Symbolic SSTを用いた文字列制約の充足可能性判定

東京工業大学大学院 情報理工学院 数理・計算科学系
学籍番号 20M30338
福田大我

指導教員 南出靖彦 教授

令和4年1月21日
修士論文

概要

プログラミング言語において文字列は基本的なデータ型の一つであり文字列変数が多用される．文字列の操作はその正当性を決めることが難しく, 意図しない動作を含むことがある．文字列操作を解析する方法の一つに文字列制約を利用したものがあり, いくつかのソルバがある．既存のソルバはオートマトン, トランスデューサーを拡張したモデルを用いて充足可能性判定を行うが, 問題点の一つとして文字列のドメインが大きくなった時指数的に実行時間が大きくなることがある．Symbolic なモデルはアルファベットの大きい場合にオートマトンの遷移で全ての文字に対する個別の遷移が必要になってしまう問題を解決するために提案されたものである．本論文では SST を Symbolic に拡張し, それによるオートマトンの逆像の構成を行った．また, その手法を用いて実際に文字列制約の充足可能性判定を行い, Rust による実装を行った．

目次

第 1 章

序論

文字列はプログラミングにおける基本的なデータ型の一つであり, 置換, 比較, 正規表現による操作などが多く含まれる. 文字列の解析手法の一つとして文字列制約があり, Web プログラミングにおける記号的実行を利用したテストケース生成や XSS 解析, モデル検査, SMT ソルバ等の様々な領域での応用が考えられている. 一般の文字列制約に対する充足可能性判定は決定不能であることが知られており, 容易に PCP(Post Corresponding Problem) に帰着できる一方で, 直線制約という acyclic な形式に制限された制約に関しては決定可能であることが分かっている [1]. 文字列制約に関するソルバには CVC4[2], Ostrich[3], SLOTH[4] などが考案されている. (ソルバについて列挙する, JSST 入れる)

Chen らによる Ostrich では本研究で構成している逆像とはかなり異なる方法だが, Cost-enriched finite automata(CEFA)[5] とトランスデューサーを用いて逆像の計算を行い, CEFA の到達可能性に帰着している. 整数の半線形集合に関する制約を含むような文字列制約の判定を行っている. 本研究とは整数制約を含まない点と直線制約に含まれるトランスダクションが SST によるものでも構わないという点で異なる.

本研究の重要な先行研究である Zhu らによるソルバ [6] では Streaming String Transducer とオートマトンを用いて文字列制約の充足可能性を判定する. String Streaming Transducer(SST) は Alur らによって考案されたトランスダクションの高い表現力を持つモデルであり, Zhu はこの SST が合成により閉じていることを利用して, 本研究のソルバで用いている SST に似た形式に変換し, 逆像ではなく合成を用いて基礎直線制約と正規制約を表す SST を作った. そしてそのソルバを出力の Parikh Image を表すトランスデューサーに変換し, その半線形集合の空性を調べることにより充足可能性を判定している. ここで用いている SST は Symbolic なモデルではなく, 決定性である. 既存のソルバの課題の一つとしてアルファベットの増加により計算量が大きくなり, 複雑な制約に対しては時間がかかってしまうことがあり, その改善策の候補として Symbolic なモデルを用いることがある.

また, 同 Chen らは [7] においても... PSST を用いた逆像によるソルバ. WIP

Symbolic Finite Automata(SFA) は [8] 等の研究がなされており, UTF8 や UTF16 など現実的に扱わなければならない巨大なアルファベットに関する問題解決のために考案されたものである. 通常の DFA や NFA に対して, アルファベットの有限性を利用したアルゴリズムなどは事情が変わることが知られており, 例えば最小化についても既存アルゴリズムの計算量とは異なる形で, 活発な研究が行われている.

(非決定性については触れた方が良い.) 本研究では SST とそれによるオートマトンの逆像により制約の充足可能性を判定する方法をベースに, モデルの遷移を論理式により表す Symbolic な性質を導入することで課題の解決を目指す. Symbolic SST を定義し, その意味論を与え, SFA の Symbolic SST による逆像計算アルゴリズムとその形式的な証明を行った. 実際に簡単な制約を解くことができる実装を Rust で行った.

第 2 章

準備

2.1 文字列制約

初めに, 本論文で扱う対象である文字列制約を定義する. 文字列の長さの半線形集合による制約 (整数制約) を含む定義も存在するが, 本論文ではそのような長さに関する制約は考えず, 文字列変数のトランスダクションと正規言語による制約を考える.

定義 2.1.1 (基礎直線制約)**.** $x_0, x_1, \ldots, x_{n-1}$ を文字列変数, Σ をアルファベットとする. $j, k < i$, T: トランスダクション, $w \in \Sigma^*$ とする. x_i に関する原子制約 φ_i を

$$\varphi \stackrel{\text{def}}{=} w \mid x_j \mid x_j \cdot x_k \mid T(x_j)$$

と定義する. $m < n$, $\varphi_i (m \leq i < n)$ を原子制約とするとき, 基礎直線制約 φ_{sl} を

$$\varphi_{sl} \stackrel{\text{def}}{=} \bigwedge_{m \leq i < n} \varphi_i$$

で定義する.

定義 2.1.2 (正規制約)**.** $x_0, x_1, \ldots, x_{n-1}$ を文字列変数, $R_i (0 \leq i \leq n)$ を正規言語とする. 正規制約 φ_{reg} を

$$\varphi_{reg} \stackrel{\text{def}}{=} \bigwedge_{0 \leq i < n} x_i \in R_i$$

で定義する

定義 2.1.3 (文字列制約)**.** 直線文字列制約 φ を

$$\varphi \stackrel{\text{def}}{=} \varphi_{sl} \wedge \varphi_{reg}$$

で定義する.

w を文字列, x を文字列変数として変数の割り当て θ を

$$\begin{aligned} \theta(w) &= w \\ \theta(x_i \cdot x_j) &= \theta(x_i) \cdot \theta(x_j) \\ \theta(T(x)) &= T(\theta(x)) \\ \theta(x \in R) &\Leftrightarrow \theta(x) \in R \end{aligned}$$

と拡張する. 直線文字列制約 φ は真となる割り当て θ が存在するとき, 充足可能であるという.

直線制約に帰着可能な制約は充足可能性が決定可能であることが示されている. ([1], Theorem 5) 本論文では直線文字列制約のみを考え, 特に注釈のない文字列制約とした時は直線文字列制約を指すものとする.

例 1. 本論文では文字列変数 x に対して代表的なトランスダクションとして $x.rev()$ により x の反転, $x.replace(R, w)$ により正規表現 R で最初にマッチする部分の w による置換, $x.replace(R, w)$ で全ての置換を表す.

文字列変数を x_0, x_1, x_2, x_3 とするとき, 以下は直線文字列制約となる.

$$
\begin{aligned}
&x_1 = x_0 \cdot \text{"}a\text{"} \\
&x_2 = x_1 \cdot x_1.reverse() \\
&x_3 = x_2.replace(\text{"}cbaabc\text{"}, \text{"}xyz\text{"}) \\
&x_3 \in \text{"}xyz\text{"}
\end{aligned}
$$

θ を

$$
\begin{aligned}
x_0 &= \text{"}cb\text{"} \\
x_1 &= \text{"}cba\text{"} \\
x_2 &= \text{"}cbaabc\text{"} \\
x_3 &= \text{"}xyz\text{"}
\end{aligned}
$$

とすると上記制約を満たすため, 充足可能である.

2.2 Symbolic Automata

直線制約を解析するための道具の一つとして Symbolic Automaton を導入する. 通常のオートマトンが各遷移のラベルとして文字を利用するのに対して, Symbolic Automaton は以下の Effective Boolean Algebra を用いて遷移のラベルを表す.

定義 2.2.1 (Effective Boolean Algebra)**.** Effective Boolean Algebra $\mathcal{A}$ は以下の 8 つ組で表される.

$$\langle \mathcal{D}, \Psi, [\![\_]\!], \top, \bot, \vee, \wedge, \neg \rangle$$

- $\mathcal{D}$: ドメイン
- Ψ: 述語全体の集合
- $[\![\_]\!] \in \Psi \to 2^{\mathcal{D}}$: 論理式を満たす集合

とする. $\top, \bot \in \Psi$ は $[\![\top]\!] = \mathcal{D}, [\![\bot]\!] = \emptyset$ なる要素で, $\varphi, \psi \in Psi$ に対して $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!], [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!], [\![\neg\varphi]\!] = \mathcal{D} - [\![\varphi]\!]$ である.

以上に加えて, effective boolean algebra はその充足可能性が決定可能であるものとする.

例 2 (Equality Algebra)**.** $\mathcal{D}$ に対して $a \in \mathcal{D}, \varphi_a = \{a\}$ と $\top, \bot$, それらの論理閉包で表される集合を Ψ とすると effective boolean algebra になる.

Symbolic Finite Automata(SFA) を以下のように定義する

定義 2.2.2 (Symbolic Finite Automata)**.** SFAA を五組 $\langle \mathcal{A}, Q, q_0, F, \delta \rangle$ により定義する.

- $\mathcal{A}$: Boolean Algebra
- Q: 状態
- q_0: 初期状態
- F: 受理状態
- $\delta \subseteq Q \times \Psi \times Q$: 遷移関係

δ は次のような形となる.

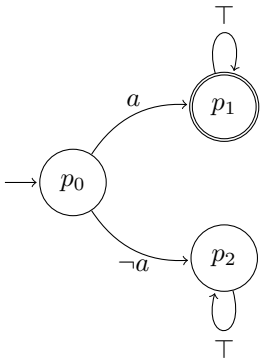
$$q \xrightarrow{\phi} q'$$

$a \in \mathcal{D}$ と A の遷移に対して, $q \xrightarrow{\phi} q'$ かつ $a \in [\![\phi]\!]$ であるとき, $q \xrightarrow{a} q'$ とする.
再帰的に, $q_0, q_1, \ldots, q_n, w = a_1 a_2 \ldots a_n$ に対して, $q_0 \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_n} q_n$ かつ $a_1 \in [\![\phi_1]\!], a_2 \in [\![\phi_2]\!], \ldots, a_n \in [\![\phi_n]\!]$ であるとき $q_0 \xrightarrow{w} q_n$ とする.
また, このようなパスがあるとき $q_0 \xrightarrow{[\phi_1, \ldots, \phi_n]} q_n$ と書くことにする.
$\mathrm{dom}(q) = \bigvee \{\phi \mid \exists q' \in Q. q \xrightarrow{\phi} q'\}$ とする.
A の言語を $\mathcal{L}(A) = \{w \in \mathcal{D}^* \mid q_0 \xrightarrow{w} q \wedge q \in F\}$ とする.

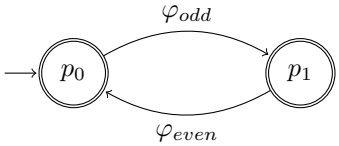
SFAA が決定的であるとは, 任意の $q \in Q$ で $q \xrightarrow{\varphi_1} q_1 \in \delta, q \xrightarrow{\varphi_2} q_2 \in \delta$ であるとき, $[\![\varphi_1 \wedge \varphi_2]\!] = \emptyset$ であることをいう.

通常のオートマトンは, 遷移のラベルを Equality Algebra に置換することで SFA に変換することができる. 一方, 論理式の Minterm を用いて SFA を有限語上の DFA に変換することで DFA のアルゴリズムを適用することはできるが, その Complexity に関してはアルファベット上の演算による分の考慮が必要で異なる部分もある. サブセット構成等, ラベル部分に関して修正を行うことでほとんど同様に適用できるものもあり, 任意の SFA を決定的に変換できることや空性判定, 和集合や積集合, 補集合に関して閉じていることなどは同じである.

例 3. 以下の SFA は a から始まる文字列を受理する.



例 4. 以下の SFA は奇数番目が奇数, 偶数番目が偶数である文字列を受理する. ただし, $\mathcal{D} = \{0,1,2,3,4,5,6,7,8,9\}$, $\varphi_{odd}(i) = (i \ mod \ 2 = 1)$, $\varphi_{even}(i) = (i \ mod \ 2 = 0)$ とする.



Symbolic Automaton と同様に Symbolic Transducer を考えることもできるが, 更新関数は元の文字に関する関数の適用により表され,　以下の function term を用いる.

2.3　Function Term

定義 2.3.1 (Function Term)**.** $\lambda : \mathcal{D} \to \mathcal{D}$ を function term という.
function term の集合を Λ で表す.

Function Term

例えば, 任意の $a \in \mathcal{D}$ に対して, $\mathrm{id}(a) = a$ なる恒等関数 id や $c \in \mathcal{D}$ として, 任意の $a \in \mathcal{D}$ に対して, $c(a) = c$ となる定数関数 c は function term である. function term 上の合成 $\circ$ を $\lambda_1, \lambda_2 \in \Lambda$ に対して

$$\lambda_1 \circ \lambda_2(a) = \lambda_1(\lambda_2(a))$$

と定義する. 同様に $\psi \in \Psi$ と $\lambda \in \Lambda$ に対して $\psi(\lambda) \in \Psi$ を

$$\psi(\lambda)(a) = \psi(\lambda(a))$$

と定義する.

Λ は文字から文字への関数であり, 文字から文字列への関数ではないことに注意する.

本研究のソルバでは Symbolic なモデルを用いて計算を行うため, 文字列は effective boolean algebra のドメイン $\mathcal{D}$ によって $\mathcal{D}^*$ と表される.

2.4　Streaming String Transducer

Streaming String Transducer(SST)[9] は Alur らにより提案されたトランスダクションを表すモデルであり, MSO Transducer, Two way Transducer[10] と等価な表現力を持つことが示されている [11].

定義 2.4.1 (SST)**.** (deterministic) Streaming String Transducer S を $\langle \Sigma, \Gamma, Q, X, \delta, \eta, q_0, F \rangle$ で定義する. Σ は入力文字列, Γ は出力文字列, Q を状態の集合, X は文字列変数の集合, q_0 は開始状態である. $\delta \subseteq Q \times \Sigma \to Q$ は遷移関数, $\eta \subseteq Q \times \Sigma \to M_{X,\Gamma}$ は変数更新関数である. $F \subseteq Q \hookrightarrow (X \cup \Gamma)^*$ は出力関数で, 受理状態に対して出力文字列を返す. ここで, $M_{X,\Gamma}$ は変数 $x \in X$ に対して X と Γ の文字列を返す関数 $\alpha : X \to (X \cup \Gamma)^*$ の集合である. SST の意味 $[\![S]\!]$ を以下のように定義する. $q_0 \xrightarrow{w/\alpha} q_f,\ q_f \in dom(F)$ のとき、

$$[\![S]\!](w) = \hat{\varepsilon}(\alpha(F(q_f)))$$

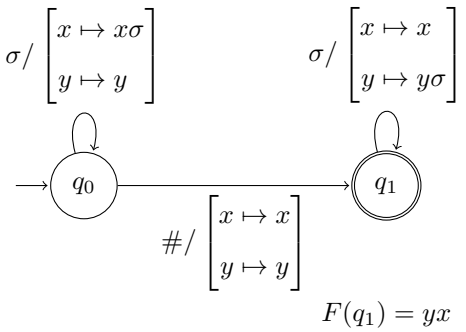
ここで、$\hat{\varepsilon} \in (\Gamma \cup X)^* \to \Gamma^*$ は変数を空文字列で置き換える関数である.

$\alpha \in M_{X,\Gamma}$ に対し, $\max_{x \in X} |\alpha|_x \leq k (k \in \mathbb{N})$ であるとき, α は k-copy であるという.

$\hat{\eta}$ を $\hat{\eta}(q, w) = \begin{cases} \mathbb{1}_{X,\Gamma}(x) & (w = \epsilon) \\ \eta(q, \sigma) \circ \hat{\eta}(\delta(q, \sigma), w') & (w = \sigma w') \end{cases}$ として, 文字列に対して変数更新関数を返すよう η を拡

張したものとする. 任意の文字列 $w \in \Sigma^*$ に対し, $\hat{\eta}(q_0, w)$ が k-copy であるとき, SSTS は k-bounded copy であるという.

以下は入力 $w_0 \# w_1$ に対し $w_1 w_0$ を出力する SST の $ab\#ba$ での動作である. 動作のわかりやすさのため, 状態, 文字列変数の中身を $\langle q, [x, y] \rangle$ の組みで表す.



- $\langle q, [x, y] \rangle = \langle q_0, [\varepsilon, \varepsilon] \rangle$ から開始.
- $\langle q_0, [\varepsilon, \varepsilon] \rangle \xrightarrow{a} \langle q_0, [a, \varepsilon] \rangle$
- $\langle q_0, [a, \varepsilon] \rangle \xrightarrow{b} \langle q_0, [ab, \varepsilon] \rangle$
- $\langle q_0, [ab, \varepsilon] \rangle \xrightarrow{\#} \langle q_1, [ab, \varepsilon] \rangle$
- $\langle q_1, [ab, \varepsilon] \rangle \xrightarrow{b} \langle q_1, [ab, b] \rangle$
- $\langle q_1, [ab, b] \rangle \xrightarrow{a} \langle q_1, [ab, ba] \rangle$
- $F(q_1) = yx$ であるから出力は $baab$ となる.

第 3 章

Symbolic SST

本研究では後に説明するように文字列制約を Alur らによる Streaming String Transducer を Symbolic に拡張したものと Symbolic Automaton を用いて表し, Automaton の SST による逆像を取ることで充足可能性判定を行う. 以下のように Symbolic SST を定義する.

定義 3.0.1 (Symbolic SST)**.** Symbolic SSTS を六組 $\langle \mathcal{A}, X, Q, q_0, f, \Delta \rangle$ で定義する.

- $\mathcal{A}$: Boolean Algebra
- X: 文字列変数
- Q: 状態
- q_0: 初期状態
- $F \subseteq Q \to (\mathcal{D} \cup X)^*$: 受理関数
- $\Delta \subseteq Q \times \Psi \times Q \times (X \to (\Lambda \cup X)^*)$: 遷移関係

出力される関数 $\alpha \subseteq X \to (\Lambda \cup X)^*$ を, function termλ に対しては $\alpha(\lambda) = \lambda$ となるよう自然に拡張された関数 $\alpha \subseteq (\Lambda \cup X)^* \to (\Lambda \cup X)^*$ として考える.

$\Lambda_k \stackrel{\text{def}}{=} \Lambda \times k, \Lambda \simeq \Lambda \times 1$ とする. 演算 $\uparrow$ を $(\lambda, i)^\uparrow = (\lambda, i+1)$ と定義し, $\forall x \in X.\ x^\uparrow = x$ として $\uparrow \subseteq (\Lambda_{[k]} \cup X)^* \to (\Lambda_{[k+1]} \cup X)^*$ に拡張する. ここで $\Lambda_{[k]} = \bigcup_{i \in \{1,\ldots,k\}} \Lambda_i$.

同様に $\forall d \in \mathcal{D}.\ d^\uparrow = d$ により $(X \cup \mathcal{D})^* \to (X \cup \mathcal{D})^*$ に対しても拡張する. この関数によって Symbolic SST の関数適用を $\forall h \in (\Lambda^{[k]} \times X)^*.\ \alpha(h) = \alpha(h^\uparrow)$ と意味付ける. $\alpha = \alpha_1 \circ \cdots \circ \alpha_n$ は $w = a_1 \cdots a_n$ に対して $\alpha(z)$ 中の (λ, i) の出現を $\lambda(a_i)$ で置換することで $\alpha_w \in X \to (\mathcal{D} \cup X)^*$ に簡約される.

$\Psi_{[k]} \in \Psi^{\{1,\ldots,k\}}$ として, $\psi \in \Psi_{[k]}$ に対して $\psi_i = \psi(i)$ と書く事にする.

$\phi \in \Psi_{[n]}$ は長さ n の文字列 $w = a_1 \ldots a_n$ が $\bigwedge_{[n]} \phi_i(a_i)$ を真にするとき $w \in [\![\phi]\!]$ と意味付ける. $\Psi^{[k]}$ に対して $\wedge, \vee, \neg$ を $i \in \{1, \ldots, k\}$

$$\begin{aligned} \varphi \wedge \psi(i) &= \varphi(i) \wedge \psi(i) \\ \varphi \vee \psi(i) &= \varphi(i) \vee \psi(i) \\ \neg\psi(i) &= \neg\psi(i) \end{aligned}$$

と定義し, $\Rightarrow$ を $\varphi \Rightarrow \psi = (\neg\varphi \vee \psi)$ とする. また, $\uparrow \subseteq \Psi^{[k]} \to \Psi^{[k+1]}$ を

$$\varphi^\uparrow(i) = \begin{cases} \top & \text{if} \quad i = 1 \\ \varphi(i-1) & \text{otherwise} \end{cases}$$

と定義する.

略記方として特に $\phi(i) = \varphi$ かつ $j(\neq i)$ に対して $\phi(j) = \top$ であるような $\phi \in \Psi^{[k]}$ を $[\varphi]_i$ と書き, そのような論理式の集合を Ψ_k とする.

Δ は次のような関数である.

$$q \xrightarrow{\varphi/\alpha} q'$$

ここで α は

$$\begin{aligned} x &= \kappa_{1,x}\kappa_{2,x}\dots\kappa_{n_x,x} \\ y &= \kappa_{1,y}\kappa_{2,y}\dots\kappa_{n_y,y} \\ &\vdots \end{aligned}$$

それぞれの $\kappa_{i,x}$ は function term または文字列変数である.

$a \in \mathcal{D}$ と S の遷移に対して, $q \xrightarrow{\varphi/\alpha} q'$ かつ $a \in [\![\varphi]\!]$ であるとき, $q \xrightarrow{a/\alpha} q'$ とする.

再帰的に, $q_0, q_1, \dots, q_n, w = a_1 a_2 \dots a_n$ に対して, $q_0 \xrightarrow{\varphi_1/\alpha_1} q_1 \xrightarrow{\varphi_2/\alpha_2} \cdots \xrightarrow{\varphi_n/\alpha_n} q_n$ かつ $w \in [\![\varphi_1 \wedge \cdots \wedge \varphi_n^{(\uparrow)^{n-1}}]\!]$ であるとき $q_0 \xrightarrow{w/\alpha_w} q_n$ とする.

S の言語を $\mathcal{L}(S) = \{(w, w') \in (\mathcal{D}^*)^2 \mid q_0 \xrightarrow{w/\alpha_w} q \wedge q \in \mathrm{dom}(F) \wedge \hat{\varepsilon}(\alpha_w(F(q))) = w'\}$ とする.

ここで $\hat{\varepsilon}$ は文字列変数を ε で置き換える関数である.

定義 3.0.2 (Bounded Copy)**.** 更新関数 α と $x \in X$ に対して $Var(\alpha(x))$ を $\alpha(x)$ に含まれる変数の数とする.

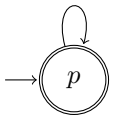
Symbolic SST$S = \langle \mathcal{A}, X, Q, q_0, f, \Delta \rangle$ が

$$\max_{w \in \mathcal{A}^*} \max_{x \in X, q_0 \xrightarrow{w/\alpha_w} q} Var(\alpha(x)) = k$$

を満たすとき k-bounded copy であるという. 特に $k = 1$ であるとき copyless という.

例 5. 以下の Symbolic SST は任意の文字列 w に対して ww^R を出力する. ただし, w^R は w を反転させた文字列とする.

$$\top / \begin{bmatrix} x = x \cdot \mathrm{id} \\ y = \mathrm{id} \cdot y \end{bmatrix}$$



$$F(p) = x \cdot y$$

この Symbolic SST は copyless である.

例 6. 以下の Symbolic SST は入力の, "$(a \cup b)^* c$" で表されるマッチを "d" で置き換えた文字列を出力する.

$\varphi_{a,b}(\sigma) = (\sigma = a \vee \sigma = b)$, $\varphi_c(\sigma) = (\sigma = d)$,

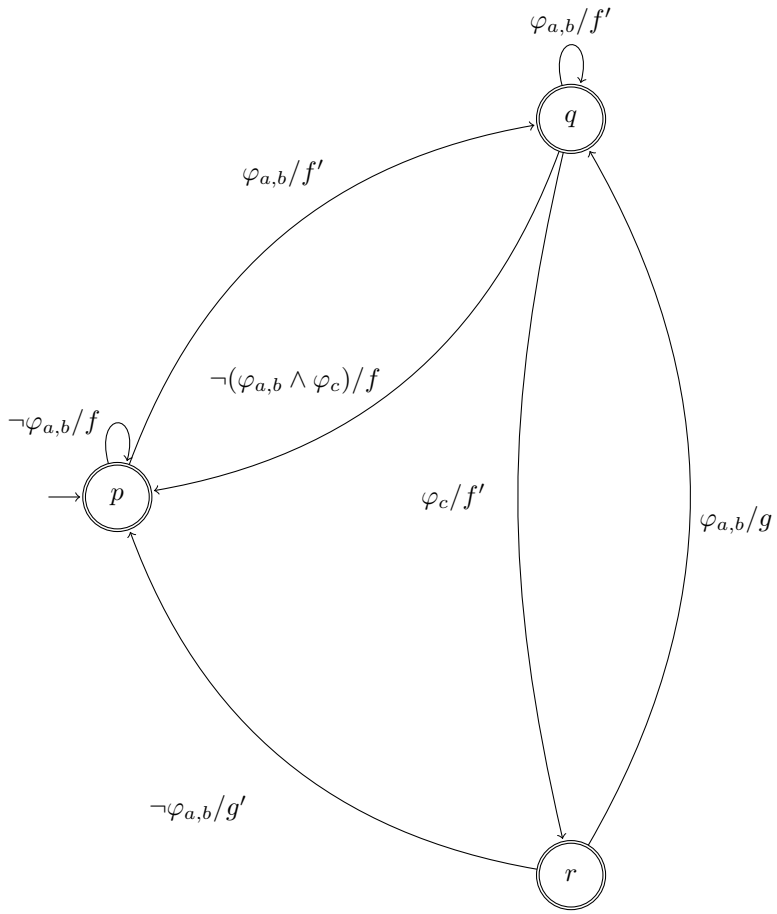
$$f = \begin{bmatrix} x = x \cdot \mathrm{id} \\ y = x \cdot \mathrm{id} \end{bmatrix}$$
$$f' = \begin{bmatrix} x = x \cdot \mathrm{id} \\ y = y \end{bmatrix}$$
$$g = \begin{bmatrix} x = y \cdot \text{"}d\text{"} \cdot \mathrm{id} \\ y = y \cdot \text{"}d\text{"} \end{bmatrix}$$
$$g' = \begin{bmatrix} x = y \cdot \text{"}d\text{"} \cdot \mathrm{id} \\ y = y \cdot \text{"}d\text{"} \cdot \mathrm{id} \end{bmatrix}$$

とする.

出力関数は

$$F(p) = x$$
$$F(q) = x$$
$$F(r) = y \cdot \text{"}d\text{"}$$

である.



上記 Symbolic SST は copyless である.

第 4 章

文字列制約の充足可能性判定

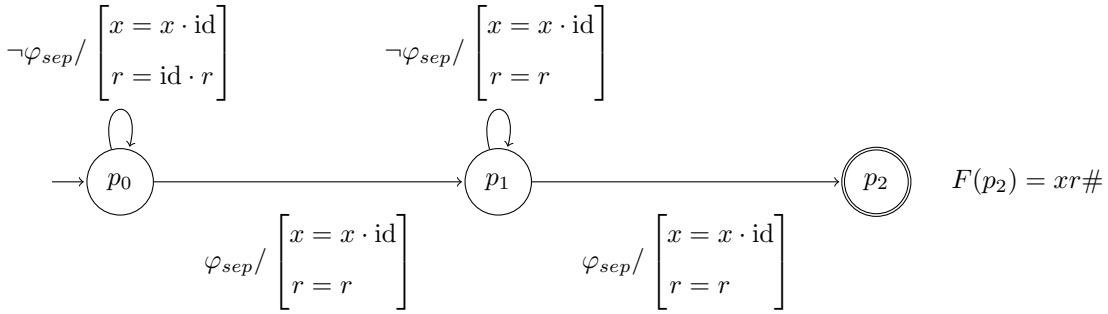
文字列制約 φ の充足可能性判定を以下のように行う．ただし, $\varphi_{sl} \stackrel{\text{def}}{=} \bigwedge_{m \le i \le n} \varphi_i$ であるとする．

1. 直線制約を Symbolic SST として, 正規制約を SFA として表現する．
2. SFA の SST による逆像を順に取る．
3. 最終的に生成された SFA の空性を判定することで解を得る．

step1

Zhu によるソルバと同様にして変換を行う．x_i に対応して得られる SSTS_i はセパレーター $\#$ として入力 $w_0\# \ldots \#w_{i-1}\#$ に対して $w_0\# \ldots \#w_i\#$ を出力するもので, w_i が x_i の制約を満たすものとなるように構成される．

例 7. $x_2 = \mathrm{reverse}(x_0)$ に対して出力される SST は以下のようになる．$\varphi_{sep}(\sigma) = (\sigma = \#)$ とする．



ただし, 正規制約の情報を SST に入れることはせず, x_i の制約 φ_i の計算に必要な情報を用いて SST の構成を行う．

正規制約 $\bigwedge_{0 \le i \le n} x_i \in R_i$ より $R_0\#R_1\# \cdots \#R_{n-1}\#$ に対応する SFA を構成する．対応する正規制約のない x_i については $x_i \in \mathcal{D}^*$ であるものとして考える．この SFA は正規制約を満たす $w_0, \ldots, w_{n-1}$ をセパレーターで連接した文字列を受理することに注意する．

これによって直線制約が $\bigwedge_{m \le i \le n} \varphi_i$ であるとき $n - m$ 個の SST と 1 個の SFA が得られる．

step2

step1 で生成された x_i に対応する SST は $x_0, \ldots, x_{i-1}$ までを入力として受け取り, それらにより得られる x_i を末尾に追加して出力する．S_{n-1} によって SFA の逆像を取ると, 得られた SFA は正規制約を満たす

$w_0, \ldots, w_{n-2}$ をセパレーターで連接した文字列を受理する. 同様にして得られた, 正規制約満たす $w_0, \ldots, w_i$ をセパレーターで連接した文字列を受理する SFA を順に S_i で逆像を取る.

step3

step2 で最終的に $0 < m$ であれば正規制約満たす $w_0 \# \ldots \# w_{m-1}$ を, $m = 0$ であれば ε を受理する SFA か, $\emptyset$ の SFA が得られる. 得られた SFA の空性を調べることで充足可能性を判定することができ, 空でない場合は受理されるパスを取ってくることで解の一つを得ることができる.

第 5 章

Symbolic SST の Symbolic Automaton による逆像

Symbolic SST $S = \langle \mathcal{A}, X, Q, q_0, F_S, \Delta \rangle$, SFA $A = \langle \mathcal{A}, P, p_0, F, \delta \rangle$ に対して,

$$\mathcal{L}(A') = S^{-1}(\mathcal{L}(A))$$

を満たすような SFA A' を構成する. すなわち, 入力に対する SST の出力で SFA が受理するか否かを判定する SFA を構成する.

基本的なアイデアとしては SST の入力に対し, 各ステップ毎の変数の値で SFA の各状態からどのように遷移しうるかを覚えておくことで, SST の各状態に対する出力で実際に SFA を動かし, 変数部分の遷移を覚えておいた遷移先で置換することにより最終的な入力の受理・拒否を判定する.

このとき, Symbolic な遷移ではたとえ決定性の SST, SFA であったとしても function term の適用結果がある論理式を満たすかどうかは入力に依存してしまうため, 決定性の SST を生成するためには全ての状態のシミュレーションを行う必要があり, 論理式のどの部分が実際に必要な条件か判定しなければならない. 本研究では最終的な出力に必要とされる変数の条件のみを非決定的に選び, 論理式として採用するため, 最終的に構成される SFA は非決定的になる.

また上記のような構成を行うために, 逆像を表す SFA は初期状態としていくつかの状態を持ちうる. 直接構成される SFA に関しては 2 章の定義と一致しないが, 新しい状態を用意し, 全ての開始状態の遷移をその状態に加え, 新しい開始状態とすることで前述の定義の形にすることは容易であり本質的には無関係なため逆像を表す SFA では開始状態の複数ある SFA を構成する.

5.1 構成法

逆像を表す SFA $A' = \langle \mathcal{A}, Q', I, F', \delta' \rangle$ とする. $Q' = Q \times (X \to \mathcal{P}(P \times P))$, $F' \subseteq Q'$, $\delta' \subseteq Q' \times \Psi \to Q'$, $I = \{(q_0, \mathrm{id}') \mid \forall x \in \mathrm{dom}(\mathrm{id}').\ \mathrm{id}'(x) \subseteq IDs.t.ID = \{(p,p) \mid p \in P\}\}$ である.

ここで状態 Q' は SST の状態 Q と各変数に対して各状態からの遷移先を非決定的に表す集合を返す関数の組である. この関数が後段の SFA での遷移をシミュレーションすることになる. $(q, g) \in Q'$ は SST の状態と変数による遷移を加えた SFA の組と見ることができる. この関数で各変数に対応する集合に含まれる状態の組も SST の出力で動作するために必要なものに限られることに注意する.

各変数の初期の値が ε であることから必要な変数のみを定義域とする関数で各状態から遷移しない関数であれ

ば全て初期状態になりうるため, I はそれらと SST の初期状態の組になっている.

$T = X \to \mathcal{P}(P \times P)$ とする. $g \in T$ に対して関係 $\underset{g}{\to} \subseteq (\Lambda_k \cup X) \times P \times \Psi_{[m]} \times P$ を $p \xrightarrow[A]{\varphi} p'$ のとき $p \xrightarrow[g]{(\lambda,i)/[\varphi(\lambda)]_i} p'$, $(p,p') \in g(x)$ のとき $p \xrightarrow{x/\top} p'$ として定義する. また,

$$p \xrightarrow[g]{h/\varphi} p' \in (X \to \mathcal{P}(P \times P)) \to (\bigcup_{m \in \mathcal{N}} ((\Lambda_{[m]} \cup X)^*) \times P \times \Psi_{[m]} \times P)$$

を以下のように帰納的に拡張する.

$$p \xrightarrow[g]{(\lambda,i)h/[\varphi(\lambda)]_i \wedge \varphi'} p' \quad \text{iff } p \xrightarrow[A]{\varphi} p'' \text{ and } p'' \xrightarrow[g]{h/\varphi'} p' \text{ and } (\lambda,i) \in \Lambda^{[k]} \text{and } k \in [m] \tag{rule 1}$$

$$p \xrightarrow[g]{xh/\varphi} p' \quad \text{iff } (p,p'') \in g(x) \wedge p'' \xrightarrow[g]{h/\varphi} p' \text{ and } x \in X \tag{rule 2}$$

$$p \xrightarrow[g]{\varepsilon/[\top]} p \tag{rule 3}$$

記法として, $p,p' \in P, h \in (\Lambda \cup X)^*$ に対して $\Delta_g(p,h,p') = \bigvee\{\varphi \mid p \xrightarrow[g]{h/\varphi} p'\}$ と定義する. ただし, $\bigvee \emptyset = \bot$. また, $\to \subseteq \bigcup_{m \in \mathcal{N}} (T \times (X \to (X \cup \Lambda^{[m]})^*)) \times (T \times \Psi_{[m]})$ を $g, g' \in T$ に対して, $\mathrm{dom}(g) = \bigcup_{y \in \mathrm{dom}(g')} Var(\alpha(y))$ のとき $\phi = \bigwedge_{x \in \mathrm{dom}(g')} \bigwedge_{(p,p') \in g'(x)} \Delta_g(p, \alpha(x), p')$ として

$$g \xrightarrow{\alpha/\phi} g'$$

と定義する. 全ての論理式の和をとり g 上の遷移を定義しても, 非決定的に全ての論理式に対して関係を付けても, 必要な変数に関する条件のみを定義しておりこの後の議論はほとんど同様の方法で成り立つ. 本研究では論理式の和を取ることにする.

$q \xrightarrow{\psi/\alpha} q' \in \Delta$ かつ $g \xrightarrow{\alpha/\phi} g'$ であるとき $(q,g) \xrightarrow{\phi \wedge \psi} (q',g') \in \delta'$ が成り立つように δ' を構成する.

関係 $p \xrightarrow[g]{w} p' \in (X \hookrightarrow P \times P) \to (\mathcal{D} \cup X)^* \times \mathcal{P}(P \times P)$ を以下で帰納的に定義する.

$$p \xrightarrow[g]{aw} p' \text{ iff } p \xrightarrow[A]{a} p'' \text{ and } p'' \xrightarrow[g]{w} p' \text{ and } a \in \mathcal{D} \tag{rule 4}$$

$$p \xrightarrow[g]{xw} p' \text{ iff } (p,p'') \in g(x) \wedge p'' \xrightarrow[g]{w/\varphi} p' \text{ and } x \in X \tag{rule 5}$$

$$p \xrightarrow[g]{\varepsilon} p \tag{rule 6}$$

と定義する.

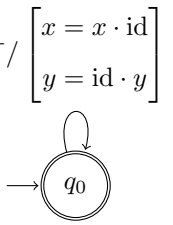
関係 $\underset{g}{\to}$ は最終的に構成する SFA の意味論には無関係だが, 変数による SFA の遷移を表すために必要となる.

$F' = \{(q,g) \in Q' \mid q \in \mathrm{dom}(F_S) \wedge p_0 \xrightarrow[g]{F_S(q)} p_f, p_f \in F\}$ を満たすように構成する. F' は SST で出力を持つ状態と SFA での遷移が受理状態となるような状態の集合となる.

5.2 構成例

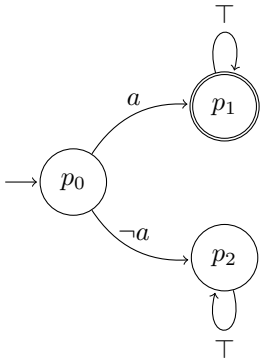
SFA A を Symbolic SST S で逆像を取ることを考える.

S:

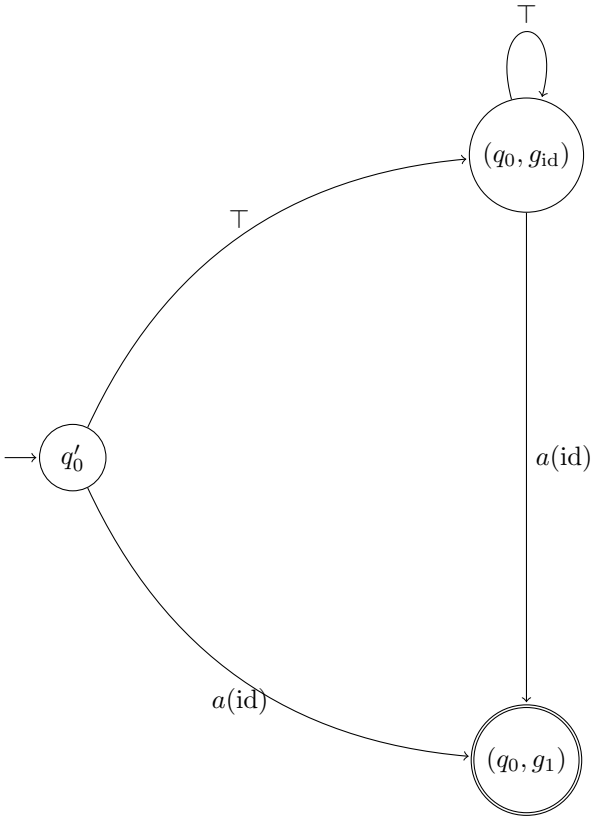


$F(q_0) = yx$

A:



$\forall c \in \Sigma.\ \mathrm{id}(c) = c, [\![a]\!] = \{a\}$ とする.
以下のようなオートマトンが得られる.

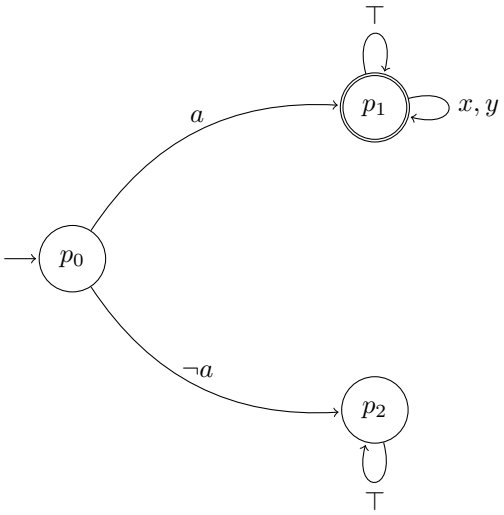


q_0' は新しく用意された開始状態で, 実質的には 2 状態である.
$g_i \subseteq P \times X \to P$ は以下の様な関数.

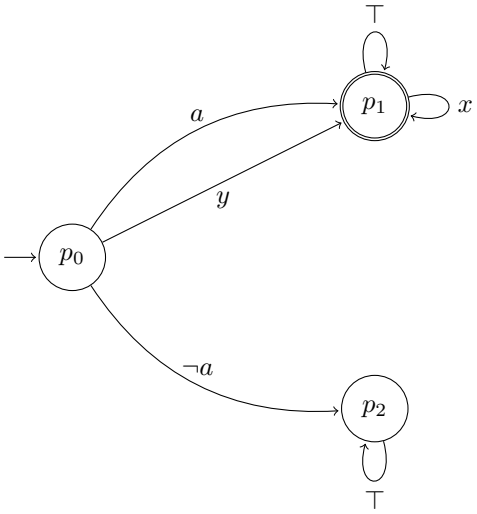
g_{id}	
x	$\{(p_1, p_1)\}$
y	$\{(p_1, p_1)\}$

g_1	
x	$\{(p_1, p_1)\}$
y	$\{(p_0, p_1)\}$

ここで g_{id}, g_1 により A がどのように拡張されているかを表す.
id では実質的に変数がループであるとみなせる.



g_1 では変数の中の文字列を読んでいくときの遷移で以下のようになる SFA を表すとみなせる.



ただし, 変数による遷移と論理式による遷移を区別するため別の矢印により表している.

5.3 上記構成の正しさ

5.3.1 $\mathcal{L}(A') \supset S^{-1}(\mathcal{L}(A))$ の証明

補題 5.3.1. ある $p_2 \in P$ に対して

$$p_1 \xrightarrow[g]{h_1/\varphi_1} p_2 \xrightarrow[g]{h_2/\varphi_2} p_3 \Leftrightarrow p_1 \xrightarrow[g]{h_1h_2/\varphi_1\wedge\varphi_2} p_3$$

証明. $|h_1|$ の帰納法で示す. ($\Rightarrow$) もほとんど同様に示すことができるため ($\Leftarrow$) のみ示す.
$|h_1| = 0$ のとき $p_1 \xrightarrow[g]{\varepsilon h_2/[\top]\wedge\varphi_2} p_3$ かつ $\top \wedge \varphi_2 = \varphi_2$ より $p_1 \xrightarrow[g]{\varepsilon/[\top]} p_1 \xrightarrow[g']{h_2/\varphi_2} p_3$ から成り立つ.
$|h_1| = k$ で成り立つと仮定して $|h_1| = k+1$ で 成り立つことを示す .
$h_1 = (\lambda, i)h, (\lambda, i) \in \Lambda^{[m]}$, $p_1 \xrightarrow[A]{\varphi_1'} p_1' \in \delta$ とする. $p_1 \xrightarrow[g]{h_1h_2/\varphi_1\wedge\varphi_2} p_3$ より rule 1 から $[\varphi_1'(\lambda)]_i \wedge \varphi_1'' = \varphi_1$ なる φ_1'' に対して $p_1' \xrightarrow[g]{hh_2/\varphi_1''\wedge\varphi_2} p_3$ でなければならない. (仮定より $p_1 \xrightarrow[A]{\varphi_1'} p_1'$ のようなパスが少なくとも一つある.) 帰納法の仮定よりある $p_2 \in P$ に対して

$$p_1' \xrightarrow[g]{h/\varphi_1''} p_2 \xrightarrow[g]{h_2/\varphi_2} p_3$$

より $p_1 \xrightarrow[g]{h_1/\varphi_1} p_2$ であることが分かるから示された. $h_1 = xh$ のとき $p_1 \xrightarrow[g]{h_1h_2/\varphi_1\wedge\varphi_2} p_3$ より rule 2 から $(p_1, p_1') \in g(x)$ なる p_1' に対して $p_1' \xrightarrow[g]{hh_2/\varphi_1\wedge\varphi_2} p_3$ でなければならない. 帰納法の仮定よりある $p_2 \in P$ に対して

$$g(p, x) \xrightarrow[g]{h/\varphi_1} p_2 \xrightarrow[g]{h_2/\varphi_2} p_3$$

より $p_1 \xrightarrow[g]{h_1/\varphi_1} p_2$ であることが分かるから示された. □

補題 5.3.2. $\alpha \in X \to (X \cup \Lambda)^*, \beta \in X \to (X \cup \Lambda^{[m]})^*$ とする. $g \xrightarrow{\alpha/\varphi_1} g' \xrightarrow{\beta/\varphi_2} g''$ であるとき,

$$g \xrightarrow{\alpha\circ\beta/\varphi_1\wedge\varphi_2^{\uparrow}} g''$$

が成り立つ.

証明. $\varphi = \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \Delta_g(p, \alpha\circ\beta(x), p')$ を計算する. $\beta(x)$ に k 個の変数が含まれるとして,

$l_0y_0\dots l_{k-1}y_{k-1}l_k$ としても一般性を失わない.$(1 \le k, l_i \in \Lambda^{[m]^*})$ 以下では $r_0 = p, r'_k = p'$ とする.

$$
\begin{aligned}
\varphi &= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \Delta_g(p, \alpha\circ\beta(x), p') \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \bigvee \left\{ \varphi_{l_0}\wedge\varphi_{y_0}\wedge\cdots\wedge\varphi_{l_{k-1}}\wedge\varphi_{y_{k-1}}\wedge\varphi_{l_k} \;\middle|\; \begin{array}{l} r_0 \xrightarrow[g]{\alpha(l_0)/\varphi_{l_0}} r'_0 \xrightarrow[g]{\alpha(y_0)/\varphi_{y_0}} r_1 \\ \xrightarrow[g]{\alpha(l_1)/\varphi_{l_1}} r'_1 \cdots \\ r_{k-1} \xrightarrow[g]{\alpha(l_{k-1})/\varphi_{l_{k-1}}} r'_{k-1} \xrightarrow[g]{\alpha(y_{k-1})/\varphi_{y_{k-1}}} r_k \\ \xrightarrow[g]{\alpha(l_k)/\varphi_{l_k}} r'_k \end{array} \right\} \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} ((\bigwedge_{i\in[0,\dots,k]} \Delta_g(r_i, \alpha(l_i), r'_i)) \wedge (\bigwedge_{i\in[0,\dots k-1]} \Delta_g(r'_i, \alpha(y_i), r_{i+1}))) \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} ((\bigwedge_{i\in[0,\dots,k]} \Delta_{g'}(r_i, l_i, r'_i))^{\uparrow} \wedge (\bigwedge_{i\in[0,\dots k-1]} \Delta_g(r'_i, \alpha(y_i), r_{i+1}))) \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} (\Delta_{g'}(p, \beta(x), p')^{\uparrow} \wedge (\bigwedge_{i\in[0,\dots,k-1]} \Delta_g(r'_i, \alpha(y_i), r_{i+1})) \\
&= \varphi_2^{\uparrow} \wedge \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \bigwedge_{i\in[0,\dots,k-1]} \Delta_g(r'_i, \alpha(y_i), r_{i+1})
\end{aligned}
$$

ここで, $\mathrm{dom}(g') = \bigcup_{y\in\mathrm{dom}(g'')} Var(\beta(y))$ に注意すると,

$$
\begin{aligned}
\varphi_1 &= \bigwedge_{x\in\mathrm{dom}(g')} \bigwedge_{(p,p')\in g'(x)} \Delta_g(p, \alpha(x), p') \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{y\in Var(\beta(x))} \bigwedge_{(p,p')\in g'(y)} \Delta_g(p, \alpha(y), p') \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{y\in Var(\beta(x))} \bigwedge_{(p,p')\in g'(y)} \bigvee_{p\xrightarrow[g]{\alpha(y)/\varphi} p'} \varphi \\
&= \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \bigwedge_{i\in[0,\dots,k-1]} \Delta_g(p'_i, \alpha(y_i), p_{i+1})
\end{aligned}
$$

故に, $\varphi_1 \wedge \varphi_2^{\uparrow} = \varphi$ が成り立つ. □

補題 5.3.3. $\alpha \in X \to (X \cup \Lambda)^*, \beta \in X \to (X \cup \Lambda^{[m]})^*$ とする. $g \xrightarrow{\alpha\circ\beta/\varphi} g''$ であるとき, ある g' が存在して,

$$g \xrightarrow{\alpha/\varphi_1} g' \xrightarrow{\beta/\varphi_2} g'' \wedge \varphi = \varphi_1 \wedge \varphi_2^{\uparrow}$$

が成り立つ.

証明. $x \in \bigcup_{y\in\mathrm{dom}(g'')} Var(\beta(y))$ に対して

$$g'(x) = \{(r, r') \mid \exists y \in \mathrm{dom}(g''). \exists (p, p') \in g''(y).\ \beta(y) = h_1 x h_2 \wedge p \xrightarrow[g]{\alpha(h_1)/\varphi_1} r \xrightarrow[g]{\alpha(x)/\varphi} r' \xrightarrow[g]{\alpha(h_2)/\varphi_2} p'\}$$

として g' を定義する. 定義より, $\mathrm{dom}(g') = \bigcup_{y\in\mathrm{dom}(g'')} Var(\beta(y))$ であり $\varphi_2 = \bigwedge_{x\in\mathrm{dom}(g'')} \bigwedge_{(p,p')\in g''(x)} \Delta_{g'}(p, \beta(x), p')$ として

$$g' \xrightarrow{\beta/\varphi_2} g''$$

が言える.

$$\begin{aligned}\mathrm{dom}(g) &= \bigcup_{y\in\mathrm{dom}(g'')} Var(\alpha\circ\beta(y)) \\ &= \bigcup_{y\in\mathrm{dom}(g'')}\ (\ \bigcup_{z\in Var(\beta(y))} Var(\alpha(z))) \\ &= \bigcup_{z\in\bigcup_{y\in\mathrm{dom}(g'')} Var(\beta(y))} Var(\alpha(z)) \\ &= \bigcup_{z\in\mathrm{dom}(g')} Var(\alpha(z))\end{aligned}$$

より $\mathrm{dom}(g)=\bigcup_{y\in\mathrm{dom}(g')} Var(\alpha(y))$ であり, $\varphi_1=\bigwedge_{x\in\mathrm{dom}(g')}\bigwedge_{(p,p')\in g'(x)}\Delta_{g'}(p,\alpha(x),p')$ として

$$g \xrightarrow{\alpha/\varphi_1} g'$$

が言える.
5.3.2 と同様の変形により,

$$\varphi=\varphi_1\wedge\varphi_2^{\uparrow}$$

である. □

補題 5.3.4. $(q,g)\xrightarrow{w}(q',g')$ のとき以下が成り立つ.

$$\begin{aligned} q &\xrightarrow{\psi/\alpha} q' \\ g &\xrightarrow{\alpha/\phi} g' \\ \psi(w)&\wedge\phi(w)\end{aligned}$$

証明. $|w|$ に関する帰納法で示す. $|w|=0$ のとき, $(q,g)\xrightarrow{\varepsilon}(q,g)$ である. $q\xrightarrow{\top/\mathrm{id}} q$, $g\xrightarrow{\mathrm{id}/\top} g$ より満たす. $|w|=k$ で成り立つとき, $w=\sigma w'$ として $|w|=k+1$ を考える. $(q,g)\xrightarrow{\sigma}(q'',g'')\xrightarrow{w'}(q',g')$ となる. $(q,g)\xrightarrow{\sigma/\alpha}(q'',g'')$ より, $\eta(\sigma)$ なる η で $(q,g)\xrightarrow{\eta}(q'',g'')$ が成り立つ. 構成法より,

$$\begin{aligned} q &\xrightarrow{\psi_1/\alpha} q'' \\ g &\xrightarrow{\alpha/\phi_1} g'' \\ \eta&=\psi_1\wedge\phi_1\end{aligned}$$

が成り立つ. $(q'',g'')\xrightarrow{w'}(q',g')$ より帰納法の仮定から,

$$\begin{aligned} q'' &\xrightarrow{\psi_2/\alpha} q' \\ g'' &\xrightarrow{\alpha/\phi_2} g' \\ \psi_2(w)&\wedge\phi_2(w)\end{aligned}$$

5.3.2 より,

$$\begin{aligned} q &\xrightarrow{\psi_1\wedge\psi_2^{\uparrow}/\alpha} q' \\ g &\xrightarrow{\alpha/\phi_1\wedge\phi_2^{\uparrow}} g'\end{aligned}$$

を満たす. $\psi_1 \wedge \psi_2^{\uparrow} = \psi, \phi_1 \wedge \phi_2^{\uparrow} = \phi$ と置くと, $\psi(w)$ かつ $\phi(w)$ を満たすから $\psi(w) \wedge \phi(w)$ より示された. □

補題 5.3.5. $w \in \mathcal{D}^*, v \in (X \cup \mathcal{D})^*$ に対して $g \xrightarrow{\alpha/\phi} g'$, $p \xrightarrow[g']{v} p'$, $\phi(w)$ ならば

$$p \xrightarrow[g]{\alpha_w(v)} p'$$

証明. $|v|$ の帰納法で示す. $|v| = 0$ ならば $p \xrightarrow[g']{\varepsilon} p$ であり, $\alpha(\varepsilon) = \varepsilon$ より成り立つ. $|v| = k$ で成り立つとして, $|v| = k+1$ を考える. $v = \sigma v', \sigma \in \mathcal{D}$ を考える. $p \xrightarrow{\sigma} p'' \xrightarrow{v'} p'$ とすると, 帰納法の仮定より, $p'' \xrightarrow[g]{\alpha_w(v')} p'$ である. $\alpha(\sigma) = \sigma$, $p \xrightarrow[g]{\sigma} p''$ より成り立つ. $v = xv', x \in X$ を考える. $(p, p'') \in g'(x)$ で $p'' \xrightarrow[g']{v'} p'$ が成り立つ. 帰納法の仮定より, $p'' \xrightarrow[g]{\alpha_w(v')} p'$ である. ここで, $g \xrightarrow{\alpha/\phi} g'$ と $\phi(w)$ よりある φ で $p \xrightarrow[g]{\alpha(x)/\varphi} p''$ かつ $\varphi(w)$ が成り立つ. このとき $p \xrightarrow[g]{\alpha_w(x)} p''$ である.
($\because$ 構成法より φ は $\alpha(x)$ に文字列を適用したときに満たさなければならない論理式を全て含む.) よって $p \xrightarrow[g]{\alpha_w(v)} p'$. □

定理 5.3.1. $L(A') \subseteq S^{-1}(L(A))$

証明. $(q_0, \mathrm{id}) \xrightarrow{w} (q_f, g_f) \wedge (q_f, g_f) \in F' \wedge (q_0, \mathrm{id}) \in I$ であるとき, $q_0 \xrightarrow[S]{w/\alpha_w} q_f$, $p_0 \xrightarrow[A]{\alpha_w(f(q_f))} p_f \wedge p_f \in F$ を満たすことを言えば良い. $(q_0, \mathrm{id}) \xrightarrow{w} (q_f, g_f) \wedge (q_f, g_f) \in F'$ であるとき, 5.3.4 より

$$q_0 \xrightarrow[S]{\psi/\alpha} q_f$$
$$\mathrm{id} \xrightarrow{\alpha/\phi} g_f$$
$$\psi(w) \wedge \phi(w)$$

が成り立つ. 故に, $q_0 \xrightarrow[S]{w/\alpha_w} q_f$ が成り立つ. $(q_f, g_f) \in F'$ より, $p_0 \xrightarrow[g_f]{f(q_f)} p_f \wedge p_f \in F$ である. 5.3.5 より $p_0 \xrightarrow[\mathrm{id}]{\alpha_w(f(q_f))} p_f$ であり, id による遷移は A での遷移と一致するから示せた. □

補題 5.3.6. $h \in (X \cup \Lambda^{[m]})^*, w \in \mathcal{D}^*$ とする.

$$p \xrightarrow[g]{h_w} p' \Rightarrow p \xrightarrow[g]{h/\varphi} p' \wedge \varphi(w)$$

証明. $|h|$ に関する帰納法で示す. $|h| = 0$ のとき $p \xrightarrow[g]{\varepsilon} p$ かつ $p \xrightarrow[g]{\varepsilon/\top} p$ より成り立つ.
$|h| = k$ で成り立つとして $k+1$ を考える. $h = (\lambda, i)h', \lambda \in \Lambda, 0 \leq i \leq m$ のとき, a_i を w の i 文字目とすると $p \xrightarrow[g]{\lambda(a_i)} p'' \xrightarrow[g]{h'_w} p'$ である. $p \xrightarrow[g]{\lambda(a_i)} p''$ よりある φ_1 で $p \xrightarrow[g]{\varphi_1} p''$ かつ $\varphi_1(\lambda(a_i))$ が成り立つ. 構成法より $p \xrightarrow[g]{(\lambda,i)/[\varphi_1(\lambda)]_i} p''$. また, 帰納法の仮定より

$$p'' \xrightarrow[g]{h'/\varphi_2} p' \wedge \varphi_2(w)$$

である. 5.3.1 より, $p \xrightarrow[g]{h/\varphi_1 \wedge \varphi_2} p'$ かつ $\varphi_1(w) \wedge \varphi_2(w)$ より成り立つ.
$h = xh'$ のとき $(p, p'') \in g(x) \wedge p'' \xrightarrow[g]{h_w} p'$ である. 帰納法の仮定より

$$p'' \xrightarrow[g]{h'/\varphi} p' \wedge \varphi(w)$$

$p \xrightarrow[g]{x/\top} p''$ であるから, 5.3.1 より, $p \xrightarrow[g]{h/\varphi} p''$ であり成り立つ. □

定理 5.3.2. $S^{-1}(L(A)) \subseteq L(A')$

証明. $q_0 \xrightarrow[S]{w/\alpha_w} q_f, q_f \in \mathrm{dom}(f)$ かつ $p_0 \xrightarrow[A]{\hat{\varepsilon}(\alpha_w(f(q_f)))} p_f, p_f \in F$ のときある g_f, id で

$$(q_0, \mathrm{id}) \xrightarrow[A']{w} (q_f, g_f) \wedge (q_f, g_f) \in F' \wedge (q_0, \mathrm{id}) \in I$$

であることを示せば良い. $\forall x \in \mathrm{dom}(\mathrm{id}).\ \mathrm{id}(x) = \{x\}, \mathrm{dom}(\mathrm{id}) = Var(\alpha_w(f(q_f)))$ とする. $p_0 \xrightarrow[A]{\hat{\varepsilon}(\alpha_w(f(q_f)))} p_f$ であるとき, $p_0 \xrightarrow[\mathrm{id}]{\alpha_w(f(q_f))} p_f$ である.

$|w| = 0$ のとき $q_0 \xrightarrow[S]{\varepsilon/\mathrm{id}} q_0, q_0 \in \mathrm{dom}(f)$ かつ $p_0 \xrightarrow[A]{\mathrm{id}(f(q_0)))} p_f, p_f \in F$ である. $(q_0, \mathrm{id}) \xrightarrow[\mathrm{id}]{\varepsilon} (q_0, \mathrm{id})$ で, このとき $p_0 \xrightarrow[\mathrm{id}]{f(q_0))} p_f$ より成り立つ.

$|w| \neq 0$ のとき, $q_0 \xrightarrow[S]{w/\alpha_w} q_f, q_f \in \mathrm{dom}(f)$ より $q_0 \xrightarrow[S]{\psi/\alpha} q_f \wedge \psi(w)$ なる遷移が存在する. また, $p_0 \xrightarrow[id]{\alpha_w(f(q_f))} p_f, p_f \in F$ だから $f(q_f)$ 中の変数が k 個だとすれば $f(q_f) = w_0 y_0 \ldots w_{k-1} y_{k-1} w_k, w_i \in \mathcal{D}^*, y_i \in X$ として一般性を失わない. $p_0 \xrightarrow[\mathrm{id}]{w_0} p'_0 \xrightarrow[\mathrm{id}]{\alpha_w(y_0)} p_1 \to \cdots \xrightarrow[\mathrm{id}]{\alpha_w(y_{k-1})} p_n \xrightarrow[\mathrm{id}]{w_k} p_f$ とかける $(p_i, p'_i \in P)$.

$x \in \bigcup_{0 \leq i \leq n} y_i$ に対して $g_f(x) = \{(p'_i, p_{i+1}) \mid p_i \xrightarrow[\mathrm{id}]{\alpha_w(x)} p'_{i+1}\}$ と g_f を定義すると $\mathrm{id} \xrightarrow{\alpha/\phi} g_f$.

また, $p_0 \xrightarrow[\mathrm{id}]{\alpha_w(f(q_f))} p_f$ と g_f の定義より $p_0 \xrightarrow[g_f]{f(q_f)} p_f$ が分かる. 5.3.6 より $p'_i \xrightarrow[\mathrm{id}]{\alpha(y_i)/\varphi_i} p_{i+1}$ かつ $\varphi_i(w)$ であるから, $\forall x \in \mathrm{dom}(g_f).\ \forall (p, p') \in g_f(x).\ \Delta_{\mathrm{id}}(p, \alpha(x), p')(w)$. 故に, $\phi(w)$. 以上より, $q_0 \xrightarrow[S]{\psi/\alpha} q_f, q_f \in \mathrm{dom}(f)$ かつ $id \xrightarrow{\alpha/\phi} g_f$ である. $w = a_0 a_1 \ldots a_n$ とおけば, $q_0 \xrightarrow[S]{\psi_0/\alpha_0} q_1 \to \cdots \to q_n \xrightarrow[S]{\psi_n/\alpha_n} q_f$ とかけて, $\psi_i(a_i)$ かつ $\alpha = \alpha_0 \circ \alpha_1 \circ \cdots \circ \alpha_n$ が成り立つ. 5.3.3 より $\mathrm{id} \xrightarrow{\alpha_0/\phi_0} g_1 \to \cdots \to g_n \xrightarrow{\alpha_n/\phi_n} g_f$ かつ $\phi = \phi_0 \wedge \phi_1 \wedge \cdots \wedge \phi_n^{\uparrow^{(n)}}$ が成り立つ. 故に構成法から $(q_0, \mathrm{id}) \xrightarrow[A']{\psi_0 \wedge \phi_0} (q_1, g_1) \to \cdots \to (q_n, g_n) \xrightarrow[A']{\psi_n \wedge \phi_n} (q_f, g_f)$ であり,

$$\begin{aligned}(\psi_0 \wedge \phi_0) \wedge (\psi_1 \wedge \phi_1)^{\uparrow} \wedge \cdots \wedge (\psi_n \wedge \phi_n)^{\uparrow^{(n)}} &= (\psi_0 \wedge \phi_0) \wedge (\psi_1^{\uparrow} \wedge \phi_1^{\uparrow}) \wedge \cdots \wedge (\psi_n^{\uparrow^{(n)}} \wedge \phi_n^{\uparrow^{(n)}}) \\ &= (\psi_0 \wedge \psi_1^{\uparrow} \wedge \cdots \psi_n^{\uparrow^{(n)}}) \wedge (\phi_0 \wedge \phi_1^{\uparrow} \wedge \cdots \phi_n^{\uparrow^{(n)}}) \\ &= \psi \wedge \phi\end{aligned}$$

より $(q_0, \mathrm{id}) \xrightarrow[A']{\psi \wedge \phi} (q_f, g_f)$ が言える. $\psi(w) \wedge \phi(w)$ であるから $(q_0, \mathrm{id}) \xrightarrow[A']{w} (q_f, g_f)$ で, $q_f \in \mathrm{dom}(f)$ かつ $p_0 \xrightarrow[g_f]{f(q_f)} p_f, p_f \in F$ より $(q_f, g_f) \in F'$. □

5.3.1 と 5.3.2 より直ちに以下が分かる.

定理 5.3.3. $L(A') = S^{-1}(L(A))$

よって, 上記構成の正当性が示せた.

第 6 章

実装

4 章, 5 章の実装を Rust で行った. smt2(https://smtlib.cs.uiowa.edu/) の形式で入力ファイルを受け取り, 実行を行う. 実際のコードは https://github.com/tgfukuda/solver_with_symbolic で確認することができる.

6.1 モデルの実装

6.1.1 smt2 形式からの変換

smt2 からの変換にはクレート smt2parser v0.6.1 を使用した. 本クレートでは smt2 ファイルから ast への変換を行い, struct として提供してくれる.

6.1.2 ドメイン

boolean algebra のドメインは以下のような trait で実装した.

Listing 6.1 Domain

```
pub trait Domain: Debug + Eq + Ord + Clone + Hash + From<char> + Into<char> {
    fn separator() -> Self;
}
```

char からの変換が必要なため上記のような実装となる.

6.1.3 boolean algebra, function term

boolean algebra の定義と共に Equality algebra を含む述語集合を前提としたため, 以下のような実装になっていることに加え, function term との合成とあるドメインが論理式を満たすかを調べるためのメソッドを含む.

Listing 6.2 boolean algebra

```
pub trait BoolAlg: Debug + Eq + Hash + Clone {
    type Domain: Domain;
    type Term: FunctionTerm<Domain = Self::Domain>;
```

```
type GetOne: Domain;

/**
 * predicate that means x == a.
 * it names `char` because
 * `eq` is already defined in trait PartialEq
 * and the name like it is confusing.
 */
fn char(a: Self::Domain) -> Self;
fn and(&self, other: &Self) -> Self;
fn or(&self, other: &Self) -> Self;
fn not(&self) -> Self;
fn top() -> Self;
fn bot() -> Self;
fn with_lambda(&self, f: &Self::Term) -> Self;

fn all_char() -> Self {
  Self::char(Self::Domain::separator()).not()
}

fn boolean(b: bool) -> Self {
  if b {
    Self::top()
  } else {
    Self::bot()
  }
}

fn separator() -> Self {
  Self::char(Self::Domain::separator())
}

/** apply argument to self and return the result */
fn denote(&self, arg: &Self::Domain) -> bool;

fn satisfiable(&self) -> bool;
```

```
    fn get_one(&self) -> Result<Self::GetOne, NoElement>;
}
```

function term も同様に以下の trait で実装している．基本的な term の生成と適用, composition が含まれる．

Listing 6.3 function term

```
pub trait FunctionTerm: Debug + Eq + Hash + Clone {
    type Domain: Domain;

    fn identity() -> Self;

    fn constant(a: Self::Domain) -> Self;

    fn separator() -> Self {
        Self::constant(Self::Domain::separator())
    }

    fn apply<'a>(&'a self, arg: &'a Self::Domain) -> &'a Self::Domain;

    /** functional composition of self (other (x)) */
    fn compose(self, other: Self) -> Self;
}
```

6.1.4 SFA, SST

SFA, SST, Transducer は共通の処理を実装する trait (入力に対する実行や 1 ステップの動作, 必要のない状態を排除するメソッド等) が別にあるが, 以下の struct で実装してある．シンプルな実装にするため定義外のフィールドは含まない．

SFA では preimage に加えて, intersection, concat, or 等の基本的な実装がされている．

Listing 6.4 SFA

```
    type Source<S, B> = (S, B);
    type Target<S> = Vec<S>;
    /**
    * symbolic automata
    * each operation like concat, or, ... corresponds to regex's one.
    */
    #[derive(Debug, PartialEq, Clone)]
    pub struct SymFa<D, B, S>
```

```
where
  D: Domain,
  B: BoolAlg<Domain = D>,
  S: State,
{
  states: HashSet<S>,
  initial_state: S,
  final_states: HashSet<S>,
  transition: HashMap<Source<S, B>, Target<S>>,
}
```

SST の実装において HashMap を更新関数で利用しており, BTreeMap を利用する方法もあるが, HashMap に Hash が実装されないため非決定的な遷移に Vec(BTreeSet でも良い) を使用している. 二つの SST の遷移を merge するためのメソッドとソルバで使用する形式の SST を作成するためのメソッドが存在する. struct とは別に入力文字列 w に対して w, w^R, replace(from, to), replace_all(from, to), constant(w') を出力する SST を生成する関数と原子制約から SST に変換する関数が存在する.

Listing 6.5 SST

```
type UpdateFunction<F, V> = HashMap<V, Vec<UpdateComp<F, V>>>;
type Source<B, S> = (S, B);
type Target<F, S, V> = (S, UpdateFunction<F, V>);
type Output<D, V> = Vec<OutputComp<D, V>>;
type Transition<B, F, S, V> = HashMap<Source<B, S>, Vec<Target<F, S, V>>>;

/** implementation of symbolic streaming string transducer (SSST) */
#[derive(Debug, PartialEq, Clone)]
pub struct SymSst<D, B, F, S, V>
where
  D: Domain,
  B: BoolAlg<Domain = D>,
  F: FunctionTerm<Domain = D>,
  S: State,
  V: Variable,
{
  states: HashSet<S>,
  variables: HashSet<V>,
  initial_state: S,
  output_function: HashMap<S, Output<D, V>>,
  /**
```

```
        * if a next transition has no correponding sequence
        * for some variable, update with identity
        * i.e. update(var) = vec![UpdateComp::X(var)]
        */
       transition: Transition<B, F, S, V>,
   }
```

6.2 逆像の計算

実際の逆像計算では全ての状態を計算することはせず, 必要な部分のみに限って構成する. Symbolic SST$\langle \mathcal{A}, X, Q, q_0, F_S, \Delta \rangle$ とするとき, 初期状態 Q' を $q_f \in \mathrm{dom}(F_S)$ に対して $F_S(q_f)$ で SFA を動作させ受理された集合とする. この際 SFA の動作に関してもバックトラックで可能性を列挙していき, 動作を調べる.

Q' の状態に対して 5 章の手続きでその状態への遷移を追加し, 新たに現れた状態に対して再帰的に計算していく. つまり, SST の遷移の探索, SFA の探索の双方に関してバックトラックで計算を行っていく. この計算に関しては到達しうる状態を調べていく方法, すなわち各ステップを通常の動作と同様の方法で計算していく方法でも同じ SFA が構成できるが, 実行時間に大きな差が出たためこの方法を採用している.

例 8.

第 7 章

結論

本研究では Symbolic SST の形式化とそれを利用した文字列制約の充足可能性判定を行った．Symbolic SST において遷移を論理式にすることは出力に入力の情報が直接関係してくるということであり, 文字のインデックスの情報を加えた論理式を使った意味論を与えることで Symbolic SST の意味を与えた．また, その逆像を与えるアルゴリズムを提案し, その正当性を示した．本論文のアルゴリズムは Symbolic SST に関して制約を与えておらず, Copyfull SST においても同様の手法により逆像計算が行える．これは状態として持たせた, 後段の SFA を変数に対して拡張する関数で冪集合を用いていることが要因であり, 部分関数にする等の手法で逆像計算の定義域を copyless(bounded copy) なものに制限することができるだろう．本研究において文字列制約は整数制約を含まないものを考えているが, これは逆像の手法に関係しており重要である．

まず, 整数制約を含む文字列制約に対して拡張するためには CEFA や Parikh Automata のような整数に関する情報をオートマトンの遷移に持たせ, 半線形集合として表現しなければならないが, それらのモデルを Symbolic に拡張することが必要である．また, それらのモデルが Symbolic SST の逆像で閉じていることが必要である．こういった理由から, 本研究の手法に何らかの修正を与えなければ適用することはできないと考えられ, 今後の課題の一つである．

(計算量周り, ボトルネックの考察入れる?)

実装面に関しても, 非常に簡単な制約に関しては動作の確認が取れているものの, うまく動かないケースがあるため改善が必要である．

第 8 章

謝辞

参考文献

[1] Anthony W Lin and Pablo Barceló. String solving with word equations and transducers: towards a logic for analysing mutation XSS. In *43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming*, Vol. 51, pp. 123–136. ACM, 2016.

[2] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pp. 171–177. Springer, 2011.

[3] Taolue Chen, Matthew Hague, Jinlong He, Denghang Hu, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. A decision procedure for path feasibility of string manipulating programs with integer data type. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 325–342. Springer, 2020.

[4] Lukáš Holík, Petr Janků, Anthony W Lin, Philipp Rümmer, and Tomáš Vojnar. String constraints with concatenation and transducers solved efficiently. *Proceedings of the ACM on Programming Languages*, Vol. 2, No. POPL, pp. 1–32, 2017.

[5] Rajeev Alur, Loris DAntoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 13–22. IEEE, 2013.

[6] Qizhen Zhu, Hitoshi Akama, and Yasuhiko Minamide. Solving string constraints with streaming string transducers. *Information Processing*, Vol. 27, pp. 810–821, 2019.

[7] Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony W Lin, Philipp Ruemmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *Proceedings of the ACM on Programming Languages*, Vol. 6, No. POPL, pp. 1–31, 2022.

[8] Loris D' Antoni and Margus Veanes. The power of symbolic automata and transducers. In *International Conference on Computer Aided Verification*, pp. 47–67. Springer, 2017.

[9] Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 599–610, 2011.

[10] Joost Engelfriet and Hendrik Jan Hoogeboom. Mso definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, Vol. 2, No. 2, pp. 216–254, 2001.

[11] Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal Lodaya

and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, Vol. 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 1–12, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.