A Formal Framework for Stringology

Neerja Mhaskar[1] and Michael Soltys[2]

[1] McMaster University
Dept. of Computing & Software
1280 Main Street West
Hamilton, Ontario L8S 4K1, CANADA
pophlin@mcmaster.ca

[2] California State University Channel Islands
Dept. of Computer Science
One University Drive
Camarillo, CA 93012, USA
michael.soltys@csuci.edu

Abstract. A new formal framework for Stringology is proposed, which consists of a three-sorted logical theory $\mathcal{S}$ designed to capture the combinatorial reasoning about finite words. A witnessing theorem is proven which demonstrates how to extract algorithms for constructing strings from their proofs of existence. Various other applications of the theory are shown. The long term goal of this line of research is to introduce the tools of Proof Complexity to the analysis of strings.

Keywords: proof complexity, string algorithms

1 Introduction

Finite strings are an object of intense scientific interest. This is due partly to their intricate combinatorial properties, and partly to their eminent applicability to such diverse fields as genetics, language processing, and pattern matching. Many techniques have been developed over the years to prove properties of finite strings, such as suffix arrays, border arrays, and decomposition algorithms such as Lyndon factorization. However, there is no unifying theory or framework, and often the results consist in clever but ad hoc combinatorial arguments. In this paper we propose a unifying theory of strings based on a three sorted logical theory, which we call $\mathcal{S}$. By engaging in this line of research, we hope to bring the richness of the advanced field of Proof Complexity to Stringology, and eventually create a unifying theory of strings.

The great advantage of this approach is that proof theory integrates proofs and computations; this can be beneficial to Stringology as it allows us to extract efficient algorithms from proofs of assertions. More concretely, if we can prove in $\mathcal{S}$ a property of strings of the form: "for all strings V, there exists a string U with property α," i.e., $\exists U \leq t\alpha(U, V)$, then we can mechanically extract an actual algorithm which computes U for any given V. For example, suppose that we show that $\mathcal{S}$ proves that every string has a certain decomposition; then, we can actually extract a procedure from the proof for computing such decompositions.

For a background on Proof Complexity see [3] which contains a complete treatment of the subject; we follow its methodology and techniques for defining our theory $\mathcal{S}$. We also use some rudimentary λ-calculus from [6] to define string constructors in our language.

Neerja Mhaskar, Michael Soltys: A Formal Framework for Stringology, pp. 90–103.
Proceedings of PSC 2015, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05787-2 © Czech Technical University in Prague, Czech Republic

2 Formalizing the theory of finite strings

We propose a three sorted theory that formalizes the reasoning about finite strings. We call our theory $\mathcal{S}$. The three sorts are *indices*, *symbols*, and *strings*. We start by defining a convenient and natural language for making assertions about strings.

2.1 The language of strings $\mathcal{L}_{\mathcal{S}}$

Definition 1. *$\mathcal{L}_{\mathcal{S}}$, the language of strings, is defined as follows:*

$$\mathcal{L}_{\mathcal{S}} = [0_{\text{index}}, 1_{\text{index}}, +_{\text{index}}, -_{\text{index}}, \cdot_{\text{index}}, \text{div}_{\text{index}}, \text{rem}_{\text{index}},$$

$$\mathbf{0}_{\text{symbol}}, \sigma_{\text{symbol}}, \text{cond}_{\text{symbol}}, ||_{\text{string}}, e_{\text{string}}; <_{\text{index}}, =_{\text{index}} <_{\text{symbol}}, =_{\text{symbol}}, =_{\text{string}}]$$

The table below explains the intended meaning of each symbol.

Formal	**Informal**	**Intended Meaning**
		Index
0_{index}	0	the integer zero
1_{index}	1	the integer one
$+_{\text{index}}$	$+$	integer addition
$-_{\text{index}}$	$-$	bounded integer subtraction
$\cdot_{\text{index}}$	$\cdot$	integer multiplication (we also just use juxtaposition)
$\text{div}_{\text{index}}$	div	integer division
$\text{rem}_{\text{index}}$	rem	remainder of integer division
$<_{\text{index}}$	$<$	less-than for integers
$=_{\text{index}}$	$=$	equality for integers
		Alphabet symbol
$\mathbf{0}_{\text{symbol}}$	$\mathbf{0}$	default symbol in every alphabet
σ_{symbol}	σ	unary function for generating more symbols
$<_{\text{symbol}}$	$<$	ordering of alphabet symbols
$\text{cond}_{\text{symbol}}$	cond	a conditional function
$=_{\text{symbol}}$	$=$	equality for alphabet symbols
		String
$\|\|_{\text{string}}$	$\|\|$	unary function for string length
e_{string}	e	binary fn. for extracting the i-th symbol from a string
$=_{\text{string}}$	$=$	string equality

Note that in practice we use the informal language symbols as otherwise it would be tedious to write terms, but the meaning will be clear from the context. When we write $i \leq j$ we abbreviate the formula $i < j \vee i = j$.

2.2 Syntax of $\mathcal{L}_{\mathcal{S}}$

We use metavariables $i, j, k, l, \ldots$ to denote indices, metavariables $u, v, w, \ldots$ to denote alphabet symbols, and metavariables $U, V, W, \ldots$ to denote strings. When a variable can be of any type, i.e., a meta-meta variable, we write it as $x, y, z \ldots$ We are going to use t to denote an index term, for example $i + j$, and we are going to use s to denote a symbol term, for example $\sigma\sigma\sigma\mathbf{0}$. We let T denote string terms. We are going to use Greek letters $\alpha, \beta, \gamma, \ldots$ to denote formulas.

Definition 2. *$\mathcal{L}_{\mathbb{S}}$-Terms are defined by structural induction as follows:*

1. *Every index variable is a term of type index (index term).*
2. *Every symbol variable is a term of type symbol (symbol term).*
3. *Every string variable is a term of type string (string term).*
4. *If t_1, t_2 are index terms, then so are $(t_1 \circ t_2)$ where $\circ \in \{+, -, \cdot\}$, and $\mathrm{div}(t_1, t_2)$, $\mathrm{rem}(t_1, t_2)$.*
5. *If s is a symbol term then so is σs.*
6. *If T is a string term, then $|T|$ is an index term.*
7. *If t is an index term, and T is a string term, then $e(T, t)$ is a symbol term.*
8. *All constant functions $(0_{\text{index}}, 1_{\text{index}}, \mathbf{0}_{\text{symbol}})$ are terms.*

We are going to employ the lambda operator λ for building terms of type string; we want our theory to be constructive, and we want to have a method for constructing bigger strings from smaller ones.

Definition 3. *Given a term t of type index, and given a term s of type symbol, then the following is a term T of type string:*

$$\lambda i \langle t, s \rangle. \tag{1}$$

The idea is that T is a string of length t and the i_0-th symbol of the string is obtained by evaluating s at i_0, i.e., by evaluating $s(i_0/i)$. Note that $s(i_0/i)$ is the term obtained by replacing every *free* occurrence of i in s with i_0. Note that (1) is a λ-term, meaning that i is considered to be a bound variable. For examples of string constructors see Section 2.4.

Definition 4. *$\mathcal{L}_{\mathbb{S}}$-Formulas are defined by structural induction as follows:*

1. *If t_1, t_2 are two index terms, then $t_1 < t_2$ and $t_1 = t_2$ are atomic formulas.*
2. *If s_1, s_2 are symbol terms, then $s_1 < s_2$ and $s_1 = s_2$ are atomic formulas.*
3. *If T_1, T_2 are two string terms, then $T_1 = T_2$ is an atomic formula.*
4. *If α, β are formulas (atomic or not), the following are also formulas:*

$$\neg\alpha, (\alpha \wedge \beta), (\alpha \vee \beta), \forall x \alpha, \exists x \alpha.$$

We are interested in a restricted mode of quantification. We say that an index quantifier is bounded if it is of the form $\exists i \leq t$ or $\forall i \leq t$, where t is a term of type index and i does not occur free in t. Similarly, we say that a string quantifier is bounded if it is of the form $\exists U \leq t$ or $\forall U \leq t$, where this means that $|U| \leq t$ and U does not occur in t.

Definition 5. *Let Σ_0^B be the set of $\mathcal{L}_{\mathbb{S}}$-formulas without string or symbol quantifiers, where all index quantifiers (if any) are bounded. For $i > 0$, let Σ_i^B (Π_i^B) be the set of $\mathcal{L}_{\mathbb{S}}$ formulas of the form: once the formula is put in prenex form, there are i alternations of bounded string quantifiers, starting with an existential (universal) one, and followed by a Σ_0^B formula.*

Given a formula α, and two terms s_1, s_2 of type symbol, then $\mathrm{cond}(\alpha, s_1, s_2)$ is a term of type symbol. We want our theory to be strong enough to prove interesting theorems, but not too strong so that proofs yield feasible algorithms. For this reason we will restrict the α in the $\mathrm{cond}(\alpha, s_1, s_2)$ to be Σ_0^B. Thus, given such an α and assignments of values to its free variables, we can evaluate the truth value of α, and output the appropriate s_i, in polytime – see Lemma 8.

The alphabet symbols are as follows, $\mathbf{0}$, $\sigma\mathbf{0}$, $\sigma\sigma\mathbf{0}$, $\sigma\sigma\sigma\mathbf{0}$, ..., that is, the unary function σ allows us to generate as many alphabet symbols as necessary. We are going to abbreviate these symbols as $\sigma_0, \sigma_1, \sigma_2, \sigma_3, \ldots$ In a given application in Stringology, an alphabet of size three would be given by $\Sigma = \{\sigma_0, \sigma_1, \sigma_2\}$, where $\sigma_0 < \sigma_1 < \sigma_2$, inducing a standard lexicographic ordering. We make a point of having an alphabet of any size in the language, rather than a fixed constant size alphabet, as this allows us to formalize arguments of the type: given a particular structure describing strings, show that such strings require alphabets of a given size (see [2]).

2.3 Semantics of $\mathcal{L}_{\mathcal{S}}$

We denote a structure for $\mathcal{L}_{\mathcal{S}}$ with $\mathcal{M}$. A structure is a way of assigning values to the terms, and truth values to the formulas. We base our presentation on [3, §II.2.2]. We start with a non-empty set M called the universe. The variables in any $\mathcal{L}_{\mathcal{S}}$ are intended to range over M. Since our theory is three sorted, the universe $M = (I, \Sigma, S)$, where I denotes the set of indices, Σ the set of alphabet symbols, and S the set of strings.

We start by defining the semantics for the three 0-ary (constant) function symbols:

$$0^{\mathcal{M}}_{\text{index}} \in I, \quad 1^{\mathcal{M}}_{\text{index}} \in I, \quad 0^{\mathcal{M}}_{\text{symbol}} \in \Sigma,$$

for the two unary function symbol:

$$\sigma^{\mathcal{M}}_{\text{symbol}} : \Sigma \longrightarrow \Sigma, \quad ||^{\mathcal{M}}_{\text{string}} : S \longrightarrow I,$$

for the six binary function symbols:

$$+^{\mathcal{M}}_{\text{index}} : I^2 \longrightarrow I, \quad -^{\mathcal{M}}_{\text{index}} : I^2 \longrightarrow I, \quad \cdot^{\mathcal{M}}_{\text{index}} : I^2 \longrightarrow I$$
$$\text{div}^{\mathcal{M}}_{\text{index}} : I^2 \longrightarrow I, \quad \text{rem}^{\mathcal{M}}_{\text{index}} : I^2 \longrightarrow I, \quad e^{\mathcal{M}}_{\text{string}} : S \times I \longrightarrow \Sigma.$$

With the function symbols defined according to $\mathcal{M}$, we now associate relations with the predicate symbols, starting with the five binary predicates:

$$<^{\mathcal{M}}_{\text{index}} \subseteq I^2, \quad =^{\mathcal{M}}_{\text{index}} \subseteq I^2, \quad <^{\mathcal{M}}_{\text{symbol}} \subseteq \Sigma^2, \quad =^{\mathcal{M}}_{\text{symbol}} \subseteq \Sigma^2, \quad =^{\mathcal{M}}_{\text{string}} \subseteq S^2,$$

and finally we define the conditional function as follows: $\text{cond}^{\mathcal{M}}_{\text{symbol}}(\alpha, s_1, s_2)$ evaluates to $s_1^{\mathcal{M}}$ if $\alpha^{\mathcal{M}}$ is true, and to $s_2^{\mathcal{M}}$ otherwise.

Note that $=^{\mathcal{M}}$ must always evaluate to true equality for all types; that is, equality is hardwired to always be equality. However, all other function symbols and predicates can be evaluated in an arbitrary way (that respects the given arities).

Definition 6. *An* object assignment τ *for a structure* $\mathcal{M}$ *is a mapping from variables to the universe* $M = (I, \Sigma, S)$*, that is,* M *consists of three sets that we call* indices*,* alphabet symbols*, and* strings*.*

The three sorts are related to each other in that S can be seen as a function from I to Σ, i.e., a given $U \in S$ is just a function $U : I \longrightarrow \Sigma$. In Stringology we are interested in the case where a given U may be arbitrarily long but it maps I to a relatively small set of Σ: for example, binary strings map into $\{0, 1\} \subset \Sigma$. Since the range of U is relatively small this leads to interesting structural questions about the mapping: repetitions and patterns.

We start by defining τ on terms: $\tau^{\mathcal{M}}[\sigma]$. Note that if $m \in M$ and x is a variable, then $\tau(m/x)$ denotes the object assignment τ but where we specify that the variable x must evaluate to m.

We define the evaluation of a term t under $\mathcal{M}$ and τ, $t^{\mathcal{M}}[\tau]$, by structural induction on the definition of terms given in Section 2.1. First, $x^{\mathcal{M}}[\tau]$ is just $\tau(x)$, for each variable x. We must now define object assignments for all the functions. Recall that t, t_1, t_2 are index terms, s is a symbol term and T is a string term.

$$(t_1 \circ_{\text{index}} t_2)^{\mathcal{M}}[\tau] = (t_1^{\mathcal{M}}[\tau] \circ_{\text{index}}^{\mathcal{M}} t_2^{\mathcal{M}}[\tau]),$$

where $\circ \in \{+, -, \cdot\}$ and

$$(\text{div}(t_1, t_2))^{\mathcal{M}}[\tau] = \text{div}^{\mathcal{M}}(t_1^{\mathcal{M}}[\tau], t_2^{\mathcal{M}}[\tau]),$$

$$(\text{rem}(t_1, t_2))^{\mathcal{M}}[\tau] = \text{rem}^{\mathcal{M}}(t_1^{\mathcal{M}}[\tau], t_2^{\mathcal{M}}[\tau]).$$

and for symbol terms we have:

$$(\sigma s)^{\mathcal{M}}[\tau] = \sigma^{\mathcal{M}}(s^{\mathcal{M}}[\tau]).$$

Finally, for string terms:

$$|\mathrm{T}|^{\mathcal{M}}[\tau] = |(\mathrm{T}^{\mathcal{M}}[\tau])|.$$

$$(\mathrm{e}(T, t))^{\mathcal{M}}[\tau] = \mathrm{e}^{\mathcal{M}}(T^{\mathcal{M}}[\tau], t^{\mathcal{M}}[\tau]).$$

Given a formula α, the notation $\mathcal{M} \vDash \alpha[\tau]$, which we read as "$\mathcal{M}$ satisfies α under τ" is also defined by structural induction. We start with the basis case:

$$\mathcal{M} \vDash (s_1 <_{\text{symbol}} s_2)[\tau] \iff (s_1^{\mathcal{M}}[\tau], s_2^{\mathcal{M}}[\tau]) \in <_{\text{symbol}}^{\mathcal{M}}.$$

We deal with the other atomic predicates in a similar way:

$$\mathcal{M} \vDash (t_1 <_{\text{index}} t_2)[\tau] \iff (t_1^{\mathcal{M}}[\tau], t_2^{\mathcal{M}}[\tau]) \in <_{\text{index}}^{\mathcal{M}},$$

$$\mathcal{M} \vDash (t_1 =_{\text{index}} t_2)[\tau] \iff t_1^{\mathcal{M}}[\tau] = t_2^{\mathcal{M}}[\tau],$$

$$\mathcal{M} \vDash (s_1 =_{\text{symbol}} s_2)[\tau] \iff s_1^{\mathcal{M}}[\tau] = s_2^{\mathcal{M}}[\tau],$$

$$\mathcal{M} \vDash (T_1 =_{\text{string}} T_2)[\tau] \iff T_1^{\mathcal{M}}[\tau] = T_2^{\mathcal{M}}[\tau].$$

Now we deal with Boolean connectives:

$$\mathcal{M} \vdash (\alpha \wedge \beta)[\tau] \iff \mathcal{M} \vDash \alpha[\tau] \text{ and } \mathcal{M} \vDash \beta[\tau],$$

$$\mathcal{M} \vdash \neg\alpha[\tau] \iff \mathcal{M} \nvDash \alpha[\tau],$$

$$\mathcal{M} \vdash (\alpha \vee \beta)[\tau] \iff \mathcal{M} \vDash \alpha[\tau] \text{ or } \mathcal{M} \vDash \beta[\tau].$$

Finally, we show how to deal with quantifiers, where the object assignment τ plays a crucial role:

$$\mathcal{M} \vDash (\exists x\alpha)[\tau] \iff \mathcal{M} \vDash \alpha[\tau(m/x)] \text{ for some } m \in M,$$

$$\mathcal{M} \vDash (\forall x\alpha)[\tau] \iff \mathcal{M} \vDash \alpha[\tau(m/x)] \text{ for all } m \in M.$$

Definition 7. *Let $\mathbb{S} = (\mathbb{N}, \Sigma, S)$ denote the standard model for strings, where $\mathbb{N}$ are the standard natural numbers, including zero, $\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \ldots\}$ where the alphabet symbols are the ordered sequence $\sigma_0 < \sigma_1 < \sigma_2, \ldots$, and where S is the set of functions $U : I \longrightarrow \Sigma$, and where all the function and predicate symbols get their standard interpretations.*

Lemma 8. *Given any formula $\alpha \in \Sigma_0^B$, and a particular object assignment τ, we can verify $\mathbb{S} \vDash \alpha[\tau]$ in polytime in the lengths of the strings and values of the indices in α.*

Proof. We first show that evaluating a term t, i.e., computing $t^{\mathbb{S}}[\tau]$, can be done in polytime. We do this by structural induction on t. If t is just a variable then there are three cases: i, u, U. $i^{\mathbb{S}}[\tau] = \tau(i) \in \mathbb{N}$, $u^{\mathbb{S}}[\tau] = \tau(u) \in \Sigma$, and $U^{\mathbb{S}}[\tau] = \tau(U) \in S$. Note that the assumption is that computing $\tau(x)$ is for free, as τ is given as a table which states which free variable gets replaced by what concrete value. Recall that all index values are assumed to be given in unary, and all the function operations we have are clearly polytime in the values of the arguments (index addition, subtraction, multiplication, etc.).

Now suppose that we have an atomic formula such as $(t_1 < t_2)^{\mathbb{S}}[\tau]$. We already established that $t_1^{\mathbb{S}}[\tau]$ and $t_2^{\mathbb{S}}[\tau]$ can be computed in polytime, and comparing integers can also be done in polytime. Same for other atomic formulas, and the same holds for Boolean combinations of formulas. What remains is to consider quantification; but we are only allowed bounded index quantification: $(\exists i \leq t\alpha)^{\mathbb{S}}[\tau]$, and $(\exists i \leq t\alpha)^{\mathbb{S}}[\tau]$. This is equivalent to computing:

$$\bigvee_{j=0}^{t^{\mathbb{S}}[\tau]} \alpha^{\mathbb{S}}[\tau(j/i)], \text{ and } \bigwedge_{j=0}^{t^{\mathbb{S}}[\tau]} \alpha^{\mathbb{S}}[\tau(j/i)].$$

Clearly this can be done in polytime. □

2.4 Examples of string constructors

The string 000 can be represented by:

$$\lambda i \langle 1 + 1 + 1, \mathbf{0} \rangle.$$

Given an integer n, let $\hat{n}$ abbreviate the term $1 + 1 + \cdots + 1$ consisting of n many 1s. Using this convenient notation, a string of length 8 of alternating 1s and 0s can be represented by:

$$\lambda i \langle \hat{8}, \mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0}) \rangle. \tag{2}$$

Note that this example illustrates that indices are going to be effectively encoded in unary; this is fine as we are proposing a theory for strings, and so unary indices are an encoding that is linear in the length of the string. The same point is made in [3], where the indices are assumed to be encoded in unary, because the main object under investigation are binary strings, and the complexity is measured in the lengths of the strings, and unary encoded indices are proportional to those lengths.

Also note that there are various ways to represent the same string; for example, the string given by (2) can also be written thus:

$$\lambda i \langle \hat{2} \cdot \hat{4}, \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0}) \rangle. \tag{3}$$

For convenience, we define the empty string ε as follows:

$$\varepsilon := \lambda i\langle 0, \mathbf{0}\rangle.$$

Let U be a binary string, and suppose that we want to define $\bar{U}$, which is U with every 0 (denoted $\mathbf{0}$) flipped to 1 (denote $\sigma\mathbf{0}$), and every 1 flipped to 0. We can define $\bar{U}$ as follows:

$$\bar{U} := \lambda i\langle |U|, \mathrm{cond}(e(U,i) = \mathbf{0}, \sigma\mathbf{0}, \mathbf{0}\rangle.$$

We can also define a string according to properties of positions of indices; suppose we wish to define a binary string of length n which has one in all positions which are multiples of 3:

$$U_3 := \lambda i\langle \hat{n}, \mathrm{cond}(\exists j \leq n(i = j + j + j), \sigma\mathbf{0}, \mathbf{0})\rangle.$$

Note that both $\bar{U}$ and U_3 are defined with the conditional function where the formula α conforms to the restriction: variables are either free (like U in $\bar{U}$), or, if quantified, all such variables are bounded and of type index (like j in U_3).

Note that given a string W, $|W|$ is its length. However, we number the positions of a string starting at zero, and hence the last position is $|W| - 1$. For $j \geq |W|$ we are going to define a string to be just $\mathbf{0}$s.

Suppose we want to define the reverse of a string, namely if $U = u_0 u_1 \cdots u_{n-1}$, then its reverse is $U^R = u_{n-1} u_{n-2} \cdots u_0$. Then,

$$U^R := \lambda i\langle |U|, e(U, (|U| - 1) - i)\rangle,$$

and the concatenation of two strings, which we denote as "$\cdot$", can be represented as follows:

$$U \cdot V := \lambda i\langle |U| + |V|, \mathrm{cond}(i < |U|, e(U,i), e(V, i - |U|))\rangle. \tag{4}$$

2.5 Axioms of the theory $\mathcal{S}$

We assume that we have the standard equality axioms which assert that equality is true equality — see [1, §2.2.1]. So we won't give those axioms explicitly.

Since we are going to use the rules of Gentzen's calculus, LK, we present the axioms as Gentzen's sequents, that is, they are of the form $\Gamma \rightarrow \Delta$, where Γ, Δ are coma-separated lists of formulas. That is, a sequent is of the form:

$$\alpha_1, \alpha_2, \ldots, \alpha_n \rightarrow \beta_1, \beta_2, \ldots, \beta_m,$$

where n or m (or both) may be zero, that is, Γ or Δ (or both) may be empty. The semantics of sequents is as follows: a sequent is valid if for any structure $\mathcal{M}$ that satisfies all the formulas in Γ, satisfies at least one formula in Δ. Using the standard Boolean connectives this can be state as follows: $\neg \bigwedge_i \alpha_i \vee \bigvee_j \beta_j$, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

The index axioms are the same as 2-BASIC in [3, pg. 96], plus we add four more axioms (B7 and B15, B8 and B16) to define bounded subtraction, as well as division and remainder functions. Keep in mind that a formula α is equivalent to a sequent $\rightarrow \alpha$, and so, for readability we sometimes mix the two.

Index Axioms	
B1. $i+1 \neq 0$	B9. $i \leq j, j \leq i \rightarrow i = j$
B2. $i+1 = j+1 \rightarrow i = j$	B10. $i \leq i+j$
B3. $i+0 = i$	B11. $0 \leq i$
B4. $i+(j+1) = (i+j)+1$	B12. $i \leq j \vee j \leq i$
B5. $i \cdot 0 = 0$	B13. $i \leq j \leftrightarrow i < j+1$
B6. $i \cdot (j+1) = (i \cdot j) + i$	B14. $i \neq 0 \rightarrow \exists j \leq i(j+1 = i)$
B7. $i \leq j, i+k = j \rightarrow j - i = k$	B15. $i \not\leq j \rightarrow j - i = 0$
B8. $j \neq 0 \rightarrow \mathrm{rem}(i,j) < j$	B16. $j \neq 0 \rightarrow i = j \cdot \mathrm{div}(i,j) + \mathrm{rem}(i,j)$

The alphabet axioms express that the alphabet is totally ordered according to "$<$" and define the function cond.

Alphabet Axioms
B17. $u \lneqq \sigma u$
B18. $u < v, v < w \rightarrow u < w$
B19. $\alpha \rightarrow \mathrm{cond}(\alpha, u, v) = u$
B20. $\neg\alpha \rightarrow \mathrm{cond}(\alpha, u, v) = v$

Note that α in cond is a formula with the following restrictions: it only allows bounded index quantifiers and hence evaluates to true or false once all free variables have been assigned values. Hence cond always yields the symbol term s_1 or the symbol term s_2, according to the truth value of α.

Note that the alphabet symbol type is defined by four axioms, B17–B20, two of which define the cond function. These four axioms define symbols to be ordered "place holders" and nothing more. This is consistent with alphabet symbols in classical Stringology, where there are no operations defined on them (for example, we do not add or multiply alphabet symbols).

Finally, these are the axioms governing strings:

String Axioms
B21. $\lvert\lambda i\langle t, s\rangle\rvert = t$
B22. $j < t \rightarrow e(\lambda i\langle t, s\rangle, j) = s(j/i)$
B23. $\lvert U\rvert \leq j \rightarrow e(U, j) = \mathbf{0}$
B24. $\lvert U\rvert = \lvert V\rvert, \forall i < \lvert U\rvert e(U,i) = e(V,i) \rightarrow U = V$

Note that axioms B22–24 define the structure of a string. In our theory, a string can be given as a variable, or it can be constructed. Axiom B21 defines the length of the constructed strings, and axiom B22 shows that if j is less than the length of the string, then the symbol in position j is given by substituting j for all the free occurrences of i in s; this is the meaning of $s(j/i)$. On the other hand, B23 says that if j is greater or equal to the length of a string, then $e(U, j)$ defaults to $\mathbf{0}$. The last axioms, B24, says that if two strings U and V have the same length, and the corresponding symbols are equal, then the two strings are in fact equal.

In axiom B24 there are three types of equalities, from left to right: index, symbol, and string, and so B24 is the axiom that ties all three sorts together. Note that formally strings are infinite ordered sequences of alphabet symbols. But we conclude that they are equal based on comparing finitely many entries ($\forall i < |U| e(U, i) = e(V, i)$). This works because by B23 we know that for $i \geq |U|$, $e(U, i) = e(V, i) = \mathbf{0}$ (since $|U| = |V|$ by the assumption in the antecedent). A standard string of length n is an object of the form:

$$\sigma_{i_0}, \sigma_{i_1}, \ldots, \sigma_{i_{n-1}}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \ldots,$$

i.e., an infinite string indexed by the natural numbers, where there is a position so that all the elements greater than that position are **0**.

A rich source of insight is to consider non-standard models of a given theory. We have described $\underline{\mathbb{S}}$, the standard theory of strings, which is intended to capture the mental constructs that Stringologists have in mind when working on problems in this field. It would be very interesting to consider non-standard strings that satisfy all the axioms, and yet are not the "usual" object.

2.6 The rules of $\mathcal{S}$

We use the Gentzen's predicate calculus, LK, as presented in [1].

Weak structural rules

exchange-left: $\dfrac{\Gamma_1, \alpha, \beta, \Gamma_2 \to \Delta}{\Gamma_1, \beta, \alpha, \Gamma_2 \to \Delta}$ exchange-right: $\dfrac{\Gamma \to \Delta_1, \alpha, \beta, \Delta_2}{\Gamma \to \Delta_1, \beta, \alpha, \Delta_2}$

contraction-left: $\dfrac{\alpha, \alpha, \Gamma \to \Delta}{\alpha, \Gamma \to \Delta}$ contraction-right: $\dfrac{\Gamma \to \Delta, \alpha, \alpha}{\Gamma \to \Delta, \alpha}$

weakening-left: $\dfrac{\Gamma \to \Delta}{\alpha, \Gamma \to \Delta}$ weakening-right: $\dfrac{\Gamma \to \Delta}{\Gamma \to \Delta, \alpha}$

Cut rule $\dfrac{\Gamma \to \Delta, \alpha \quad \alpha, \Gamma \to \Delta}{\Gamma \to \Delta}$

Rules for introducing connectives

$\neg$-left: $\dfrac{\Gamma \to \Delta, \alpha}{\neg\alpha, \Gamma \to \Delta}$ $\neg$-right: $\dfrac{\alpha, \Gamma \to \Delta}{\Gamma \to \Delta, \neg\alpha}$

$\wedge$-left: $\dfrac{\alpha, \beta, \Gamma \to \Delta}{\alpha \wedge \beta, \Gamma \to \Delta}$ $\wedge$-right: $\dfrac{\Gamma \to \Delta, \alpha \quad \Gamma \to \Delta, \beta}{\Gamma \to \Delta, \alpha \wedge \beta}$

$\vee$-left: $\dfrac{\alpha, \Gamma \to \Delta \quad \beta, \Gamma \to \Delta}{\alpha \vee \beta, \Gamma \to \Delta}$ $\vee$-right: $\dfrac{\Gamma \to \Delta, \alpha, \beta}{\Gamma \to \Delta, \alpha \vee \beta}$

Rules for introducing quantifiers

$\forall$-left: $\dfrac{\alpha(t), \Gamma \to \Delta}{\forall x \alpha(x), \Gamma \to \Delta}$ $\forall$-right: $\dfrac{\Gamma \to \Delta, \alpha(b)}{\Gamma \to \Delta, \forall x \alpha(x)}$

$\exists$-left: $\dfrac{\alpha(b), \Gamma \to \Delta}{\exists x \alpha(x), \Gamma \to \Delta}$ $\exists$-right: $\dfrac{\Gamma \to \Delta, \alpha(t)}{\Gamma \to \Delta, \exists x \alpha(x)}$

Note that b must be free in Γ, Δ.

Induction rule

$$\text{Ind:} \quad \frac{\Gamma, \alpha(i) \to \alpha(i+1), \Delta}{\Gamma, \alpha(0) \to \alpha(t), \Delta}$$

where i does not occur free in Γ, Δ, and t is any term of type index. By restricting the quantifier structure of α, we control the strength of this induction. We call Σ_i^B-Ind to be the induction rule where α is restricted to be in Σ_i^B. We are mainly interested in Σ_i^B-Ind where $i = 0$ or $i = 1$.

Definition 9. *Let $\mathcal{S}_i$ to be the set of formulas (sequents) derivable from the axioms B1-24 using the rules of LK, where the α formula in* cond *is restricted to be in Σ_0^B and where we use Σ_i^B-Ind.*

Theorem 10 (Cut-Elimination). *If Φ is a $\mathcal{S}_i$ proof of a formula α, then Φ can always be converted into a Φ' $\mathcal{S}_i$ proof where the cut rule is applied only to formulas in Σ_i^B.*

We do not prove Theorem 10, but the reader is pointed to [5] to see the type of reasoning that is required. The point of the Cut-Elimination Theorem is that in any $\mathcal{S}_i$ proof we can always limit all the intermediate formulas to be in Σ_i^B, i.e., we do not need to construct intermediate formulas whose quantifier complexity is more than that of the conclusion.

As an example of the use of S_i we outline an S_0 proof of the equality of (2) and (3). First note that by axiom B21 we have that:

$$|\lambda i\langle \hat{8}, \mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0})\rangle| = \hat{8}$$
$$|\lambda i\langle \hat{2} \cdot \hat{4}, \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0})\rangle| = \hat{2} \cdot \hat{4},$$

and by axioms B1-16 we can prove that $\hat{8} = \hat{2} \cdot \hat{4}$ (the reader is encouraged to fill in the details), and so we can conclude by transitivity of equality (equality is always true equality) that:

$$|\lambda i\langle \hat{8}, \mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0})\rangle| = |\lambda i\langle \hat{2} \cdot \hat{4}, \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0})\rangle|.$$

Now we have to show that:

$$\forall i < \hat{8}(\mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0}) = \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0})) \quad (5)$$

and then, using axiom B24 and some cuts on Σ_0^B formulas we can prove that in fact the two terms given by (2) and (3) are equal.

In order to prove (5) we show that:

$$i < \hat{8} \wedge (\mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0}) = \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0})) \quad (6)$$

and then we can introduce the quantifier with $\forall$-intro right. We prove (6) by proving:

$$i < \hat{8} \rightarrow \mathrm{cond}(\exists j \leq i(j + j = i), \mathbf{0}, \sigma\mathbf{0}) = \mathrm{cond}(\exists j \leq i(j + j = i + 1), \sigma\mathbf{0}, \mathbf{0}) \quad (7)$$

Now to prove (7) we have to show that:

$$\mathcal{S}_0 \vdash \exists j \leq i(j + j = i) \leftrightarrow \neg\exists j \leq i(j + j = i + 1),$$

which again is left to the reader. Then, using B19 and B20 we can show (7).

3 Witnessing theorem for $\mathcal{S}$

Recall that $\mathcal{S}_1$ is our string theory restricted to Σ_1^B-Ind. For convenience, we sometimes use the notation bold-face V, $\boldsymbol{V}$, to denote several string variables, i.e., $\boldsymbol{V} = V_1, V_2, \ldots, V_\ell$.

We now prove the main theorem of the paper, showing that if we manage to prove in $\mathcal{S}_1$ the existence of a string U with some given properties, then in fact we can construct such a string with a polytime algorithm.

Theorem 11 (Witnessing). *If $\mathcal{S}_1 \vdash \exists \boldsymbol{U} \leq t\alpha(U, \boldsymbol{V})$, then it is possible to compute $\boldsymbol{U}$ in polynomial time in the total length of all the string variables in $\boldsymbol{V}$ and the value of all the free index variables in α.*

Proof. We give a bare outline of the proof of the Witnessing theorem.

By Lemma 8 we know that we can evaluate any $\mathcal{L}_{\mathcal{S}}$-term in $\underline{\mathbb{S}}$ in polytime in the length of the free string variables and the values of the index variables. In order to simplify the proof we show it for $\mathcal{S}_1 \vdash \exists U \leq t\alpha(U, \boldsymbol{V})$, i.e., U is a single string variable rather than a set, i.e., rather than a block of bounded existential string quantifiers. The general proof is very similar.

We argue by induction on the number of lines in the proof of $\exists U \leq t\alpha(U, \boldsymbol{V})$ that U can be witnessed by a polytime algorithm. Each line in the proof is either an axiom (see Section 2.5), or follows from previous lines by the application of a rule (see Section 2.6). By Theorem 10 we know that all the formulas in the $\mathcal{S}_1$ proof of $\exists U \leq t\alpha(U, \boldsymbol{V})$ can be restricted to be Σ_1^B. It is this fundamental application of Cut-Elimination that allows us to prove our Witnessing theorem.

The Basis Case is simple as the axioms have no string quantifiers. In the induction step the two interesting cases are $\exists$-right and the induction rule. In the former case we have:

$$\exists\text{-right: } \frac{|T| \leq t, \Gamma \rightarrow \Delta, \alpha(T, \boldsymbol{V}, \boldsymbol{i})}{\Gamma \rightarrow \Delta, \exists U \leq t\alpha(U, \boldsymbol{V}, \boldsymbol{i})}$$

which is the $\exists$-right rule adapted to the case of bounded string quantification. We use $\boldsymbol{V}$ to denote all the free string variables, and $\boldsymbol{i}$ to denote explicitly all the free index variables. Then U is naturally witnessed by the function f:

$$f(\boldsymbol{A}, \boldsymbol{b}) := T^{\mathbb{S}}[\tau(\boldsymbol{A}/\boldsymbol{V})(\boldsymbol{b}/\boldsymbol{i})].$$

Note that f is polytime as evaluating T under $\underline{\mathbb{S}}$ and any object assignment can by done in polytime by Lemma 8.

The induction case is a little bit more involved. We restate the rule as follows in order to make all the free variables more explicit:

$$\frac{U \leq t, \alpha(U, \boldsymbol{V}, i, \boldsymbol{j}) \rightarrow \exists U \leq t\alpha(U, \boldsymbol{V}, i+1, \boldsymbol{j})}{U \leq t, \alpha(U, \boldsymbol{V}, 0, \boldsymbol{j}) \rightarrow \exists U \leq t\alpha(U, \boldsymbol{V}, t', \boldsymbol{j})}$$

where we ignore Γ, Δ for clarity, and we ignore existential quantifiers on the left side, as it is quantifiers on the right side that we are interested in witnessing. The algorithm is clear: suppose we have a U such that $\alpha(U, \boldsymbol{V}, 0, \boldsymbol{V})$ is satisfied. Use top of rule to compute U's for $i = 1, 2, \ldots, t^{\mathbb{S}}[\tau]$. $\square$

4 Application of $\mathcal{S}$ to Stringology

In this section we state various basic Stringology constructions as $\mathcal{L}_{\mathcal{S}}$ formulas.

4.1 Subwords

The prefix, suffix, and subword are basic constructs of a given string V. They can be given easily as $\mathcal{L}_{\mathbb{S}}$-terms as follows: $\lambda k\langle i, e(V,k)\rangle$, $\lambda k\langle i, e(V, |V|-i+1+k)\rangle$, and since any subword is the prefix of some suffix, it can also be given easily.

We can state that U is a prefix of V with the Σ_0^B predicate:

$$\mathrm{pre}(U,V) := \exists i \leq |V| (U = \lambda k\langle i, e(V,k)\rangle),$$

The predicates for suffix $\mathrm{suf}(U,V)$ and subword $\mathrm{sub}(U,V)$ predicates can be defined with Σ_0^B formulas in a similar way.

4.2 Counting symbols

Suppose that we want to count the number of occurrences of a particular symbol σ_i in a given string U; this can be defined with the notation $(U)_{\sigma_i}$, but we need to define this function with a new axiom (it seems that the language given thus far is not suitable for defining $(U)_{\sigma_i}$ with a term). First, define the projection of a string U according to σ_i as follows:

$$U|_{\sigma_i} := \lambda k\langle |U|, \mathrm{cond}(e(U,k) = \sigma_i, \sigma_1, \sigma_0)\rangle.$$

That is, $U|_{\sigma_i}$ is effectively a binary string with 1s where U had σ_i, and 0s everywhere else, and of the same length as U. Thus, counting σ_i's in U is the same as counting 1's in $U|_{\sigma_i}$. Given a binary string V, we define $(V)_{\sigma_1}$ as follows:

C1. $|V| = 0 \rightarrow (V)_{\sigma_1} = 0$
C2. $|V| \geq 1, e(V,0) = \sigma_0 \rightarrow (V)_{\sigma_1} = (\lambda i\langle |V|-1, e(V,i+1)\rangle)_{\sigma_1}$
C3. $|V| \geq 1, e(V,0) = \sigma_1 \rightarrow (V)_{\sigma_1} = 1 + (\lambda i\langle |V|-1, e(V,i+1)\rangle)_{\sigma_1}$

Having defined $(U)_{\sigma_1}$ with axioms C1-3, and $U|_{\sigma_i}$ as a term in $\mathcal{L}_{\mathbb{S}}$, we can now define $(U)_{\sigma_i}$ as follows: $(U|_{\sigma_i})_{\sigma_1}$. Note that C1-3 are Σ_0^B sequents.

4.3 Borders and border arrays

Suppose that we want to define a border array. First define the border predicate which asserts that the string V has a border of size i; note that by definition a border is a (proper) prefix equal to a (proper) suffix. So let:

$$\mathrm{Brd}(V,i) := \lambda k\langle i, e(V,k)\rangle = \lambda k\langle i, e(V, |V|-i+1+k)\rangle \wedge i < |V|,$$

We now want to state that i is the largest possible border size:

$$\mathrm{MaxBrd}(V,i) := \mathrm{Brd}(V,i) \wedge (\neg\mathrm{Brd}(V,i+1) \vee |U| = |V|-1).$$

Thus, if we want to define the function $\mathrm{BA}(V,i)$, which is the border array for V indexed by i, we can define it by adding the following as an axiom:

$$\mathrm{MaxBrd}(\lambda k\langle i, e(V,k)\rangle, \mathrm{BA}(V,i)).$$

4.4 Periodicity

See [4, pg. 10] for the definition of a period of a string, but for our purpose let us define $p = |U|$ to be a period of V if $V = U^r U'$ where U' is some prefix, possibly empty, of U. The Periodicity Lemma state the following: Suppose that p and q are two periods of V, $|V| = n$, and $d = \gcd(p, q)$. Then, if $p + q \le n + d$, then d is also a period of V.

Let $\mathrm{Prd}(V, p)$ be true if p is a period of the string V. Note that U is a border of a string V if and only if $p = |V| - |U|$ is a period of V. Using this observation we can define the predicate for a period as a Σ_0^B formula:

$$\mathrm{Prd}(V, p) := \exists i < |V|(p = |V| - i \wedge \mathrm{Brd}(V, i))$$

We can state with a Σ_0^B formula that $d = \gcd(i, j)$: $\mathrm{rem}(d, i) = \mathrm{rem}(d, j) = 0$, and $\mathrm{rem}(d', i) = \mathrm{rem}(d', j) = 0 \supset d' \le d$. We can now state the Periodicity Lemma as the sequent $\mathrm{PL}(V, p, q)$ where all formulas are Σ_0^B:

$$\mathrm{Prd}(V, p), \mathrm{Prd}(V, q), \exists d \le p(d = \gcd(p, q) \wedge p + q \le |V| + d) \to \mathrm{Prd}(V, d).$$

Lemma 12. $\mathcal{S}_0 \vdash \mathrm{PL}(V, p, q)$.

Proof. The proof relies on a formalization of the observation stated above linking periods and borders. □

4.5 Regular and context-free strings

We are now going to show that regular languages can be defined with a Σ_1^B formula. This means that given any regular language, described by a regular expression R, there exists a Σ_1^B formula Ψ_R such that $\Psi_R(U) \iff U \in L(R)$.

Lemma 13. *Regular languages can be defined with a Σ_1^B formula.*

Proof. We have already defined concatenation of two strings in (4), but we still need to define the operation of union and Kleene's star. All together this can be stated as:

$$\begin{aligned} &\Psi_{\cdot}(U, V, W) := W = U \cdot V \\ &\Psi_{\cup}(U, V, W) := (W = U \vee W = V) \\ &\Psi_{*}(U, W) := \exists i \le |W|(W = \lambda i \langle i \cdot |u|, e(U, \mathrm{rem}(i, |U|)) \rangle) \end{aligned}$$

Now we show that R can be represented with a Σ_1^B formula by structural induction on the definition of R. The basis case is simple as the possibilities for R are as follows: a, ε, σ, and they can be represented with $W = a, |W| = 0, 0 = 1$, respectively.

For the induction step, consider R defined from $R_1 \cdot R_2$, $R_1 \cup R_2$ and $(R_1)^*$:

$$\begin{aligned} R = R_1 \cdot R_2 \qquad & \exists U_1 \le |W| \exists U_2 \le |W|(\Psi_{R_1}(U_1) \wedge \Psi_{R_2}(U_2) \wedge \Psi_{\cdot}(U_1, U_2, W)) \\ R = R_1 \cup R_2 \qquad & \exists U_1 \le |W| \exists U_2 \le |W|(\Psi_{R_1}(U_1) \wedge \Psi_{R_2}(U_2) \wedge \Psi_{\cup}(U_1, U_2, W)) \\ R = (R_1)^* \qquad & \exists U_1 \le |W| \Psi_{*}(U_1, W) \end{aligned}$$

Thus, we obtain a Σ_1^B formula $\Psi_R(W)$ which is true iff $W \in L(R)$. □

Note that in the proof of Lemma 13, when we put $\Psi_R(W)$ in prenex form all the string quantifiers are bounded by $|W|$, and they can be viewed as "witnessing" intermediate strings in the construction of W.

Lemma 14. *Context-free languages can be defined with a Σ_1^B formula.*

Proof. Use Chomsky's normal form and the CYK algorithm. □

5 Conclusion and future work

We have just touched the surface of the beautiful interplay between Stringology and Proof Complexity. Lemma 8 can likely be strengthened to say that evaluating $\mathcal{L}_{\mathcal{S}}$-terms can be done in $\mathbf{AC}^0$ rather than polytime. As was mentioned in the paper, the richness of the field of Stringology arises from the fact that a string U is a map $I \longrightarrow \Sigma$, where I can be arbitrarily large, while Σ is small. This produces repetitions and patterns that are the object of study for Stringology. On the other hand, Proof Complexity has studied in depth the varied versions of the Pigeonhole Principle that is responsible for these repetitions. Thus the two may enrich each other. Finally, Regular languages can be decided in $\mathbf{NC}^1$; how can this be reflected in the proof of Lemma 13? Also, prove Lemma 14.

Due to the lack of space, and the fact that it usually requires a rather lengthy construction, we did not illustrate an application of the Witnessing theorem. A very nice application can be found in the Lyndon decomposition of a string (see [4, pg. 29]). Recall that our alphabet is ordered — this was precisely so these types of arguments could be carried out naturally in our theory. Since $\sigma_0 < \sigma_1 < \sigma_2 \ldots$, we can easily define a lexicographic ordering of strings; define a predicate $U <_{\text{lex}} V$. We can define a Lyndon word with a Σ_0^B formula as follows: $\forall i < |V|(V <_{\text{lex}} \lambda k\langle i, e(V, |V| - i + 1 + k)\rangle)$.

Let V be a string; then $V = V_1 \cdot V_2 \cdots V_k$ is a Lyndon decomposition if each V_i is a Lyndon word, and $V_k <_{\text{lex}} V_{k-1} <_{\text{lex}} \cdots <_{\text{lex}} V_1$. The existence of a Lyndon decomposition can be proven as in [4, Theorem 1.4.9], and we assert that the proof itself can be formalized in $\mathcal{S}_1$. We can therefore conclude that the actual decomposition can be computed in polytime. As one can see, this approach provides a deep insight into the nature of strings.

References

1. S. R. Buss: *An introduction to proof theory*, in Handbook of Proof Theory, S. R. Buss, ed., North Holland, 1998, pp. 1–78.
2. S. R. Buss and M. Soltys: *Unshuffling a square is NP-hard*. Journal of Computer and System Sciences, 80(4) 2013, pp. 766–776.
3. S. A. Cook and P. Nguyen: *Logical Foundations of Proof Complexity*, Cambridge Univeristy Press, 2010.
4. B. Smyth: *Computing Patterns in Strings*, Pearson Education, 2003.
5. M. Soltys: *A model-theoretic proof of the completeness of LK proofs*, Tech. Rep. CAS-06-05-MS, McMaster University, 1999.
6. M. Soltys and S. Cook: *The proof complexity of linear algebra*. Annals of Pure and Applied Logic, 130(1–3) December 2004, pp. 207–275.