多項式時間判定アルゴリズムを提案・実装し，実世界における DTD と XPath 式にさらに対応することを考えている．更に，XML スキーママッピングの際に得られる情報を用いて, ターゲットスキーマに従うデータに対する XPath クエリについて, 問合せ最適化を行うという拡張も予定している．

参考文献

[1] M. Benedikt, W. Fan, and F. Geerts. “XPath satisfiability in the presence of DTDs.” Journal of the ACM, 55(2) (2008).

[2] P. Genevès and N. Layaida. “A system for the static analysis of XPath.” ACM Transactions on Information Systems, 24(4), pp. 475-502 (2006).

[3] P. Genevès and N. Layaida. “Deciding XPath containment with MSO.” Data & Knowledge Engineering, 63(1), pp. 108-136 (2007).

[4] P. Genevès, N. Layaida and A. Schmitt. “Efficient static analysis of XML paths and types.” In: Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, pp. 342-351 (2007).

[5] Y. Ishihara, T. Morimoto, S. Shimizu, K. Hashimoto, T. Fujiwara. “A tractable subclass of DTDs for XPath satisfiability with sibling axes.” In: Gardner, P., Geerts, F. (eds.) Database Programming Languages, LNCS, vol. 5708, pp. 68-83 (2009).

[6] Y. Ishihara, S. Shimizu, and T. Fujiwara. “Extending the tractability results on XPath satisfiability with sibling axes.” In Proceedings of the 7th International XML Database Symposium, pp. 33-47 (2010).

[7] 桑田 逸人. “実用的な DTD クラスにおける XML スキーママッピングの整合性および絶対整合性判定問題.” 大阪大学大学院情報科学研究科修士学位論文 (2014).

[8] M. Franceschet. “Document Type Definition.”`http://users.dimi.uniud.it/~massimo.franceschet/caffe-xml/dtd/dtd-xmark.html`.

[9] M. Franceschet. “XPathMark.” `http://sole.dimi.uniud.it/~massimo.franceschet/xpathmark/`.

[10] W3C. “XML Path Language (XPath).” `http://www.w3.org/TR/xpath/`.

[11] W3C. “Extensible Markup Language (XML) 1.1 (Second Edition)” `http://www.w3.org/TR/xml11/`

[12] Y. Ishihara, K. Hashimoto, S. Shimizu, and T. Fujiwara. “XPath satisfiability with downward and sibling axes is tractable under most of real-world DTDs.” In The 12th International Workshop on Web Information and Data Management, pp. 11-18 (2012).

[13] Y. Ishihara, N. Suzuki, K. Hashimoto, S. Shimizu, and T. Fujiwara. “XPath satisfiability with parent axes or qualifiers is tractable under many of real-world DTDs.” In Proceedings of the 14th International Symposium on Database Programming Languages `http://arxiv.org/abs/1308.0769` (2013).