String Constraint Solving with Transducers, ReplaceAll and Reverse

TAOLUE CHEN, Birkbeck, University of London, United Kingdom
MATTHEW HAGUE, Royal Holloway, University of London, United Kingdom
ANTHONY W. LIN, University of Oxford, United Kingdom
PHILIPP RUEMMER, Uppsala University, Sweden
ZHILIN WU, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China

To be done.

CCS Concepts: • **Theory of computation → Automated reasoning**; **Verification by model checking**; **Program verification**; **Program analysis**; *Logic and verification*; Complexity classes;

Additional Key Words and Phrases: String Constraints, Transducers, ReplaceAll, Reverse, Decision Procedures, Straight-Line Programs

ACM Reference Format:
Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Ruemmer, and Zhilin Wu. 2019. String Constraint Solving with Transducers, ReplaceAll and Reverse . *Proc. ACM Program. Lang.* 1, XXXX, Article 1 (January 2019), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

2 PRELIMINARIES

General Notation. Let $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers and natural numbers respectively. For $k \in \mathbb{N}$, let $[k] = \{1, \cdots, k\}$. For a vector $\vec{x} = (x_1, \cdots, x_n)$, let $|\vec{x}|$ denote the length of $\vec{x}$ (i.e., n) and $\vec{x}[i]$ denote x_i for each $i \in [n]$.

Regular Languages. Fix a finite *alphabet* Σ. Elements in Σ^* are called *strings*. Let ε denote the empty string and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. We will use $a, b, \cdots$ to denote letters from Σ and $u, v, w, \cdots$ to denote strings from Σ^*. For a string $u \in \Sigma^*$, let $|u|$ denote the *length* of u (in particular, $|\varepsilon| = 0$). A *position* of a nonempty string u of length n is a number $i \in [n]$ (Note that the first position is 1, instead of 0). In addition, for $i \in [|u|]$, let $u[i]$ denote the i-th letter of u. For two strings u_1, u_2, we use $u_1 \cdot u_2$ to denote the *concatenation* of u_1 and u_2, that is, the string v such that $|v| = |u_1| + |u_2|$ and for each $i \in [|u_1|]$, $v[i] = u_1[i]$ and for each $i \in |u_2|$, $v[|u_1| + i] = u_2[i]$. Let u, v be two strings.

Authors' addresses: Taolue Chen, Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London, WC1E 7HX, United Kingdom, taolue@dcs.bbk.ac.uk; Matthew Hague, Department of Computer Science, Royal Holloway, University of London, Egham Hill, Egham, Surrey, TW20 0EX, United Kingdom, matthew.hague@rhul.ac.uk; Anthony W. Lin, Department of Computer Science, University of Oxford, Wolfson Buildin, Parks Road, Oxford, OX1 3QD, United Kingdom, anthony.lin@cs.ox.ac.uk; Philipp Ruemmer, Department of Information Technology, Uppsala University, Box 337, Uppsala, SE-751 05, Sweden, philipp.ruemmer@it.uu.se; Zhilin Wu, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China.

2475-1421/2019/1-ART1
https://doi.org/10.1145/nnnnnnn.nnnnnnn

If $v = u \cdot v'$ for some string v', then u is said to be a *prefix* of v. In addition, if $u \neq v$, then u is said to be a *strict* prefix of v. If u is a prefix of v, that is, $v = u \cdot v'$ for some string v', then we use $u^{-1}v$ to denote v'. In particular, $\varepsilon^{-1}v = v$.

A *language* over Σ is a subset of Σ^*. We will use $L_1, L_2, \ldots$ to denote languages. For two languages L_1, L_2, we use $L_1 \cup L_2$ to denote the union of L_1 and L_2, and $L_1 \cdot L_2$ to denote the concatenation of L_1 and L_2, that is, the language $\{u_1 \cdot u_2 \mid u_1 \in L_1, u_2 \in L_2\}$. For a language L and $n \in \mathbb{N}$, we define L^n, the *iteration* of L for n times, inductively as follows: $L^0 = \{\varepsilon\}$ and $L^n = L \cdot L^{n-1}$ for $n > 0$. We also use L^* to denote the iteration of L for arbitrarily many times, that is, $L^* = \bigcup\limits_{n \in \mathbb{N}} L^n$. Moreover, let $L^+ = \bigcup\limits_{n \in \mathbb{N} \setminus \{0\}} L^n$.

Definition 2.1 (Regular expressions RegExp*).*

$$e \stackrel{\text{def}}{=} \emptyset \mid \varepsilon \mid a \mid e + e \mid e \circ e \mid e^*, \text{ where } a \in \Sigma.$$

Since $+$ is associative and commutative, we also write $(e_1 + e_2) + e_3$ as $e_1 + e_2 + e_3$ for brevity. We use the abbreviation $e^+ \equiv e \circ e^*$. Moreover, for $\Gamma = \{a_1, \cdots, a_n\} \subseteq \Sigma$, we use the abbreviations $\Gamma \equiv a_1 + \cdots + a_n$ and $\Gamma^* \equiv (a_1 + \cdots + a_n)^*$.

We define $\mathcal{L}(e)$ to be the language defined by e, that is, the set of strings that match e, inductively as follows: $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(a) = \{a\}$, $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$, $\mathcal{L}(e_1 \circ e_2) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2)$, $\mathcal{L}(e_1^*) = (\mathcal{L}(e_1))^*$. In addition, we use $|e|$ to denote the number of symbols occurring in e.

A *nondeterministic finite automaton* (NFA) $\mathcal{A}$ on Σ is a tuple (Q, δ, q_0, F), where Q is a finite set of *states*, $q_0 \in Q$ is the *initial* state, $F \subseteq Q$ is the set of *final* states, and $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*. For a string $w = a_1 \ldots a_n$, a *run* of $\mathcal{A}$ on w is a state sequence $q_0 \ldots q_n$ such that for each $i \in [n]$, $(q_{i-1}, a_i, q_i) \in \delta$. A run $q_0 \ldots q_n$ is *accepting* if $q_n \in F$. A string w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. We use $\mathcal{L}(\mathcal{A})$ to denote the language defined by $\mathcal{A}$, that is, the set of strings accepted by $\mathcal{A}$. We will use $\mathcal{A}, \mathcal{B}, \cdots$ to denote NFAs. For a string $w = a_1 \ldots a_n$, we also use the notation $q_1 \xrightarrow[\mathcal{A}]{w} q_{n+1}$ to denote the fact that there are $q_2, \ldots, q_n \in Q$ such that for each $i \in [n]$, $(q_i, a_i, q_{i+1}) \in \delta$. For an NFA $\mathcal{A} = (Q, \delta, q_0, F)$ and $q, q' \in Q$, we use $\mathcal{A}(q, q')$ to denote the NFA obtained from $\mathcal{A}$ by changing the initial state to q and the set of final states to $\{q'\}$. The *size* of an NFA $\mathcal{A} = (Q, \delta, q_0, F)$, denoted by $|\mathcal{A}|$, is defined as $|Q|$, the number of states. For convenience, we will also call an NFA without initial and final states, that is, a pair (Q, δ), as a *transition graph*.

It is well-known (e.g. see [Hopcroft and Ullman 1979]) that regular expressions and NFAs are expressively equivalent, and generate precisely all *regular languages*. In particular, from a regular expression, an equivalent NFA can be constructed in linear time. Moreover, regular languages are closed under Boolean operations, i.e., union, intersection, and complementation. In particular, given two NFA $\mathcal{A}_1 = (Q_1, \delta_1, q_{0,1}, F_1)$ and $\mathcal{A}_2 = (Q_2, \delta_2, q_{0,2}, F_2)$ on Σ, the intersection $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ is recognised by the *product automaton* $\mathcal{A}_1 \times \mathcal{A}_2$ of $\mathcal{A}_1$ and $\mathcal{A}_2$ defined as $(Q_1 \times Q_2, \delta, (q_{0,1}, q_{0,2}), F_1 \times F_2)$, where δ comprises the transitions $((q_1, q_2), a, (q_1', q_2'))$ such that $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$.

Graph-Theoretical Notation. A DAG (*directed acyclic graph*) G is a finite directed graph (V, E) with no directed cycles, where V (resp. $E \subseteq V \times V$) is a set of vertices (resp. edges). Equivalently, a DAG is a directed graph that has a topological ordering, which is a sequence of the vertices such that every edge is directed from an earlier vertex to a later vertex in the sequence. An edge (v, v') in G is called an *incoming* edge of v' and an *outgoing* edge of v. If $(v, v') \in E$, then v' is called a *successor* of v and v is called a *predecessor* of v'. A *path* π in G is a sequence $v_0 e_1 v_1 \cdots v_{n-1} e_n v_n$ such that for each $i \in [n]$, we have $e_i = (v_{i-1}, v_i) \in E$. The *length* of the path π is the number n of edges in π. If there is a path from v to v' (resp. from v' to v) in G, then v' is said to be *reachable* (resp. *co-reachable*) from v in G. If v is reachable from v' in G, then v' is also called an *ancestor* of v in G. In addition, an edge (v', v'') is said to be reachable (resp. co-reachable) from v if v' is reachable from v (resp. v'' is co-reachable from v). The *in-degree* (resp. *out-degree*) of a vertex v is the number of incoming (resp. outgoing) edges of v. A *subgraph* G' of $G = (V, E)$ is a directed graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Let G' be a subgraph of G. Then $G \setminus G'$ is the graph obtained from G by removing all the edges in G'.

Computational Complexity. In this paper, we study not only decidability but also the complexity of string logics. In particular, we shall deal with the following computational complexity classes (see [Hopcroft and Ullman 1979] for more details): PSPACE (problems solvable in polynomial space and thus in exponential time), and EXPSPACE (problems solvable in exponential space and thus in double exponential time). Verification problems that have complexity PSPACE or beyond (see [Baier and Katoen 2008] for a few examples) have substantially benefited from techniques such as symbolic model checking [McMillan 1993].

3 THE CORE CONSTRAINT LANGUAGE

We will consider the straight-line fragment of string constraints with one-way transducers, ReplaceAll function, and Reverse function.

4 DECISION PROCEDURE

We will present a decision procedure for the core constraint language. Also the non-elementary lower bound by Matt.

5 FRAGMENTS

5.1 One-way transducers, Concatenation, Reverse

5.2 One-way functional transducers, ReplaceAll, Reverse

6 EXTENSIONS

6.1 Two-way Transducers

6.2 Length constraints

7 RELATED WORK

8 CONCLUSION

ACKNOWLEDGMENTS

REFERENCES

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series).* The MIT Press.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.

K. L. McMillan. 1993. *Symbolic model checking.* Kluwer.