Macro Forest Transducer からの XML ストリーム処理器の自動導出*

中野圭介 †
Keisukie NAKANO

† 東京大学大学院情報理工学系研究科数理情報学専攻
Department of Mathematical Informatics, University of Tokyo
ksk@mist.i.u-tokyo.ac.jp

プログラマに負担をかけることなく効率的な XML 変換器を得るための一つの方法として，XML 文書木変換器からの XML ストリーム処理器の自動導出がある．従来の研究では，属性文法で表された XML 変換からの導出方法が示されていたが，本研究では，近年 Perst らにより提案された，更に表現力の高い macro forest transducer (MFT) から導出する方法を示す．MFT は，関数型言語に似た方法で XML 変換を定義することができるため，既存の XML 変換言語の代替としても利用できる．

1 はじめに

XML は木構造をもつデータ表現として扱うため，XML 変換はその木の上の再帰的なプログラムとして定義できる．実際，XDuce [9] や CDuce [2]，XSLT [19] などの XML 変換言語では，XML の木構造上の再帰関数の集合として変換プログラムが記述されている．XDuce や CDuce では，正規パターンによる再帰呼び出しを行っており，XSLT では，XPath 式による再帰呼び出しを行っている．

厳密にいえば，これらの XML 変換言語で扱われるデータは，木（tree）ではなく，**森**（forest）と呼ばれることが多い．これは，一般に各ノードの子の数やパターンに照合するノードの数が任意であるということに起因している．森は順序づけられた木のリストであり，木はラベル（要素名）とその子となる森から成っている．森は，以下の文法で与えられる．

$$\mathit{Forest} ::= \varepsilon \mid \sigma\langle \mathit{Forest}\rangle \mathit{Forest}$$

ここで，ε は空リストを表し，σ はある文字集合 Σ の元である．また，$\sigma\langle f_1\rangle f_2$ は森であり，その先頭は，森 f_1 を子とし名前 σ でラベルづけされた木で，f_2 はそれに後続する森である．森の末尾の ε はしばしば省略され，木 $\sigma\langle\varepsilon\rangle$ を単に σ と書くこともある．実際の XML では，テキストノードや属性ノードなどが扱われるが，これらは要素の一種として容易に拡張できるので，上のようなモデルに限定しても全く問題はない．たとえば，XML データ

```
<p>XML is <em>not</em> tree.</p>
```

は，森

$$\mathtt{p}\langle \mathit{XML}\ \mathit{is}\ \mathtt{em}\langle \mathit{not}\rangle\ \mathit{tree}.\rangle$$

で表現される．ここで，*XML* や *is* などは，$\mathit{XML}\langle\varepsilon\rangle$，$\mathit{is}\langle\varepsilon\rangle$ の略記表現である．この例では，テキストと要素名が混同しないように字体を変えて表現したが，当論文で扱われるモデルでは森のノードのラベルとして同様に扱われる．

Perst ら [16] は，XML 変換のモデルとして **macro forest transducer** (MFT) を提案した．MFT では，森から森への変換を，森の上の相互再帰関数として定義することができる．各関数は第一引数をパターン照合しながら，再帰を進めることができる．MFT は，各関数に関するルールの集合で定義されており，関数 f のルールは，

$$f(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to \mathit{Rhs} \qquad \text{または，}$$
$$f(\varepsilon, y_1, \ldots, y_n) \to \mathit{Rhs}$$

の形をしており，f は第一引数として入力となる森の一部を受け取っている[1]．変数 $y_1, \ldots, y_n$ は蓄積変数であり，*Rhs* では，これらの変数を用いて新たな森を生成したり，$x_i (i = 1, 2)$ を第一引数として関数を再び呼び出すこともできる．各関数は上のようなルールに従い，第一引数をパターン照合し，すべての変数 $x_1, x_2, y_1, \ldots, y_n$ に値を束縛し，*Rhs* の式を計算する．第二引数以降はパターン照合されない．

*本研究の一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託を受けた東京大学において実施したものである．

[1] Tree transducer ではオートマトンなどと同様に f を状態と呼んでいる．

このような森の上の再帰は，二分木

$$Tree ::= \varepsilon \mid \sigma(Tree, Tree)$$

の上で定義される再帰と同じものであるとも考えられる．実際，macro tree transducer（MTT）[5] はこの二分木の上の再帰関数を定義することができるが，表現力は MFT よりも弱いとこうこともPerst ら [16] によって示されている．これは，MFT がルールの右辺 *Rhs* において森と森の結合操作を許していることに起因する差である．Perst らは，MTT でこの結合操作を扱うには，もう一つ別の MTT が必要になるということも示した．

MFT のような再帰的な関数定義は，森を操作する上で非常に自然であるが，メモリ消費や実行時間などが非効率になることが多い．ユーザの定義する再帰関数は森の構造に従って変換が定義されており，入力全体の森の構造をメモリに確保する必要があるためである．通常の XML 変換の実装では，入力として文字列ストリームで与えられた XML を最後まで読み，それに対する完全な森の構造をメモリに確保してから，ようやく計算が始まる．実世界には，巨大な XML を扱うアプリケーションも少なくなく，この非効率性は著しい弊害を招く．

SAX [18] をはじめとする XML ストリーム処理は，この問題を解決する手段としてよく使われてきた．ストリーム処理では，XML の構造全体をメモリに確保することがなく，なおかつ入力が終わる前に変換結果の出力を開始できるため，メモリ消費や実行時間の両方を最小限に抑えることが可能である．一般に，ストリーム処理のプログラムは，入力イベントに付随する関数の集合として与えられる．ここでいうイベントとは，XML の開始タグや終了タグなどの一つの読み込みのことである．各関数は以下のように定義される．

$$
\begin{aligned}
&\mathit{initInfo} : \mathit{info}\\
&\mathit{initInfo} = \ldots\\
&\mathit{beginElement}(\mathit{name}, \mathit{info}) : \mathit{string} \times \mathit{info} \to \mathit{info}\\
&\mathit{beginElement}(\mathit{name}, \mathit{info}) =\\
&\ldots;\ \mathit{print}\ldots;\ \ldots\\
&\mathit{endElement}(\mathit{name}, \mathit{info}) : \mathit{string} \times \mathit{info} \to \mathit{info}\\
&\mathit{endElement}(\mathit{name}, \mathit{info}) =\\
&\ldots;\ \mathit{print}\ldots;\ \ldots\\
&\qquad\vdots
\end{aligned}
$$

XML ストリーム処理は，ある型 *info* をもつ情報（初期値は *initInfo*）をイベント毎に更新していくことで進められる．入力から開始タグイベント<a>が読み込まれ，現在の情報が i である場合には，a と i を引数として関数 *beginElement* が呼ばれる．同様に，終了タグイベントが読み込まれた場合には関数 *endElement* が呼ばれる．各関数が呼ばれる毎に，*print* 関数により変換結果の一部を出力をすることも可能である．

XML ストリーム処理はメモリ消費や実行時間が節約できる反面，プログラミングが困難であるという深刻な問題を抱えている．型 *info* をどのような情報のデータにすればよいか，また，どのようなタイミングでどのような出力をするべきかということは，すべてプログラマに委ねられており，設計も非常に難しくバグも引き起こされやすい．

本研究では，MFT のような森の上の再帰関数による XML 変換プログラムから，SAX のような XML ストリーム処理プログラムを自動導出する手法を提案する．これにより，プログラマに負担をかけることなく，効率的な実行コードを得ることが可能になる．本手法は，tree transducer の融合理論に基づいている．XML ストリーム処理器は，与えられた MFT M に対し，

$$\text{XML アンパーサ} \circ M \circ \text{XML パーサ}$$

という関数合成で表現できる．ここで，XML パーサは XML ストリームから森への変換，XML アンパーサは森から XML ストリームへの変換である．XML アンパーサと MFT の融合は，MFT のルールの変形で容易に可能であるが，XML パーサと MFT の融合はやや難しい問題である．一般に，XML パーサは，文脈自由言語の構文解析と同様，スタックを用いなければならないが，可算無限の状態があるような top-down tree transducer（TDTT）として表現できる．TDTT は，蓄積変数を一切許さないような MTT であり，MTT と融合できることが Engelfriet らによって示されている [5]．しかし，今回適用する XML ストリーム処理器の導出の場合，無限の状態があるような TDTT を扱う必要がある上，MTT を拡張した MFT を融合の対象としているため，この融合はまだ確立していない．彼らの手法を自然に拡張することにより融合は可能であるが，拡張された手法の正しさは新たに証明する必要がある．本論文では，その融合の証明は避け，XML パーサに特化した融合アル

ゴリズムを与え，得られた XML ストリーム処理器が元の MFT と同じ役割を果たすことを直接証明している．

本研究の基盤となる著者らによる研究 [14, 15] では，XML 変換を属性文法で表現し XML パーサと融合することにより，ストリーム処理器の自動導出を実現していたが，本研究よりも表現力の低いクラスを扱っている．彼らの扱う属性文法は，ほぼ attrributed tree transducer (ATT) [6] に相当する表現力を持つが，表現力としては MFT より劣っていることは，

$$\mathrm{ATT} \subsetneq \mathrm{MTT} \subsetneq \mathrm{MFT}$$

という包含関係から容易に確認できる [5, 16]．

本稿は，この章を含め 6 つの章から構成されている．第 2 節では，本枠組で扱う XML のモデルである森およびストリームを形式化し，第 3 節で森の上の変換としての MFT を導入する．第 4 節では，XML ストリーム処理器の形式化を与え，MFT からの自動導出アルゴリズムとその正当性を示す．第 5 節で関連研究について触れ，第 6 節で今後の展望を含め，本稿のまとめとする．

2 XML データモデル

本節では，形式化された XML データモデルを導入する．簡単のため，本研究で扱われるモデルでは，いくつかの制限を設けている．

第一に，要素ノードのみを扱い，テキストノードや属性ノードは扱わないものとする．第 1 節でも述べた通り，これは本質的な制限ではない．テキストノードや属性ノードは特殊な要素ノードとして定義することができ，それらのノードに対しても，本稿が提供するアルゴリズムも容易に拡張できる．

第二に，ストリーム処理の入力となる XML は，**well-formed** であると仮定する．すなわち，すべての入力 XML は開始タグと終了タグのバランスがとれているものとする．それ以外の入力に関する動作についてはここでは議論しない．この仮定により，終了タグの名前は無視してもよい．そこで，本稿における枠組では，入出力ともに終了タグの名前は考えないこととする．入力は単に無視すればよいし，出力に関しても開始タグの名前を積むスタックを用意するだけで復帰することができる．そのスタックの深さは文書の深さを超えないので，このスタックによるメモリ消費は，変換自体による消費に比べればごく僅かなものである．

森による XML 構造モデル　XML はルートが 1 つしかないと仕様で定められているが，多くの XML 変換言語では，複数のノードを同時に扱う形で変換を定義している．そこで，子の数だけでなく，ルートの数も任意であるような森のモデルを考える．**文字集合 Σ 上の森**は，文法

$$\mathit{Forest} ::= \varepsilon \mid \sigma\langle \mathit{Forest} \rangle \mathit{Forest}$$

で定義される．ここで，$\sigma \in \Sigma$ である．記号 ε は空列を表し，$\sigma\langle f_1 \rangle f_2$ は，森 f_1 を子とし σ でラベル付けられた木を先頭とし，森 f_2 を後続する弟の列とするような森である．また，Σ 上の森の集合を $\mathcal{F}_\Sigma$ で表すものとする．二つの森 f_1, f_2 の結合は，単に $f_1 f_2$ と並べて表現する．

XML ストリームのモデル　XML ストリームは，名前付きの開始タグと名前無しの終了タグの列で与えられる．**文字集合 Σ 上の XML ストリーム**は，文法

$$\mathit{Stream} ::= \varepsilon \mid \sigma\texttt{[}, \mathit{Stream}, \texttt{]}, \mathit{Stream}$$

で定義される．ここで，$\sigma \in \Sigma$ である．記号 ε は空文字列，σ[は開始タグ，] は終了タグを表す．_, _（コンマ）は文字列の結合を表しているが，便宜上，以後はコンマを省略するものとする．また，Σ 上の XML ストリームの集合を $\mathcal{S}_\Sigma$，記号の集合 $\{\sigma\texttt{[} \mid \sigma \in \Sigma\} \cup \{\texttt{]}\}$ を $\Sigma_{[]}$ で表すものとする．上の文法は，$\Sigma_{[]}$ の元から成る列の集合 $\Sigma_{[]}^*$ のうち，well-defined な XML ストリームのみを受け付けていることがわかる．$\Sigma_{[]}$ の元は **XML イベント**とも呼ばれ，入力の終了を表すイベント \$ を合わせて，$\Sigma_{[]}^+$ で表す．すなわち，$\Sigma_{[]}^+ = \Sigma_{[]} \cup \{\$\}$ である．また，森 f に対応する XML ストリームを $\lfloor f \rfloor$ で表す．この変換は以下のように自然に定義できる．

$$\lfloor \varepsilon \rfloor = \varepsilon \qquad \lfloor \sigma\langle f_1 \rangle f_2 \rfloor = \sigma\texttt{[}\, \lfloor f_1 \rfloor\, \texttt{]}\, \lfloor f_2 \rfloor$$

3 Macro Forest Transducers

Macro forest transducer (MFT) は，Perst ら [16] が提案した森から森への変換のモデルである．このモデルは，**macro tree transducer** (MTT) [5] の拡張で，森同士の結合を許可しているため，MTT を超える表現力を持っている．

3.1 MFT の定義

本稿では，Perst らによる MFT の定義と同じものを利用する．tree transducer に慣れ親しんでいない

読者は，MFT の各ルール

$$q(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to Rhs$$

を名前 q の関数の定義と見なすことによって，理解がしやすくなるかもしれない．各関数の第一引数は森であり，$\sigma\langle x_1\rangle x_2$ というパターンに照合したときに，変数 x_1, x_2 が束縛され，このルールの右辺が適用される．右辺では，関数 q の他の引数 $y_1, \ldots, y_n$ も何度でも利用することができる．関数 in は，変換の最初に呼ばれる主関数として理解される．

定義 3.1 Macro forest transducer（MFT）は，5 つ組 $M = (Q, \Sigma, \Delta, in, R)$ で与えられ，各要素は以下の項目に従う．

- Q は**状態**の集合である．各状態には 1 以上のランクが与えられており，その状態に関するルールにおける引数の数が固定されている．
- Σ は**入力に関する文字集合**を表し，Δ は**出力に関する文字集合**を表す．ただし，$Q \cap (\Sigma \cup \Delta) = \emptyset$ とする．
- $in \in Q$ は**初期状態**を表す．
- R は**ルール**の集合であり，各ルールは

  $$q(Pat, y_1, \ldots, y_n) \to Rhs$$

  の形で与えられる．ここで，$q \in Q$ はランク $n+1$ の状態であり，$y_1, \ldots, y_n$ は変数である．左辺の Pat は森に照合するパターンであり，ε または $\sigma\langle x_1\rangle x_2$ で表される．ただし，$\sigma \in \Sigma$ とし，x_1, x_2 は変数とする．右辺の Rhs は以下の文法で与えられる式である．

  $$\begin{aligned} Rhs ::= {} & q'(x_i, Rhs, \ldots, Rhs) \mid y_j \\ & \mid \varepsilon \mid \delta\langle Rhs\rangle Rhs \mid Rhs\,Rhs \end{aligned}$$

  ここで，$q' \in Q$，$i = 1, 2$，$j = 1, \ldots, n$，$\delta \in \Delta$ である．ただし，$Pat = \varepsilon$ の場合には，右辺に変数 x_i が現れないものとする．ルールの集合 R のうち，左辺の状態が q であるものを **q-ルール**と呼ぶ．

3.2 MFT の意味論

Perst ら [16] の手法により，MFT の意味論を定義する．この意味論は，各状態を関数として翻訳することにより与えられる．

定義 3.2 $M = (Q, \Sigma, \Delta, in, R)$ を MFT とし，$f \in \mathcal{F}_\Sigma$ をその入力となる森とする．ランク $n+1$ の状態 $q \in Q$ の**動作関数**は，$[\![q]\!] : \mathcal{F}_\Sigma \times (\mathcal{F}_\Delta)^n \to \mathcal{F}_\Delta$ で表され，q-ルールに従って以下のように定義される．

- $[\![q]\!](\varepsilon, \phi_1, \ldots, \phi_n) = [\![Rhs]\!]_\rho$ である．ただし，$(q(\varepsilon, y_1, \ldots, y_n) \to Rhs) \in R$ とし，ρ は $\rho(y_j) = \phi_j (j = 1, \ldots, n)$ で与えられる代入とする．
- $[\![q]\!](\sigma\langle f_1\rangle f_2, \phi_1, \ldots, \phi_n) = [\![Rhs]\!]_\rho$ である．ただし，$(q(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to Rhs) \in R$ とし，ρ は $\rho(x_i) = f_i (i = 1, 2)$ および $\rho(y_j) = \phi_j (j = 1, \ldots, n)$ で与えられる代入とする．

ここで，$[\![\_\_]\!]_\rho$ は代入 ρ によって定まるルールの右辺の評価関数で次のように帰納的に定義される．

$$\begin{aligned} & [\![q'(x_i, r_1, \ldots, r_m)]\!]_\rho = [\![q']\!](\rho(x_i), [\![r_1]\!]_\rho, \ldots, [\![r_n]\!]_\rho) \\ & [\![y_j]\!]_\rho = \rho(y_j), \qquad [\![\varepsilon]\!]_\rho = \varepsilon \\ & [\![\delta\langle r_1\rangle r_2]\!]_\rho = \delta\langle [\![r_1]\!]_\rho\rangle [\![r_2]\!]_\rho, \quad [\![r_1 r_2]\!]_\rho = [\![r_1]\!]_\rho [\![r_2]\!]_\rho \end{aligned}$$

Perst らの定義では非決定的な動作を与えているため，動作関数 $[\![\_\_]\!](\_\_)$ の返り値は集合であったが，本稿では，決定的かつ全域，すなわち，入力となるすべての森に対し出力がただ一つに定まるような MFT のみを考え，動作関数の返り値を単一の森とした．

MFT による森から森への変換は，初期状態の動作関数によって定められ，与えられた入力に対する変換結果は，その動作関数に第一引数として入力を適用することによって得られる．蓄積引数はすべて空列を与えるものとする．

定義 3.3 $M = (Q, \Sigma, \Delta, in, R)$ を MFT とする．M による**変換** $\tau_M : \mathcal{F}_\Sigma \to \mathcal{F}_\Delta$ は，

$$\tau_M(f) = [\![in]\!](f, \varepsilon, \ldots, \varepsilon)$$

によって定義される．

3.3 MFT の例

MFT の例として，ある書籍データを入力とし，以下の手続きを同時にする変換を考えよう．

- 章番号の追加
- `chapter` 要素の下の `title` を `name` に変更
- 書籍データの最後に `key` を列挙

```
book⟨title⟨What is MFT?⟩
     chapter⟨title⟨Introduction⟩
              XML is key⟨Forest⟩.⟩
     chapter⟨title⟨About MFT⟩
              key⟨MFT⟩ transforms Forests.⟩
     chapter⟨title⟨Conclusion⟩
              MFT transforms XML.⟩
     ⟩
```

(a) 書籍データを表す森（変換前）

```
book⟨title⟨What is MFT?⟩
     chapter⟨S Z
              name⟨Introduction⟩
              XML is key⟨Forest⟩.⟩
     chapter⟨S S Z
              name⟨About MFT⟩
              key⟨MFT⟩ transforms Forests.⟩
     chapter⟨S S S Z
              name⟨Conclusion⟩
              MFT transforms XML.⟩
     index⟨entry⟨Forest⟩
           entry⟨MFT⟩⟩
     ⟩
```

(b) 書籍データを表す森（変換後）

図 1: 変換前と変換後の森

具体的には，図 1(a) の森から図 1(b) の森を生成するような変換である．厳密な MFT ではすべての操作を抽象化しており，整数やその上の計算などは定義されていないため，章番号として，サクセサ S とゼロ Z を組み合わせた表現を用いた．この制限の緩和は難しくない．目的の変換は，以下を満たす $\mathrm{MFT}(Q, \Sigma, \Delta, in, R)$ として与えられる．ただし，Σ と Δ は十分な数の文字列が含まれているものとし，R には，明示的に与えられているルール以外に，各 $q \in Q$，$\sigma \in \Sigma$ に対し，

$$q(\varepsilon, y_1, \ldots, y_n) \to \varepsilon,$$
$$q(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to \sigma\langle q(x_1, y_1, \ldots, y_n)\rangle q(x_2, y_1, \ldots, y_n)$$

に対応するルールが重複のない範囲で省略されているものとする．

$$\begin{aligned}
Q &= \{in, idx, t2n, id\}\\
R &= \{\ in(\texttt{book}\langle x_1\rangle x_2, y_1) \to \\
&\qquad \texttt{book}\langle in(x_1, Z)\ \texttt{index}\langle idx(x_1)\rangle\rangle\\
&\quad in(\texttt{chapter}\langle x_1\rangle x_2, y_1) \to\\
&\qquad \texttt{chapter}\langle S\ y_1\ t2n(x_1)\rangle in(x_2, S\ y_1)\\
&\quad idx(\texttt{key}\langle x_1\rangle x_2) \to \texttt{entry}\langle id(x_1)\rangle idx(x_2)\\
&\quad idx(\sigma\langle x_1\rangle x_2) \to idx(x_1) idx(x_2)\ (\sigma \neq \texttt{key})\\
&\quad t2n(\texttt{title}\langle x_1\rangle x_2) \to \texttt{name}\langle id(x_1)\rangle t2n(x_2)\\
&\qquad \vdots\\
\}&
\end{aligned}$$

4 XML ストリーム処理器とその自動導出

本節では，SAX スタイルの XML ストリーム処理器を形式化し，MFT からのどのように自動導出されるかを示す．

4.1 XML ストリーム処理器

XML ストリーム処理は，ある「情報」をイベント毎に更新することによって行われる．本枠組では，「情報」として部分的に評価された出力結果を保持するものとし，入力の終了イベントを読んだ段階で，その「情報」は出力そのものとなる．XML ストリーム処理器は，MFT と同様に定義されるが，各ルールは，一つのイベントを読むことによって現在の情報がどのように更新されるかを与えている．

定義 4.1 XML ストリーム処理器は，5 つ組 $S = (Q, \Sigma, \Delta, in, R)$ で表され，

- Q は**状態**の（可算無限）集合である．各状態には 0 以上のランクが与えられており，その状態に関するルールにおける引数の数が固定されている．
- Σ は**入力に関する文字集合**を表し，Δ は**出力に関する文字集合**を表す．ただし，$Q \cap (\Sigma \cup \Delta) = \emptyset$ とする．
- $in \in Q$ は**初期状態**を表す．
- R は**ルール**の集合であり，各ルールは，

$$q(y_1, \ldots, y_n) \xrightarrow{\chi} Rhs$$

の形で与えられる．ここで，$q \in Q$ はランク n の状態であり，$y_1, \ldots, y_n$ は変数である．また，$\chi \in \Sigma^+_{[]}$ は入力イベントであり，右辺 Rhs は以下の文法で与えられる式である．

$$Rhs ::= q'(Rhs_1, \ldots, Rhs_m) \mid y_j \mid \varepsilon \mid \delta\texttt{[}Rhs\texttt{]}\,Rhs \mid Rhs\,Rhs$$

ここで，$q' \in Q$，$j = 1, \ldots, n$，$\delta \in \Delta$ である．ただし，$\chi = \$$ の場合には，右辺は状態を含む式 $q'(\ldots)$ は全く現れないものとする．ルールの集合 R のうち，左辺の状態が q で，イベント χ に付随するルールを (q, χ)-**rule** と呼ぶ．

4.2 XML ストリーム処理器の意味論

XML ストリーム処理器の各ルールは，イベント毎の読み込みに対する処理のみを与えているため，MFT の意味論の定義とは異なる．XML ストリーム処理は，イベント毎に，部分評価された出力結果の暫定的な情報を更新することにより進められるが，この「暫定的な情報」は次のように定義される．

定義 4.2 $S = (Q, \Sigma, \Delta, in, R)$ を XML ストリーム処理器とする．S に関する**暫定情報** Tmp は文法

$$Tmp ::= q(Tmp, \ldots, Tmp) \mid \varepsilon \mid \delta\texttt{[}Tmp\texttt{]}\,Tmp \mid Tmp\,Tmp$$

で表される．S に関する暫定的な情報を $\mathcal{B}_S$ で表す．

XML ストリーム処理器 S の意味論は，各状態を，暫定情報を更新する関数に翻訳することにより与えられる．

定義 4.3 $S = (Q, \Sigma, \Delta, in, R)$ を XML ストリーム処理器とする．XML イベントに関する $\mathcal{B}_S$ の**更新関数**は，$\langle\!\langle \_\_, \_\_ \rangle\!\rangle : \mathcal{B}_S \times \Sigma^+_{[]} \to \mathcal{B}_S$ で表され，R 内のルールに従って以下のように定義される．

- $\langle\!\langle q(e_1, \ldots, e_n), \chi \rangle\!\rangle = [\![Rhs]\!]_\rho$ である．ただし，$(q(y_1, \ldots, y_n) \xrightarrow{\chi} Rhs) \in R$ とし，ρ は $\rho(y_j) = \langle\!\langle e_j, \chi \rangle\!\rangle$ $(j = 1, \ldots, n)$ で与えられる代入である．
- $\langle\!\langle \varepsilon, \chi \rangle\!\rangle = \varepsilon$.
- $\langle\!\langle \delta\texttt{[}e_1\texttt{]}e_2, \chi \rangle\!\rangle = \delta\texttt{[}\langle\!\langle e_1, \chi \rangle\!\rangle\texttt{]}\langle\!\langle e_2, \chi \rangle\!\rangle$.
- $\langle\!\langle e_1 e_2, \chi \rangle\!\rangle = \langle\!\langle e_1, \chi \rangle\!\rangle \langle\!\langle e_2, \chi \rangle\!\rangle$.

ここで，$[\![\_\_]\!]_\rho$ は代入 ρ によって定まるルールの右辺の評価関数で次のように帰納的に定義される．

$$[\![q'(r_1, \ldots, r_n)]\!]_\rho = q'([\![r_1]\!]_\rho, \ldots, [\![r_2]\!]_\rho)$$
$$[\![y_j]\!]_\rho = \rho(y_j), \qquad [\![\varepsilon]\!]_\rho = \varepsilon$$
$$[\![\delta\texttt{[}r_1\texttt{]}r_2]\!]_\rho = \delta\texttt{[}[\![r_1]\!]_\rho\texttt{]}[\![r_1]\!]_\rho, \quad [\![r_1 r_2]\!]_\rho = [\![r_1]\!]_\rho [\![r_1]\!]_\rho$$

XML ストリーム処理は，次のように進められる．暫定情報の初期値を $in(\varepsilon, \ldots, \varepsilon)$ とし，読まれるイベント毎に，更新関数を利用してこの情報を更新することにより，ストリーム処理が行われる．終了イベント $\$$ が読み込まれると，この更新は終了し，その時点での暫定情報が出力結果そのものになっている．$\$$ に関するルールに状態を含む式がないことおよび更新関数の定義を考慮すると，この更新の後の暫定情報は $\mathcal{S}_\Delta$ の元であることが容易にわかる．S を XML ストリーム処理器とし，$\chi_1 \chi_2 \ldots \chi_k$ を S の入力となる XML ストリームとすると，XML ストリーム処理の結果は，

$$\langle\!\langle \ldots \langle\!\langle in(\varepsilon, \ldots, \varepsilon), \chi_1 \rangle\!\rangle, \chi_2 \rangle\!\rangle, \ldots, \chi_k \rangle\!\rangle, \$\rangle\!\rangle \qquad (1)$$

で表せる．

しかしながら，式 (1) に従うようなストリーム処理は我々の満足するような効率的なものではない．理想的なストリーム処理器は，入力を読み終えるよりも前に出力を開始するべきである．式 (1) では，出力の開始に関しては何も触れておらず，終了イベント $\$$ を読むまで変換結果が分からない．

本枠組では，この問題を解決することはさほど難しいことではない．実は，XML ストリーム処理器の意味論は，理想的なストリーム処理を実現することが容易であるように定義されている．たとえば，$l (1 \le l < k)$ 番目の入力イベント χ_l を読むまでの結果が $\delta\texttt{[}e$ の形をしていたとしよう．すなわち，

$$\langle\!\langle \ldots \langle\!\langle in(\varepsilon, \ldots, \varepsilon), \chi_1 \rangle\!\rangle, \chi_2 \rangle\!\rangle, \ldots, \chi_l \rangle\!\rangle = \delta\texttt{[}e$$

である．このとき，式 (1) から最終的な結果は，

$$\langle\!\langle \ldots \langle\!\langle \delta\texttt{[}e, \chi_{l+1} \rangle\!\rangle, \ldots, \chi_k \rangle\!\rangle, \$\rangle\!\rangle$$

で表すことができる．定義 4.3 から，この値が

$$\delta\texttt{[}\langle\!\langle \ldots \langle\!\langle e, \chi_{l+1} \rangle\!\rangle, \ldots, \chi_k \rangle\!\rangle, \$\rangle\!\rangle$$

と等しいことが示すことができるため，出力結果が $\delta\texttt{[}$ で始まることは，少なくとも l 番目の入力イベントを読んだ段階で知ることができる．つまり，各イ

ベントを読むたびに先頭にある文字列をチェックすることで，変換結果を少しずつ出力することができるのである．このような先頭のチェックを**文字列の絞り出し**と呼び，ストリーム処理器の実装の際には，イベントの読み出し毎に絞り出しを行うようにするのが望ましい．

ただ，本研究の目的は，MFT からの XML ストリーム処理器の自動導出およびその正当性の証明であり，絞り出しを行わない簡潔な定義の方が取り扱いやすいため，式 (1) のような定式化を扱うことにする．本枠組で自動導出された XML ストリーム処理器は，絞り出しをするストリーム処理器へも容易に変形できる．

定義 4.4 $S = (Q, \Sigma, \Delta, in, R)$ を XML ストリーム処理器とする．S による**変換** $\tau_S : \mathcal{S}_\Sigma \to \mathcal{S}_\Delta$ は，

$$\tau_S(s) = \theta_S(in(\varepsilon, \ldots, \varepsilon), s\$)$$

によって定義される．ここで，関数 $\theta_S : \mathcal{B}_S \times \Sigma^*_{[]} \to \mathcal{B}_S$ は

$$\theta_S(e, \varepsilon) = e, \qquad \theta_S(e, \chi s) = \theta_S(\langle\!\langle e, \chi \rangle\!\rangle, s)$$

によって与えられるものとする．

4.3 XML ストリーム処理器の導出

与えられた MFT からの XML ストリーム処理器の導出は，既存の tree transducer との融合の拡張により実現できる．第 1 節で述べた通り，本稿では，一般的な tree transducer との融合は考えず，XML ストリーム処理器の導出に特化したアルゴリズムを提案し，その正当性を示す．

与えられた MFT M の状態集合を Q とすると，自動導出で得られる XML ストリーム処理器の状態集合は $Q \times \mathbb{N}$ という可算無限集合で与えられる．ここで，$\mathbb{N}$ を非負整数の集合とする．

定義 4.5 MFT $M = (Q, \Sigma, \Delta, in, R)$ から自動導出される XML ストリーム処理器 $\mathcal{SP}(M) = (Q', \Sigma, \Delta, in', R')$ は，以下のように与えられる．

- $Q' = \{q[i] \mid q \in Q,\ i \in \mathbb{N}\}$ であり，$q[i] \in Q'$ のランクは，M における q のランクを $n(\geq 1)$ とすると，$n-1$ で表される．
- $in' = in[0] \in Q'$.
- R' は以下の項目より与えられるルールの集合である．
  - すべての $q \in Q$ と $\sigma \in \Sigma$ とルール $q(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to Rhs \in R$ に対し，R' は $(q[0], \sigma\texttt{[})$-ルール

    $$q[0](y_1, \ldots, y_n) \xrightarrow{\sigma\texttt{[}} \mathcal{A}(Rhs),$$

    を含む．
  - すべての $q \in Q$ と $\sigma \in \Sigma$ と $i \in \mathbb{N}$ とルール $q(\varepsilon, y_1, \ldots, y_n) \to Rhs \in R$ に対し，R は次のような $(q[0], \texttt{]})$-ルールと $(q[i], \$)$-ルールを含む．

    $$q[0](y_1, \ldots, y_n) \xrightarrow{\texttt{]}} \mathcal{A}(Rhs)$$
    $$q[i](y_1, \ldots, y_n) \xrightarrow{\$} \mathcal{A}(Rhs)$$

  - すべての $q \in Q$ と $\sigma \in \Sigma$ と $i \geq 1$ に対し，R' は次のような $(q[i], \sigma\texttt{[})$-ルールと $(q[i], \texttt{]})$-ルールを含む．

    $$q[i](y_1, \ldots, y_n) \xrightarrow{\sigma\texttt{[}} q[i+1](y_1, \ldots, y_n),$$
    $$q[i](y_1, \ldots, y_n) \xrightarrow{\texttt{]}} q[i-1](y_1, \ldots, y_n),$$

  ここで，$\mathcal{A}$ は R における右辺 Rhs の式の上に定義された関数で，以下のように与えられる．

$$\mathcal{A}(q(x_1, r_1, \ldots, r_n)) = q[0](\mathcal{A}(r_1), \ldots, \mathcal{A}(r_n))$$
$$\mathcal{A}(q(x_2, r_1, \ldots, r_n)) = q[1](\mathcal{A}(r_1), \ldots, \mathcal{A}(r_n))$$
$$\mathcal{A}(y_j) = y_j, \qquad \mathcal{A}(\varepsilon) = \varepsilon$$
$$\mathcal{A}(\delta\langle r_1\rangle r_2) = \delta\texttt{[}\mathcal{A}(r_1)\texttt{]}\mathcal{A}(r_2)$$
$$\mathcal{A}(r_1 r_2) = \mathcal{A}(r_1)\mathcal{A}(r_2)$$

4.4 導出アルゴリズムの正当性

先に提示した XML ストリーム処理器の導出アルゴリズムが正当であるとは，すべての MFT とその入力となる森に対し，二つのストリーム

- MFT によるその森の変換結果に対応する XML ストリーム
- MFT から導出アルゴリズムにより得られる XML ストリーム処理器による，その森に対応するストリームの変換結果

が一致することである．この正当性は，以下の定理によって定式化できる．

定理 4.6 $M = (Q, \Sigma, \Delta, in, R)$ を MFT とすると，すべての森 $f \in \mathcal{F}_\Sigma$ に対して，

$$\tau_{\mathcal{SP}(M)}(\lfloor f \rfloor) = \lfloor \tau_M(f) \rfloor$$

が成り立つ．

この定理は，$\lfloor \_\_ \rfloor$ や $\langle\!| \_\_ |\!\rangle$ や θ_S の拡張の性質に関するいくつかの補題によって証明される．これらの補題を議論する前に，**後続森スタック**という概念を導入する．この概念は，あくまで自動導出の正当性を証明するために導入するものであり，自動導出アルゴリズムや導出される XML ストリーム処理器の実装に関わるものではない．森 f に関する後続森スタックは，f の部分森を要素とするようなスタックであり，文法

$$L ::= [] \mid f' :: L$$

によって定義される．ここで，f' は f の部分森である．森 f に関する後続森スタックの集合は FF_f で表す．後続森スタック L の i 番目の要素は $L.i$ で表され，$(f' :: L).0 = f'$ と $(f' :: L).i = L.(i-1)$ によって与えられる．

森 f に関する後続森スタックは，ある特定の方法で更新していくことにより，XML ストリーム $\lfloor f \rfloor$ のイベントを前から順番に 1 つずつ出力することができるという，非常に便利な性質を持っている．森 f の後続森スタックの初期値は $f :: []$ で与えられ，その更新関数 ud は，現在の後続森スタックを受け取り，次の XML イベントと更新後の後続森スタックを返す関数で，以下のように定義される．

$$\begin{aligned} ud(\sigma\langle f_1 \rangle f_2 :: l) &= (\sigma\texttt{[}, f_1 :: f_2 :: l) \\ ud(\varepsilon :: f :: l) &= (\texttt{]}, f :: l) \\ ud(\varepsilon :: []) &= (\texttt{\$}, []). \end{aligned}$$

ここで，$ud([])$ は未定義とするが，本稿ではこの計算は出現しない．この更新関数により，f に関する後続森スタックから $\lfloor f \rfloor$ のイベントが 1 つずつ出力できるということが，次の補題によって確かめられる．

補題 4.7 後続森スタック上の関数 g を以下のように定義する．

$$g(l) = \begin{cases} \varepsilon & \text{if } l = [] \\ \chi\, g(l') & \text{otherwise} \end{cases}$$

ただし，$(\chi, l') = ud(l)$ とする．このとき，

$$g(f :: []) = \lfloor f \rfloor \texttt{\$} \tag{2}$$

が成り立つ．

証明 まず，任意の森 f, f' と後続森スタック l に対し，等式

$$g(f :: f' :: l) = \lfloor f \rfloor \texttt{]}\, g(f' :: l) \tag{3}$$

が成り立つことを，f の構造に関する帰納法で示す．$f = \varepsilon$ のとき，$g, ud, \lfloor \_\_ \rfloor$ の定義から式 (3) は成り立つ．$f = \sigma\langle f_1 \rangle f_2$ のとき，$g, ud, \lfloor \_\_ \rfloor$ の定義と帰納法の仮定から，

$$\begin{aligned} g(\sigma\langle f_1 \rangle f_2 :: f' :: l) &= \sigma\texttt{[}\, g(f_1 :: f_2 :: f' :: l) \\ &= \sigma\texttt{[}\, \lfloor f_1 \rfloor \texttt{]}\, g(f_2 :: f' :: l) \\ &= \sigma\texttt{[}\, \lfloor f_1 \rfloor \texttt{]}\, \lfloor f_2 \rfloor \texttt{]}\, g(f' :: l) \\ &= \lfloor \sigma\langle f_1 \rangle f_2 \rfloor \texttt{]}\, g(f' :: l) \end{aligned}$$

が成り立つ．よって，任意の f について，式 (3) が成り立つ．

次に式 (2) を f の構造に関する帰納法で示す．$f = \varepsilon$ のとき，$g, ud, \lfloor \_\_ \rfloor$ の定義から式 (2) が成り立つ．$f = \sigma\langle f_1 \rangle f_2$ のとき，$g, ud, \lfloor \_\_ \rfloor$ の定義および式 (3) と帰納法の仮定から，

$$\begin{aligned} g(\sigma\langle f_1 \rangle f_2 :: []) &= \sigma\texttt{[}\, g(f_1 :: f_2 :: []) \\ &= \sigma\texttt{[}\, \lfloor f_1 \rfloor \texttt{]}\, g(f_2 :: []) \\ &= \sigma\texttt{[}\, \lfloor f_1 \rfloor \texttt{]}\, \lfloor f_2 \rfloor \texttt{\$} \\ &= \lfloor \sigma\langle f_1 \rangle f_2 \rfloor \texttt{\$} \end{aligned}$$

が成り立つ．よって，任意の f について，式 (2) が成り立つ． ■

この補題の証明は，$g(f :: [])$ の計算の停止性も同時に示している．

さて，θ_S の後続森スタックによる拡張 Θ_S を考えよう．S を XML ストリーム処理器とし，f を，$\lfloor f \rfloor$ が S の入力となるような森とする．θ_S が暫定情報と残りの XML ストリームを受け取るのに対し，Θ_S は暫定情報と後続森スタックを受け取る．関数 Θ_S は同様に以下のように定義される．暫定情報 $e \in \mathcal{B}_S$ と後続森スタック $l \in FF_f$ に対し，

$$\begin{aligned} &\Theta_S(e, []) = e \\ &\Theta_S(e, l) = \Theta(\langle\!| e, \chi |\!\rangle, l') \quad \text{where } (\chi, l') = ud(l) \end{aligned}$$

とする．次の補題は，森 f に対し，XML ストリーム $\lfloor f \rfloor$ の τ_S による変換が，Θ_S に利用して計算できることを示している．

補題 4.8 $S = (Q^1, Q^2, \Sigma, \Delta, in, R)$ を XML ストリーム処理器とする．任意の森 $f \in \mathcal{F}_\Sigma$ に対し，

$$\tau_S(\lfloor f \rfloor) = \Theta_S(in(\varepsilon, \ldots, \varepsilon), f :: []) \tag{4}$$

が成り立つ．

証明 τ_S の定義から，

$$\theta_S(in(\varepsilon, \ldots, \varepsilon)), \lfloor f \rfloor \, \$) = \Theta_S(in(\varepsilon, \ldots, \varepsilon), f :: []). \tag{5}$$

を示せばよいが，ここではより一般的な等式を証明するために，補題 4.7 で与えられた関数 g を拡張した関数を G を導入する．

後続森スタック l に対し，$g(l)$ の計算途中で現れる g の引数となる後続森スタックの集合を $G(l)$ で表すことにする．すなわち，

$$G(l) = \begin{cases} \{l\} & \text{if } l = [] \\ \{l\} \cup G(l') & \text{otherwise} \end{cases} \tag{6}$$

ただし，$(\chi, L') = ud(L)$ とする．$G(f :: [])$ の計算も $g(f :: [])$ と同様に停止する．

本証明では，式 (5) を示す代わりに，等式

$$\theta_S(e, g(l)) = \Theta_S(e, l) \tag{7}$$

が，すべての暫定情報 $e \in Tmp_S$ と後続森スタック $l \in G(f :: [])$ に対して成り立つことを示す．補題 4.7 から，式 (5) は，この式における $e = in(\varepsilon, \ldots, \varepsilon)$ かつ $l = f :: []$ の場合に過ぎない．

式 (7) は，$G(l)$ の元の個数による帰納法により示される．G の定義から，最小の個数は 1 であり，$l = []$ の場合である．このとき，式 (7) の両辺は e となり，等号が成り立つ．$G(l)$ の元の個数が $n > 1$ の場合，$g(l) = \chi\, g(l')$ かつ $(\chi, l') = ud(l)$ なる l' があり，G の停止性から $l \notin G(l')$ であり，$G(l') = n - 1$ である．よって，帰納法の仮定および Θ_S の定義から，

$$\begin{aligned} \theta_S(e, g(l)) &= \theta_S(e, \chi\, g(l')) \\ &= \theta_S(\langle\!|e, \chi|\!\rangle, g(l')) \\ &= \Theta_S(\langle\!|e, \chi|\!\rangle, l') \\ &= \Theta_S(e, l) \end{aligned}$$

が成り立ち，式 (7) が示された． ■

次に，暫定情報を出力となる森に翻訳する関数 $\mathcal{I}$ を定義する．関数 $\mathcal{I}$ は，$\Theta_S(e, l)$ が $\Theta_S(e', l')$ によって計算されるならば $\mathcal{I}(e, l) = \mathcal{I}(e', l')$ が成り立つというよい性質を持っている．これは，補題 4.9 によって示される．関数 $\mathcal{I}$ は以下のように定義される．

$$\mathcal{I}(q[i](e_1, \ldots, e_n), l) = [\![q]\!](l.i, \mathcal{I}(e_1, l), \ldots, \mathcal{I}(e_n, l))$$
$$\mathcal{I}(\varepsilon) = \varepsilon, \qquad \mathcal{I}(\delta\,\texttt{[}\,e_1\,\texttt{]}\,e_2, l) = \delta\langle\mathcal{I}(e_1, l)\rangle\mathcal{I}(e_2, l)$$
$$\mathcal{I}(e_1 e_2, l) = \mathcal{I}(e_1, l)\,\mathcal{I}(e_2, l)$$

補題 4.9 $M = (Q, \Sigma, \Delta, in, R)$ を MFT，$\mathcal{SP}(M) = (Q', \Sigma, \Delta, in, R')$ を自動導出された XML ストリーム処理器，$f \in \mathcal{F}_\Sigma$ を M への入力，$l \in G(f :: [])$ とする．ここで，G は式 (6) で定義された関数とする．このとき，$(\chi, l') = ud(l)$ ならば，

$$\mathcal{I}(e, l) = \mathcal{I}(\langle\!|e, \chi|\!\rangle, l') \tag{8}$$

が成り立つ．

証明 e の構造に関する帰納法により示す．

$e = \varepsilon$ のとき，式 (8) の両辺はともに ε で成り立つ．$e = \delta\,\texttt{[}\,e_1\,\texttt{]}\,e_2$ のとき，$(\chi, l') = ud(l)$ とすると，$\mathcal{I}, \langle\!|\_\_|\!\rangle$ の定義および帰納法の仮定より，

$$\begin{aligned} \mathcal{I}(\delta\,\texttt{[}\,e_1\,\texttt{]}\,e_2, l) &= \delta\langle\mathcal{I}(e_1, l)\rangle\mathcal{I}(e_2, l) \\ &= \delta\langle\mathcal{I}(\langle\!|e_1, \chi|\!\rangle, l)\rangle\mathcal{I}(\langle\!|e_2, \chi|\!\rangle, l) \\ &= \mathcal{I}(\delta\,\texttt{[}\langle\!|e_1, \chi|\!\rangle\,\texttt{]}\,\langle\!|e_2, \chi|\!\rangle, l) \\ &= \mathcal{I}(\langle\!|\delta\,\texttt{[}\,e_1\,\texttt{]}\,e_2, \chi|\!\rangle, l) \end{aligned}$$

である．よって，式 (8) が成り立つ．$e = e_1 e_2$ のときも同様に示すことができる．

$e = q[0](e_1, \ldots, e_n)$ $(q \in Q)$ のとき，$\mathcal{I}$ の定義から，式 (8) の左辺は $[\![q]\!](l.0, \mathcal{I}(e_1, l), \ldots, \mathcal{I}(e_n, l))$ に等しい．ここで，$l.0$ に関して場合分けを行う．

$l.0 = \sigma\langle f_1\rangle f_2'$ すなわち $l = \sigma\langle f_1\rangle f_2 :: l''$ のとき，$l' = f_1 :: f_2 :: l''$ とすると $ud(l) = (\sigma\,\texttt{[}, l')$ である．よって，式 (8) の左辺は，$[\![\_\_]\!], ud$ の定義と帰納法の仮定から，

$$\begin{aligned} &[\![q]\!](l.0, \mathcal{I}(e_1, l), \ldots, \mathcal{I}(e_n, l)) \\ &= [\![q]\!](\sigma\langle f_1\rangle f_2, \mathcal{I}(e_1, l), \ldots, \mathcal{I}(e_n, l)) \\ &= [\![q]\!](\sigma\langle f_1\rangle f_2, \mathcal{I}(\langle\!|e_1, \sigma\,\texttt{[}|\!\rangle, l'), \ldots, \mathcal{I}(\langle\!|e_n, \sigma\,\texttt{[}|\!\rangle, l')) \\ &= [\![Rhs^{q,\sigma}]\!]_\rho \end{aligned}$$

ここで，$(q(\sigma\langle x_1\rangle x_2, y_1, \ldots, y_n) \to Rhs^{q,\sigma}) \in R$ であり，ρ は，$\rho(x_i) = f_i$ $(i = 1, 2)$，$\rho(y_j) = \mathcal{I}(\langle\!|e_j, \sigma\,\texttt{[}|\!\rangle, l')$ $(j = 1, \ldots, n)$ を満たす代入とする．一方，式 (8) の右辺は，

$$\mathcal{I}(\langle\!|q[0](e_1, \ldots, e_n), \sigma\,\texttt{[}|\!\rangle, l') = \mathcal{I}([\![\mathcal{A}(Rhs^{q,\sigma})]\!]_{\rho'}, l')$$

である．ここで，ρ' は，$\rho'(y_j) = \langle\!|e_j, \sigma[|\!\rangle$ $(j = 1, \ldots, n)$ である．今，$l'.0 = f_1$，$l'.1 = f_2$ および $\mathcal{I}, \mathcal{A}$ の定義を利用することにより，$Rhs^{q,\sigma}$ に関する帰納法により，

$$[\![Rhs^{q,\sigma}]\!]_\rho = \mathcal{I}([\![\mathcal{A}(Rhsq, \sigma)]\!]_{\rho'}, l')$$

を示すことができ，式 (8) が成り立つ．

また，$l.0 = \varepsilon$ のとき，すなわち $l = \varepsilon :: f :: l''$ または $L = \varepsilon :: []$ のときも同様に式 (8) を示すことができる． ■

以上の補題から，定理 4.6 を証明することができる．M を MFT とし，$S = \mathcal{SP}(M)$ を自動導出された XML ストリーム処理器とする．補題 4.9 から，$\Theta_S(e_0, e_0)$ に関する計算が

$$\Theta_S(e_0, e_0) = \Theta_S(e_1, l_1) = \cdots = \Theta_S(e_n, l_n) = e_n \tag{9}$$

（ここで $l_n = []$）より与えられるならば，すべての $k = 0, \ldots, n$ に対し $\mathcal{I}(e_k, l_k)$ は同じ値を持つ．$\mathcal{SP}(M)$ の $ に関するルールの右辺に状態を含む式がないこと，および $e_n = \langle\!|e_{n-1}, $|\!\rangle$ であることを考慮すると，$e_n \in \mathcal{S}_\Delta$ であることがわかる．よって，$\mathcal{I}, \lfloor\_\_\rfloor$ の定義から，$\lfloor \mathcal{I}(e_n, l_n) \rfloor = e_n$ が成り立つ．補題 4.9 および式 (9) から，

$$\begin{aligned} \lfloor \mathcal{I}(e_0, l_0) \rfloor &= e_n \\ &= \Theta_S(e_0, l_0) \end{aligned}$$

いま，$e_0 = in[0](\varepsilon, \ldots, \varepsilon)$ とし，M の入力となる森 f に対して $l_0 = f :: []$ とすると，

$$\begin{aligned} \lfloor \tau_M(f) \rfloor &= \lfloor [\![in]\!](f, \varepsilon, \ldots, \varepsilon) \rfloor \\ &= \lfloor \mathcal{I}(e_0, l_0) \rfloor \\ \tau_S(\lfloor f \rfloor) &= \Theta_S(e_0, l_0). \end{aligned}$$

したがって，定理 4.6 が証明される．

5 関連研究

XML ストリーム処理器の自動導出に関しては様々な研究がなされているが，主として XPath [1, 4, 7, 8] や XQuery [12] などのクエリ言語からの導出であり，XML 変換言語よりも表現力の低いものを対象にしている．クエリ言語以外では以下のような研究がある．

Kiselyov [10] は森の上の一般的な畳み込み関数 `foldts` を使うことのできる XML パーサを提案した．この関数は，種を親方向にどう蓄積するか，子方向にどう蓄積するか，葉ではどう蓄積するかなどに関する情報を引数として取るが，プログラミングはやや難しく，高階関数が多く現れることにより効率的な出力が可能かどうかは，その実装方法に強く依存する．実際，彼の実装による SSAX は巨大なファイルには対応できていない．

STX [3] は，XSLT のようなテンプレート型の言語で，XML イベントを自由に扱うことができる．しかしながら，どのタイミングで必要な情報をバッファリングするかは，すべてプログラマに委ねられている．

Scherzinger らによる TransformX [17] は，入力の型スキーマとなる正規木の各ノードに，以下の部分木を読み込む前と読み込んだ後の動作を java プログラムとして与えることにより，変換を可能にしている．しかしながら，この枠組でも STX と同様にバッファリングをプログラマが意識する必要がある．

児玉ら [11] は，線形順序型を用いてストリーム処理器導出の可能性をチェックする機構を提案した．彼らの提案した言語では，関数型プログラムとして木の変換を自由に定義することができるが，終了タグを見ずにパースできる二分木を対象としているため，XML に応用するにはまだ困難が多い．

6 おわりに

本稿では，macro forest transducer（MFT）から XML ストリーム処理器を自動導出するアルゴリズムを提案した．MFT では，XML のような森の構造に対する再帰関数によって変換を定義するため，自然な形で XML 変換を表すプログラムを書くことができる．また，複数の関数による相互再帰や蓄積変数を利用することもできるため，より柔軟な形のプログラミングも可能である．本稿では，MFT の強い制限から，蓄積変数やその操作について禁じており，整数や論理値および条件分岐などの操作については導入していないが，出力の文字集合の一部としてこれらを加えることにより，同様のストリーム処理器の導出が可能である．これらの基本型やそれを操作する関数を用意すれば，XSLT や XDuce，CDuce といった既存の XML 変換言語へ適用することも十分可能である．なお，本枠組は，著者による XML 変換言語 XTiSP [14, 13] の次期バージョンに実装される予定である．

参考文献

[1] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. *International Journal on Very Large Data Bases*, pages 53–64, 2000.

[2] V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. In *Proceedings of the 8th International Conference of Functional Programming*, pages 51–63, 2003.

[3] P. Cimprich, O. Becker, C. Nentwich, M. K. H. Jiroušek, P. Brown, M. Batsis, T. Kaiser, P. Hlavnička, N. Matsakis, C. Dolph, and N. Wiechmann. Streaming transformations for XML (STX) version 1.0. `http://stx.sourceforge.net/documents/`.

[4] Y. Diao and M. J. Franklin. High-performance XML filtering: An overview of YFilter. In *IEEE Data Engineering Bulletin*, volume 26(1), pages 41–48, 2003.

[5] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985.

[6] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[7] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Transactions on Database Systems*, 29(4):752–788, 2004.

[8] A. K. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 419–430, 2003.

[9] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[10] O. Kiselyov. A better XML parser through functional programming. In *4th International Symposium on Practical Aspects of Declarative Languages*, volume 2257 of *Lecture Notes in Computer Science*, pages 209–224, 2002.

[11] K. Kodama, K. Suenaga, N. Kobayashi, and A. Yonezawa. Translation of tree-processing programs into stream-processing programs based on ordered linear type. In *The 2nd ASIAN Symposium on Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 41–56, 2004.

[12] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A transducer-based XML query processor. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 227–238, 2002.

[13] K. Nakano. XTiSP: XML transformation language intended for stream processing. `http://xtisp.psdlab.org/`.

[14] K. Nakano. An implementation scheme for XML transformation lauguages through derivation of stream processors. In *The 2nd ASIAN Symposium on Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 74–90, 2004.

[15] S. Nishimura and K. Nakano. XML stream transformer generation through program composition and dependency analysis. *Science of Computer Programming*, 54:257–290, 2005.

[16] T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89:141–149, 2004.

[17] S. Scherzinger and A. Kemper. Syntax-directed transformations of XML streams. In *The workshop on Programming Language Technologies for XML*, pages 75–86, 2005.

[18] SAX: the simple api for XML. `http://www.saxproject.org/`.

[19] XSL transformations (XSLT). `http://www.w3c.org/TR/xslt/`.