命題μ計算を用いたXML変換言語のための型検証アルゴリズム

戸沢 晶彦[*]

日本 IBM 東京基礎研究所

要旨

XSLT は XML の応用分野で広く使われている XML 変換言語であり, 入力の XML 文書を走査しながら, その内容をもとに出力の文書を構築するという動作をする. 本稿ではまず XSLT の「型検証アルゴリズム」すなわち「ある型の XML 文書が別のある型の文書に変換されるのか」という問題の判定アルゴリズムを考案し, 次にとり扱いの難しい XSLT の入力文書の木の上方向への走査を考慮した場合の「型構築アルゴリズム」すなわち「正しい変換が行われるための入力文書の型」を推論するアルゴリズムを与える. 2 つめの成果のために本稿では命題 μ 計算による XML の型のモデルを新しく提案する.

1　はじめに

XML (eXtensible Markup Language) およびそれに関連する

- XSLT – XML の変換のための言語, ブラウザ表示用のスタイルシートとしてもよく使われる.
- XPath – XML 文書の内部を検索するために用いられる式. XSLT から利用される.
- DTD, スキーマ – XML 文書の型、たとえば XHTML のような XML 文書のひとつの形式を定義するもの.

のような技術が普及し重要性をましてきた. ただし現状ではこれらの技術はあまり理論的裏付けをもたないアドホックなものであるといえる. そこで, 形式的なアプローチでこれらの技術にきれいな体系を与えられないか, そのようなアプローチを通したよい成果を得られないかということを目的とし, XML の形式的なモデルやそれに基づいた解析, 最適化などの研究が活発になってきた.

モデルの面からこれらの研究を概観してみる. 木オートマトン (TA) およびそれが受理する言語クラスである正規木言語が DTD, スキーマを形式化したものとしてよく研究されている. 特に XML 文書は枝の数の決まっていない木であるが, これを二分木とみなし, これに対して二分木オートマトンを用いることによって, 簡易な型のモデルが作れることも知られている. 正規木言語と同等の表現力を持つ二分木上での単項二階論理 (MSO) などもよく言及されており, MSO を XPath のような検索言語のモデルとして使うこともできる.

一方, これらのモデルを用いた研究トピックとして XML の検索, 変換, 処理言語の静的解析は興味深い. 本稿ではこのような解析の対象として XSLT とその型検証の問題, つまり「ある型の XML 文書が正しい型の文書に変換されるのか」という問題を扱う. XSLT は入力の XML 文書を走査しながら, その内容をもとに出力の文書を構築するという動作をする言語である. XSLT に対する型検証には入力文書が常にブラウザで出力できる XHTML の文書に変換されることの静的な検査, そしてその検査を利用した XSLT スタイルシート作成者のためのデバッグ環境の提供といった実用性がある.

著者は既発表論文 [6] において, XSLT の出力文書の型からその入力文書の型を推論する「型構築アルゴリズム」を考案した. 本稿は, この論文の結果を補おうとするものである. 改善されるべき点は 2 点あって, ひとつにはこの「型構築アルゴリズム」はオートマトンを用いていたために, 直観的にわかりやすいものではなかった. また既発表論文では XSLT の動作のうち入力木を上方向へ移動する動作は扱っていなかった. そこで本研究では (1) 入力木を下方向にしか移動しない場合の「型検証」アルゴリズムと (2) 入力木を上方向にも移動する場合の「型構築」アルゴリズムをより理解しやすい記述で与える.

後者 (2) のアルゴリズムにおいて入力木を上方向に移動する動作は XSLT や XPath で特徴的なものであって, しかも既存のモデルでは扱いが困難であったものである. 本稿ではこれを解決するため新しく様相論理とくに命題 μ 計算 (propositional μ-calculus) を用い, とくにこれに入力木を上方向に移動する動作を表す様相 (時相論理における過去) を加える.

様相論理は MSO と同様たかだかオートマトンと同等の表現力をしか持たないものであり, XML の型のモ

[*]atozawa@trl.ibm.co.jp

デルとしては都合がよいものである. また様相論理は記述の簡潔さ (succinctness) において MSO におとるが, その分オートマトンへの変換などの計算は楽になる. 本稿では XML 文書のモデルとマッチした二分木上での命題 μ 計算を提案する. この命題 μ 計算式は (2) の場合の入力文書の「型」として用いられる.

2 関連研究

XSLT [1] とは XML 技術の応用分野において最も一般的に使われる XML 変換言語であり, 入力 XML 文書を走査しながら得られた情報にもとづき出力 XML を構築するという動作をする.

XSLT の型検証に関しては Milo ら [3], Rose ら [5] と著者 [6] の試みがある. Milo らは XSLT というよりより原始的な k-pebble 変換器というオートマトンベースの木変換器について, その型検証の可能性を示した. 彼らは入力文書の型を出力文書の型から推論する. 彼らはこの推論された入力型は最終的には MSO の論理式で表現できることをしめし, その上で知られている MSO と TA との変換の可能性から型検証の可能性を示した. ただし一般に MSO から TA の変換は非初等的 (non-elementary) と呼ばれる非常に高い計算量を要する.

Rose らは TA で表現される正規木言語ではなく, その部分クラスである局所木言語を用いて XSLT の型検証を試みたものである. 局所木言語は XML で一般に文書の型として使われてきた DTD に対応する言語クラスである.

本稿ではもうひとつ様相論理を取り扱う. 様相論理はプログラム検証の分野でよく研究されてきた論理である. たとえば Vardi [7] が過去を持つ命題 μ-計算に関する充足可能性判定を議論したのもこの文脈におけるものである. この Vardi の結果は過去を持つ命題 μ-計算の TA への変換および, その充足可能性判定がたかだか指数的であることを示すものであり, これは, 様相論理を XSLT の型推論の問題で用いる手法の MSO を使った手法に対するひとつの利点である. この Vardi の研究は例外として過去を持つような様相論理は今まであまり研究されてきたとは言いがたい. 一方 XSLT プログラムは入力木を上下左右に移動しながら動作するため, これを取り扱う様相論理は過去を持つようなものでなくてはならない. また MSO と違い, 有限の二分木で動作するような様相論理は過去あまり研究されてこなかった.

3 XML と二分木

本稿では XML 文書を二分木構造とよぶもので表現する. 例えば以下の文書を例にとる.

```
<table>
  <one/>
  <two/>
  <three/>
</table>
```

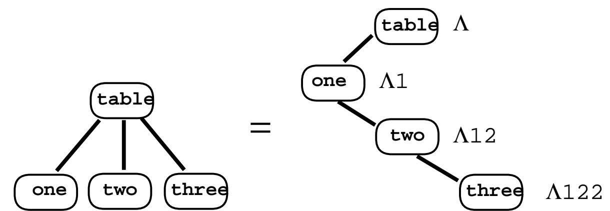


図 1: XML 木と二分木

この XML 文書は 図 1 の右側に示すような二分木で表現される. ただし, 本稿では XML 文書を単純な二分木としては定義せず, これを以下に形式的に定義されるような二分木構造 M (様相論理における Kripke structure) で表現する.

L をラベルの集合, V をノードの集合とする. ノード v は $\{1,2\}$ 上の文字列であり, 空文字 Λ (ルート) と後続演算子 $\cdot 1$ と $\cdot 2$ によって作られるものとする. 今 V は木を表現するので $v\cdot 1 \in V \vee v\cdot 2 \in V \Rightarrow v \in V$ が成立するものとする. $\models$ はノードとラベルの間の充足関係を示す. ただしラベル $\{\mathbf{\Lambda}, -\mathbf{1}, -\mathbf{2}, \mathbf{1}, \mathbf{2}, T, F\}$ については, $\models$ の意味は $\Lambda \models \mathbf{\Lambda}$, $v\cdot 1 \models -\mathbf{1}$, $v\cdot 2 \models -\mathbf{2}$, $v \models \mathbf{1}$ ($v\cdot 1 \in V$ の場合), $v \models \mathbf{2}$ ($v\cdot 2 \in V$ の場合), $v \models T$, $v \not\models F$ のように当初から与えられているものとする. 以下ではラベルの集合 L は固定されているものとする. ノード V と充足関係 $\models$ の組 $M = \langle V, \models \rangle$ を二分木構造と呼ぶ.

文書は次のように翻訳される (任意のラベルは a で表す).

- <a> ... </a> の間の内容つまり列 ... (列に複数の要素がある場合はその先頭の要素) は $\cdot 1$ による後続ノードに対応する.
- </a> の次に来る要素は $\cdot 2$ による後続ノードに対応する.
- 要素 <a>$\cdots$</a> に対応するノードは a を満たす.

よって冒頭の文書は以下の二分木構造で表現される

$$\left\langle \begin{array}{l} \{\Lambda, \Lambda 1, \Lambda 12, \Lambda 122\}, \\ \{\Lambda \models \texttt{table},\ \Lambda 1 \models \texttt{one},\ \Lambda 12 \models \texttt{two},\ \Lambda 122 \models \texttt{three}\} \end{array} \right\rangle$$

XML ではルートノードのとなりのノードなどは存在しない, つまり $\Lambda 2$ というノードはありえないが, 本稿ではそのようなノードが存在する場合があるものとする. このような木 (の並び) を村田は生け垣 (hedge) と呼んだ [4].

3.1 二分木の連結と要素生成

二分木の連結演算 MM' および要素生成 $a\langle M\rangle$ を定義する. XML の視点でみると, 前者は XML の要素列 M と M' を連結するという演算であり, $a\langle M\rangle$ はその内容が M であるような a 要素を作る演算である. 二分木構造の言葉で定義すると以下のようになる.

定義 1. 二分木構造 M と M' について M の右端 (ここでは最上位のならびの右端, たとえば冒頭の文書ならばルート table のとなり) に M' を繋込む操作を M と M' の連結とよぶ. M の右端を v' ($\Lambda 2\cdots 2 \notin V$ で最も短いもの) として $MM' = M \cup v'\cdot M'$ と表現できる.

定義 2. 要素生成 $a\langle M\rangle$ は ルート Λ が a で, M がその内容であるようなものである. $a\langle M\rangle = \langle\{\Lambda\}, \Lambda \models a\rangle \cup \Lambda 1\cdot M$ と表現できる.

以上で a は $\{\mathbf{\Lambda}, -\mathbf{1}, -\mathbf{2}, \mathbf{1}, \mathbf{2}, T, F\}$ ではないものする. また $v\cdot M$ を $M = \langle V^M, \models^M\rangle$ に対してノード集合 $v\cdot V^M$ と充足関係 $v\cdot v' \models^{v\cdot M} a$ ($v' \models^M a$ に対して) による二分木構造とし, $M\cup M'$ を $\langle V^M \cup V^{M'}, \models^M \cup \models^{M'}\rangle$ とする. (上のようにしばしば M のノード集合を V^M で, M の充足関係を $\models^M$ で表現する.)

定義 3. 空の二分木構造 $\langle\emptyset,\emptyset\rangle$ を ϵ と表現する.

以上よりさきほどの文書は $\mathtt{table}\langle\mathtt{one}\langle\epsilon\rangle\mathtt{two}\langle\epsilon\rangle\mathtt{three}\langle\epsilon\rangle\rangle$ と表すことができる.

4 XML 文書型

XML では, 以下のような DTD (Document Type Definition) 形式で文書の型を表現する.

```
<!ELEMENT table (one|two|three)*>
<!ELEMENT one EMPTY>
<!ELEMENT two EMPTY>
<!ELEMENT three EMPTY>
```

この文書型は, ルートが table でその子ノードに任意の数, 順序の one, two, three の並びが続くような文書を認めるものである.

本稿では XML の文書型は次のような形で定義される (たとえば細谷ら [2] は μ-束縛子のかわりに x から τ への写像 (type definition) を用い, また村田 [4] は同じものを生け垣オートマトンとして表現し, また正規表現の Kleene 閉包 $\tau^* \equiv \mu x.(\tau x|\epsilon)$ を用いたりするが, どれも本質的には違いがない [1]).

$$\tau ::= \epsilon \mid \tau\tau \mid \tau|\tau \mid a\langle\tau\rangle \mid \mu x.\tau \mid x$$

ただし a は $\{\mathbf{\Lambda}, -\mathbf{1}, -\mathbf{2}, \mathbf{1}, \mathbf{2}, T, F\}$ ではないものとし, 以上の定義を満たすもので自由変数を持たないものが文書型であるとする. ϵ は空文書, $\tau\tau$ は連結, $\tau|\tau$ は選択, $a\langle\tau\rangle$ は要素をそして $\mu x.\tau$ は有限回の再帰を示す. ここで (1) 変数 x は常に連結の最後に出現するとする, すなわち $x\tau$ や $(\tau|x)\tau'$ などは許さない. これは細谷らによる right linearity とよばれる制約である (これがないと型として文脈自由言語を許すことになる). (2) 型 τ は空ではない. つまり $\mu x.\tau$ の $\tau = t_1|\cdots|\tau_n$ の中のどれかひとつは必ず変数 x を持たない. また (3) $\tau\tau$ は結合則 $\tau(\tau'\tau'') = (\tau\tau')\tau''$ を満たし, また (4) $\tau\epsilon = \epsilon\tau = \tau$ であるものとする (これらを便宜上, 同一視する).

[1] このように定義された文書型 τ によって表現される言語のクラスは「二分木上の正規言語のクラス」(二分木オートマトンによって受理される言語クラス) [2], あるいは「unranked な木上の正規言語のクラス」(生け垣オートマトンによって受理される言語クラス) [4] とちょうど一致する.

文書型の意味は次のように帰納的に与えられる. 上の (3), (4) により, 自由変数を持たない文書型は $\epsilon, (\tau|\tau')\tau'', a\langle\tau\rangle\tau'$ または $(\mu x.\tau)\tau'$ のどれかの形であるとみなせる.

定義 4. 与えられた $M = \langle V,\models\rangle$ の $\models$ の定義を以下のように帰納的に拡張する.

$$\frac{v\notin V}{v\models\epsilon}\qquad \frac{v\models\tau\tau''}{v\models(\tau|\tau')\tau''}\qquad\frac{v\models\tau'\tau''}{v\models(\tau|\tau')\tau''}$$

$$\frac{v\models a\quad v\cdot 1\models\tau\quad v\cdot 2\models\tau'}{v\models a\langle\tau\rangle\tau'}\qquad\frac{v\models\tau[\mu x.\tau/x]\tau'}{v\models(\mu x.\tau)\tau'}$$

ここで $\tau[\mu x.\tau/x]$ とは τ 中に自由に出現する変数 x を $\mu x.\tau$ に書きかえることをしめす. 今, M において $\Lambda\models\tau$ であるならば (これを $\models^M \tau$ と書く), M は型 τ に含まれるという意味となる.

上の DTD は型 $\mathtt{table}\langle\mu x.((\mathtt{one}\langle\epsilon\rangle|\mathtt{two}\langle\epsilon\rangle|\mathtt{three}\langle\epsilon\rangle)x|\epsilon)\rangle$ に対応する. ちなみにこの型は本節の冒頭の XML 文書を含むような型である.

4.1 文書型に対する演算

次に型 τ に関して演算 $split(\tau)$ と $a^{-1}\langle\tau\rangle$ を用意する. 正規表現 a^*b^* を考えるとき, これに属する任意の連結語 w_1w_2 について, $w_1\in a^*$ かつ $w_2 \in a^*b^*$ であるかまたは, $w_1\in a^*b^+$ かつ $w_2\in b^*$ であることが確かめられる. 文書型 τ に対してもこのような分割を $\tau\equiv\tau_1\tau_1'|...|\tau_n\tau_n'$ (ただし常に $\models\tau \Leftrightarrow \models\tau'$ ならば $\tau\equiv\tau'$) であるような有限個の $\tau_i\tau_i'$ の組で表現できる. 本稿ではこのような分割演算の存在は自明として詳細は与えず, 非操作的に次のように定義する.

定義 5. 型 τ について $\tau\equiv\tau_1\tau_1'|...|\tau_n\tau_n'$ という同値変形であって, 任意の木の連結 MM' が型 τ に属する (つまり $\models^{MM'}\tau$) ならば, ある i について $\models^M\tau_i$ かつ $\models^{M'}\tau_i'$ が成立するようなものが存在する. このような変形は無数に存在するが, 本稿では各 τ についてこのような変形をひとつ選ぶような関数が与えられているものとする. この関数を

$$split(\tau) = \{\langle\tau_1,\tau_1'\rangle, ..., \langle\tau_n,\tau_n'\rangle\}$$

と定義する.

もうひとつの演算 $a^{-1}\langle\tau\rangle$ は τ の $a\langle\tau'\rangle$ という形の部分型から τ' を取り出す演算である.

$$\frac{}{\epsilon(_) \Downarrow \epsilon} \quad \frac{v\cdot m \notin V}{me(v) \Downarrow \epsilon} \quad \frac{e(v\cdot m) \Downarrow M \quad v\cdot m \in V}{me(v) \Downarrow M} \quad \frac{v \not\models a}{\mathrm{if}(a)e(v) \Downarrow \epsilon} \quad \frac{e(v) \Downarrow M \quad v \models a}{\mathrm{if}(a)e(v) \Downarrow M}$$

$$\frac{e[\mu x.e/x](v) \Downarrow M}{\mu x.e(v) \Downarrow M} \quad \frac{e(v) \Downarrow M \quad e'(v) \Downarrow M'}{ee'(v) \Downarrow MM'} \quad \frac{e(v) \Downarrow M}{a\langle e\rangle(v) \Downarrow a\langle M\rangle}$$

図 2: 操作意味論 (帰納的定義)

定義 6. τ が与えられた時 $a^{-1}\langle\tau\rangle$ を $\models^{a\langle M\rangle} \tau$ ならば, またその時にかぎり $\models^{M} a^{-1}\langle\tau\rangle$ であるような型として定義する. もしそのような型がない場合は $a^{-1}\langle\tau\rangle = F$ とする.

4.2 文書型演算の閉包

以上に定義した $split(\tau)$ と $a^{-1}\langle\tau\rangle$ は本稿の型推論を行うときに, 木変換器の出力と出力型をマッチさせて, その出力の逆操作を出力型に対して適用するためのものである. よってたとえば 8 節の型構築アルゴリズムの過程で再帰的に評価される出力型は次の閉包 $C(\tau)$ の要素である.

$$\frac{}{\tau \in C(\tau)} \quad \frac{\tau' \in C(\tau) \quad \langle\tau_i, \tau_i'\rangle \in split(\tau')}{\tau_i, \tau_i' \in C(\tau)}$$

$$\frac{\tau' \in C(\tau)}{a^{-1}\langle\tau\rangle \in C(\tau)}$$

型構築アルゴリズムの実行可能性を保証するためにはこの $C(\tau)$ の数がたかだか有限であるかどうかが問題となるが, これについては次がいえる.

命題 1. $C(\tau)$ の大きさがたかだか $O(|\tau|^2)$ となるように $split(\tau)$ と $a^{-1}\langle\tau\rangle$ を定義することができる.

詳細は省くが, 細谷ら [2] などのアルゴリズムによって, τ と等価な $O(|\tau|)$ の大きさの 2 分木オートマトンを作ることができる. この二分木オートマトンの任意の 2 つの状態のペアで $C(\tau)$ の要素を表現することができるのである. ただし本稿では, 簡単のためにオートマトンは用いず τ で通すことにする.

例 1. $\mu x.(a\langle\epsilon\rangle x|\epsilon)$ という型に対して $split$ と a^{-1} を次のように定義できる.

$$split(\mu x.(a\langle\epsilon\rangle x|\epsilon)) = \{\langle \mu x.(a\langle\epsilon\rangle x|\epsilon),\ \mu x.(a\langle\epsilon\rangle x|\epsilon)\rangle\}$$
$$split(\epsilon) = \{\epsilon, \epsilon\}$$
$$a^{-1}\langle \mu x.(a\langle\epsilon\rangle x|\epsilon)\rangle = \epsilon$$
$$a^{-1}\langle\epsilon\rangle = F$$

この場合 $C(\mu x.(a\langle\epsilon\rangle x|\epsilon)) = \{\mu x.(a\langle\epsilon\rangle x|\epsilon), \epsilon, F\}$ である.

5 木変換器

XSLT は広く使われている XML の変換言語である. 次は XSLT のプログラムの例であり, 入力の XML 木を出力にそのままコピーするものである.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org
/1999/XSL/Transform" version="1.0">
 <xsl:output method="xml" indent="yes"/>

 <xsl:template match="*">
    <xsl:copy>
       <xsl:apply-templates select="./*"/>
    </xsl:copy>
 </xsl:template>

</xsl:stylesheet>
```

ここでは XSLT の構文の説明には立ち入らない. 以下ではこの XSLT のモデルとなるような木変換器を設計する. 木変換器と XSLT との関連は, 既発表論文 [6] により詳しいのでそちらも参考にしてほしい.

様相演算子として以下のものを導入する.

$$m ::= 1 \mid 2$$

この様相演算子上での木変換器を次のように定義する.

$$e ::= \epsilon \mid me \mid \mathrm{if}(a)e \mid ee \mid a\langle e\rangle \mid \mu x.e \mid x$$

移動文 me と条件文 $\mathrm{if}(a)e$ は入力木に対する操作、空文 ϵ, 連結文 ee と要素生成文 $a\langle e\rangle$ は出力木を構築する操作である. 今, 入力木 M_{in} は固定されているものとする. 木変換器の動作はこの入力木のルートノード Λ から開始し, 入力木を調べその情報をもとに出力木を構築する.

各構文の意味は次のようである.

- 空文 ϵ は空の出力木を生成する.
- 移動文 me は入力木のカレントノード (現在注目しているノード) を v から $v\cdot m$ に移動し, その上で e を実行する. もし移動できない場合, すなわち $v\cdot m \notin V$ であれば空の出力木をかえす.
- 条件文 $\mathrm{if}(a)e$ はカレントノードで a が成立していれば e を実行し, 成立していなければ空の出力木をかえす.

- 連結文 ee' は e と e' の評価結果の二つの出力木の連結を出力する．

- 要素生成文 $a\langle e\rangle$ は要素 a を生成し，その要素の下位ノードの内容として e の評価結果を用いる．

- 再帰文 $\mu x.e$ および e 中の x の出現で再帰呼出しを表現する．

木変換器の出力部分の定義は文書型の定義と似ているが，今回は right linearity の制約がないことに注意する．すなわち $\mu x.exe'$ のような文が許される．

図 2 に e の評価のための操作意味論を与える．

定義 7. 述語 $e(v) \Downarrow M$ は入力木 M_{in} の上のノード v に対して e を評価した結果が M であることを示す (M_{in} を明示的に示したい場合は $e(v^{M_{\mathrm{in}}}) \Downarrow M$ と書くことにする)．ルートノードに対して $e(\Lambda) \Downarrow M$ であるような M を e を M_{in} に適用した場合の出力木とよぶ

ここで，$\mu x.x$ のような停止しないプログラムの場合は $\mu x.x(\Lambda)$ が評価結果を持たないことに注意する．このようにプログラムが停止しない状況は上方向の様相演算子を加えた場合にはより頻繁に起こりうる．本稿の型検証アルゴリズム，および型構築アルゴリズムはこのプログラムの停止性も同時に検証するものである．

例 2. 冒頭の XSLT スタイルシートは入力文書を出力文書にコピーするものであり，次のような木変換器に対応する．

$$\mu x.(\mathrm{copy}(\mathrm{children}(x)))$$

ただし，$\mathrm{children}(e) = 1\mu x.e(2x)$ としまた各 $a_1, .., a_n \in L$ について

$$\mathrm{copy}(e) = (\mathrm{if}(a_1)a_1\langle e\rangle) \cdots (\mathrm{if}(a_n)a_n\langle e\rangle)$$

であるとする．

6 型検証アルゴリズム

木変換器，入力型，出力型を与えられて，「入力型に合致するどんな木も出力型に合致する木に書換えられるか」ということを判定するアルゴリズムを与えるのが本稿の目的である．このようなアルゴリズムははじめに述べたように，XSLT の静的な検査，あるいはデバッグなどの応用がある．提案されるアルゴリズムはふたつあり，ひとつは本節で考察する型付け規則 $e : \tau \to \tau'$ を直接評価する手法である．もうひとつは型構築アルゴリズムと様相論理の充足可能性判定を組み合わせる方法であり，これは 8 節で与える．

本節ではまず型付け規則を与える際に問題となる 2 点を説明したのち，$e : \tau \to \tau'$ を定義する (図 5)．

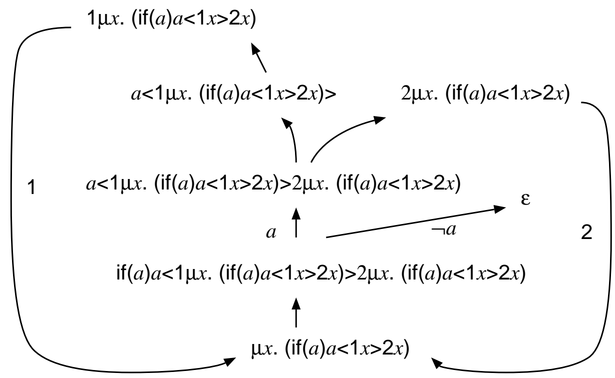


図 3: 閉包 C'

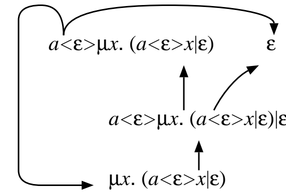


図 4: 閉包 C''

6.1 停止性判定と入力型

木変換器の評価 $e(v) \Downarrow M$ の際に再帰的に出現する式 e' もたかだか e のサイズに対して線形個しか存在しない (このような e' の集合を $C'(e)$ で表現することにする．図 3 参照)．木変換器の停止性を確認するためには，木の同じノードに対して同じ e' を (たかだか有限個なので) 適用しないことが必要十分である．たとえば図 3 のプログラムはループ中 1 や 2 (me の実行) の遷移の部分を実行するために，入力木をたえず下方向に移動するため，ループが同じノードにとどまることはなくよって停止する．

しかし，一般には木変換器が停止するかどうかは木がどのようなものであるかに依存する．つまり，入力型によって木の形が制限されることによって，一般には停止しないはずのプログラムが停止する可能性があるのである．また次節のように上方向への動作を持つ木変換器を考えると $C'(e)$ 上で 1 や 2 を通ることだけに着目してもプログラムの停止性はいえない．

そこで本稿の型検証では $C'(e)$ をみるかわりに，入力型 τ に対する $\models^{M_{\mathrm{in}}} \tau$ の証明木の有限性によって，木変換器の停止性を保証するという方法をとる．明らかに $\models^{M_{\mathrm{in}}} \tau$ の証明木中に出現するある $v' \models^{M_{\mathrm{in}}} \tau'$ の上方には $v' \models^{M_{\mathrm{in}}} \tau'$ はこれ以上出現しない．いまこの証明木から v' を忘れたものを考える (これを $C''(\tau)$ で表現する)．この場合 τ' の書換えが一回でも進んだならば，以

$$\frac{\epsilon \in \tau'}{\epsilon : _ \to \tau'} \qquad \frac{\epsilon \in \tau'}{_ : \epsilon \to \tau'} \qquad \frac{e : \tau\tau'' \to \tau''' \quad e : \tau'\tau'' \to \tau'''}{e : (\tau|\tau')\tau'' \to \tau'''}$$

$$\frac{e[\mu x.e/x] : \tau \to \tau'}{\mu x.e : \tau \to \tau'} \qquad \frac{e : \tau \to \tau'}{1e : a\langle\tau\rangle_ \to \tau'} \qquad \frac{e : \tau \to \tau'}{2e : a\langle_\rangle\tau \to \tau'} \qquad \frac{e : a\langle\tau\rangle\tau' \to \tau''}{\text{if}(a)e : a\langle\tau\rangle\tau' \to \tau''} \qquad \frac{e : b\langle\tau\rangle\tau' \to \epsilon \quad a \neq b}{\text{if}(a)e : b\langle\tau\rangle\tau' \to \epsilon}$$

$$\frac{e : \tau \to a^{-1}\langle\tau'\rangle \quad a^{-1}\langle\tau'\rangle \neq F}{a\langle e\rangle : \tau \to \tau'} \qquad \frac{\begin{array}{l} split(\tau') = \{\langle\tau_1, \tau_1{}'\rangle, ..., \langle\tau_n, \tau_n{}'\rangle\}, \\ \forall I \subseteq \{1..n\}.\ e : \tau \to \tau_{I_1}|\cdots|\tau_{I_k} \vee e' : \tau \to \tau'_{\bar{I}_1}|\cdots|\tau'_{\bar{I}_j} \\ \qquad (I = \{I_1, ..I_k\},\ \bar{I} = \{1..n\} \setminus I = \{\bar{I}_1, ..., \bar{I}_j\}) \end{array}}{ee' : \tau \to \tau'} \qquad \frac{\begin{array}{c} \left[e : (\mu x.\tau)\tau' \to \tau''\right] \\ \vdots \\ e : \tau[\mu x.\tau/x]\tau' \to \tau'' \end{array}}{e : (\mu x.\tau)\tau' \to \tau''}$$

図 5: 型付け規則 (帰納的定義)

降に τ' が再び出現したとしてもそれは違うノード v'' に対応するものとなる. 図 4 は $C''(\mu x.(a\langle\epsilon\rangle x \mid \epsilon))$ を示しているが, これはループとなっている部分においても, ループをまわるたびに違うノードに対する評価を行っているということを示す図である.

直観的に言うと, 本節の型検証アルゴリズムでは $C'(e)$ のグラフと $C''(\tau)$ のグラフとの直積 (のうち妥当な型つけを示す枝のみを残したもの) をとっている. そして得られたグラフのループのうち, $C''(\tau)$ 上では遷移が進んでいないようなループを well-founded ではないものとして除外するということを行う. 残ったループは $C'(e)$ 上では同じ部分を実行していても必ず違うノードに対応するものであることが保証される.

6.2 ee' 文の取り扱い

停止性判定とならぶもうひとつのポイントは, ee' 文の取り扱いである. ee' によって型 τ の木が型 τ' の木に変換されるという述語,

$$ee' : \tau \to \tau'$$

を検証するとき, ある $\langle\tau_i, \tau_i{}'\rangle \in split(\tau')$ について $e : \tau \to \tau_i$ かつ $e' : \tau \to \tau_i{}'$ であるかを調べるだけでは十分条件になっていない. なぜなら, τ に属するある入力文書 M に対しては $e(\Lambda^M)$ と $e'(\Lambda^M)$ の結果がそれぞれ $\langle\tau_i, \tau_i{}'\rangle$ に属しているが, 同じ τ に属する別の入力文書 M' に対しては変換の結果が $\langle\tau_j, \tau_j{}'\rangle$ に属している場合がありうるからである.

細谷ら [2] は $\tau \subseteq \tau'$ という木言語の包含を判定するアルゴリズムを考案した. 彼らのアルゴリズムは似たような状況を取り扱っており, 一般化すると集合の直積の間の以下の包含

$$\tau \times \tau' \subseteq \tau_1 \times \tau_1{}' \cup \cdots \cup \tau_n \times \tau_n{}'$$

を調べるために次の式を調べればよいことを用いている.

$$\forall I \subseteq \{1, .., n\}.\ \tau \subseteq \tau_{I_1} \cup \cdots \cup \tau_{I_k} \vee \tau' \subseteq \tau'_{\bar{I}_1} \cup \cdots \cup \tau'_{\bar{I}_j}$$

ただし $I = \{I_1, ..I_k\}$ また $\bar{I} = \{1, .., n\} \setminus I = \{\bar{I}_1, ..., \bar{I}_j\}$ とする.

本節で提案するアルゴリズムも $ee' : \tau \to \tau'$ の検証のために同様のアイディアを用いる. すなわち $ee' : \tau \to \tau'$ の検証が成立するためには次を調べればよい.

$$\forall I \subseteq \{1..n\}.\ e : \tau \to \tau_{I_1}|\cdots|\tau_{I_k} \vee e' : \tau \to \tau'_{\bar{I}_1}|\cdots|\tau'_{\bar{I}_j}$$

ただし, $split(\tau') = \{\langle\tau_1, \tau_1{}'\rangle, ..., \langle\tau_n, \tau_n{}'\rangle\}$ とする. 彼らの木言語の包含判定アルゴリズムは一般の場合には組み合わせ爆発を起こすが, 実用上は XML 処理言語 XDuce の実装に使われるなど, 問題なく動作する. 本節のアルゴリズムのボトルネックもこの部分であり, $split(\tau')$ の大きさに対して組み合わせ爆発を起こす可能性があるが, 実用上どうかについてはまだ確認していない. [2]

6.3 アルゴリズム

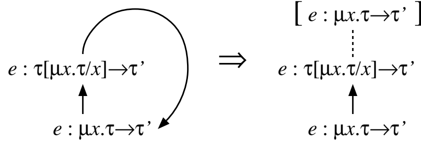
以上の 2 点を考慮して作った型付け規則を図 5 に示す. $\to$ の左辺に関しては 1 段目のルール群および 3 段目の最後のルール (後述) をくり返し適用することにより $a\langle\tau\rangle\tau'$ にまで分解しそこで if や *me* を適用する. また $\to$ の右辺に関しては 3 節で定義した操作を用いて 3 段目のルール群で処理している. 最後のルール,

$$\frac{\begin{array}{c} \left[e : (\mu x.\tau)\tau' \to \tau''\right] \\ \vdots \\ e : \tau[\mu x.\tau/x]\tau' \to \tau'' \end{array}}{e : (\mu x.\tau)\tau' \to \tau''}$$

は本節冒頭の議論に対応している. これは, $e : (\mu x.\tau)\tau' \to \tau''$ を仮定して, $e : \tau[\mu x.\tau/x]\tau' \to \tau''$ が示せるならば, $e : (\mu x.\tau)\tau' \to \tau''$ 自体が成立するという意味である. $^{[}\vdots^{]}$ の部分のないままでは帰納的定義はどの

[2] 一般に $C(\tau)$ (4.2 節) の要素の部分集合の和 $\tau_1|\cdots|\tau_k$ が $e : \tau \to \tau'$ の右辺 τ' に出現することになる. ただし, この和についての *split* や a^{-1} 演算の結果も, 同様に $C(\tau)$ の要素の部分集合の和で表現することができる. これにより型検証アルゴリズムは実行可能である.

ような定義のループも許さないが, 冒頭での議論により, → の左辺の展開が一回でも起こっている場合はループしていても入力木のノードを移動するようなループであり, 許されなければならない. そこで, このようなループを下図,



のような切り離しを用いて, 帰納的に評価できるようにするということを行っている.

型検証アルゴリズムはこの型付け規則を, 下から上へ評価することによって実現でき, 探索空間が有限であることから停止する. $e : (\mu x.\tau)\tau' \to \tau''$ のルールに関しては, この 3 つ組を証明木の上方で次回出会うまで記憶しもし出会ったならば, そこで成立とすればよい.

定理 1. $e : \tau \to \tau'$ であるとき, またその時にかぎり $\models^{M_{\mathrm{in}}} \tau$ ならば変換 $e(\Lambda^{M_{\mathrm{in}}}) \Downarrow M$ が停止して $\models^{M} \tau'$ である. [3]

証明. 証明すべき補題は以下のようなものである.

$$e : \tau \to \tau' \Leftrightarrow \forall v, M. (v \models^{M} \tau \Rightarrow \ \models^{e(v^M)} \tau')$$

ここでは $\models^{e(v^M)} \tau'$ とは

$$\begin{array}{l}(v^M = \epsilon \Rightarrow \epsilon \in \tau') \\ \wedge (v^M \neq \epsilon \Rightarrow \exists M'. \, e(v^M) \Downarrow M' \wedge \ \models^{M'} \tau')\end{array}$$

の略記とする.

まず (⇒) 方向を示す. 右辺の ∀ を外して

$$\forall M. \, e : \tau \to \tau' \ \wedge \ v \models^{M} \tau \Rightarrow \ \models^{e(v^M)} \tau'$$

を証明する. 帰納法は外側では $v \models^{M} \tau$ の導出に関するものであり, 内部で $e : \tau \to \tau'$ の導出に関する帰納法を使う.

帰納法のベースは $e : \tau \to \tau'$ の上段 1 番目, 2 番目で, 両方とも容易である. 帰納ステップについて, ここでは $ee' : \tau \to \tau'$ の導出 (v と τ は不変) を使ったものについて示す.

$$\begin{array}{l}ee' : \tau \to \tau' \ \wedge \ v \models^{M} \tau \\ \Rightarrow \forall I \subseteq \{1..n\}.(e : \tau \to \tau_{I_1}|\cdots|\tau_{I_k} \vee e' : \tau \to \tau'_{\bar{I}_1}|\cdots|\tau'_{\bar{I}_j}) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \ v \models^{M} \tau \\ \Rightarrow \forall I \subseteq \{1..n\}.(\models^{e(v^M)} \tau_{I_1}|\cdots|\tau_{I_k} \vee \ \models^{e'(v^M)} \tau'_{\bar{I}_1}|\cdots|\tau'_{\bar{I}_j}) \\ (\text{帰納法の仮定}) \\ \Rightarrow \exists \langle \tau_i, \tau_i' \rangle \in split(\tau').(\models^{e(v^M)} \tau_i \ \wedge \ \models^{e'(v^M)} \tau_i') \\ (\text{積和標準形 } (cf. \text{ 細谷 [2]})) \\ \Rightarrow \ \models^{(ee')^{v^M}} \tau' \\ (ee' \text{ の操作}, MM' \text{ の定義}, split(\tau') \text{ の定義})\end{array}$$

(⇐) 方向を考える. 基本的なアイディアは $e : \tau \to \tau'$ の成立を保証するためには, 全ての入力木を調べる必要はなくたかだか $e : \tau \to \tau'$ の証明木の大きさと関連する大きさの入力木のみを調べればよいということである. 証明は入力の型 τ に出現する μ を展開した回数 k, τ の大きさ, e の大きさのこの順での辞書式順序に関する帰納法により, 以下を示す.

$$\forall v, M. (v \models^{M}_{k} \tau \Rightarrow \ \models^{e(v^M)} \tau') \Rightarrow e :_k \tau \to \tau'$$

ここで

$$\frac{v \notin V}{v \models_0 \epsilon} \qquad \frac{v \models_k \tau[\mu x.\tau/x]\tau'}{v \models_{k+1} (\mu x.\tau)\tau'}$$

また,

$$\frac{\epsilon \in \tau'}{\epsilon :_k _ \to \tau'} \qquad \frac{\epsilon \in \tau'}{_ :_k \epsilon \to \tau'}$$

$$\frac{}{e :_0 (\mu x.\tau)\tau' \to \tau''} \qquad \frac{e :_k \tau[\mu x.\tau/x]\tau' \to \tau''}{e :_{k+1} (\mu x.\tau)\tau' \to \tau''}$$

と定義する. (これら以外の定義は元の定義を k を変えないように使う), これらの述語は μ を展開する回数を数えるものである. 後者の述語は k 回の μ の展開で導出ができない場合は真になるように定義されている. 帰納法のベースは $k = 0$ かつ $\tau = \epsilon$ または $e = \epsilon$ の場合であり容易. 帰納ステップは, $\mu x.\tau$ 以外たとえば $v \models_k a\langle\tau\rangle\tau'$ は $v{\cdot}1 \models_k \tau$, $v{\cdot}2 \models_k \tau'$ に帰着して示すことができる. $\mu x.\tau$ に関しては k が増える.

$$\begin{array}{l}\forall v, M. (v \models^{M}_{k+1} (\mu x.\tau)\tau' \Rightarrow \ \models^{e(v^M)} \tau'') \\ \Rightarrow \forall v, M. (v \models^{M}_{k} \tau[\mu x.\tau/x]\tau' \Rightarrow \ \models^{e(v^M)} \tau'') \\ \Rightarrow e :_k \tau[\mu x.\tau/x]\tau' \to \tau'' \\ (\text{帰納法の仮定}) \\ \Rightarrow e :_{k+1} (\mu x.\tau)\tau' \to \tau''\end{array}$$

ここで, $e : \tau \to \tau'$ の証明木の大きさはたかだか $|C(\tau')| \times |C'(e)| \times |C''(\tau)|$ 以下であり, これより大きい k で $e :_k \tau \to \tau'$ ならば証明木中の $\mu x.\tau$ に関するループを $e : (\mu x.\tau)\tau' \to \tau'$ のルールを使って除去し, $e :_0 (\mu x.\tau)\tau' \to \tau''$ のルールを使わないような証明木にかえることができ, $e : \tau \to \tau'$ が得られる. よって (⇐) が言える. □

7　上方向への移動

本節と次節では木変換器が移動文 me によって下方向だけでなく, 上方向にも移動する場合を考察する. やるべきことは様相演算子を以下のように拡張することである.

$$m ::= 1 \mid 2 \mid -1 \mid -2$$

図 2 では me 文の評価は $v{\cdot}m$ の定義にしたがっている. $v{\cdot}1, v{\cdot}2$ の意味は文字の連結であるが, −1, −2 に対し

[3] $e(\Lambda^{M_{\mathrm{in}}}) \Downarrow M$ は $M_{\mathrm{in}} = \epsilon$ の場合を定義していないが ($\Lambda \notin V$ のため), 定理の記述を簡単にするためにここでは $M_{\mathrm{in}} = \epsilon$ ならば変換結果 $M = \epsilon$ であると仮定する.

ては $v\cdot 1\cdot -1 = v$, $v\cdot 2\cdot -2 = v$ またそれ以外の場合は $v\cdot -1, v\cdot -2 \notin V$ と定義する. $m = 1, 2$ は木を下降する動作に対応しているのに対し, $-1, -2$ は木を上方に動く動作に対応している.

例 3. 次のような XSLT プログラムを考えることができる.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org
/1999/XSL/Transform" version="1.0">
 <xsl:output method="xml" indent="yes"/>

 <xsl:template match="*">
    <xsl:if test="self::two">
       <xsl:apply-templates select=".."/>
    </xsl:if>
    <xsl:apply-templates select="./*"/>
 </xsl:template>

</xsl:stylesheet>
```

このプログラムは次のような木変換器に対応する.

$$\mu x.((\text{if}(\texttt{two})\text{parent}(x))\text{children}(x))$$

ただし $\text{parent}(e) = \mu x.((\text{if}(-\mathbf{1})-1e)\text{if}(-\mathbf{2})-2x)$ であるものとする. プログラムの動作はもし two がルート以外のノードに存在するならば, 停止しないというものである.

7.1 二分木上の命題 μ-計算

二分木命題 μ-計算の式 φ は以下のように定義される.

$$\varphi ::= a \mid m\varphi \mid \neg\varphi \mid \varphi|\varphi \mid \varphi\&\varphi \mid \mu x.\varphi \mid x$$

ただし x は否定的には (二重否定も含めて) 現れない. 通常の命題 μ-計算においては最大不動点 $\nu x.\varphi$ を考えるが, 本稿では (本稿の目的のためには十分なため) 最小不動点 $\mu x.\varphi$ だけを考える. 最小不動点 $\mu x.\varphi$ のみしか存在しない (さらに否定の現れない) μ-計算の意味は以下のような簡単な帰納的定義である. [4]

$$\frac{v \models \varphi \quad v \models \psi}{v \models \varphi\&\psi} \quad \frac{v \models \varphi}{v \models \varphi|\psi} \quad \frac{v \models \psi}{v \models \varphi|\psi}$$

$$\frac{v \cdot m \notin V}{v \models m\varphi} \quad \frac{v\cdot m \models \varphi}{v \models m\varphi} \quad \frac{v \models \varphi[\mu x.\varphi/x]}{v \models \mu x.\varphi}$$

否定に関しては次のように定義する.

$$v \models \neg\varphi \Leftrightarrow v \not\models \varphi$$

定義 8. x が否定的に現れないために φ の意味は上の帰納的定義と否定に関するルールを式の大きさに関する帰納法で適用していけば一意に定まる.

[4]注意: (1) τ とは違い $v \notin V$ については未定義. (2) 3 番目のルールにより $m\varphi$ が $v \cdot m \notin V$ において必ず成立する.

以上のように定義された φ を用いると, 木変換器の条件判定文 $\text{if}(a)e$ を $\text{if}(\varphi)e$ のように拡張することができる. これより, XML の検索に使われる XPath 式の持つ「上方向」に関する条件を自然に表現できる. 以下の例における not(parent::table) は Xpath の述語の一例である.

例 4. XSLT では <xsl:if test="not(parent::table)"/> によって親ノードの値が table でないかどうかで条件分岐ができる. これは $\text{if}(\text{parent}(\neg\texttt{table}))e$ によって表現できる. ただし $\text{parent}(\varphi) = \mu x.(-1\varphi\&-2x)$ とする (親要素が存在すれば φ が成立するという意味になる).

7.2 文書型から μ-論理式への変換

文書型 τ を与えられ, 等価な μ-論理式に変換するのは容易である. 以下にアルゴリズムを示す.

定義 9. 以下の関数 $conv$ によって $\models \tau \Leftrightarrow \models conv(\tau)$ を満たす μ-論理式が得られる.

$$\begin{aligned}
&conv(\tau) = conv_{\emptyset}(\tau)\\
&conv_{\Gamma}(\epsilon) = F\\
&conv_{\Gamma}((\tau|\tau')\tau'') = conv_{\Gamma}(\tau\tau'')|conv_{\Gamma}(\tau'\tau'')\\
&conv_{\Gamma}((\mu x.\tau)\tau') = conv_{\Gamma}(\tau[\mu x.\tau/x]\tau')\\
&conv_{\Gamma}(a\langle\tau\rangle\tau') = \mu x_{a\langle\tau\rangle\tau'}.\ a\&1conv_{\Gamma\cup\{a\langle\tau\rangle\tau'\}}(\tau)\\
&\qquad\qquad \&2conv_{\Gamma\cup\{a\langle\tau\rangle\tau'\}}(\tau') \quad (a\langle\tau\rangle\tau' \notin \Gamma)\\
&conv_{\Gamma}(a\langle\tau\rangle\tau') = x_{a\langle\tau\rangle\tau'} \qquad\qquad (a\langle\tau\rangle\tau' \in \Gamma)
\end{aligned}$$

ただし, $x_{a\langle\tau\rangle\tau'}$ とは $a\langle\tau\rangle\tau'$ によって識別される新しい変数とする. たとえば, $\mu x.(\mathsf{a}\langle x\rangle x|\mathsf{b}\langle\epsilon\rangle)$ は $\mu x.(\mathsf{a}\&1\mu y.(\mathsf{a}\&1y\&2y)\&2x|\mathsf{b}\&1F\&2F)$ に変換される. ただし, それぞれ x は $x_{\mathsf{a}\langle\mu x.(\mathsf{a}\langle x\rangle x|\epsilon)\rangle\mu x.(\mathsf{a}\langle x\rangle x|\epsilon)\mathsf{b}\langle\epsilon\rangle}$ のように, y は $x_{\mathsf{a}\langle\mu x.(\mathsf{a}\langle x\rangle x|\epsilon)\rangle\mu x.(\mathsf{a}\langle x\rangle x|\epsilon)}$ のようにラベル付けされていた変数である. また μ-論理式の意味から $1F \equiv \neg\mathbf{1}$ また $2F \equiv \neg\mathbf{2}$ である.

7.3 命題 μ-計算の表現力と充足可能性

過去の様相をもつ一般の命題 μ 計算について次のような定理が知られているが, これらの結果は有限二分木上の命題 μ 計算に関するものにも適用できるものである.

定理 2. (Vardi [7]) (1) φ を与えられたとき任意の M について $\models \varphi \Leftrightarrow \models \tau$ であるような τ を構築できる. (2) 与えられた φ の充足可能性をたかだか指数時間で判定できる.

定理の (1) は φ が定義する言語クラスがちょうど TA で認識できる正規木言語のクラスと一致するということを示す.

8 入力型構築アルゴリズム

論文 [6] で議論されているように, 木変換器に対する順方向の型構築では正確な型を正規言語として求めるこ

$$
\begin{aligned}
&\epsilon^{-1}(\tau) = T && (\epsilon \in \tau),\\
&\epsilon^{-1}(\tau) = F && (\epsilon \notin \tau),\\
&(me)^{-1}(\tau) = m(e^{-1}(\tau)) && (\epsilon \in \tau),\\
&(me)^{-1}(\tau) = m(e^{-1}(\tau))\&(m) && (\epsilon \notin \tau),\\
&(\text{if}(\varphi)e)^{-1}(\tau) = \varphi\&e^{-1}(\tau)|\neg\varphi\&\epsilon^{-1}(\tau),\\
&(ee')^{-1}(\tau) = e^{-1}(\tau_1)\&e'^{-1}(\tau'_1)|\cdots|e^{-1}(\tau_n)\&e'^{-1}(\tau'_n)\\
&\qquad\qquad (split(\tau) = \{\langle\tau_1,\tau'_1\rangle,...,\langle\tau_n,\tau'_n\rangle\}),\\
&a\langle e\rangle^{-1}(\tau) = e^{-1}(a^{-1}\langle\tau\rangle),\\
&(\mu x.e)^{-1}(\tau) = \mu x_\tau.(e^{-1}(\tau))[(\forall\tau')\ \mu x_{\tau'}.(e^{-1}(\tau'))/x_{\tau'}]^*,\\
&x^{-1}(\tau) = x_\tau
\end{aligned}
$$

図 6: 入力型構築アルゴリズム

とができない. 実際,

$$e = (\mu x.(\text{copy}(1x)2x))\mu x.(\text{copy}(1x)2x)$$

というプログラムは木 M を MM に変換するものである. しかしよく知られた事実として正規言語 τ について $e(\tau) = \{MM \mid M \in \tau\}$ は正規言語たりえない. ところが以下で示すように出力の型から入力の型を構築するアルゴリズムにおいては構築される正規言語の型は正確である. すなわち後の定理において $\Rightarrow$ 方向 (健全性) と $\Leftarrow$ 方向 (完全性) の両方が成立するような型構築が可能となる.

以下のアルゴリズムは, 木を上方向にも動作するような木変換器と出力文書の型を与えられ, -1, -2 方向への様相演算子を持った μ 論理式を入力文書の型として推論する.

8.1 アルゴリズム

本節の型構築アルゴリズムは入力型を構築するだけであるため, 6 節の型検証アルゴリズムに比べると簡素なものになっている. やるべきことは木変換器の出力動作が出力型 τ を模倣するように両者の直積をとり, それに応じて入力文書の型を構築することだけである. 木変換器と出力文書型の直積をとる際, 木変換器の各 μx 束縛子と, 出力型 τ の書き換え中にあらわれる式 τ' との組をとり, 新しい変数 $x_{\tau'}$ を導入する. また図 6 の $\mu x_\tau.(e^{-1}(\tau))[(\forall\tau')\ \mu x_{\tau'}.(e^{-1}(\tau'))/x_{\tau'}]^*$ とは $\mu x_\tau.(e^{-1}(\tau))$ に自由にあらわれる全ての $x_{\tau'}$ に $\mu x_{\tau'}.(e^{-1}(\tau'))$ を代入するという動作を $x_{\tau'}$ という形の自由変数がなくなるまでくりかえすことを表す (一旦行われた代入 $\mu x_{\tau'}.(e^{-1}(\tau'))/x_{\tau'}$ について $x_{\tau'}$ は束縛されるために, $e^{-1}(\tau')$ がもう一度呼ばれることはない. これより定義が well-defined となる). また定義中の (m) は $(1) = \mathbf{1}$ などと解釈する.

定義 10. 木変換器 e の出力の型を τ とする時, その入力の型 $e^{-1}(\tau)$ は図 6 の再帰的定義にしたがって構築される.

例 5. 例 3 において出力型を ϵ とすると

$$
\begin{aligned}
&(\mu x.((\text{if}(\texttt{two})\text{parent}(x))\text{children}(x)))^{-1}(\epsilon)\\
&= \mu x_\epsilon.((\texttt{two}\&\text{parent}(x)^{-1}(\epsilon)|\neg\texttt{two})\&1\text{children}(x)^{-1}(\epsilon))\\
&\equiv \mu x_\epsilon.((\texttt{two}\Rightarrow\mu y_\epsilon.(-1x_\epsilon\&-2y_\epsilon)\&1(\mu y_\epsilon.x_\epsilon\&2y_\epsilon))\\
&\equiv \mu x_\epsilon.((\texttt{two}\Rightarrow\text{parent}(x_\epsilon))\&\text{children}(x_\epsilon))
\end{aligned}
$$

ただし $\varphi\Rightarrow\psi \equiv \neg\varphi|\psi$ とし, $\text{children}(\varphi) = 1\mu x.(\varphi\&2x)$ で全ての子要素に対して φ が成立するという意味とする. たとえば, 冒頭にあげた XML 文書は `two` 要素を持つためこの $\mu x_\epsilon.((\texttt{two}\Rightarrow\text{parent}(x_\epsilon))\&\text{children}(x_\epsilon))$ を満たさない.

定理 3. $\models^{M_{\text{in}}} e^{-1}(\tau)$ ならば $(\Rightarrow)$, またその時にかぎり $(\Leftarrow)$ 変換 $e(\Lambda)\Downarrow M$ が停止して $\models^M \tau$ である.

証明. まず代入列 $\theta = \mu x_1.e_1/x_1,...,\mu x_k.e_k/x_k$ に対して $mod(\theta)$ を次のように定義する.

$$mod(\theta) = (\forall\tau)\ \mu x_{1\tau}.(e_1^{-1}(\tau))/x_{1\tau},...,(\forall\tau)\ \mu x_{k\tau}.(e_k^{-1}(\tau))/x_{k\tau}$$

これは $x_{i\tau}$ の形の変数をすべて $\mu x_{i\tau}.(e_i^{-1}(\tau))$ に置き換える代入である. 補題 (帰納法の不変式) は $v \in V$ において,

$$v \models^{M_{\text{in}}} e^{-1}(\tau)[mod(\theta)]^* \Leftrightarrow \exists M.\ e[\theta](v)\Downarrow M\ \wedge\ \models^M \tau \quad \text{...(不変式)}$$

となる. 証明は $(\Rightarrow)$ については $v \models^{M_{\text{in}}} e^{-1}(\tau)[mod(\theta)]^*$ の導出に関する帰納法で, $(\Leftarrow)$ については $e[\theta](v)\Downarrow M$ の導出に関する帰納法で証明する. $(\Leftarrow)$ のベースは $e[\theta](v)\Downarrow\epsilon$ の場合, すなわち $e = \epsilon$, $e = \text{if}(a)e' \wedge v \not\models a$ または $e = me' \wedge m{\cdot}v \notin V$ の場合であり, どの場合も $v \models^{M_{\text{in}}} e^{-1}(\tau) \Leftarrow\ \models^\epsilon \tau$. 一方 $(\Rightarrow)$ のベースは, $e^{-1}(\tau) = T$ または $e^{-1}(\tau) = m\varphi\ \wedge\ v{\cdot}m \notin V$ の場合 ($\models^{M_{\text{in}}}$ の定義) でありやはり容易 ($M = \epsilon$ とする). 帰納ステップは e の形による場合わけで $(\Rightarrow)$ と $(\Leftarrow)$ を同時に証明する. ここでは $\mu x.e$ の場合を示す.

$$
\begin{aligned}
&v \models^{M_{\text{in}}} (\mu x.e)^{-1}(\tau)[mod(\theta)]^*\\
&\Leftrightarrow v \models^{M_{\text{in}}} \mu x_\tau.(e^{-1}(\tau))[(\forall\tau')\ \mu x_{\tau'}.(e^{-1}(\tau'))/x_{\tau'}, mod(\theta)]^*\\
&\Leftrightarrow v \models^{M_{\text{in}}} e^{-1}(\tau)[mod(\mu x.e/x,\theta)]^*\\
&\Leftrightarrow \exists M.\ e[\mu x.e/x,\theta](v)\Downarrow M\ \wedge\ \models^M \tau\\
&(\text{ 不変式 })\\
&\Leftrightarrow \exists M.\ \mu x.e[\theta](v)\Downarrow M\ \wedge\ \models^M \tau\\
&(\ \mu x.e(v)\Downarrow M \text{ の定義})
\end{aligned}
$$

よって証明された. □

8.2 型構築の応用

アルゴリズム $e^{-1}(\tau)$ の時間計算量と作られる μ-論理式のサイズはたかだか $O(|e||\tau|^3)$ なので, 型構築のアルゴリズム自体は効率的に動作する. 作られた μ-論理式を入力文書の型を検査するために使うことはもちろん可能である. たとえば, 出力文書が正しい XHTML の文書

であることが保証されるのかどうかを与えられた入力文書を前もって検査することで確かめられる.

一方, このアルゴリズムを用いて静的に型検証 $e : \tau' \to \tau$ の判定をするためには, $conv(\tau') \Rightarrow e^{-1}(\tau)$ が恒真であること, つまり $conv(\tau') \& \neg e^{-1}(\tau)$ が充足不可能であることを示せばよい. μ 論理式の充足可能性判定については定理 2(2) の結果があり, これより上方向への動作をもった XSLT プログラムの型検証が可能となる. ただし, Vardi の充足可能性判定の枠組をそのまま使うのか, あるいはより効率のよい判定手法があるかなど課題はいくつかある. また上方向への動作を木変換器に加えた場合に, 6 節で提案したような型検証アルゴリズムを直接使って, 型の整合性を判定する方法についてはよくわかっていない.

9　まとめ

本稿では XML 変換言語 XSLT を対象としてその型検査および型構築のアルゴリズムを考案した. とくに後者のアルゴリズムは, 入力木を上下方向に走査するような XSLT プログラムを取り扱うものであった.

本稿では記述を簡単にするために関連文献では有限遷移系を用いて書かれていたものをすべて μ 演算子をもつ式をつかって表現している.

- 木オートマトン $\longleftrightarrow$ μ 式をもつ文書型式
- 木変換器 $\longleftrightarrow$ μ をもつ木変換器式
- 交替 2 方向木オートマトン $\longleftrightarrow$ 二分木命題 μ 計算式

このように表現することで, たとえば既存研究 [6] の同等のアルゴリズム に比べてはるかに簡易かつ直観的にアルゴリズムが記述できるようになった. また本稿で提案する上方向への動作を持った命題 μ 計算式や木変換器は取り扱いの難しい木を上に移動するような動作や, 木の上部に関する条件などを自然に表現できるものである. 本稿で提案する形式化手法は XSLT の型検証の問題にとどまらない XML のさまざまな分野に広く応用が効く可能性がある.

10　謝辞

本研究を進める際に, 村田真氏および萩谷昌己氏の助言や知識が大変助けになったことに感謝します. また詳しく読んでくださった査読者の方々, 証明の手直しを待ってくださったプログラム委員の方々に感謝します.

参考文献

[1] XSL transformations (XSLT) version 1.0, 2000. `http://www.w3.org/TR/xslt`.

[2] H. Hosoya and B. Pierce. Regular expression pattern matching for XML. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2001.

[3] T. Milo, D. Suciu, and V. Vianu. Type-checking for XML transformers. In *Proceedings of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–22, 2000.

[4] M. Murata. Extended path expressions for XML. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 126–137, 2001.

[5] K. Rose and P. Audebaud. Stylesheet validation. ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2000/RR2000-37.ps.Z.

[6] A. Tozawa. Towards static type checking for XSLT. In *Proceedings of the 1st ACM Symposium on Document Engineering*. ACM Press, 2001.

[7] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proceedings of 25th Intl. Colloquium on Automata, Languages and Programming*, pages 628–641, 1998.