情報処理学会論文誌　プログラミング　Vol.8　No.3　1–10 (Sep. 2015)

Streaming String Transducerの等価性判定と正規表現による文字列置換への応用

加賀江 優幸[1,a)]　南出 靖彦[2,b)]

受付日 2015年2月10日, 採録日 2015年5月25日

概要：文字列から新たな文字列を生成する計算モデルとして Alur と Černý によるストリーミング文字列トランスデューサ（Streaming String Transducer: SST）がある．SST は，文字を受け取るごとに文字列を保存できる変数の内容を更新し，最終的には変数の内容を用いて出力文字列を構成するトランスデューサである．本研究では，関数的非決定性ストリーミング文字列トランスデューサ（関数的 SST）の等価性判定を正規表現による文字列置換の等価性判定に応用する．ここで，関数的とは非決定性であっても出力がたかだか 1 つであることを示す．関数的 SST の等価性判定問題の決定可能性は Alur らによって証明されている．しかしながら，判定方法は複雑であり直感的な理解が困難であった．そこで，本論文では関数的 SST の等価性判定アルゴリズムの簡略化について述べる．また，本アルゴリズムの実装および実験結果についても述べる．正規表現による文字列置換はグループ変数にマッチした箇所を複数回出力することができることから有限状態トランスデューサでは模倣できない．そこで，本論文では正規表現による文字列置換を関数的 SST で模倣することにより，関数的 SST 上の検証問題に帰着させる．最終的には，関数的 SST の等価性判定アルゴリズムを正規表現による文字列置換の等価性判定に応用する．

キーワード：正規表現，ストリーミング文字列トランスデューサ，等価性判定

Equivalence Checking of Streaming String Transducers and Its Application to Regular Expression Replacement

Masayuki Kagae[1,a)]　Yasuhiko Minamide[2,b)]

Received: February 10, 2015, Accepted: May 25, 2015

Abstract: A streaming string transducer (SST) proposed by Alur and Černý is a powerful computation model from strings to strings. A SST equips with several variables that store strings and each transition updates the contents of the variables. We apply functional nondeterministic streaming string transducers (Functional SST) to model regular expression replacement in programming languages. Alur et al. showed the decidability of equivalence checking of Functional SST. However, they did not show the details of their algorithm and no implementation and experimental results are reported, as long as we know. In this paper, we reformulate and describe the details of equivalence checking of Functional SST. We also report our implementation and experimental results. Regular expression replacement cannot be precisely modeled by a finite state transducer. It is because it may output a string matched with a group variable more than once. We construct Functional SST that simulates regular expression replacement and apply equivalence checking of Functional SST to equivalence checking of regular expression replacement.

Keywords: regular expression, streaming string transducer, equivalence checking

[1] 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[a)] kagae@score.cs.tsukuba.ac.jp
[b)] minamide@cs.tsukuba.ac.jp

1. はじめに

文字列から新たな文字列を生成する計算モデルとして Alur と Černý によるストリーミング文字列トランスデューサ（Streaming String Transducer: SST）がある [3]．SST は，文字を受け取るごとに文字列を保存できる変数の内容

を更新し，最終的には変数の内容を用いて出力文字列を構成するトランスデューサである．SST は，有限状態トランスデューサよりも表現能力が高く，2 方向有限状態トランスデューサや MSO 文字列トランスデューサと同等の表現能力を持つ [1]．この SST において，関数的非決定性ストリーミング文字列トランスデューサ（Functional SST）の等価性判定問題は決定可能であることが Alur らにより証明されている [2], [3]．

また，有限状態トランスデューサでは表現できないが SST で表現できる例として正規表現による文字列置換があげられる．正規表現による文字列置換は，書き換え対象の文字列を正規表現を用いて検索し，マッチした箇所を置換文字列で書き換える文字列処理である．たとえば，PHP では以下のようなプログラムによって正規表現による文字列置換を用いた文字列操作を行うことができる．

```
preg_replace("!<([a-z]+)/>!", "<$1></$1>", $w)
```

このプログラムは，変数`$w` に格納されている文字列を正規表現`!<([a-z]+)/>!`[*1]で検索し，マッチした箇所を`<$1></$1>`の形に書き換える．`$1` はキャプチャ変数と呼ばれ，正規表現の `([a-z]+)` にマッチした部分文字列を参照することができる．これにより，たとえば文字列中の `<br/>`などのタグは `<br></br>`に書き換えることができる．

正規表現による文字列置換の解析に関する先行研究として，佐久間らによる正規表現マッチングの意味付けおよび正規表現マッチングを行う有限状態トランスデューサの構築がある [10]．しかしながら，正規表現による文字列置換には正規表現の一部にマッチした文字列を変数に格納しておくキャプチャ構文が存在し，正規表現の一部にマッチした文字列の複数回挿入や順番の入れ替えといったことが可能であるため，有限状態トランスデューサでの解析には限界がある．

本研究では，関数的 SST の等価性判定アルゴリズムを応用して正規表現による文字列置換の等価性判定を行う．そのために，関数的 SST の等価性判定アルゴリズムの簡略化を行い実装する．SST をモノイドの要素を出力するトランスデューサと見る視点を導入することにより，出力文字列の情報を監視する計算モデルを準同型写像を用いた変換で容易に構成できる．さらに，正規表現による文字列置換を模倣する関数的 SST の構築を行い，正規表現による文字列置換への適用を試みる．

まず 2 章で本論文で使用する計算モデルの導入を行う．3 章では関数的 SST の等価性判定アルゴリズムとその簡略化について述べる．4 章では正規表現による文字列置換を模倣する関数的 SST の構成方法について述べる．5 章および 6 章で正規表現による文字列置換の等価性判定の実装とその実験結果について述べる．

[*1] 正規表現の区切り文字として通常は/で囲んで表すが，PHP では他の文字を区切り文字として使用でき，この例では!を用いている．

2. 準備

2.1 モノイドを出力するトランスデューサ

モノイドを出力するトランスデューサは，入力文字を受け取るごとに状態遷移すると同時にモノイドの要素を出力するトランスデューサである．入力アルファベット Σ，出力モノイド $\langle M, 1_M, \cdot\rangle$ 上のトランスデューサの集合は $\mathcal{T}_{\Sigma,M}$ で表し，$\langle Q, Q_0, F, \Delta\rangle$ の 4 個組で定義される．ここで，Q は状態の空でない有限集合，$Q_0 \subseteq Q$ は初期状態の有限集合，$F \subseteq Q$ は最終状態の有限集合，$\Delta \subseteq Q \times \Sigma \times M \times Q$ は遷移規則である．ただし，遷移規則 Δ は有限集合とする．

遷移は $q \xrightarrow[T]{w/m} q'$ の形式で書き，状態 q において入力文字列 w が入力された場合にモノイド M の要素 m を出力して状態 q' に遷移することを表す．入力文字列 $w = \epsilon$ である場合は，任意の状態 q において $q \xrightarrow[T]{\epsilon/1_M} q$ となる．入力文字列 $w = w'\sigma$ である場合は，遷移 $q \xrightarrow[T]{w'/m_1} q''$ かつ遷移規則 $(q'', \sigma, m_2, q') \in \Delta$ ならば $q \xrightarrow[T]{w'\sigma/m_1\cdot m_2} q'$ となる．

以下，本論文ではモノイドを出力するトランスデューサを単にトランスデューサということとする．トランスデューサは，準同型写像を用いて出力するモノイドを変換することができる．2 つのモノイド $\langle M, 1_M, \cdot_M\rangle, \langle M', 1_{M'}, \cdot_{M'}\rangle$ において準同型写像 $h : M \to M'$ がある場合，M を出力するトランスデューサ T に準同型写像 h を適用することにより M' を出力するトランスデューサ T' を構成することができる．新たなトランスディーサ T' は，T に対して遷移規則 Δ を $\Delta_{T'} = \{(q, \sigma, h(m), q') \mid (q, \sigma, m, q') \in \Delta\}$ に変更することにより実現できる．

補題 2.1　M を出力するトランスデューサ T に対して準同型写像 $h : M \to M'$ を用いて M' を出力するトランスデューサ T' を構築した場合，以下の性質が成り立つ．

$$q \xrightarrow[T]{w/\alpha} q' \Rightarrow q \xrightarrow[T']{w/h(\alpha)} q'$$
$$q \xrightarrow[T']{w/\alpha'} q' \Rightarrow \exists\alpha.q \xrightarrow[T]{w/\alpha} q' \wedge h(\alpha) = \alpha'$$

2.2 1 カウンタオートマトン

1 カウンタオートマトン（1 Counter Automaton: 1CA）は，1 個のカウンタを持つ有限状態オートマトンである．入力アルファベット Σ 上の 1 カウンタオートマトン A は，整数モノイド $\langle \mathbb{Z}, 0, +\rangle$ を出力するトランスデューサとして定義できる．

また，入力を持たない 1 カウンタオートマトンとして 1 カウンタ機械（1 Counter Machine: 1CM）がある．1CM は，4 個組 $\langle Q, Q_0, F, \Delta\rangle$ で与えられ，遷移規則の集合は

$\Delta \subseteq Q \times \mathbb{Z} \times Q$ となる．1CM $M = \langle Q, Q_0, F, \Delta \rangle$ が与えられたとき，ある状態 $q_0 \in Q_0, q_f \in F$ が存在し，

$$q_0 \xrightarrow[M]{0} q_f$$

となる遷移が存在するとき，0 到達可能であるという．また，ある $n \neq 0$ に対して，

$$q_0 \xrightarrow[M]{n} q_f$$

となる遷移が存在するとき，非 0 到達可能であるという．0 到達可能性判定問題および非 0 到達可能性判定問題は決定可能であり，NLOGSPACE で解くことができる [3].

3. 関数的 SST の等価性判定とその簡略化

Alur と Černý によって決定性 SST の等価性判定が決定可能であることが証明されている [3]．また，Alur と Deshmukh は，Alur と Černý の証明が容易に関数的 SST にも適用できることを示している [2]．既存研究における関数的 SST S_1, S_2 の等価性判定は，以下の等価でない 3 パターンの可能性を判定してどれも起こりえない場合に等価であると判定する．

1) S_1 と S_2 の定義域が異なる．
2) ある w に対して $S_1(w)$ と $S_2(w)$ の長さが異なる．
3) ある w に対して $S_1(w)$ と $S_2(w)$ のある p 文字目が異なる．

本研究では，2) 出力文字列長の非一致可能性判定および 3) 異なる文字の同位置への出現可能性判定の簡略化を行った．Alur と Černý による証明では，3) を行うために 2 つの SST S_1, S_2 から以下の性質を満たす 1CM M を構成している．

- S_1 と S_2 が 3) を満たす $\Leftrightarrow$ M が最終状態に 0 到達可能

しかしながら，1CM M の構成方法は複雑であり直感的な理解が困難であった．また，2) についても 3) よりも容易に行うことができるとされているが，判定方法の詳細は示されていなかった．本研究では，2) の判定方法を構築し，3) の判定では 2) の判定を利用する方法で簡略化を行う．SST をモノイドを出力するトランスデューサと見ることによって，準同型写像を用いて出力文字列長に関係する情報のみを抽出することができる．抽出した情報を元に出力文字列長をカウントする 1CA を構成することができ，2 つの 1CA のカウンタの差をカウントする 1CM を構成する．これにより構成された 1CM は 2 つの SST S_1，S_2 に同じ文字列を与えた場合の出力文字列長の差をカウントする 1CM となっている．この 1CM に対して非 0 到達可能性判定を行うことによって 2) を判定することができる．3) は，特定の文字の出現位置までを非決定的に出力する SST を構成することにより，出力文字列長の一致可能性判定に帰着する方法で判定する．

本章ではまずストリーミング文字列トランスデューサの定義を行う．その後に，簡略化後の出力文字列長の一致可能性判定について説明し，簡略化後の異なる文字の同位置への出現可能性判定について述べる．

3.1 ストリーミング文字列トランスデューサ

入力アルファベット Σ，出力アルファベット Γ 上のストリーミング文字列トランスデューサ（Streaming String Transducer: SST）S は，6 個組 $\langle Q, q_0, X, x_{out}, F, \Delta \rangle$ である．ここで，Q は状態の空でない有限集合，$q_0 \in Q$ は初期状態，X は変数の有限集合，$x_{out} \in X$ は出力する変数，$F \subseteq Q$ は最終状態の有限集合，$\Delta \subseteq Q \times \Sigma \times M_{X,\Gamma} \times Q$ は遷移規則である．$M_{X,\Gamma} \subset X \to (X \cup \Gamma)^*$ は変数環境の更新であり，各変数に対して新たに格納する文字列を変数と出力文字の列で表す．ただし，変数環境の更新 $\alpha \in M_{X,\Gamma}$ は，以下の条件を満たすものに制限される．

> $\{\alpha(y) \mid y \in X\}$ 中に任意の変数 $x \in X$ はたかだか 1 回しか出現しない．

すなわち，$M_{X,\Gamma}$ は以下のように定義される．

$$M_{X,\Gamma} = \{\alpha \in X \to (X \cup \Gamma)^* \mid \forall x \in X. \sum_{y \in X} \#_x(\alpha(y)) \leq 1\}$$

ここで，$\#_x : (X \cup \Gamma)^* \to \mathbb{N}$ は与えられた文字列に対して，文字 x が出現する回数を返す準同型写像である．

ストリーミング文字列トランスデューサは，出力文字列の定義を除くとモノイド $(M_{X,\Gamma}, id, \bullet)$ を出力するトランスデューサと見ることができる．$id \in X \to (X \cup \Gamma)^*$ は $\forall x. id(x) = x$ を満たす関数であり，$\bullet$ は $M_{X,\Gamma}$ 上の合成で次のように定義される．

$$\alpha_1 \bullet \alpha_2 = \hat{\alpha_1} \circ \alpha_2$$

ここで，$\circ$ は関数の合成であり，$\hat{\alpha} : (X \times \Gamma)^* \to (X \times \Gamma)^*$ は文字列中の変数 x を $\alpha(x)$ で書き換える準同型写像である．

SST S の意味 $[\![S]\!] : \Sigma^* \to 2^{\Gamma^*}$ を以下のように定義する．ここで，$\pi : (X \cup \Gamma)^* \to \Gamma^*$ は文字列中の変数を空文字列に書き換える準同型写像とする．

$$[\![S]\!](w) = \{\pi(\alpha(x_{out})) \mid q_0 \xrightarrow[S]{w/\alpha} q \wedge q \in F\}$$

また，任意の入力文字列に対して出力文字列がたかだか 1 つである場合は関数的（Functional）であるといい，関数的 SST の意味は以下の性質を満たす部分関数 $S : \Sigma^* \to \Gamma^*$ で表すこととする．

$$S(w) = w' \Leftrightarrow [\![S]\!](w) = \{w'\}$$

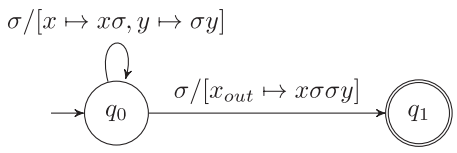
変数環境の更新のための記法として，$[x_1 \mapsto w_1, \ldots, x_n \mapsto w_n]$ と $\alpha[x \mapsto w]$ を以下のように定義する．

$$[x_1 \mapsto w_1, \ldots, x_n \mapsto w_n](y) = \begin{cases} w_i & \text{if } y = x_i \\ \epsilon & \text{otherwise} \end{cases}$$

$$\alpha[x \mapsto w](y) = \begin{cases} w & \text{if } y = x \\ \alpha(y) & \text{otherwise} \end{cases}$$

ただし，$x, x_i \in X$, $w, w_i \in X \cup \Gamma$, $\alpha \in M_{X,\Gamma}$ とする．

例 3.1　文字列 $w(\neq \epsilon)$ を与えられた際に ww^R を出力する関数的 SST を以下のように定義することができる．



ただし，$\sigma \in \Sigma$ は任意の文字である．

文字列 abb が入力された場合は，以下のような手順で出力文字列が決定される．

(1) $w = \epsilon$ での遷移により，$q_0 \xrightarrow[S]{\epsilon/id} q_0$.

(2) q_0 から q_0 への遷移を用いて，$q_0 \xrightarrow[S]{a/[x\mapsto xa, y\mapsto ay]} q_0$.

(3) q_0 から q_0 への遷移を用いて，$q_0 \xrightarrow[S]{ab/[x\mapsto xab, y\mapsto bay]} q_0$.

(4) q_0 から q_1 への遷移を用いて，$q_0 \xrightarrow[S]{abb/[x_{out}\mapsto xabbbbay]} q_1$.

(5) x_{out} の内容から変数を除いた結果，出力文字列 $abbbba$ が出力される．

3.2　SST の文字列長の一致可能性判定

SST の文字列長の一致可能性判定は以下の流れで行う．

(1) SST から不要な情報を除去したトランスデューサを構築

(2) 文字列長をカウントする 1 カウンタオートマトンを構築

(3) 文字列長の差をカウントする 1 カウンタ機械を構築

(4) 0 到達可能性判定問題へ帰着

3.2.1　不要な情報を除去したトランスデューサの構築

SST が出力するモノイド $(M_{X,\Gamma}, id, \bullet)$ から不要な情報を除去する．例として変数の更新 $[x \mapsto abxcy, y \mapsto \epsilon]$ を考える．このとき，出力文字列の長さのみを考えると，変数 x が $abxcy$ に更新されるという情報すべては必要ない．変数 x に挿入される文字数 (a, b, c の 3 文字) と挿入される変数の集合 $\{x, y\}$ の情報があれば十分である．このような必要な情報を準同型写像 $h : (X \to (X \cup \Gamma)^*) \to ((X \to 2^X) \times (X \to \mathbb{N}))$ を用いて抽出する．準同型写像 h は，$\alpha \in M_{X,\Gamma}$ に対して $\alpha(x) = w$ ならば $\beta(x) =$ (w 中の変数の集合)，$\gamma(x) =$ (w 中の出力文字数) となる (β, γ) を生成する準同型写像である．厳密には準同型写像 h は，準同型写像 $h_\beta : (X \cup \Gamma)^* \to 2^X$ と準同型写像 $h_\gamma : (X \cup \Gamma)^* \to \mathbb{N}$ を用いて $h(\alpha) = (\lambda x.h_\beta(\alpha(x)), \lambda x.h_\gamma(\alpha(x)))$ と定義される．h_β は変数 x に対して $h_\beta(x) = \{x\}$，文字 σ に対して $h_\beta(\sigma) = \emptyset$ と定義される準同型写像であり，h_γ は変数 x に対して $h_\gamma(x) = 0$，文字 σ に対して $h_\gamma(\sigma) = 1$ と定義される準同型写像である．

出力されるモノイドは $(P_X, (\lambda x.\{x\}, \lambda x.0), \bullet)$ となる．ここで，$P_X \subset (X \to 2^X) \times (X \to \mathbb{N})$ と演算子 $\bullet$ は以下のように定義される．

$$P_X = \{(\beta, \gamma) \mid \forall x \in X. \sum_{y \in X} \#_x(\beta(y)) \leq 1\}$$

$$(\beta_1, \gamma_1) \bullet (\beta_2, \gamma_2) = \lambda x. \left(\bigcup_{y \in \beta_2(x)} \beta_1(y), \sum_{y \in \beta_2(x)} \gamma_1(y) + \gamma_2(x) \right)$$

3.1 節で述べたように，SST S はモノイド $(M_{X,\Gamma}, id, \bullet)$ を出力するトランスデューサと見ることができる．また，2.1 節で述べたとおり，準同型写像 h を用いてモノイド $(P_X, (\lambda x.\{x\}, \lambda x.0), \bullet)$ を出力する新たなトランスデューサを構築することができる．すなわち，補題 2.1 を SST と準同型写像 h に特化することで以下の補題を得る．

補題 3.1　*SST S に対して準同型写像 $h : (X \to (X \cup \Gamma)^*) \to ((X \to 2^X) \times (X \to \mathbb{N}))$ を用いて P_X を出力するトランスデューサ T を構築した場合，以下の性質が成り立つ．*

$$q \xrightarrow[S]{w/\alpha} q' \Rightarrow q \xrightarrow[T]{w/h(\alpha)} q'$$

$$q \xrightarrow[T]{w/(\beta,\gamma)} q' \Rightarrow \exists \alpha. q \xrightarrow[S]{w/\alpha} q' \wedge h(\alpha) = (\beta, \gamma)$$

3.2.2　出力文字列長をカウントする 1CA の構築

出力文字列長をカウントする 1CA を構成する．モノイド $(P_X, (\lambda x.\{x\}, \lambda x.0), \bullet)$ を出力するトランスデューサ $\langle Q, Q_0, F, \Delta \rangle$ に対して，1CA $A = \langle Q_A, Q_0^A, F_A, \Delta_A \rangle$ を定義する．各構成要素は以下のように定義される．

$$Q_A = Q \times 2^X$$

$$Q_0^A = \{(q_0, B_0) \mid q_0 \in Q_0 \wedge B_0 \subseteq X\}$$

$$F_A = \{(q_f, \{x_{out}\}) \mid q_f \in F\}$$

$$\Delta_A = \{((q, \beta(B')), \sigma, \gamma(B'), (q', B')) \mid B' \subseteq X \wedge (q, \sigma, (\beta, \gamma), q') \in \Delta\}$$

β および γ は変数を入力とする関数であるが，$\beta(B) = \bigcup_{x \in B} \beta(x)$, $\gamma(B) = \sum_{x \in B} \gamma(x)$ とすることにより変数の集合に対して拡張することができる．状態中に付加される変数の集合は，SST の出力に含まれる文字列を格納している変数の集合である．1CA A は出力に使用される変数の集合を非決定的に選択し，各変数に格納されている文字数の和をカウントする．SST では最終状態 $q_f \in F$ に到達したときに x_{out} を出力して終了するため，1CA A の最終状態中の変数集合は x_{out} のみとなる．このように構成することにより，出力文字数をカウントする 1CA を構成することができる．以下の補題は，遷移の長さに関する帰納法によって容易に証明できる．

補題 3.2　*P_X を出力するトランスデューサ $T = \langle Q, Q_0, F, \Delta \rangle$ と出力文字数をカウントする 1CA $A = \langle Q_A, Q_0^A, F_A, \Delta_A \rangle$ との間に以下の性質が成り立つ．*

- $q_0 \xrightarrow[T]{w/(\beta,\gamma)} q$ ならば，任意の変数の集合 $B \subseteq X$ において $(q_0, \beta(B)) \xrightarrow[A]{w/\gamma(B)} (q, B)$ が成り立つ．
- $(q_0, B_0) \xrightarrow[A]{w/n} (q, B)$ ならば，$\beta(B) = B_0$ および $\gamma(B) = n$ となる β, γ において $q_0 \xrightarrow[T]{w/(\beta,\gamma)} q$ が成り立つ．

3.2.3 出力文字列長の差をカウントする 1CM の構築

2 つの SST S_1, S_2 の出力文字列長の差をカウントする 1CM を構成する．1CA $A_1 = \langle Q_{A_1}, Q_0^{A_1}, F_{A_1}, \Delta_{A_1} \rangle$ および 1CA $A_2 = \langle Q_{A_2}, Q_0^{A_2}, F_{A_2}, \Delta_{A_2} \rangle$ に対して，1CM $M = \langle Q_M, Q_0^M, F_M, \Delta_M \rangle$ を定義する．ここで，$Q_M = Q_{A_1} \times Q_{A_2}$ を状態の有限集合，$Q_0^M = Q_0^{A_1} \times Q_0^{A_2}$ を初期状態の有限集合，$F_M = F_{A_1} \times F_{A_2}$ を最終状態の有限集合とする．遷移規則 Δ_M は以下のように定義する．

$$\Delta_M = \{((q_{A_1}, q_{A_2}), n_1 - n_2, (q'_{A_1}, q'_{A_2})) \mid \exists \sigma \in \Sigma.\ (q_{A_1}, \sigma, n_1, q'_{A_1}) \in \Delta_{A_1} \wedge (q_{A_2}, \sigma, n_2, q'_{A_2}) \in \Delta_{A_2}\}$$

このとき，補題 3.1，3.2 から次の定理が成立つ．

定理 3.1　*2 つの SST S_1, S_2 の出力文字列長の差をカウントする 1CM M は以下の性質を満たす．*

$$\exists w, w_1, w_2.(w_1 \in [\![S_1]\!](w) \wedge w_2 \in [\![S_2]\!](w) \wedge |w_1| - |w_2| = n)$$
$$\Updownarrow$$
$$\exists q_0^M \in Q_0^M, q_f^M \in F_M . q_0^M \xrightarrow[M]{n} q_f^M$$

これにより，2 つの SST S_1，S_2 の文字列長の一致可能性判定は 1CM M の 0 到達可能性判定に帰着される．

3.3 異なる文字の同位置への出現可能性判定の簡略化

関数的 SST S，S' に対する異なる文字の同位置への出現可能性判定は以下の流れで行う．

(1) 異なる出力文字 $a, b \in \Gamma$ を選択
(2) 関数的 SST S に対して a の出現位置までを非決定的に出力する SST S_a を構成し，関数的 SST S' に対しても同様に SST S'_b を構成する．
(3) S_a と S'_b に対して文字列長の一致可能性判定を行う．文字列長が一致する可能性がある場合には同位置への異なる文字の出現可能性があることになる．
(4) (1) から (3) をすべての異なる文字の組み合わせを網羅するように繰り返す．

この方法中での関数的 SST S に対して a の出現位置までを非決定的に出力する SST S_a の構成方法を示す．遷移中に出力文字が変数に挿入される時点で非決定的に末尾にするかを決定し，状態中に末尾にする変数を保持し続けることにより実現する．関数的 SST $S = \langle Q, q_0, X, x_{out}, F, \Delta \rangle$ に対して，SST $S_a = \langle Q', q'_0, X, x_{out}, F', \Delta' \rangle$ を定義する．ここで，$Q' = Q \times (X \cup \{\bot\})$ を状態の有限集合，$q'_0 = (q_0, \bot)$ を初期状態，$F' = \{(q_f, x_{out}) \mid q_f \in F\}$ を最終状態の有限集合とする．遷移規則 Δ' は以下の 3 種類の遷移規則の和集合とする．

- 末尾にする文字を決定する前の遷移

$$\{((q, \bot), \sigma, \alpha, (q', \bot)) \mid (q, \sigma, \alpha, q') \in \Delta\}$$

- 新たに挿入される文字を末尾にする遷移

$$\{((q, \bot), \sigma, \alpha[x \mapsto w_L a], (q', x)) \mid (q, \sigma, \alpha, q') \in \Delta \wedge \alpha(x) = w_L a w_R\}$$

- すでに変数に格納されている文字列を末尾にする遷移

$$\{((q, x), \sigma, \alpha[x' \mapsto w_L x], (q', x')) \mid (q, \sigma, \alpha, q') \in \Delta \wedge \alpha(x') = w_L x w_R\}$$

補題 3.3　上記により定義された *SST S_a* は以下の性質を満たす．

$$[\![S_a]\!](w) = \{w_L a \mid S(w) = w_L a w_R\}$$

上の補題は，S と S_a の間で成り立つ以下の性質を示すことにより証明することができる．

- $(q_0, \bot) \xrightarrow[S_a]{w/\alpha} (q, \bot) \Leftrightarrow q_0 \xrightarrow[S]{w/\alpha} q$.
- $q_0 \xrightarrow[S]{w/\alpha} q$ かつ $\alpha(x) = w_L a w_R$ ならば，$(q_0, \bot) \xrightarrow[S_a]{w/\alpha[x \mapsto w_L a]} (q, x)$.
- $(q_0, \bot) \xrightarrow[S_a]{w/\alpha} (q, x)$ ならば，ある w_L, w_R に対して，$\alpha(x) = w_L a$ かつ $q_0 \xrightarrow[S]{w/\alpha[x \mapsto w_L a w_R]} q$.

これにより，SST S_a の出力文字列長は関数的 SST S の出力における a の出現位置に一致する．よって，SST S_a と SST S'_b に対して出力文字列長の一致可能性判定を行うことにより，関数的 SST S における a の出力位置と関数的 SST S' における b の出力位置が一致する可能性を判定することができる．

定理 3.2　関数的 *SST* の同位置への異なる文字の出現可能性判定問題は，出力文字列長の一致可能性判定問題に帰着可能である．

4. 正規表現による文字列置換の等価性判定への応用

本論文では，関数的 SST の等価性判定の決定可能性を応用して正規表現による文字列置換の等価性判定アルゴリズムの構築を行う．そのために，正規表現による文字列置換を模倣する関数的 SST の構築を行う．

4.1 正規表現マッチングを行うトランスデューサ

本論文で扱う正規表現は Perl 互換のものとする．まず，正規表現の構文を以下のように定義する．

$$
\begin{array}{rcll}
r & ::= & \epsilon & \text{(空文字列)} \\
 & | & c & \text{(文字)} \\
 & | & r_1 r_2 & \text{(連接)} \\
 & | & r_1 | r_2 & \text{(選択)} \\
 & | & r_1^* & \text{(貪欲な反復)} \\
 & | & r_1^{*?} & \text{(非貪欲な反復)} \\
 & | & (r_1)_y & \text{(キャプチャ)}
\end{array}
$$

キャプチャ構文 $(r_1)_y$ は，この構文にマッチした部分文字列を対応するキャプチャ変数 y に格納する構文である．キャプチャ変数 y に格納された文字列は，正規表現による文字列置換や言語によってはプログラム中で使用することができる．正規表現 r におけるキャプチャ変数の集合を Y_r とする．ここで，各キャプチャ変数は正規表現中に 1 度しか現れないとする．

正規表現マッチングを行う際にマッチングの優先度が重要となる．選択の構文 $r_1|r_2$ においては，r_1 の方が r_2 よりも優先順位が高い．そのため，r_1 と r_2 の両方にマッチする場合には r_1 にマッチしたとみなされる．また，反復には r_1^* と $r_1^{*?}$ があるが，それぞれ貪欲な反復と非貪欲な反復である．r_1^* は $r_1 r_1^* | \epsilon$ と，$r_1^{*?}$ は $\epsilon | r_1 r_1^{*?}$ と解釈される．

正規表現マッチングを考えるにあたり，対象とする正規表現 r は $\epsilon \notin L(r)$ かつ部分正規表現として $\epsilon \in L(r_1)$ となる r_1^* や $r_1^{*?}$ を含まないものとする．このような正規表現によるマッチングは，バックトラックに基づく実装において停止性を保証するための工夫を必要とし実装によって異なる振る舞いをする．そのため，佐久間らによる正規表現によるマッチングの意味論の定義もより複雑なものとなっている．このような正規表現による文字列置換は実際に使われることは稀であるため，本論文では除外して議論することとする．

正規表現による文字列置換を行うために，まず与えられた文字列のどの箇所に正規表現がマッチするかをチェックする正規表現マッチングを行う必要がある．正規表現 r のマッチングを行う先読み付きトランスデューサ T_r が佐久間らによって与えられている [10]．正規表現マッチングは文字列の先頭から正規表現で繰り返しマッチングを行い，マッチした箇所の情報を取得する文字列検索である．T_r では，正規表現マッチングによりマッチした箇所に (と) の記号を付加し，キャプチャ変数 $y \in Y$ のキャプチャ構文にマッチした箇所に $(_y$ と $)_y$ を付加することによってマッチング情報の付加を行う．入力アルファベットを Σ とすると，出力アルファベットは $\Sigma_r = \Sigma \cup \{(_y \mid y \in Y_r\} \cup \{)_y \mid y \in Y_r\} \cup \{(,)\}$ となる．T_r は，検索対象の文字列 $w \in \Sigma^*$ を受け取りマッチング情報を付加した文字列 $w' \in \Sigma_r^*$ を出力する．たとえば，以下のように動作する．

例 4.1　文字列 $abaaab$ に対して，正規表現 $r_1 = aa|a$ でマッチングを行った場合．

$$T_{r_1}(abaaab) = (a)b(aa)(a)b$$

文字列 $abccda$ に対して，正規表現 $r_2 = a(b^*)_y(c^*)_z$ でマッチングを行った場合．

$$T_{r_2}(abccda) = (a(_y b)_y(_z cc)_z)d(a(_y)_y(_z)_z)$$

文字列 $abac$ に対して，正規表現 $r_3 = ((a|b)_y)^*$ でマッチングを行った場合．

$$T_{r_3}(abac) = ((_y a)_y(_y b)_y(_y a)_y)c$$

正規表現 r による文字列置換の動作は，T_r を適用することによって得られるマッチング情報を付加した文字列に対しては，容易に定義することができる．厳密な定義は煩雑になるためここでは省略するが，正規表現マッチング中にキャプチャ構文 $(r)_y$ が複数回マッチした場合の動作は，明確にしておく必要がある．そのような場合は，変数 y には最後にマッチした文字列が保存される．たとえば，文字列 $ababcabcb$ に対して，正規表現 $a((b|c)_y)^*$ でマッチングを行い，マッチング箇所を置換文字列 y に書き換えると文字列 bcb が得られる．

4.2　正規表現による文字列置換を模倣する関数的 SST

マッチング箇所を置換文字列 v に書き換える決定性 SST を定義し，前述の繰返しマッチングを行う関数的非決定性トランスデューサと合成することにより正規表現による文字列置換を模倣する関数的 SST を構成する．

マッチング箇所を置換文字列 v に書き換える決定性 SST を定義するためにいくつか準備をする．キャプチャ変数を含む文字列 $v \in (Y_r \cup \Gamma)^*$ に対して自然数でインデックス付けされた文字列 $index(v, n)$ を以下のように定義する．ここで，:: は文字列の連結を表すものとする．

$$
\begin{aligned}
&index(\epsilon, n) = \epsilon \\
&index(\gamma v, n) = \gamma :: index(v, n) \\
&index(xv, n) = x_n :: index(v, n+1)
\end{aligned}
$$

たとえば，$x, y \in Y_r$ のとき，$index(axyx, 1) = ax_1y_2x_3$ となる．また，キャプチャ変数を含む文字列に含まれる変数の集合を $vars(w)$ で表す．たとえば，$vars(index(axyx, 1)) = \{x_1, y_2, x_3\}$ となる．

定義 4.1　入力文字列 $w \in \Sigma_r$ を受け取りマッチング箇所を置換文字列 v に書き換える決定性 *SST* $S_{r,v} = \langle Q, q_0, X, x_{out}, F, \Delta \rangle$ を定義する．ここで，状態の有限集合は $Q = \{q_0\} \cup \{q_A \mid A \subseteq Y_r\}$，変数の有限集合は $X = \{x_{out}\} \cup vars(index(v, 1))$，最終状態の有限集合は $F = \{x_{out}\}$ とする．遷移規則 Δ は以下の *5* 種類の遷移規則からなる．

- 非マッチング箇所の文字を出力する遷移

$(q_0, \sigma, id[x_{out} \mapsto x_{out}\sigma], q_0) \in \Delta$

ただし，$\sigma \in \Sigma$.

- マッチング開始および終了位置を検出する遷移

$(q_0, (, [x_{out} \mapsto x_{out}], q_\emptyset) \in \Delta$

$(q_\emptyset,), [x_{out} \mapsto x_{out} :: index(v, 1)], q_0) \in \Delta$

- キャプチャ変数対応箇所の開始を検出する遷移

$(q_A, (_x, id[x_i \mapsto \epsilon], q_{A\cup\{x\}}) \in \Delta$

ただし，$x \notin A$．また $id[x_i \mapsto \epsilon]$ は，任意のインデックス i に対して $id[x_i \mapsto \epsilon](x_i) = \epsilon$ となり，その他の変数 x' に対しては $id[x_i \mapsto \epsilon](x') = x'$ となる環境の更新とする．

- キャプチャ変数対応箇所の終了を検出する遷移

$(q_{A\cup\{x\}},)_x, id, q_A) \in \Delta$

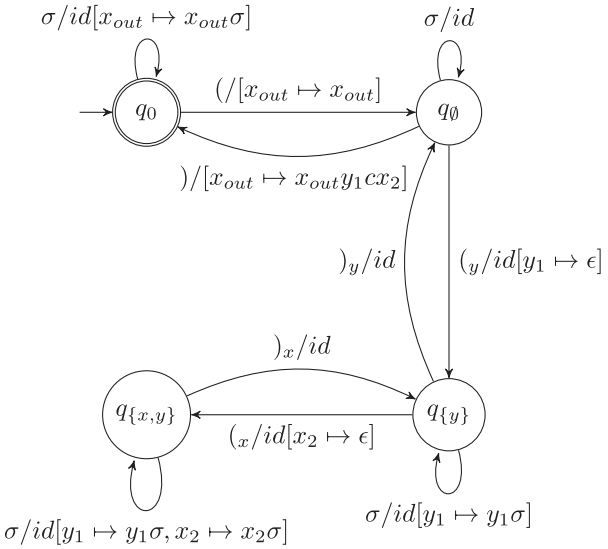
ただし，$x \notin A$.

- キャプチャ変数対応箇所を変数に記録する遷移

$(q_A, \sigma, id[x_i \mapsto x_i\sigma](x \in A), q_A) \in \Delta$

$id[x_i \mapsto x_i\sigma](x \in A)$ は，任意のインデックス i と任意の変数 $x \in A$ に対して $id[x_i \mapsto x_i\sigma](x \in A)(x_i) = x_i\sigma$ となり，その他の変数 x' に対しては $id[x_i \mapsto x_i\sigma](x \in A)(x') = x'$ となる環境の更新とする．

例 4.2　$Y_r = \{x, y\}$ をキャプチャ変数とする正規表現 $r = ((a)_x b)_y$ と置換文字列 $v = ycx$ に対する決定性 *SST* $S_{r,v}$ は以下のように構築する．ここで，図の遷移中の σ は Σ 中の任意の文字とする．



正規表現 r と置換文字列 v を用いて正規表現による文字列置換を行う関数的 SST を，正規表現マッチングを行う関数的非決定性トランスデューサ T_r とマッチング箇所を置換文字列に書き換える決定性 SST $S_{r,v}$ を合成することにより構成することができる．関数的 SST は合成で閉じていることは Alur と Deshmukh により示されている [2]．ただし，ここでは関数的非決定性トランスデューサと決定性 SST の合成であるため，以下のように直積構成で合成した関数的 SST を得ることができる．

$((q_1, q_1'), \sigma, \alpha, (q_2, q_2')) \in \Delta$
　if $(q_1, \sigma, w, q_2) \in \Delta_{T_r} \wedge q_1' \xrightarrow[S_{r,v}]{w/\alpha} q_2'$

5. 実装

文字列置換の等価性判定を行うプログラムは OCaml にて実装を行った．プログラムコードの行数は空行とコメント含めて約 4,700 行になっている．正規表現と置換文字列のペアを 2 組与えることにより同じ動作をするかを判定する．正規表現のパーサは佐久間らによって実装された Perl 準拠のものを使用する．正規表現マッチングを行う先読み付きトランスデューサは佐久間ら [10] および Frisch と Cardelli[8] によって示されているものを使用する．また，本プログラムが対応する正規表現は 4 章で導入した構文定義にさらに以下の構文を追加したものとする．

$$
\begin{array}{rcl}
r & ::= & \cdots \\
 & | & . \\
 & | & [\hat{}c_1c_2\ldots c_{n-1}c_n] \\
 & | & [c_1c_2\ldots c_{n-1}c_n] \\
 & | & r? \\
 & | & r+ \\
 & | & r\{i\} \\
 & | & r\{i,\} \\
 & | & r\{i,j\}
\end{array}
$$

1CM の 0 到達可能性判定は Esparza らによるプッシュダウンシステムの到達可能性判定 [6] を使用して行う．以下の流れで 1CM の 0 到達可能性判定を行っている．

(1) 1CM の各遷移でのカウンタの増減値がたかだか 1 となるように変形する．

(2) 1CM から以下のようにカウンタの値をスタックに保持するプッシュダウンシステムを構成する．スタックアルファベットは $\{P, M, Z\}$ とし，カウンタ n を以下のようにスタック上で表現する．

- $n \geq 0$ の場合，

$$\overbrace{P\cdots P}^{|n|}Z$$

- $n < 0$ の場合，

$$\overbrace{M\cdots M}^{|n|}Z$$

(3) Esparza らによるプッシュダウンシステムの到達可能

表 1　等価性判定結果
Table 1　Results of equivalence checking.

r_1	v_1	r_2	v_2	判定結果	実行時間 (秒)
(b+)	$1$1	b	bb	○	36.45
(a*)(b*)c	$1$2$1$2	(a*b*)c	$1$1	○	123.00
((aa|aaa)+)	$1$1	((aaa|aa)+)	$1$1	×(文字列長不一致)	18.47

性判定 [6] を用いてスタックに Z のみが積まれた状態での最終状態への到達可能性判定を行う．

異なる文字の同位置への出現可能性判定は，3 章では異なる出力文字 $a, b \in \Gamma$ に対して同位置への出現可能性を判定していたが，実装では重複のない出力文字の集合 $\Gamma_1, \Gamma_2 \subseteq \Gamma$ に対する同位置への出現可能性の判定を行っている．これは，特定の文字の出現位置までを非決定的に出力する SST の構成方法を応用することにより容易に実現することができる．異なる出力文字 $a, b \in \Gamma$ に対する判定では，異なるすべての出力文字のペアを網羅するためには出力アルファベット数 n に対して $O(n^2)$ 回判定を繰り返す必要がある．重複のない出力文字の集合 $\Gamma_1, \Gamma_2 \subseteq \Gamma$ に対する判定に拡張することにより，判定の繰り返し回数を出力アルファベット数 n に対して $O(\log n)$ 回の判定で行うことができるようになる．

たとえば，$\Gamma = \{a_{00}, a_{01}, a_{10}, a_{11}\}$ の場合は，以下の 4 つの集合を考える．

$$\Gamma_{0\perp} = \{a_{00}, a_{01}\}$$
$$\Gamma_{1\perp} = \{a_{10}, a_{11}\}$$
$$\Gamma_{\perp 0} = \{a_{00}, a_{10}\}$$
$$\Gamma_{\perp 1} = \{a_{01}, a_{11}\}$$

関数的 SSTS_1，S_2 が，異なる文字を同位置に出力することと以下のどれかが成立つことが同値になる．

- S_1 と S_2 が，同位置にそれぞれ $\Gamma_{0\perp}, \Gamma_{1\perp}$ に含まれる文字を出力する．
- S_1 と S_2 が，同位置にそれぞれ $\Gamma_{1\perp}, \Gamma_{0\perp}$ に含まれる文字を出力する．
- S_1 と S_2 が，同位置にそれぞれ $\Gamma_{\perp 0}, \Gamma_{\perp 1}$ に含まれる文字を出力する．
- S_1 と S_2 が，同位置にそれぞれ $\Gamma_{\perp 1}, \Gamma_{\perp 0}$ に含まれる文字を出力する．

このように検査をすることで，文字集合が m ビットで表現できれば，$2\,m$ 回の文字列長の一致可能性判定で検査することができる．

6. 実験

本研究において実装した正規表現による文字列置換の等価性判定アルゴリズムの動作検証を行う．

実装では，正規表現は構文解析された後に以下のように展開してから解析が行われる．

- $. \Rightarrow [abc\ldots]$　　$(\Gamma = \{a, b, c, \ldots\})$
- $[\hat{}c_1c_2\ldots c_{n-1}c_n] \Rightarrow [d_1d_2d_3\ldots]$
  $(\Gamma - \{c_1, c_2, c_3, \ldots\} = \{d_1, d_2, d_3, \ldots\})$
- $r^? \Rightarrow r|\epsilon$
- $r^+ \Rightarrow rr^*$

本実験では，Γ は ASCII 配列での!から~の 94 文字に設定して行う．

6.1 等価性判定実験

いくつかの正規表現による文字列置換に対して等価性判定アルゴリズムの適用を行う．各正規表現による文字列置換のペアに対して等価性判定結果および実行時間の計測を行う．

等価性判定を行う文字列置換およびその動作結果を**表 1** に示す．1 つ目のサンプル中の r_1, v_1 は文字列中の b の連続を 2 つ並べたもので置き換える文字列処理であり，r_2, v_2 は文字列中の b を bb に置き換える文字列処理である．どちらも文字列中の b が出現する箇所を長さが 2 倍の b の列に書き換える動作をし，まったく同じ動作をする．2 つ目のサンプルは，どちらも正規表現 a^*b^*c で検索し，マッチした箇所を末尾の c 以外を 2 回繰り返した文字列で書き換える文字列処理であり等価である．3 つ目のサンプルは一見同じ動作をするように見えるが，たとえば文字列 aaa に対して文字列置換を行った場合に出力文字列が異なる．r_1, v_1 では，マッチング結果は $((_1aa)_1)a$ となり，文字列置換の結果は $aaaaa$ となる．r_2, v_2 では，マッチング結果は $((_1aaa)_1)$ となり，文字列置換の結果は $aaaaaa$ となり長さが異なる．

6.2 有限状態トランスデューサでは表現できない文字列置換に対する性能評価

オープンソースとして公開されている WordPress，MediaWiki，PHPMyAdmin に含まれる関数 preg_replace の使用箇所を対象とする．正規表現と置換文字列に PHP 言語の変数の埋め込みのないものは重複なしで全 544 パターンあった．その中で，先読みやアトミックグループ・接頭辞マッチ・末尾辞マッチなどの拡張構文を用いていない本研究対象の正規表現を用いているものは 387 パターンであり，有限状態トランスデューサで表現できない文字列置換は 2 パターン存在した．有限状態トランスデューサで表現できない文字列置換を以下に示す．

表 2　有限状態トランスデューサで表現できない文字列置換の実験結果
Table 2　Results for regular expression replacement that cannot be represented with a finite state transducer.

	正規表現の大きさ	T_r		$S_{r,v} \circ T_r$		PDS	
		状態数	遷移数	状態数	遷移数	状態数	遷移数
r_1	40	46	4,449	63	5,573	712–756	5,523–6,021
r_2	11	14	1,452	16	1,505	375–392	2,592–2,751

	正規表現	置換文字列
r_1	`<a href="(.*?)".*?>(.*?)<\/a>`	`$2 &lt;$1&gt;`
r_2	`<([a-z]+)/>`	`<$1></$1>`

これらは，以下の理由から有限状態トランスデューサでは表現できない．

- r_1 の中に現れる 2 箇所の`"(.*?)"`には，それぞれ任意の長さの文字列がマッチする可能性がある．また，置換文字列では`$2`が`$1`より前に現れているので，その順番を変えて出力する必要がある．有限状態トランスデューサでは`$2`を出力している間，その状態として`$1`の内容を記憶しておくことができない．
- r_2 と`<$1></$1>`による文字列置換に関しては，理論的に説明することもできる．r_2 で表現される文字列の集合に，この文字列置換を適用すると以下の集合が得られる．

  $$\{\texttt{<}w\texttt{></}w\texttt{>} \mid w \in L(\texttt{[a-z]+})\}$$

  この集合は正規言語ではなく，この文字列置換が有限状態トランスデューサで表現できるとすると次の閉包性に矛盾する．有限状態トランスデューサを正規言語に適用して得られる言語は，正規言語である．

上の 2 つの文字列置換に対して，同一の文字列置換に対する等価性判定を実行する．マッチング情報を付加する有限状態トランスデューサの状態数および遷移数，正規表現による文字列置換を模倣する関数的 SST の状態数および遷移数，異なる文字の同位置への出現判定で最終的に生成されるプッシュダウンシステムの状態数と遷移数を集計する．

実験結果として**表 2** に集計結果を示す．異なる文字の同位置への出現判定では文字集合 $\Gamma' \subseteq \Gamma$ を変更し，繰り返しプッシュダウンシステムを構成する．そのため，プッシュダウンシステムは複数回生成される．表 2 ではこの複数回構築されるプッシュダウンシステムの状態数と遷移数の範囲を示している．

7. 関連研究

関数的 SST と同等の能力を計算モデルとして決定性 2 方向有限状態トランスデューサ（Deterministic Two-Way Finite State Transducer）と決定性 MSO 文字列トランスデューサ（Deterministic MSO Definable String Transducer）がある [1]．決定性 MSO 文字列トランスデューサは，単項二階述語論理（Monadic Second Order Logic: MSO）を用いてコピーする文字や文字順序を決定することにより新たな文字列を構成するトランスデューサである．また，決定性 2 方向有限状態トランスデューサは 2 方向に移動して適宜出力できる有限状態トランスデューサである．これら 2 つの計算モデルに対しても等価性判定問題は決定可能であることが証明されている [5], [9]．

決定性 MSO 文字列トランスデューサの等価性判定 [5] では，2 つの決定性 MSO 文字列トランスデューサに対して出力される文字列中に現れる異なる文字の出現位置をその文字の頂点数とする辺のないグラフを非決定的に出力する MSO グラフトランスデューサ（MSO Definable Graph Transducer）を構成する．MSO グラフトランスデューサ（MSO Definable Graph Transducer）の各文字ごとの頂点数は準線形集合で表現することができる [4]．準線形集合に対して同数であるものが含まれるかを判定することにより同位置への異なる文字の出力判定を行うことができる．

決定性 2 方向有限状態トランスデューサの等価性判定 [9] は，読み込み方向の反転およびカウンタの増加減少の切り替えの回数が有限回に制限された反転制限付き 2 方向カウンタ機械（Reversal-Bounded m-Counter Machine）の空判定に帰着させる方法で証明されている．

これらの 2 つの計算モデルもまた正規表現による文字列置換の等価性判定に利用することができるが，キャプチャ変数とストリーミング文字列トランスデューサ上の変数の対応により相性が良いことからストリーミング文字列トランスデューサを使用している．

また，変数の更新中に同じ変数を複数回を使用することが可能であるストリーミング文字列トランスデューサに対する等価性判定の証明もされている [7]．

8. 結論

本研究では，関数的 SST の等価性判定アルゴリズムの簡略化を行った．SST をモノイドを出力するトランスデューサと見ることにより，準同型写像を用いて容易に文字列長の情報を取得できるようにした．異なる文字の同位置への出現判定を文字列長の一致可能性判定に帰着させる方法で簡略化を行った．

また，関数的 SST による正規表現による文字列置換の動作の模倣を行うために，マッチング箇所の書き換えを行う関数的 SST を定義した．佐久間らにより正規表現マッチングを行う先読み付きトランスデューサが提案されており，マッチング箇所の書き換えを行う決定性 SST と合成することにより正規表現による文字列置換を模倣する関数的 SST の構築を行った．

これらを合わせることによって正規表現による文字列置換の等価性判定を行うことができることを示した．正規表現による文字列置換の等価性判定アルゴリズムの実装および実験を行った．実験結果から等価性判定器としての一定の成果とトランスデューサでは表現できない文字列置換への効果を示すことができた．

今後の課題として，正規表現による文字列置換を含むプログラムに対しての等価性判定に拡張することを考える．関連研究では変数の更新における制約を持たない関数的 SST の等価性判定の決定可能性も示されている．正規表現による文字列置換だけでなくより広い範囲のプログラムにも適用できる可能性を持っている．また，本論文におけるアルゴリズムの性能もまだまだ改善の余地があるため，最適化を行っていく必要がある．さらに，2 つの文字列置換に対して等価でないと判定された際に反例をあげることも考えられる．より広い範囲のシステムに対してより高速な検証を行えるように改良を加えていきたい．

謝辞　本研究は JSPS 科研費 24500028, 15K00087 の助成を受けたものです．

参考文献

[1] Alur, R. and Černý, P.: Expressiveness of streaming string transducers, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science* (*FSTTCS 2010*), LIPIcs, Vol.8, pp.1–12 (2010).

[2] Alur, R. and Deshmukh, J.V.: Nondeterministic streaming string transducers, *Proc. 38th International Colloquium on Automata, Languages and Programming* (*ICALP*), Lecture Notes in Computer Science, Vol.6756, pp.1–20 (2011).

[3] Alur, R. and Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs, *Proc. 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pp.599–610 (2011).

[4] Courcelle, B.: Monadic second-order definable graph transductions: A survey, *Theoretical Computer Science*, Vol.126, No.1, pp.53–75 (1994).

[5] Engelfriet, J. and Maneth, S.: The Equivalence Problem for Deterministic MSO Tree Transducers Is Decidable, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Vol.3821, pp.495–504 (2005).

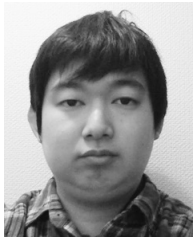
[6] Esparza, J., Hansel, D., Rossmanith, P. and Schwoon, S.: Efficient Algorithms for Model Checking Pushdown Systems, *Computer Aided Verification*, Lecture Notes in Computer Science, Vol.1855, pp.232–247 (2000).

[7] Filiot, E. and Reynier, P.: On Streaming String Transducers and HDT0L Systems, *CoRR*, Vol.abs/1412.0537 (2014).

[8] Frisch, A. and Cardelli, L.: Greedy regular expression matching, *Proc. 31th International Colloquium on Automata, Languages and Programming* (*ICALP*), Lecture Notes in Computer Science, Vol.3142, pp.618–629 (2004).

[9] Gurari, E.: The Equivalence Problem for Deterministic Two-Way Sequential Transducers is Decidable, *SIAM Journal on Computing*, Vol.11, No.3, pp.448–452 (1982).

[10] Sakuma, Y., Minamide, Y. and Voronkov, A.: Translating Regular Expression Matching into Transducers, *Journal of Applied Logic*, Vol.10, No.1, pp.32–41 (2012).



加賀江 優幸

2013 年筑波大学情報学群情報科学類卒．現在，同大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程在籍．

南出 靖彦　（正会員）

1993 年京都大学大学院理学研究科数理解析専攻修士課程修了．同年同大学数理解析研究所助手．1999 年筑波大学電子・情報工学系講師．2004 年筑波大学大学院システム情報工学研究科講師．2007 年同准教授．現在，筑波大学システム情報系准教授．博士（理学）．プログラミング言語およびソフトウェア検証に興味を持つ．