A formal framework for Stringology

Michael Soltys[a], Neerja Mhaskar[b,*]

[a] *California State University Channel Islands, Dept. of Computer Science, One University Drive, Camarillo, CA 93012, USA*
[b] *McMaster University, Dept. of Computing & Software, 1280 Main Street West, Hamilton, Ontario L8S 4K1, CANADA*

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 strings. We propose a language $\mathcal{L}_{\mathcal{S}}$ for expressing assertions about strings, and study in detail two sets of formulas Σ_0^B, a set of formulas decidable in polytime, and Σ_1^B, a set of formulas with the property that those provable in $\mathcal{S}$ yield polytime algorithms.

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 in 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.

*Corresponding author
Email addresses: michael.soltys@csuci.edu (Michael Soltys), pophlin@mcmaster.ca (Neerja Mhaskar)

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 $\boldsymbol{v}$, there exists a string $\boldsymbol{u}$ with property α," i.e., $\exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \boldsymbol{v})$, where $|\boldsymbol{u}| \leq I$, then we can mechanically extract an actual algorithm which computes $\boldsymbol{u}$ for any given $\boldsymbol{v}$.

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. Furthermore, we can refine $\mathcal{S}$ to make guarantees about the complexity of the algorithm. This refinement is usually brought about by restricting induction to be over formulas of a certain syntactic form, and by choosing the axioms to reflect a certain level of computational complexity.

Our contribution is conceptual; we do not propose new or faster algorithms. Rather, we propose a unified logical theory of strings. The reward of this exercise is that we make many implicit assumptions of Stringology explicit; for example, we define alphabet symbols with the successor function, a constant, and an ordering relation. We also discover that the theory of strings consists in rudimentary number theory on the indices of a one-dimensional array, with combinatorial sophistication arising from the drive that make all algorithms linear time. Thus Stringology can be viewed as the study of one-dimensional arrays with entries from a (normally small) ordered set. Such observations are valuable insights into this beautiful field.

For a background on Proof Complexity see [CN10] 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 in the style of [SC04].

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 $\mathcal{L}_{\mathbb{S}}$

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

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

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

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 (informally we 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 arbitrary numbers of 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 function 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 $\mathcal{L}_{\mathbb{S}}$

We use metavariables $i, j, k, l, \ldots$ to denote indices, metavariables $a, b, c, \ldots$ to denote alphabet symbols, and metavariables $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \ldots$ 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 use I to denote an index term, for example $i + j$, S to denote a symbol term, for example $\sigma\sigma\sigma\mathbf{0}$, and T to denote string terms. We use $t, t_1, t_2, \ldots$, to denote terms of any type. Finally, we use Greek letters $\alpha, \beta, \gamma, \ldots$, to denote formulas.

Definition 2. *$\mathcal{L}_S$-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 I_1, I_2 are index terms, then so are $(I_1 \circ I_2)$ where $\circ \in \{+, -, \cdot\}$,* $\mathrm{div}(I_1, I_2)$*, and* $\mathrm{rem}(I_1, I_2)$*.*
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 I is an index term, and T is a string term, then $e(T, I)$ is a symbol term.*
8. *All constant functions $(0_{\text{index}}, 1_{\text{index}}, \mathbf{0}_{\text{symbol}})$ are terms.*

We define the term cond *once we define formulas (see Definition 6).*

Definition 3 (Bound and Free variables)**.** *An occurrence of a variable x in a formula α is said to be bound iff it is in a subformula β of α of the form $\forall x\beta$ or $\exists x\beta$. Otherwise the occurrence is free.*

Definition 4 (Substitution)**.** *Let t_1, t_2 be terms, and α a formula. Then $t_2(t_1/x)$ is the result of replacing all occurrences of x in t_2 by t_1, and $\alpha(t_1/x)$ is the result of replacing all free occurrences of x in α by t_1.*

We are going to employ the lambda operator λ for building terms of type string.

Definition 5. *Given a term I of type index, and given a term S of type symbol, the following is a term T of type string:*

$$\lambda i\langle I, S\rangle. \tag{1}$$

T is a term of type string, and is of length I and i occurs in S. The j-th symbol of the string T is obtained by evaluating S at j, i.e., by evaluating $S(j/i)$. Note that, $S(j/i)$ is the term obtained by replacing every free *occurrence of i in S with j. Since (1) is a λ-term, i is considered to be a bound variable and its value ranges in $[0..I-1]$. This concept is formalized in Axiom B22.*

For examples of string constructors see Section 2.4.

Definition 6. *$\mathcal{L}_S$-Formulas are defined by structural induction as follows:*

1. *If I_1, I_2 are two index terms, then $I_1 < I_2$ and $I_1 = I_2$ 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, so are the following: $\neg\alpha, (\alpha \wedge \beta), (\alpha \vee \beta), \forall x\alpha, \exists x\alpha$, where x is a variable.*
5. *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.*

Note that in order to define all the terms (Definition 5) we need to first define formulas (Definition 6), as cond is defined in terms of both; but formulas require terms. In order to avoid the appearance of circularity, we could combine Definitions 5 and 6, but it is clearer to have two definitions, where care needs to be taken in the definition of cond.

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

Definition 7. *Let Σ_0^B be the set of $\mathcal{L}_S$-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 $\mathcal{L}_S$ 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.*

We want our theory to be strong enough to prove interesting theorems, but not too strong so that proofs yield feasible algorithms. When a theory is strong (say the axioms express strong properties, such as correctness of $f(\alpha) = t$, where f is a function computing a satisfying assignment t for formula α, when it exists), then the function witnessing the existential quantifiers will be of a higher complexity. 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 10.

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 [BS13] or [HRSS17]).

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

Assigning a standard interpretation to the syntax of a given formalism is the ensign of developing a logical theory. In our case it is more than just a formal exercise as we are giving a formal meaning to the notations and syntactical constructions of Stringology. In order to provide an interpretation of $\mathcal{L}_{\mathcal{S}}$, we first define a structure and then give a standard interpretation for Stringology, which we call $\underline{\mathbb{S}} = (\mathbb{N}, \Sigma, \mathcal{S})$.

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 [CN10, §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 is $M = (\mathcal{I}, \Sigma, \mathcal{S})$, where $\mathcal{I}$ denotes the set of indices, Σ the set of alphabet symbols, and $\mathcal{S}$ the set of strings.

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

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

for the two unary function symbols:

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

for the six binary function symbols:

$$+^{\mathcal{M}}_{\text{index}} : \mathcal{I}^2 \longrightarrow \mathcal{I}, \quad -^{\mathcal{M}}_{\text{index}} : \mathcal{I}^2 \longrightarrow \mathcal{I}, \quad \cdot^{\mathcal{M}}_{\text{index}} : \mathcal{I}^2 \longrightarrow \mathcal{I}$$

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

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

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

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 8. *An* object assignment τ *for a structure* $\mathcal{M}$ *is a mapping from variables to the universe* $M = (\mathcal{I}, \Sigma, \mathcal{S})$*, 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 $\mathcal{S}$ can be seen as a function from $\mathcal{I}$ to Σ, i.e., a given $\boldsymbol{u} \in \mathcal{S}$ is just a function $\boldsymbol{u} : \mathcal{I} \longrightarrow \Sigma$. In Stringology we are interested in the case where a given $\boldsymbol{u}$ may be arbitrarily long but it maps $\mathcal{I}$ to a relatively small set of Σ: for example, binary strings map into $\{0, 1\} \subset \Sigma$. Since the range of $\boldsymbol{u}$ is relatively small this leads to interesting structural questions about the mapping: repetitions and patterns.

We start by defining τ on terms: $t^{\mathcal{M}}[\tau]$. 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 I, I_1, I_2 are index terms, S is a symbol term and T is a string term.

$$(I_1 \circ_{\text{index}} I_2)^{\mathcal{M}}[\tau] = (I_1^{\mathcal{M}}[\tau] \circ^{\mathcal{M}}_{\text{index}} I_2^{\mathcal{M}}[\tau]),$$

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

$$(\text{div}(I_1, I_2))^{\mathcal{M}}[\tau] = \text{div}^{\mathcal{M}}(I_1^{\mathcal{M}}[\tau], I_2^{\mathcal{M}}[\tau]),$$

$$(\text{rem}(I_1, I_2))^{\mathcal{M}}[\tau] = \text{rem}^{\mathcal{M}}(I_1^{\mathcal{M}}[\tau], I_2^{\mathcal{M}}[\tau]).$$

and for symbol terms we have:

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

Finally, for string terms:

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

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

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:

$$\mathcal{M} \vDash (I_1 <_{\text{index}} I_2)[\tau] \iff (I_1^{\mathcal{M}}[\tau], I_2^{\mathcal{M}}[\tau]) \in <_{\text{index}}^{\mathcal{M}},$$

$$\mathcal{M} \vDash (I_1 =_{\text{index}} I_2)[\tau] \iff I_1^{\mathcal{M}}[\tau] = I_2^{\mathcal{M}}[\tau],$$
$$\mathcal{M} \vDash (S_1 =_{\text{symbol}} S_2)[\tau] \iff S_1^{\mathcal{M}}[\tau] = S_2^{\mathcal{M}}[\tau],$$
$$\mathcal{M} \vDash (T_1 =_{\text{string}} T_2)[\tau] \iff T_1^{\mathcal{M}}[\tau] = T_2^{\mathcal{M}}[\tau].$$

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 9. *Let $\underline{\mathbb{S}} = (\mathbb{N}, \Sigma, \mathcal{S})$ 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 $\mathcal{S}$ is the set of functions $\boldsymbol{u} : \mathcal{I} \longrightarrow \Sigma$, and where all the function and predicate symbols get their standard interpretations.*

The following lemma is important in that it reassures us of the "reasonableness" of the logical theory that we proposed: given a formula without unbounded quantifiers, we can always check its truth in the standard model, for particular parameter values, in polynomial time (in the size of the encoding of those values).

lemma 10. *Given any formula $\alpha \in \Sigma_0^B$, and a particular object assignment τ, we can verify $\underline{\mathbb{S}} \vDash \alpha[\tau]$ in polytime in the lengths of the strings and values of the indices in α.*

Proof. Let t, t_1, t_2 denote terms of any type (index, symbol or strings). We first show that evaluating a term t, i.e., computing $t^{\underline{\mathbb{S}}}[\tau]$, can be done in polytime. We do this by structural induction on t. If t is just a variable then there are three cases:

- t is i, an index variable: $i^{\underline{\mathbb{S}}}[\tau] = \tau(i) \in \mathbb{N}$.
- t is u, a symbol variable: $u^{\underline{\mathbb{S}}}[\tau] = \tau(u) \in \Sigma$.
- t is $\boldsymbol{u}$, a string variable: $\boldsymbol{u}^{\underline{\mathbb{S}}}[\tau] = \tau(\boldsymbol{u}) \in \mathcal{S}$.

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. Therefore evaluating t in the base case when it is just a variable is done in constant time.

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.). Therefore, evaluating t in this case is done in polytime in the values of the arguments. Hence so is evaluating the atomic formulas $(t_1 < t_2)^{\underline{\mathbb{S}}}[\tau]$ and $(t_1 = t_2)^{\underline{\mathbb{S}}}[\tau]$, where t_1 and t_2 are terms of any type (index, symbols or strings), or their Boolean combinations.

Finally, we consider quantification; but we are only allowed bounded index quantification: $(\exists i \leq I\alpha)^{\underline{\mathbb{S}}}[\tau]$, and $(\forall i \leq I\alpha)^{\underline{\mathbb{S}}}[\tau]$. This is equivalent to computing:

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

Since this can be done in polytime, we conclude that evaluating any formula can be done in polytime. □

2.4. Examples of string constructors

In the previous sections we have formalized Stringology and we have given it a formal interpretation. We now seek to demonstrate that our theory is useful in the sense that it can express standard properties of strings in a

succinct manner. We are going to show more examples of applications in Section 4, once we introduce the axioms and rules of inference for $\mathcal{S}$.

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 0s and 1s 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 [CN10], 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 as:

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

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

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

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

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

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:

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

Note that both $\bar{\boldsymbol{u}}$ and $\boldsymbol{v}$ are defined with the conditional function where the formula α conforms to the restriction: variables are either free (like $\boldsymbol{u}$ in $\bar{\boldsymbol{u}}$),

or, if quantified, all such variables are bounded and of type index (like j in $\boldsymbol{v}$).

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

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

$$\boldsymbol{u}^R := \lambda i \langle |\boldsymbol{u}|, e(\boldsymbol{u}, (|\boldsymbol{u}|-1)-i) \rangle,$$

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

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

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 [Bus98, §2.2.1]. So we will not 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 $\Gamma \to \Delta$, where Γ, Δ are comma-separated lists of formulas. That is, a sequent is of the form:

$$\alpha_1, \alpha_2, \dots, \alpha_n \to \beta_1, \beta_2, \dots, \beta_m,$$

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 stated as follows: $\neg \bigwedge_i \alpha_i \vee \bigvee_j \beta_j$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. To prove a formula α we need to derive $\to \alpha$ and to refute it we need to derive $\alpha \to$.

The index axioms are the same as 2-BASIC in [CN10, 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. The 2-BASIC axioms: $B1, B2$, define that there are no negative numbers and that the successor function is a bijection. The axioms $B3, B4$, and $B5, B6$ provide recursive definitions of addition and multiplication respectively. Axioms $B9, B10$ provide basic properties of $\leq$ and axiom $B12$ expresses that the elements of the index sort are totally ordered. Finally, the axiom B11 defines that 0 is the minimum number, B13 defines discreteness, and B14 defines the predecessor function.

Keep in mind that a formula α is equivalent to a sequent $\to \alpha$, and so, for readability we sometimes mix the two.

Index Axioms	
B1. $i+1 \neq 0$	B9. $i \leq j, j \leq i \to i = j$
B2. $i+1 = j+1 \to i = j$	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. $i \neq 0 \to \exists j \leq i(j+1 = i)$
B7. $i \leq j, i+k = j \to j-i = k$	B15. $i \not\leq j \to j-i = 0$
B8. $j \neq 0 \to \mathrm{rem}(i,j) < j$	B16. $j \neq 0 \to i = j \cdot \mathrm{div}(i,j) + \mathrm{rem}(i,j)$

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

Alphabet Axioms
B17. $a \lneqq \sigma a$
B18. $a < b, b < c \to a < c$
B19. $\alpha \to \mathrm{cond}(\alpha, a, b) = a$
B20. $\neg\alpha \to \mathrm{cond}(\alpha, a, b) = b$

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, in the standard model, 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. $\lvert\lambda i\langle I, S\rangle\rvert = I$
B22. $j < I \to e(\lambda i\langle I, S\rangle, j) = S(j/i)$
B23. $\lvert\boldsymbol{u}\rvert \leq j \to e(\boldsymbol{u}, j) = \boldsymbol{0}$
B24. $\lvert\boldsymbol{u}\rvert = \lvert\boldsymbol{v}\rvert, \forall i < \lvert\boldsymbol{u}\rvert e(\boldsymbol{u}, i) = e(\boldsymbol{v}, i) \to \boldsymbol{u} = \boldsymbol{v}$

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 than or equal to the length of a string, then $e(\boldsymbol{u}, j)$ defaults to $\mathbf{0}$. The last axioms, B24, says that if two strings $\boldsymbol{u}$ and $\boldsymbol{v}$ 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

$$\forall i < |\boldsymbol{u}| e(\boldsymbol{u}, i) = e(\boldsymbol{v}, i).$$

This works because by B23 we know that for $i \geq |\boldsymbol{u}|$, $e(\boldsymbol{u}, i) = e(\boldsymbol{v}, i) = \mathbf{0}$ (since $|\boldsymbol{u}| = |\boldsymbol{v}|$ 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 $\mathbf{0}$.

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 [Bus98].

2.6.1. 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: $\dfrac{\Gamma \to \Delta}{\Gamma \to \Delta, \alpha}$

2.6.2. Cut rule

$$\frac{\Gamma \to \Delta, \alpha \quad \alpha, \Gamma \to \Delta}{\Gamma \to \Delta}$$

2.6.3. 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: $\dfrac{\Gamma \to \Delta, \alpha, \beta}{\Gamma \to \Delta, \alpha \vee \beta}$

2.6.4. 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.

2.6.5. Induction rule

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

where i does not occur free in Γ, Δ, and I 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 $i = 1$, as otherwise, it might not be possible to compute the actual value of the existential quantifier feasibly.

Definition 11. *Let $\mathcal{S}_i$ 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 12 (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 12, but the reader is pointed to [Sol99] 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 $\mathcal{S}_i$ we outline an $\mathcal{S}_0$ proof of the equality of (2) and (3). Note that the truth of the statement is obvious, and the proof is tedious, but we use it to illustrate the application of the axioms and rules of our theory.

First note that by axiom B21 we have that:

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

Using Axioms B3, B5 and B6 we can show that $\hat{2} \cdot \hat{4}$ equals the sum of eight 1s, and since that is $\hat{8}$ by definition, we have $\hat{8} = \hat{2} \cdot \hat{4}$. And so,

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

Now we have to show that:

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

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:

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

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

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

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

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

Then, using B19 and B20 we can show (7).

In the next section we are going to show the interplay between proofs and computations. In particular, we are going to show how to extract efficient algorithms from $\mathcal{S}$-proofs.

3. Witnessing theorem for $\mathcal{S}$

Recall that we defined $\mathcal{S}_1$ to be a fragment of our theory where the induction is restricted to Σ_1^B formulas. For convenience, we sometimes use the notation $\vec{\boldsymbol{v}}$, to denote several string variables, i.e., $\vec{\boldsymbol{v}} = \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_\ell$.

If we prove a formula $\exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \boldsymbol{v})$, then the function f which on input $\boldsymbol{v}$ outputs $\boldsymbol{u}$, that is $f(\boldsymbol{v}) = \boldsymbol{u}$, is called the *witnessing function*. Armed with this informal definition, 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 $\vec{\boldsymbol{u}}$ with some given properties, then in fact we can construct such a string with a polytime algorithm.

Theorem 13 (Witnessing)**.** *If $\mathcal{S}_1 \vdash \exists \vec{\boldsymbol{u}} \leq I\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}})$, then it is possible to compute $\vec{\boldsymbol{u}}$ in polynomial time in the total length of all the string variables in $\vec{\boldsymbol{v}}$ and the value of all the free index variables in α.*

Proof. In order to simplify the proof we show it for $\mathcal{S}_1 \vdash \exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}})$, i.e., $\boldsymbol{u}$ 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 $\exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}})$ that $\boldsymbol{u}$ 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 12 we know that all the formulas in the $\mathcal{S}_1$ proof of $\exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}})$ 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 cases are $\exists$-right and the induction rule. In the former case we have:

$$\exists\text{-right: } \frac{|T| \leq I, \Gamma \to \Delta, \alpha(T, \vec{\boldsymbol{v}}, \vec{\boldsymbol{i}})}{\Gamma \to \Delta, \exists \boldsymbol{u} \leq I\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, \vec{\boldsymbol{i}})}$$

which is the $\exists$-right rule adapted to the case of bounded string quantification, where T is a string term and I is an index term. We use $\vec{\boldsymbol{v}}$ to denote all the

free string variables, and $\vec{\boldsymbol{i}}$ to denote explicitly all the free index variables. Then $\boldsymbol{u}$ is witnessed by the function $f(\vec{\boldsymbol{v}}, \vec{\boldsymbol{i}})$ and when f is evaluated at $\vec{\boldsymbol{A}}$ and $\vec{\boldsymbol{b}}$ we get $\boldsymbol{u}$, that is:

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

$\vec{\boldsymbol{A}}$ and $\vec{\boldsymbol{b}}$ are terms replacing variables. By Lemma 10 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. Therefore, f is polytime as evaluating T under $\underline{\mathbb{S}}$ and any object assignment can be done in polytime.

For the induction case we restate the rule as follows in order to make all the free variables more explicit:

$$\frac{\boldsymbol{u} \leq I, \alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, i, \vec{\boldsymbol{j}}) \rightarrow \exists \boldsymbol{u} \leq I \alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, i+1, \vec{\boldsymbol{j}})}{\boldsymbol{u} \leq I, \alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, 0, \vec{\boldsymbol{j}}) \rightarrow \exists \boldsymbol{u} \leq I \alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, I', \vec{\boldsymbol{j}})}$$

where $\vec{\boldsymbol{j}}$ denotes all the free variables and I' is an index term. 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 $\boldsymbol{u}$ such that $\alpha(\boldsymbol{u}, \vec{\boldsymbol{v}}, 0, \vec{\boldsymbol{v}})$ is satisfied. Use top of rule to compute $\boldsymbol{u}$'s for $i = 1, 2, \ldots, I^{\mathbb{S}}[\tau]$. □

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

In this section we state standard properties of strings, and we outline proofs of those properties in bounded fragments of $\mathcal{S}$.

4.1. Prefix, Suffix and Substring

The prefix, suffix, and substring are basic constructs of a given string $\boldsymbol{v}$. The $\mathcal{L}_{\mathcal{S}}$-term for a prefix of length i is given by $\lambda k\langle i, e(\boldsymbol{v}, k)\rangle$, and the $\mathcal{L}_{\mathcal{S}}$-term for a suffix of length i is given by $\lambda k\langle i, e(\boldsymbol{v}, |\boldsymbol{v}| - i + k)\rangle$, and since any substring of length i is a prefix (of the same length) of some suffix of length j, the $\mathcal{L}_{\mathcal{S}}$-term for a substring is given by $\lambda l\langle i, e(\lambda k\langle j, e(\boldsymbol{v}, |\boldsymbol{v}| - j + k)\rangle, l)\rangle$.

We can state that $\boldsymbol{u}$ is a prefix of $\boldsymbol{v}$ with the following Σ_0^B predicate:

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

The predicate for suffix $\mathrm{suf}(\boldsymbol{u}, \boldsymbol{v})$ is defined similarly and is given by:

$$\mathrm{suf}(\boldsymbol{u}, \boldsymbol{v}) := \exists i \leq |\boldsymbol{v}|(\boldsymbol{u} = \lambda k\langle i, e(\boldsymbol{v}, |\boldsymbol{v}| - i + k)\rangle),$$

Finally, the predicate for substring $\mathrm{sub}(\boldsymbol{u}, \boldsymbol{v})$ is given by:

$$\mathrm{sub}(\boldsymbol{u}, \boldsymbol{v}) := \exists i, j \leq |\boldsymbol{v}|(\boldsymbol{u} = \lambda l \langle i, e(\lambda k \langle j, e(\boldsymbol{v}, |\boldsymbol{v}| - j + k)\rangle, l)\rangle).$$

Note that we could also do $\mathrm{sub}'(\boldsymbol{u}, \boldsymbol{v}) := \exists i \leq |\boldsymbol{v}|(\boldsymbol{u} = \lambda j \langle |\boldsymbol{u}|, e(\boldsymbol{v}, i + j)\rangle)$, where of course $\mathrm{sub}(\boldsymbol{u}, \boldsymbol{v}) \iff \mathrm{sub}'(\boldsymbol{u}, \boldsymbol{v})$. It would be an instructive exercise for the reader to prove that $\mathcal{S}_0 \vdash \forall \boldsymbol{u} \forall \boldsymbol{v}[\mathrm{sub}(\boldsymbol{u}, \boldsymbol{v}) \leftrightarrow \mathrm{sub}'(\boldsymbol{u}, \boldsymbol{v})]$.

4.2. Counting symbols

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

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

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

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

Having defined $(\boldsymbol{u})_{\sigma_1}$ with axioms C1-3, and $\boldsymbol{u}|_{\sigma_i}$ as a term in $\mathcal{L}_{\mathcal{S}}$, we can now define $(\boldsymbol{u})_{\sigma_i}$ as follows: $(\boldsymbol{u}|_{\sigma_i})_{\sigma_1}$. Note that C1-3 are Σ_0^B sequents. It would be an instructive exercise for the reader to prove that $\mathcal{S}_0'$, which is $\mathcal{S}$ together with C1-3 and the new function symbol $(\cdot)_\sigma$, is in fact conservative over $\mathcal{S}_0$. That is, if $\mathcal{S}_1' \vdash \alpha$ where α does not contain $(\cdot)_\sigma$, then $\mathcal{S}_0 \vdash \alpha$ as well.

4.3. Borders and border arrays

Suppose that we want to define a border array. We first define the border predicate which asserts that the string $\boldsymbol{v}$ has a border of size i; note that by definition a border is a (proper) prefix equal to a (proper) suffix. So let:

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

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

$$\mathrm{MaxBrd}(\boldsymbol{v}, i) := \mathrm{Brd}(\boldsymbol{v}, i) \wedge (j > i \supset \neg \mathrm{Brd}(\boldsymbol{v}, j)).$$

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

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

We also have the conservativity property as stated previously, which can be checked by the reader. This is important as it allows us to extend our theory $\mathcal{S}_0$ with convenient function definitions without increasing the power of the theory — and thus preserving the Witnessing theorem for the underlying complexity class.

4.4. Periodicity

See [Smy03, pg. 10] for the definition of a period of a string, but for our purpose let us define $p = |\boldsymbol{u}|$ to be a period of $\boldsymbol{v}$ if $\boldsymbol{v} = \boldsymbol{u}^r\boldsymbol{u}'$ where $\boldsymbol{u}'$ is some prefix, possibly empty, of $\boldsymbol{u}$. The Periodicity Lemma states the following: Suppose that p and q are two periods of $\boldsymbol{v}$, $|\boldsymbol{v}| = n$, and $d = \gcd(p, q)$. Then, if $p + q \leq n + d$, then d is also a period of $\boldsymbol{v}$.

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

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

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

$$\mathrm{Prd}(\boldsymbol{v}, p), \mathrm{Prd}(\boldsymbol{v}, q), \exists d \leq p(d = \gcd(p, q) \wedge p + q \leq |\boldsymbol{v}| + d) \rightarrow \mathrm{Prd}(\boldsymbol{v}, d).$$

lemma 14. $\mathcal{S}_0 \vdash \mathrm{PL}(\boldsymbol{v}, p, q)$.

The proof of Lemma 14 relies on a formalization of the observation stated above linking periods and borders. The details are left to the reader.

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 $\Psi_R(\boldsymbol{u}) \iff \boldsymbol{u} \in L(R)$.

lemma 15. *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:

$$\begin{aligned}
&\Psi_{.}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) := \boldsymbol{w} = \boldsymbol{u} \cdot \boldsymbol{v} \\
&\Psi_{\cup}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) := (\boldsymbol{w} = \boldsymbol{u} \vee \boldsymbol{w} = \boldsymbol{v}) \\
&\Psi_{*}(\boldsymbol{u}, \boldsymbol{w}) := \exists i \leq |\boldsymbol{w}|(\boldsymbol{w} = \lambda i \langle i \cdot |u|, e(\boldsymbol{u}, \mathrm{rem}(i, |\boldsymbol{u}|))\rangle)
\end{aligned}$$

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 $\boldsymbol{w} = a, |\boldsymbol{w}| = 0, 0 = 1$, respectively.

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

$$\begin{aligned}
R = R_1 \cdot R_2 \qquad & \exists \boldsymbol{u}_1 \leq |\boldsymbol{w}| \exists \boldsymbol{u}_2 \leq |\boldsymbol{w}|(\Psi_{R_1}(\boldsymbol{u}_1) \wedge \Psi_{R_2}(\boldsymbol{u}_2) \wedge \Psi_{.}(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{w})) \\
R = R_1 \cup R_2 \qquad & \exists \boldsymbol{u}_1 \leq |\boldsymbol{w}| \exists \boldsymbol{u}_2 \leq |\boldsymbol{w}|(\Psi_{R_1}(\boldsymbol{u}_1) \vee \Psi_{R_2}(\boldsymbol{u}_2) \wedge \Psi_{\cup}(\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{w})) \\
R = (R_1)^* \qquad & \exists \boldsymbol{u}_1 \leq |\boldsymbol{w}| \Psi_{*}(\boldsymbol{u}_1, \boldsymbol{w})
\end{aligned}$$

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

Note that in the proof of Lemma 15, when we put $\Psi_R(\boldsymbol{w})$ in prenex form all the string quantifiers are bounded by $|\boldsymbol{w}|$, and they can be viewed as "witnessing" intermediate strings in the construction of $\boldsymbol{w}$.

lemma 16. *Context-free languages can be defined with a Σ_1^B formula.*

We leave the proof of Lemma 16 as an exercise to the reader.

5. Conclusion and future work

We have just touched the surface of the beautiful interplay between Stringology and Proof Complexity. Lemma 10 can likely be strengthened to say that evaluating $\mathcal{L}_S$-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 $\boldsymbol{u}$ is a map $\mathcal{I} \longrightarrow \Sigma$, where $\mathcal{I}$ 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 15?

A nice application of the Witnessing Theorem can be found in the Lyndon decomposition of a string (see [Smy03, 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 $\boldsymbol{u} <_{\text{lex}} \boldsymbol{v}$. We can define a Lyndon word with a Σ_0^B formula as follows: $\forall i < |\boldsymbol{v}|(\boldsymbol{v} <_{\text{lex}} \lambda k\langle i, e(\boldsymbol{v}, |\boldsymbol{v}| - i + k)\rangle)$.

Let $\boldsymbol{v}$ be a string; then $\boldsymbol{v} = \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 \cdot \ldots \cdot \boldsymbol{v}_k$ is a Lyndon decomposition if each $\boldsymbol{v}_i$ is a Lyndon word, and $\boldsymbol{v}_k <_{\text{lex}} \boldsymbol{v}_{k-1} <_{\text{lex}} \cdots <_{\text{lex}} \boldsymbol{v}_1$. The existence of a Lyndon decomposition can be proven as in [Smy03, 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

[BS13] Samuel R. Buss and Michael Soltys. Unshuffling a square is NP-hard. *Journal of Computer and System Sciences*, 80(4):766–776, 2013.

[Bus98] Samuel R. Buss. An introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, pages 1–78. North Holland, 1998.

[CN10] Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge Univeristy Press, 2010.

[HRSS17] Joel Helling, P. J. Ryan, W. F. Smyth, and Michael Soltys. Constructing an indeterminate string from its associated graph. *Accepted for publication in the Journal of Theoretical Computer Science*, 2017.

[SC04] Michael Soltys and Stephen Cook. The proof complexity of linear algebra. *Annals of Pure and Applied Logic*, 130(1–3):207–275, December 2004.

[Smy03] Bill Smyth. *Computing Patterns in Strings*. Pearson Education, 2003.

[Sol99] Michael Soltys. A model-theoretic proof of the completeness of LK proofs. Technical Report CAS-06-05-MS, McMaster University, 1999.