A Language-theoretic View on Network Protocols

Pierre Ganty[1,⋆], Boris Köpf[1], and Pedro Valero[1,2] (0000-0001-7531-6374)

[1] IMDEA Software Institute, Madrid, Spain
[2] Universidad Politécnica de Madrid, Spain
{pierre.ganty,boris.koepf,pedro.valero}@imdea.org

Abstract. Input validation is the first line of defense against malformed or malicious inputs. It is therefore critical that the validator (which is often part of the parser) is free of bugs.
To build dependable input validators, we propose using parser generators for context-free languages. In the context of network protocols, various works have pointed at context-free languages as falling short to specify precisely or concisely common idioms found in protocols. We review those assessments and perform a rigorous, language-theoretic analysis of several common protocol idioms. We then demonstrate the practical value of our findings by developing a modular, robust, and efficient input validator for HTTP relying on context-free grammars and regular expressions.

1 Introduction

Input validation, often carried out during parsing, is the first line of defense against malformed or maliciously crafted inputs. As the following reports demonstrate, bugs make parsers vulnerable, hence prone to attacks: a bug in the URL parser enabled attackers to recover user credentials from a widely used password manager [14], a bug in the RTF parser led to a vulnerability in Word 2010 [19], and a lack of input validation in the Bash shell has been used for privilege escalation by a remote attacker [13] just to cite a few. To stop the flow of such reports improved approaches for building input validators are needed.

To build dependable input validators, an approach is to rely on mature parsing technologies. As a candidate, consider parser generators for context-free languages (hereafter CFLs). Their main qualities are:

1. The code for parsing is synthesized automatically from a grammar specification, shifting the risk of programming errors away from the architect of the validator to the designer of the parser generator;
2. CFL is the most expressive class of languages supported by trustworthy implementation of parser generators. Here by trustworthy we mean an implementation that either stood the test of time like Flex, Bison, or ANTLR, or that

⋆ Pierre Ganty has been supported by the Madrid Regional Government project S2013/ICE-2731, *N-Greens Software - Next-GeneRation Energy-EfficieNt Secure Software*, and the Spanish Ministry of Economy and Competitiveness project No. TIN2015-71819-P, *RISCO - RIgorous analysis of Sophisticated COncurrent and distributed systems.*

has been formally verified like the certified implementations of Valiant's [3] and CYK [11] algorithms.

Relying on established CFL technology is an asset compared to existing solutions which are either programmed from scratch [8] or generated from ad hoc parser generators [2, 21].

However, even though CFL is the most expressive language class with trustworthy parser generators, previous works suggest CFLs are not enough for network messages. Specifically, they claim the impossibility or difficulty to specify precisely the various idioms found in network messages using CFLs. For example, some authors argue that CFL are insufficiently expressive because "data fields that are preceded by their actual length (which is common in several network protocols) cannot be expressed in a context-free grammar" [7]. Yet other authors suggest that going beyond CFLs is merely required for conciseness of expression, because "it is possible to rewrite these grammars to [...] be context-free, but the resulting specification is much more awkward" [4]. Surprisingly, the arguments made are not backed up by any formalization or proof.

In this paper we formally analyze which idioms can and which cannot be (concisely) specified using CFLs, and we turn the results into practice by building an input validator for HTTP messages entirely based on CFL technology.

As the main contributions of our analysis we find out that:

- Length fields of *bounded size* are finite and hence form a regular language. However, while they can be concisely represented in terms of a context-free grammar, every finite automaton that recognizes them grows exponentially with the bound. In contrast, length fields of *unbounded size* cannot even be expressed as a finite intersection of CFLs.
- Equality tests between words of *bounded size* again form a regular language but, as opposed to length fields, they cannot be compactly represented in terms of a context-free grammar. They do, however, allow for a concise representation in terms of a finite intersection of context-free grammars. Specifically, we show that both the grammar and the number of membership checks grows only linearly with the size bound, which has interesting practical implications, see below. As in the case of length fields, equality checks between words of *unbounded size* cannot be expressed as a finite intersection of CFLs.

We consider finite intersections of CFLs because they are (a) strictly more expressive than CFLs, and (b) checking membership in the intersection of CFLs is equivalent to checking membership in each individual CFL.

These results lead to a principled and modular approach to input validation: several CFL parsers are run on the input and their boolean results (whether the input belongs or not to the CFL) are combined following a predefined logic to decide whether or not the input message conforms to the standard (that specifies what valid messages are).

We demonstrate that this approach is practical by implementing a proof of concept input validator for a large subset of the HTTP protocol, covering a significant number of the idioms found in network messages. Our input validator, called HTTPValidator [24] draws inspiration from HTTPolice [8], a state of the

art input validator for HTTP messages built from scratch by the open source community. HTTPValidator is close to achieve feature parity (in terms of checks) with HTTPolice and offers competitive performance.

Summary of Contributions. In summary, our contributions are both foundational and applied. On the foundational side we perform a language-theoretic analysis of important protocol idioms, making a step towards more rigor in the field. On the applied side we show how to implement an input validator for HTTP using off-the-shelf parser generators.

Paper structure. Section 2 introduces basic language-theoretic and input validation concepts, Section 3 discusses the case of length fields, including the chunked messages, whereas Section 4 considers the case of comparisons. We show the practicality of our approach in Section 5, before concluding with related work (Section 6), conclusions and future work (Section 7).

2 Preliminaries

Language Theory. We begin by introducing the language-theoretic context needed for our development. An *alphabet* Σ is a nonempty finite set of *symbols*. A *word* w is a finite sequence of symbols of Σ where the empty sequence is denoted ε. A *language* is a set of words and the set of all words over Σ is denoted Σ^*. We denote by $|w|$ the *length* of w. Further define $(w)_i$ as the i-th symbol of w if $1 \leq i \leq |w|$ and ε otherwise. Given a nonempty subset X of Σ and $i \in \mathbb{N}$ define X^i as $\{w \in X^* \mid |w| = i\}$.

We assume the reader is familiar with common operations on languages such as concatenation and boolean combinations. Likewise, we count on the reader's familiarity with regular languages and finite-state automata. Yet we next give a description of context-free grammars, which are the formal basis of our work.

A *context-free grammar* (or grammar for short) is a tuple $G = (V, \Sigma, S, R)$ where V is a finite set of *variables* (or *non-terminals*) including the *start variable* S; Σ is an alphabet (or set of *terminals*), $R \subseteq V \times (\Sigma \cup V)^*$ is a finite set of *rules*. We often write $X \to w$ for a rule $(X, w) \in R$. We define a *step* as the binary relation $\Rightarrow$ on $(V \cup \Sigma)^*$ given by $u \Rightarrow v$ if there exists a rule $X \to w$ of G such that $u = \alpha\, X\, \beta$ and $v = \alpha\, w\, \beta$ for some $\alpha, \beta \in (V \cup \Sigma)^*$. Define $u \Rightarrow^* v$ if there exists a $n \geq 0$ steps sequence $u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n$ such that $u_0 = u$ and $u_n = v$. A step sequence $u \Rightarrow^* w$ is called a *derivation* whenever $u = S$ and $w \in \Sigma^*$. Define $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$ and call it the language generated by G. A language L is said to be *context-free*, or *CFL*, if there exists a grammar G such that $L = L(G)$. The *size* of a grammar is the sum of the sizes of its production rules R, that is, it is given by $\sum_{(X,w)\in R} 1+|w|$.

Input Validation. In this paper, validating an input means checking whether it belongs to a language. In particular, no data structure is filled and no information is extracted from the input other than its membership status. Thus, validating

an input w for a language L means deciding whether w is a member of L which is also known as the *membership problem*. To specify L, we use context-free grammars or regular expressions.

3 Formal Analysis of Length Fields

Length fields, whose role is to specify the length of subsequent fields, are commonly found in network protocols such as HTTP [9], SIP [23], DNS [20] and UDP [22]. As an example, consider the following HTTP POST message:

```
POST /1/notification/list HTTP/1.1\r\n
Content-Length: 47\r\n\r\n
{"header":{},"query":{"count":100},"answer":{}}
```

The length field begins after the keyword `Content-Length:` and terminates before the carriage return/newline `\r\n`. Its content, i.e. 47, describes the length of the message body, which is the string coming after the double `\r\n`.

In this section, we characterize length fields from the point of view of formal language theory. We begin with a formalization aiming to capture their essence, and then characterize the class of languages specifying them in the bounded and unbounded cases. We consider both cases because some protocols, such as DNS and UDP, require length fields to have fixed size, while others, such as HTTP and SIP, have no such restriction. We conclude by leveraging these results to analyze chunked transfer encoding.

3.1 Modeling Length Fields

To model length fields, we will work with formal languages over an alphabet Σ. For the example of HTTP, Σ would be the ASCII character set.

Fixed Size. To describe length fields of finite size $n > 0$ we define the language $L_{len}(n)$ over $\Sigma = B \cup W$ as follows:

$$L_{len}(n) \stackrel{\text{def}}{=} \{x\ w \mid x \in B^n, w \in W^*, |w| = \textstyle\sum_{i=0}^{n-1} (x)_{i+1} \cdot b^i\}$$

where $B = \{0, \ldots, b-1\}$ for an integer $b > 1$. Intuitively, $L_{len}(n)$ represents the same number twice, using two different encodings: first b-ary as x and then unary as w, where the relationship between both encodings is given by $|w| = \sum_{i=0}^{n-1} (x)_{i+1} \cdot b^i$. For example, let $n = 3$, $B = \{\mathtt{0}, \mathtt{1}\}$ and $W = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ the word `110abc` consists of the binary representation of $3 = (1 \cdot 2^0) + (1 \cdot 2^1) + (0 \cdot 2^2)$ followed by a word (`abc`) of length 3 and, therefore, $\mathtt{110abc} \in L_{len}(3)$. We choose this unconventional "least significant digit first" to keep notation simple. The results of this section stay valid for the "most significant digit first" convention.

Unbounded Size. For describing length fields of unbounded size, observe that any overlap between the alphabets W and B for describing the body of the message and its length, respectively, introduces ambiguity as to where the length field ends. A common approach to remove such ambiguities is to use a *delimiter*, which is a special symbol $\sharp$ not occurring in x whose aim is to separate explicitly the length field from the body of the message. We extend the definition of $L_{len}(n)$ to account for such delimiters:

$$L^{\sharp}_{len}(n) \stackrel{\text{def}}{=} \{x \sharp w \mid x \in B^n, w \in W^*, |w| = \textstyle\sum_{i=0}^{n-1}(x)_{i+1} \cdot b^i\} \ .$$

We are now in position to define a language for describing length fields of arbitrary and unbounded size:

$$L^{\sharp}_{len} \stackrel{\text{def}}{=} \textstyle\bigcup_{i>0} L^{\sharp}_{len}(i) \ .$$

Results shown in this section remain valid when there is no overlap between alphabets W and B. In such case the delimiter is no longer needed and removing it from the results in Section 3.2 has no effect on them.

3.2 Unbounded Length Fields

The following theorem shows that length fields of unbounded size cannot be specified using intersection of finitely many CFLs. This means that we need to impose restrictions, such as size bounds, in order to specify length fields using CFLs. We will study fixed size length fields in Section 3.3.

Theorem 1. *$L^{\sharp}_{len}$ is not a finite intersection of CFLs.*

To prove this result, we begin by defining the following subset of $L^{\sharp}_{len}$:

$$L_{\angle} \stackrel{\text{def}}{=} L^{\sharp}_{len} \cap 1^*\sharp^* a^* \ .$$

Lemma 2. *$L_{\angle}$ is not a finite intersection of CFLs. Moreover, no infinite subset of $L_{\angle}$ is a finite intersection of CFLs.*

The proof argument relies on semilinear sets which we recall next: a subset of $\mathbb{N}^k$, with $k > 0$, is called *semilinear*, if it can be specified as a finite union of linear sets. A set $S \subseteq \mathbb{N}^k$ is called *linear* if there exists $\boldsymbol{b} \in \mathbb{N}^k$ and a finite subset $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m\}$ of $\mathbb{N}^k$ such that $S = \{\boldsymbol{b} + \lambda_1 \boldsymbol{p}_1 + \cdots + \lambda_m \boldsymbol{p}_m \mid \lambda_1, \ldots, \lambda_m \in \mathbb{N}\}$.

Let $\bar{w} = \langle w_1, \ldots, w_k \rangle$ be a tuple of $k > 0$ words, define a mapping $f_{\bar{w}} \colon \mathbb{N}^k \to w_1^* \ldots w_k^*$ by $f_{\bar{w}}(i_1, \ldots, i_k) = w_1^{i_1} \ldots w_k^{i_k}$, that is, the output of $f_{\bar{w}}$ is a word in which the i-th component of $\bar{w}$ is repeated a number of times that corresponds to the i-th input to $f_{\bar{w}}$. We define the preimage of $f_{\bar{w}}$ and liftings of $f_{\bar{w}}$ from elements to subsets of $\mathbb{N}^k$ in the natural way.

The following result by Latteux [15] establishes a fundamental correspondence between languages given by finite intersection of CFLs and semilinear sets.

Proposition 3 ([15, Prop 7]). *Let $\bar{w} = \langle w_1, \ldots, w_k \rangle$, $k > 0$, and $L \subseteq w_1^* \ldots w_k^*$: $f_{\bar{w}}^{-1}(L)$ is semilinear if and only if L is a finite intersection of CFLs.*

Now we meet the requirements to prove Lemma 2.

Proof (Sketch). The proof of Lemma 2 relies on the observation that

$$L_{\angle} = \{1^n \sharp a^{val} \mid val = \textstyle\sum_{i=0}^{n-1} b^i\} \ .$$

Let $\bar{w} = \langle 1, \sharp, a\rangle$, since $\sum_{i=0}^{n-1} b^i = \frac{b^n-1}{b-1}$ for all $b > 1$, we obtain:

$$f_{\bar{w}}^{-1}(L_{\angle}) = \left\{ \left(i, 1, \tfrac{b^i-1}{b-1}\right) \mid i \ \in \mathbb{N} \right\} \ . \tag{1}$$

Next, we show this set is not semilinear using the facts that (a) the third component grows exponentially in i, and (b) $f_{\bar{w}}^{-1}(L_{\angle})$ is an infinite set. The definition of semilinear set then shows that by taking two elements in (b) we can obtain a third one. We then show that those three elements violate (a) unless they all coincide. The same reasoning remains valid when considering an infinite subset of $L_{\angle}$. The details of the proof are given in Appendix A. □

Once Lemma 2 is proved, the proof of Theorem 1 easily follows.

Proof (of Theorem 1). Assume $L_{len}^{\sharp}$ is a finite intersection of CFLs. Since $1^*\sharp^* a^*$ is a CFL, $L_{\angle}$ is also a finite intersection of CFLs contradicting Lemma 2. □

Our definitions of $L_{len}(n)$ and $L_{len}^{\sharp}$ do not put any constraints on the structure of the word w that follows the length field and the delimiter (if any). In practice, however, the word w may need to satisfy constraints beyond those on its length, such as containment in a specific language.

Theorem 4. *The language $L_{len}^{\sharp} \cap \{x \sharp w \mid x \in B^*,\ w \in L_C\}$ is a finite intersection of CFLs for **no** infinite CFL $L_C \subseteq W^*$.*

The proof of this Theorem follows the same argument used to prove Theorem 1. Hence, we begin by defining a subset of $L_{len}^{\sharp} \cap \{x \sharp w \mid x \in B^*,\ w \in L_C\}$ for which Proposition 3 holds.

Let S be the start symbol of the grammar generating language L_C. Since language L_C is infinite the following must hold: for some non terminal A and $\alpha_i \in W^*$, we have $S \Rightarrow^* \alpha_1 A \alpha_5$; $A \Rightarrow^* \alpha_2 A \alpha_4$; $A \Rightarrow^* \alpha_3$ with $\alpha_2 \neq \epsilon$ or $\alpha_4 \neq \epsilon$.

It follows that $\{\alpha_1 \alpha_2^i \alpha_3 \alpha_4^i \alpha_5 \mid i \geq 0\} \subseteq L_C$ and, thus,

$$L_{\measuredangle} \stackrel{\text{def}}{=} L_{len}^{\sharp} \cap \{x \sharp w \mid x \in B^*,\ w \in L_C\} \cap 1^*\sharp^*\alpha_1^*\alpha_2^*\alpha_3^*\alpha_4^*\alpha_5^*$$

is an infinite language contained in $1^*\sharp^*\alpha_1^*\alpha_2^*\alpha_3^*\alpha_4^*\alpha_5^*$.

Lemma 5. *Language $L_{\measuredangle}$ is not a finite intersection of CFLs. Moreover, no infinite subset of $L_{\measuredangle}$ is a finite intersection of CFLs.*

This Lemma is similar to Lemma 2 and so is the proof, whose details are given in Appendix A. Finally, we proceed to prove Theorem 4 by contradiction.

Proof (of Theorem 4). Assume that $L_{len}^{\sharp} \cap \{x \sharp w \mid x \in B^*,\ w \in L_C\}$ is a finite intersection of CFLs. Since $1^*\sharp^*\alpha_1^*\alpha_2^*\alpha_3^*\alpha_4^*\alpha_5^*$ is context-free, $L_{\measuredangle}$ is also a finite intersection of CFLs, which contradicts Lemma 5. □

3.3 Fixed Size Length Fields

In this section we sidestep the negative results of Section 3.2 by assuming an upper bound on the length field which indeed occurs in some network protocols. Such is the case of the IP, UDP and DNS protocols, whose specifications [1, 20, 22] define 16-bit fields containing the length of the data in terms of bytes. In some cases, assuming an upper bound on the length field, even if it is not defined by the standard, yields no loss of generality for all practical purposes. It is the case for HTTP where the majority of implementations do assume a bound on the size of length fields (e.g. major web browsers all do).

We start with the family of languages $L_{len}(n)$ where the length field is n symbols long. It is easy to see that each language of this family is finite, hence regular. Now we turn to the size of specifications for $L_{len}(n)$. In terms of finite state automata, all automata specifying $L_{len}(n)$ grow exponentially in n. Let $b > 1$ be the base in which the length is encoded, then there are b^n possible encodings for the length. By the pigeonhole principle, having less than b^n reachable states after reading the first n symbols implies that two distinct length encodings end up in the same state, making them indistinguishable for the automaton. Hence, it cannot decide $L_{len}(n)$. However, when $L_{len}(n)$ is specified using context-free grammars, we show that it admits a more compact description.

Theorem 6. *Let Σ be fixed alphabet and $n > 0$, there exists a context-free grammar $G_{len}(n)$ of size $\mathcal{O}(n)$ such that $L(G_{len}(n)) = L_{len}(n)$.*

Proof. For simplicity of presentation we assume that length fields are encoded in binary, i.e. $b = 2$ in the definition of $L_{len}(n)$. The generalization to any $b > 2$ is tedious but straightforward.

The grammar $G_{len}(n)$ is defined by the alphabet Σ, variables $\{S\} \cup \{X_i \mid 0 \leq i \leq n\} \cup \{F_i \mid 0 \leq i \leq n-1\}$ and the following rules:

$$\begin{array}{ll} \{S \to X_0\} & \{X_n \to \varepsilon\} \\ \{X_i \to 0\ X_{i+1} \mid 0 \leq i < n\} & \{X_i \to 1\ X_{i+1}\ F_i \mid 0 \leq i < n\} \\ \{F_j \to F_{j-1}\ F_{j-1} \mid 1 \leq j \leq n-1\} & \{F_0 \to c \mid c \in W\} \end{array}$$

It follows by construction that $L(G_{len}(n)) = L_{len}(n)$. A closer look reveals that, since the alphabet is fixed and therefore so is $|\Sigma| \geq |W|$, the size of the rules of each set is bounded and independent from n while there are $3n+2+|W|$ rules so the size of $G_{len}(n)$ is $\mathcal{O}(n)$. □

Next, we show that $110abc \in L_{len}(3)$ is also contained in $L(G_{len}(3))$.

$$\begin{aligned} S \Rightarrow X_0 \Rightarrow 1X_1F_0 \Rightarrow 11X_2F_1F_0 \Rightarrow 110X_3F_1F_0 \\ \Rightarrow 110F_1F_0 \Rightarrow 110F_0F_0F_0 \Rightarrow^* 110abc \end{aligned}$$

3.4 Chunked Messages

Closely related to length fields are chunked messages, a feature found in the HTTP protocol. According to the standard, the header `Transfer-Encoding: chunked`

signals that the body of the message is divided into chunks, each of which has its size defined by a variable size length field as shown below:

```
HTTP/1.1 200 OK\r\n
Transfer-Encoding: chunked\r\n\r\n
12\r\nThe file is \r\n
16\r\n3,400 bytes long\r\n
0\r\n\r\n
```

Relying on previous definitions we model chunked messages by defining the languages $L^{\sharp}_{chunk} \stackrel{\text{def}}{=} \left(L^{\sharp}_{len}\ \{\sharp\}\right)^{+}$ and $L^{\sharp}_{chunk}(n) \stackrel{\text{def}}{=} \left(L_{len}(n)\ \{\sharp\}\right)^{+}$ for unbounded and fixed (given by n) length field size, respectively. We further assume $\sharp \notin W$ and $\Sigma = B \cup W \cup \{\sharp\}$ to recognize the end of each chunk and thus avoid ambiguity.

Next, we turn to the claims found in the literature [7] about the impossibility of specifying chunked messages using CFLs. The proofs, which are slight variations of proofs for length fields, can be found in Appendix A.

Theorem 7. *$L^{\sharp}_{chunk}$ is not a finite intersection of CFLs.*

Theorem 8. *Let Σ be a fixed alphabet and $n > 0$. The language $L^{\sharp}_{chunk}(n)$ is regular and can be specified by a context-free grammar of size $\mathcal{O}(n)$.*

4 Formal Analysis of (In)equalities

Input validation sometimes requires comparing different parts of a message, e.g., to check that two subwords are identical or that the first one represents a number or a date that is greater than the second one. For instance, an HTTP GET message is valid only if the field `last-byte-pos` is greater than `first-byte-pos`.

4.1 Equality Check

Consider the case of HTTP when a client is asking for a transition to some other protocol. As the standard of mandates, equality should hold between the `Upgrade` fields of the request and its response.

```
======== REQUEST ========    ======== RESPONSE ========
GET /example HTTP/1.1\r\n    HTTP/1.1 101 Switching Protocols\r\n
Upgrade: h2c\r\n             Connection: Upgrade\r\n
                             Upgrade: h2c\r\n
```

Modeling Equality Check. We begin our study of comparisons with the case of two contiguous subwords compared for equality. To this end consider the following language over alphabet Σ given by

$$L^{\sharp}_{=} \stackrel{\text{def}}{=} \{x \sharp y \mid x = y\}\ .$$

This language consists of twice the same word with '$\sharp$' in between. Again, we assume $\sharp$ occurs in x for no x.

When the size of the words x and y is fixed, the delimiter is no longer needed. Thus, we define $L_{=}(n)$

$$L_{=}(n) \stackrel{\text{def}}{=} \{x\ y \mid x, y \in \Sigma^n \ \wedge\ x = y\}\ .$$

Unbounded Size. We now consider the case where the length of the subwords to compare is unbounded. The example at the top of the section requires, when validating a request-response pair of HTTP messages, to check equality across `Upgrade` fields.

This situation is described by the language $L^{\sharp}_{=}$. Next, we recall results by Liu and Weiner [17] and Brough [5] enabling us to show that $L^{\sharp}_{=}$ is not a finite intersection of context-free languages.

Proposition 9 ([5, Prop 2.1]). *For every $k > 0$, the set of languages that are an intersection of k CFLs is closed under (i) inverse GSM mappings, and (ii) union with context-free languages.*

Theorem 10 ([17, Thm 8]). *Let $a_1, \ldots a_k$ be $k > 0$ distinct symbols. Then $L_{(k)} \stackrel{\text{def}}{=} \{a_1^{i_1} a_2^{i_2} \ldots a_k^{i_k} a_1^{i_1} a_2^{i_2} \ldots a_k^{i_k} \mid i_j \geq 0$ for all $j\}$ is an intersection of ℓ context-free languages for **no** $\ell < k$.*

We are now in position to prove our impossibility result about $L^{\sharp}_{=}$.

Theorem 11. *$L^{\sharp}_{=}$ is not a finite intersection of CFLs.*

Proof (Sketch.). For the proof sketch we deliberately ignore the delimiter. Details about how to deal with it can be found in Appendix A.

Assume $L_{=}$ (the delimiterless version of $L^{\sharp}_{=}$) is an intersection of m CFLs. Now observe that $L_{(k)} = L_{=} \cap a_1^* a_2^* \ldots a_k^* a_1^* a_2^* \ldots a_k^*$ This implies $L_{(k)}$ is an intersection of $m + 1$ context-free languages, which contradicts Theorem 10 for $k > m + 1$. □

Fixed Size. Because of the negative result of Theorem 11 we turn back again to the restriction assuming an upper bound on the length of the subwords to compare. We argue next that, in practice, such a restriction is reasonable.

Consider the following HTTP message.

```
HTTP/1.1 200 OK\r\n
Date: Sat, 25 Aug 2012 23:34:45 GMT\r\n
Warning: 112 - "Net down" "Sat, 25 Aug 2012 23:34:45 GMT"\r\n
```

The RFC mandates that the date in the `Warning` header be equal to `Date`. Since date formats have bounded length we immediately have an upper bound of the length of the subwords to compare.

Another example is given by the MIME protocol which allows to split messages into multiple parts provided they are flanked by a user-defined delimiter string. Let us consider an example:

```
MIME-Version: 1.0\r\n
Content-type: multipart/mixed; boundary="Mydelimiter"\r\n\r\n
PREAMBLE to be ignored\r\n--Mydelimiter\r\n
Plain ASCII text.\r\n--Mydelimiter\r\n
Plain ASCII text.\r\n--Mydelimiter--\r\n
EPILOGUE to be ignored.\r\n
```

Observe that the delimiter is first declared, `boundary="Mydelimiter"`, and then `Mydelimiter` is used three times, the first two times as `--Mydelimiter` the last time as `--Mydelimiter--`.

Equality checks can ensure each part is flanked with the same delimiter. In the case of MIME, the standard [12] imposes a maximum length of 69 symbols for the delimiter giving us an upper bound.

Equality checks for a fixed number n of symbols are specified by $L_{=}(n)$. For every n, the language $L_{=}(n)$ is finite, hence regular. Nonetheless Theorem 12, due to Filmus [10], states that this language has no "compact" specification as a grammar.[3] Still it can be represented "compactly" as a finite intersection of CFLs, as shown by Theorem 13. In this section we will study the size of different grammars assuming the alphabet Σ is fixed and, thus, $|\Sigma|$ is a constant.

Theorem 12 ([10, Thm 7]). *Let $|\Sigma| > 2$, every context-free grammar for $L_{=}(n)$ has size* $\Omega\left(\frac{|\Sigma|^{n/4}}{\sqrt{2n}}\right)$.

Recall that $f(n) = \Omega(g(n))$ means that f is bounded from below[4] by g for sufficiently large n, which implies that context-free grammars for $L_{=}(n)$ exhibit exponential growth in n.

Our next theorem based on the observation that $x = y$ iff $(x)_i = (y)_i$ for all i allows to capture $L_{=}(n)$ as a intersection of n CFLs.

Theorem 13. *Let the alphabet Σ be fixed, the language $L_{=}(n)$ is an intersection of n CFLs, each of which is specified by a grammar of size $\mathcal{O}(n)$.*

Proof. Given $i \in \{1, \ldots, n\}$, define the language $L_{=_i}(n)$ over the alphabet Σ given by

$$L_{=_i}(n) \stackrel{\text{def}}{=} \{x\, y \mid x, y \in \Sigma^n,\ (x)_i = (y)_i\} \ .$$

Clearly, for every word u we have $u \in L_{=}(n)$ iff $u \in L_{=_i}(n)$ for all $i \in \{1, \ldots, n\}$. Next, define $G_{=_i}$ as the grammar for $L_{=_i}(n)$ with variables S and T, alphabet Σ and the rules:$\{S \to T^{i-1}\, c\, T^{n-1}\, c\, T^{n-i} \mid c \in \Sigma\}$, $\{T \to c \mid c \in \Sigma\}$. It is routine to check that the size of the grammar is $\mathcal{O}(n)$. □

Above, we studied specification of equality checks for two contiguous subwords. In practice, however, comparisons are often more general. In the previous HTTP example, the dates to compare for equality are not necessarily contiguous. Also, to specify the split messages of MIME using equality checks we need to generalize to the cases where equality covers more than two subwords (each of the multiple

[3] This implies it has no "compact" specification by a finite state automaton either.

[4] $f(n) = \Omega(g(n))$ iff $\exists$ positive c, n_0 s.t. $\forall n > n_0,\ f(n) \geq c \cdot g(n)$

parts is flanked with the same delimiter) and those are not necessarily contiguous (some parts are non empty).

We show this generalization of equality checks can still be specified concisely by a finite intersection of CFLs. For the sake of space, results are deferred to Appendix B.

4.2 Inequality Checks

Thus far, we have focused on languages whose words consist of two equal subwords. However, comparisons sometimes require that the first subword represents a lower number than the second or an earlier date. The following request is asking for bytes of `BigBuckBunny.mp4` between offsets `2833` and `7026`. To be valid the requested range should describe a non empty set.

```
GET /BigBuckBunny.mp4\r\n
Range: bytes=2833-7026\r\n
```

Modeling Inequality Checks. Let $\preceq$ define a total order on Σ. We extend $\preceq$ to Σ^* and denote it $\preceq^*$ as follows. We first define $\preceq^*$ when its arguments have equal length, then we proceed with the general case.

Given $x, y \in \Sigma^*$ of equal length, let p be the least position such that $(x)_p \neq (y)_p$. Then $(x)_p \preceq (y)_p$ iff $x \preceq^* y$. Otherwise (no such position p exists) we also have $x \preceq^* y$ since the two words are equal.

Let us now proceed to the case where x and y have different length and assume x is the shortest of the two words (the other case is treated similarly). Then we have $x \preceq^* y$ iff $x' \preceq^* y$ where $x' = \min_{\preceq}(\Sigma)^{|y|-|x|}x$, that is x' is the result of padding x with the minimal element of Σ so that the resulting word and y have equal length. For instance, $5 \preceq^* 21$ because $05 \preceq^* 21$ where Σ is the set of all digits and $\preceq$ is defined as expected. It is an easy exercise to check that $\preceq^*$ is a total order (hint: $\preceq$ is a total order).

Unbounded Size. Let us turn back to the `Range` field example at the top of the section. To specify the language of valid ranges, since the two subwords are unbounded, a delimiter is needed to indicate the end of the first word. In our example the delimiter is the dash symbol.

Next we define $L^{\sharp}_{\preceq}$, the language deciding unbounded size inequality check using $\sharp$ as a delimiter, as follows:

$$L^{\sharp}_{\preceq} \stackrel{\text{def}}{=} \{x \sharp y \mid x, y \in \Sigma^*,\ x \preceq^* y\} \ .$$

Theorem 14. *The language $L^{\sharp}_{\preceq}$ is not a finite intersection of CFLs.*

Proof. We begin by defining the order $\succeq$ over Σ as $\preceq^{-1}$ and define $\succeq^*$ by replacing $\preceq$ with $\succeq$ in the definition of $\preceq^*$. Clearly $a \preceq b$ iff $b \succeq a$ holds, hence there exists a permutation γ on Σ. Indeed, we can write Σ as the set $\{a_1, \ldots, a_n\}$ such that $a_i \preceq a_j$ iff $i \leq j$. Now define $\gamma \colon a_i \mapsto a_{n+1-i}$. It follows that $a \preceq b$ iff $\gamma(a) \succeq \gamma(b)$.

The previous equivalence naturally lifts to words ($\preceq^*$ and $\succeq^*$), e.g. $v \preceq^* w$ iff $\gamma(v) \succeq^* \gamma(w)$.

Next define

$$L^{\sharp}_{\succeq} \stackrel{\text{def}}{=} \{x \sharp y \mid x, y \in \Sigma^*,\ x \succeq^* y\} \ .$$

Notice that the following equality holds: $L^{\sharp}_{\succeq} = \{\gamma(x) \sharp \gamma(y) \mid x \sharp y \in L^{\sharp}_{\preceq}\}$. Stated equivalently, $\gamma(L^{\sharp}_{\preceq}) = L^{\sharp}_{\succeq}$ where γ is lifted to be a language homomorphism and also $L^{\sharp}_{\preceq} = \gamma^{-1}(L^{\sharp}_{\succeq})$ since γ is a bijection.

Following Proposition 9 (i) finite intersections of CFLs are closed under inverse GSM mapping. This implies that they are also closed under inverse homomorphism such as γ^{-1}.

Assume $L^{\sharp}_{\preceq}$ is a finite intersection of CFLs. It follows from above that so is $L^{\sharp}_{\succeq}$. Finally, consider the equivalence $v = w$ iff $v \preceq^* w$ and $v \succeq^* w$. Lifted to the languages the previous equivalence becomes: $L^{\sharp}_{=} = L^{\sharp}_{\preceq} \cap L^{\sharp}_{\succeq}$.

Since both $L^{\sharp}_{\preceq}$ and $L^{\sharp}_{\succeq}$ are finite intersection of CFLs we conclude that so is $L^{\sharp}_{=}$ which contradicts Theorem 11. □

Fixed Size. With the same motives as for equality checks we turn to the case in which the size of the words to be compared is fixed, say n. As opposed to the unbounded case, we can discard the delimiter because n – the last position of the first word – is known. The message below illustrates an inequality check between fixed size subwords.

```
HTTP/1.1 304 Not Modified\r\n
Date: Tue, 29 Mar 2016 09:05:57 GMT\r\n
Last-Modified: Wed, 24 Feb 2016 15:23:38 GMT\r\n
```

To ensure that this response is valid the `Last-Modified` field must contain a date earlier than the `Date` field.

Let $n > 0$, we define $L_{\preceq}(n)$ to be:

$$L_{\preceq}(n) \stackrel{\text{def}}{=} \{x\ y \mid x, y \in \Sigma^n,\ x \preceq^* y\} \ .$$

Theorem 15. *Let the alphabet Σ be fixed and $n > 0$, $L_{\preceq}(n)$ is a boolean combination of $2n$ languages each one specified by a grammar of size $\mathcal{O}(n)$.*

Proof. Let $G_{=_i}$ be the grammars used in the proof of Theorem 13 and let $G_{\preceq_i}$ a grammar for the language $L_{\preceq_i}(n) \stackrel{\text{def}}{=} \{x\ y \mid x, y \in \Sigma^n,\ (x)_i \preceq (y)_i\}$. Then, by definition of the order $\preceq$ over Σ^n, we write

$$w \in L_{\preceq}(n) \Leftrightarrow w \in L_{=_{1..n}}(n) \bigvee_{i=1}^{n} \left(w \in L_{=_{1..i-1}}(n) \wedge w \notin L_{=_i}(n) \wedge w \in L_{\preceq_i}(n)\right)$$

where $w \in L_{=_{1..i}}(n)$ is equivalent to $w \in \bigcap_{j=1}^{i} L_{=_j}(n)$.

The size of each grammar $G_{=_i}$ was shown to be $\mathcal{O}(n)$. On the other hand, each grammar $G_{\preceq_i}$ is defined by the alphabet Σ, variables S, T and $\{T_a \mid a \in \Sigma\}$ and the rules:

$$\{S \to T^{i-1}\ a\ T^{n-1}\ T_a\ T^{n-i} \mid a \in \Sigma\} \quad \{T_a \to c \mid c \in \Sigma, a \preceq c\} \quad \{T \to c \mid c \in \Sigma\}\ .$$

It is routine to check that the size of the grammar is $\mathcal{O}(n)$. □

The language $L_{\preceq}(n)$ can be extended to describe the situation in which x and y represent dates and $\preceq$ means "earlier than". To this end, whenever the month is given by its name instead of the number thereof we should read it as a single symbol, considering each one as an element of the alphabet. Otherwise, a comparison between numbers as described in proof of Theorem 15 will work. Once we know how to compare the years, months, and days of two dates, combining them to construct the language comparing two dates is straightforward.

5 Practical Evaluation

The results given in Sections 3 and 4 characterize the extent to which (intersections of) CFL can be used to specify common idioms of network protocols. In this section, we demonstrate that the positive theoretical results can be turned into practical input validators for real-world network protocols. We begin by discussing practical encoding issues, before we present an input validator for HTTP.

5.1 Encoding Real-World Protocols as CFG

Encoding effort. The manual effort of translating protocol specification into grammars is facilitated by the RFC format: Protocol RFCs usually consist of a grammar accompanied by a list of additional constraints written in English. This grammar is typically given in ABNF format [6] which easily translates to a context-free grammar. The additional constraints translate to regular expressions or CFGs, along the lines described in this paper. Then the set of valid messages of the protocol is described by a boolean combination of small CFLs.

Encoding size. The grammars required to perform the validation against the idioms discussed in this paper remain small even for real-world protocols:

Length Fields. The CFG for $L_{len}(n)$ consists of $3n + 2$ rules, i.e. it grows linearly in the size of the length field. This implies that it grows only *logarithmically* with the size of the message body, which makes the CFG encoding practical for real-world scenarios.

Comparisons. To compare two strings of length n we need $2n$ grammars each with no more than $3|\Sigma|$ rules where $|\Sigma|$ is the size of the alphabet. In practice, n is small because it is the length of the encoding of a position within a file, a timestamp, a hash value,...

5.2 An Input Validator for HTTP

Next we report on HTTPValidator [24], a proof of concept implementation to validate HTTP messages based on mere CFGs and regular expressions, without using attributes nor semantic actions.

Why HTTP? First, HTTP contains almost all of the features that have been used in the literature [4, 21] to dispute the suitability of CFLs for parsing network protocols. Second, HTTP is a widely used and complex protocol, making it an ideal testbed for our approach. Finally, HTTPolice [8] is a lint for HTTP messages which checks them for conformance to standards and best practices and provides a reference for comparison.

HTTP as CFG. The ABNF described by the standard [9] is translated into a single CFG while constraints such as "A client MUST send a Host header field in all HTTP/1.1 request messages." and "A client MUST NOT send the chunked transfer coding name in TE" are translated into regular expressions and CFGs.

Implementation. Regular expressions and grammars are compiled with Flex and Bison respectively. We avoid conflicts altogether by relying on the `%glr-parser` declaration, which forces Bison to produce a generalized LR parser [5] that copes with unresolved conflicts without altering the specified language. Finally, a script runs all these validators sequentially and combines their boolean outputs to conclude the validation. Table 1 describes the sizes of each separate element of our validator. Further details can be found in the repository [24].

Table 1. Sizes of the formal languages required to validate an HTTP message

Feature	**Size**
HTTP ABNF as a CFG	1013 grammar rules
Decimal length field of size up to 80	871 grammar rules
Comparison of version numbers	3 grammars with 13 grammar rules each
Constraints (91 different ones)	260 regular expressions

Evaluation. We evaluate HTTPValidator on messages obtained from real-world traffic (using Wireshark) and on messages provided with HTTPolice as test cases. In total we thus obtain 239 test cases of which HTTPolice classifies 116 as valid and 123 as invalid HTTP. We run HTTPValidator on these test cases obtaining the same classification as HTTPolice but for two false positives. These errors are due to well-formedness checks on message bodies in JSON and XML format, which we currently do not consider in HTTPValidator but HTTPolice does.

[5] `https://www.gnu.org/software/bison/manual/html_node/GLR-Parsers.html`

The time required for evaluating all test cases [6] is 16.1s for HTTPValidator and 60s for HTTPolice, i.e. we achieve a 4-fold speedup. Note that this comparison is slightly biased towards HTTPValidator because HTTPolice relies on interpreted Python code whereas the parsers in HTTValidator are compiled to native code. Moreover, we store each of the test cases in a single file, forgoing HTTPolice's ability to process several HTTP messages in a single file. On the other hand, we have put ease of implementation before performance so no parallelization has been implemented so far.

Overall, the experimental evaluation shows that, on our testbed, HTTPValidator achieves coverage and performance that is competitive with the state-of-the-art in the field, thereby demonstrating the practicality of our approach.

6 Related Work

We discussed related work on language theory and input validation in the paper body. Here we focus on discussing recent efforts for building parser generators for network protocols.

In recent years, a number of parser generators for network protocols have emerged. They are often parts of larger projects, but can be used in a standalone fashion. Important representatives are BinPac [21], which is part of the Bro Network Security Monitor [7], UltraPac [16], which is part of the NetShield Monitor, Gapa [4], FlowSifter [18], and Nail [2]. The difference to our approach is that they are all are built from scratch, whereas we rely on established CFG parsing technology. Moreover, they rely on user-provided code for parsing idioms such as length fields, whereas we specify everything in terms of (intersections of) CFG. However, we emphasize that the focus of our approach lies on the task of *input validation*, whereas those approaches deal with *parsing*, i.e. they additionally fill a data structure.

Among the previous parser generators, Gapa and Nail stand out in terms of their safety features. Gapa achieves a degree of safety by generating parsers in a memory-safe language. Note that this does not prevent runtime error, e.g., dividing by zero still remains possible. Nail also aims at safety by providing some automated support for filling user-defined data structure therefore reducing the risk of errors introduced by user-defined code. In contrast, we do not rely on any user-provided code.

Another line of work [7] relies on the use of the so-called attribute grammars, an extension of context-free grammar that equips rules with attributes that can be accessed and manipulated. For the parser generator, the authors use Bison and encode the attribute aspect of grammars through user-defined C code annotating the grammar rules which, as we argued before, augments the risk of errors.

[6] We run our experiments on an Intel Core i5-5200U CPU 2.20GHz with 8GB RAM.
[7] `https://www.bro.org/`

7 Conclusions and Future Work

Input validation is an important step for defending against malformed or malicious inputs. In this paper we perform the first rigorous, language theoretic study of the expressiveness required for validating a number of common protocol idioms. We further show that input validation based on formal languages is practical and build a modular input validator for HTTP from dependable software components such as off-the-shelf parser generators for context-free languages. Our experimental result shows that our approach is competitive with the state-of-the-art input validator for HTTP in terms of coverage and speed.

There are some promising avenues for extending our work. For instance our approach can be generalized to boolean closures of CFLs, which are known to be strictly more expressive than the finite intersection we deal with in this paper [5]. Besides, our approach can be extended with a notion of state that is shared between protocol participants which will allow us to implement, e.g., stateful firewalls using our approach.

References

[1] Internet Protocol. RFC 791 (Proposed Standard), Sept. 1981.
[2] J. Bangert and N. Zeldovich. Nail: A practical tool for parsing and generating data formats. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.
[3] J.-P. Bernardy and P. Jansson. Certified context-free parsing: A formalisation of valiant's algorithm in agda. *Logical Methods in Computer Science*, 12(2), 2016.
[4] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo. Generic application-level protocol analyzer and its language. In *NDSS 2007*, 2007.
[5] T. Brough. Groups with poly-context-free word problem. *Groups Complexity Cryptology*, 6(1), Jan 2014.
[6] P. D. Crocker, Ed. Brandenburg InternetWorking. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (Proposed Standard), Jan. 2008.
[7] D. Davidson, R. Smith, N. Doyle, and S. Jha. Protocol normalization using attribute grammars. In *ESORICS 2009*, LNCS, 2009.
[8] V. Faronov. HTTPolice. `https://github.com/vfaronov/httpolice`, 2017.
[9] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014.
[10] Y. Filmus. Lower bounds for context-free grammars. *Information Processing Letters*, 111(18), 2011.
[11] D. Firsov and T. Uustalu. Certified CYK parsing of context-free languages. *Journal of Logical and Algebraic Methods in Programming*, 83(5-6), 2014.
[12] N. Freed and N. Borenstein. Multipurpose internet mail extensions (mime). RFC 1341 (Proposed Standard), Nov. 1996.
[13] J. Graham-Cumming. Inside shellshock: How hackers are using it to exploit systems. `https://blog.cloudflare.com/inside-shellshock/`.
[14] Lastpass security updates. `https://blog.lastpass.com/2016/07/lastpass-security-updates.html/`.
[15] M. Latteux. Intersections de langages algébriques bornés. *Acta Informatica*, 11(3), 1979.

[16] Z. Li, G. Xia, H. Gao, Y. Tang, Y. Chen, B. Liu, J. Jiang, and Y. Lv. NetShield: massive semantics-based vulnerability signature matching for high-speed networks. In *ACM SIGCOMM 2010*, 2010.
[17] L. Y. Liu and P. Weiner. An infinite hierarchy of intersections of context-free languages. *Mathematical systems theory*, 7(2), 1973.
[18] C. R. Meiners, E. Norige, A. X. Liu, and E. Torng. Flowsifter: A counting automata approach to layer 7 field extraction for deep flow inspection. In *IEEE INFOCOM 2012*, 2012.
[19] Microsoft releases security advisory 2953095. `https://technet.microsoft.com/library/security/2953095`.
[20] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035 (Proposed Standard), Nov. 1987.
[21] R. Pang, V. Paxson, R. Sommer, and L. L. Peterson. binpac: a yacc for writing application protocol parsers. In *ACM SIGCOMM IMC 2006*, 2006.
[22] J. Postel. User Datagram Protocol. RFC 768 (Proposed Standard), Aug. 1980.
[23] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002.
[24] P. Valero. HTTPValidator. `https://github.com/pevalme/HTTPValidator`, Apr 2017.

A Deferred Proofs

A.1 Proof of Lemma 2

Proof (of Lemma 2). According to Proposition 3, if $L_\angle$ is a finite intersection of CFLs then the set $f_w^{-1}(L_\angle)$ is semilinear, which implies that it is a finite union of linear sets. Since the set is infinite, at least one of the linear sets must be infinite. We call this set S.

Any element in S will have the form defined in 1 so define $\boldsymbol{x}, \boldsymbol{y} \in S$ as follows:

$$\boldsymbol{x} = \left(i_x, 1, \frac{b^{i_x}-1}{b-1}\right), \quad \boldsymbol{y} = \left(i_y, 1, \frac{b^{i_y}-1}{b-1}\right) \ .$$

Let $\delta = i_y - i_x$, we write

$$\boldsymbol{y} = \left(i_x + \delta, 1, \frac{b^{i_x+\delta}-1}{b-1}\right) \ .$$

Without loss of generality, assume $\delta > 0$ (for otherwise swap $\boldsymbol{x}$ and $\boldsymbol{y}$). Next define

$$\Delta = \boldsymbol{y} - \boldsymbol{x} = \left(\delta, 0, \frac{b^{i_x}(b^\delta - 1)}{b-1}\right) \ .$$

Let $\boldsymbol{z} = \boldsymbol{x} + 2\Delta$:

$$\boldsymbol{z} = \left(i_x + 2\delta, 1, \frac{b^{i_x} - 1 + 2b^{i_x}(b^\delta - 1)}{b-1}\right) = \left(i_x + 2\delta, 1, \frac{b^{i_x}(2b^\delta - 1) - 1}{b-1}\right) \ . \quad (2)$$

Since the set S is linear we find that $\boldsymbol{z} \in S$ so we can write:

$$\boldsymbol{z} = \left(i_z, 1, \frac{b^{i_z} - 1}{b - 1}\right)$$

which means that $i_z = i_x + 2\delta$ obtaining:

$$\boldsymbol{z} = \left(i_x + 2\delta, 1, \frac{b^{i_x+2\delta} - 1}{b - 1}\right) \quad . \tag{3}$$

Let us now derive a contradiction using (2) and (3). For this, we start by deriving the following equivalence:

$$b^{i_x}(2b^\delta - 1) = b^{i_x+2\delta} \text{ iff } 2b^\delta - 1 = b^{2\delta} \quad .$$

Applying the change of variable $t = b^\delta$ we obtain

$$2t - 1 = t^2 \text{ iff } t^2 - 2t + 1 = 0 \quad .$$

Solving the equation give us $t = 1$ iff $\delta = 0$.

So the vector $\boldsymbol{z}$ obtained basing on the linearity of the set S will belong to S if and only if $\boldsymbol{z} = \boldsymbol{y} = \boldsymbol{x}$.

We conclude that the set S is not infinite, in fact, if it is linear then it contains only one element. Thus the set $f_w^{-1}(L_\angle)$ can not be written as a finite union of linear set so the language $L_\angle$ is not a finite intersection of CFLs. □

A.2 Proof of Lemma 5

Proof (of Lemma 5). This proof is conceptually identical to the previous one but requires a slight modification in the notation used. Now, vectors in S have 7 components and the linear combination of the last five ones equals to the *old* third component.

Thus, vector $\boldsymbol{x} \in S$ should be defined as:

$$\boldsymbol{x} = \left(i_x, 1, n_x^{(1)}, n_x^{(2)}, n_x^{(3)}, n_x^{(4)}, n_x^{(5)}\right) \quad \text{with } \frac{b^{i_x} - 1}{b - 1} = \sum_{k=1}^{5} |\alpha_k| \times n_x^{(k)} \quad .$$

Having this definition, the proof follows that of Lemma 2, considering the sum of the last five components rather than their concrete values. Assuming $L_\measuredangle$ is a finite intersection of context-free languages, there exist three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in S$ such that $\boldsymbol{z} = \boldsymbol{x} + 2\boldsymbol{\Delta}$ where $\boldsymbol{\Delta} = \boldsymbol{y} - \boldsymbol{x}$. For the sum of the last five components of vectors $\boldsymbol{z}$ and $\boldsymbol{x} + 2\boldsymbol{\Delta}$ to be equal we require $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ to have the same first component ($\delta = 0$).

Sharing the first component implies that each linear set, S, relates (by f_w^{-1}) only with vectors with the same first component. Therefore assuming $f_w^{-1}(L_\measuredangle)$ is as a finite union of linear sets requires $L_\measuredangle$ to only contain words with a finite number of different lengths, which contradicts the fact that it is infinite. By Proposition 3 we conclude that $L_\measuredangle$ is not a finite intersection of CFLs.

□

A.3 Proof of Theorem 7

Proof (of Theorem 7). Let $a \in \Sigma$ be such that $a \neq 1$ and $a \neq \sharp$. One can easily see that

$$L_{\angle}\{\sharp\} \stackrel{\text{def}}{=} L^{\sharp}_{chunk} \cap 1^*\sharp^* a^* \sharp^* \ .$$

Assuming $L^{\sharp}_{chunk}$ is a finite intersection of CFLs then we find that so is $L_{\angle}\{\sharp\}$. Let $\bar{w} = \langle 1, \sharp, a, \sharp \rangle$, then

$$f^{-1}_{\bar{w}}(L_{\angle}\{\sharp\}) = \left\{ \left(i, 1, \tfrac{b^i - 1}{b-1}, 1\right) \mid i \ \in \mathbb{N} \right\} \ .$$

From there, it is straightforward to extend the result of Lemma 2 to show that $L_{\angle}\{\sharp\}$ is not a finite intersection of CFLs, hence derive a contradiction. □

A.4 Proof of Theorem 8

Proof (of Theorem 8). Recall the grammar $G_{len}(n)$ for the language $L_{len}(n)$. The grammar $G_{chunk}(n)$ is defined by adding to $G_{len}(n)$ a fresh start variable Z and the following two rules: $\{Z \to S \ \sharp \ Z, Z \to S \ \sharp\}$. Clearly, $L(G_{chunk}(n)) = L^{\sharp}_{chunk}(n)$ and the size of $G_{chunk}(n)$ is $\mathcal{O}(n)$ since the size of $G_{len}(n)$ is $\mathcal{O}(n)$ due to Theorem 6

□

A.5 Proof of Theorem 11

Proof (of Theorem 11). We first set up languages and a GSM mapping to take proper care of the delimiter. Then we proceed with the proof of theorem. Let $L^{\sharp}_{(k)}$ be the language obtained by adding the delimiter to $L_{(k)}$, that is

$$L^{\sharp}_{(k)} = \{a_1^{i_1} a_2^{i_2} \dots a_k^{i_k} \sharp a_1^{i_1} a_2^{i_2} \dots a_k^{i_k} \mid i_j \geq 0 \text{ for all } j\} \ .$$

Now given $k > 0$ define[8]

$$L_{aux} = L^{\sharp}_{(k)} \cap \overline{\{a_i^j \sharp z \mid 1 \leq i \leq k, \ j \geq 0, \ z \in \Sigma^*\}} \ . \tag{4}$$

Intuitively, L_{aux} is obtained from $L^{\sharp}_{(k)}$ by filtering out the words where at most one symbol occurs to the left of the delimiter, e.g. $a_3a_3\#a_1a_2a_3$ is such a word whereas $a_1a_2\#$ is not. Observe that the set $\{a_i^j \sharp z \mid 1 \leq i \leq k, \ j \geq 0, \ z \in \Sigma^*\}$ is regular, and so is its complement.

Next, we define the GSM mapping π which intuitevely inserts the delimiter at the "right" place. Unless stated otherwise, when describing π, we assume the GSM outputs exactly what it reads.

The GSM reads the first symbol, say a_i, and moves to a state q_i keeping track of that symbol. Notice that the first symbol need not be the delimiter following

[8] Given L over Σ, let $\overline{L}$ denotes the complement of L, that is $\Sigma^* \setminus L$.

equation (4). Then π keeps reading the same a_i symbol without leaving q_i. Upon reading some symbol $x \neq a_i$, π moves to a different state wherein it keeps reading symbols $x \neq a_i$ until it reads a_i again. At that point, π has reached the middle of the word and so it outputs $\sharp$ (besides a_i) and keeps processing the remainder of the word.

From above, we find that $\pi^{-1}(L_{aux})$ coincides with $L_{(k)}$ minus the words where at most one symbol occurs, i.e. $\{(a_i a_i)^j \mid 1 \leq i \leq k,\ j \geq 0\}$.

Now we reinstate these words and obtain the following equality:

$$L_{(k)} = \pi^{-1}(L_{aux}) \cup \{(a_i a_i)^j \mid 1 \leq i \leq k,\ j \geq 0\} \ .$$

We are in position to prove the statement of the theorem.

It is easy to see that the following equality holds:

$$L^{\sharp}_{(k)} = L^{\sharp}_{=} \cap a_1^* a_2^* \dots a_k^* \sharp a_1^* a_2^* \dots a_k^*$$

Now assume $L^{\sharp}_{=}$ is a finite intersection of m context-free languages. Then L_{aux} is a finite intersection of $m+2$ CFLs by equation 4. Proposition 9 (*i*) shows that $\pi^{-1}(L_{aux})$ is a finite intersection of $m+2$ CFLs, hence we find that $L_{(k)}$ is an intersection of $m+2$ CFLs by Proposition 9 (*ii*). Notice that this holds for all values of k and in particular for $k > m+2$ which contradicts Theorem 10. □

B General equality checks

To specify general equality checks we consider the following language over Σ where n is the upper bound on the size of word to check equality for:

$$L_d(n) \stackrel{\text{def}}{=} \{w\ x\ w\ y_1 \dots w\ y_k\ w\ z \ \mid |w| = n \wedge x \in L_{fm} \wedge \bigwedge_j y_j \in L_{mm} \wedge z \in L_{lm}\}$$

where L_{fm}, L_{lm} and L_{mm} are CFLs.

It is routine to check that $L_d(n)$ is context-free for every value of n. However, Theorem 12 shows there is no concise specification for it. Therefore, we study the size of specifications given as a finite intersection of CFLs.

Theorem 16. *Assuming a fixed size grammar specification for the languages L_{fm}, L_{mm} and L_{lm}, the language $L_d(n)$ over fixed size alphabet Σ is a finite intersection of n CFLs, each one specified by a grammar of size $\mathcal{O}(n)$.*

Proof. Given $i \in \{1, \dots, n\}$, define the language $L_{d_{=_i}}(n)$ over alphabet Σ given by:

$$\{w_0\ x\ w_1\ y_1 \dots w_k\ y_k\ w_{k+1}\ z \ \mid \bigwedge_\ell (|w_\ell| = n \wedge (w_\ell)_i = (w_{\ell+1})_i) \\ \wedge x \in L_{fm} \wedge \bigwedge_j y_j \in L_{mm} \wedge z \in L_{lm}\} \ .$$

It is easily seen that give a word u we have $u \in L_d(n)$ iff $u \in L_{d_{=_i}}(n)$ for all $i \in \{1, \ldots, n\}$.

Let S_{fm}, S_{mm} and S_{lm} be the start symbols of the grammars defining languages L_{fm}, L_{mm} and L_{lm} respectively. Next, define $G_{d_{=_i}}(n)$ as the grammar for $L_{d_{=_i}}(n)$ with alphabet Σ, set of variables $\{S, T, P, Q, S_{fm}, S_{mm}, S_{lm}\}$ and rules:

$$\{S \to P\ S_{fm}\ Q\ P\ S_{lm}\} \qquad \{T \to c \mid c \in \Sigma\}$$
$$\{P \to T^{i-1}\ c\ T^{n-i} \mid c \in \Sigma\} \qquad \{Q \to P\ S_{mm}\ Q\} \qquad \{Q \to P\ S_{mm}\}\ .$$

$G_{d_{=_i}}(n)$ also contains all variables and rules of the grammars specifying languages L_{fm}, L_{mm} and L_{lm}. Because the size of the grammars defining the languages L_{fm}, L_{mm} and L_{lm} is fixed and independent of n, it is routine to check that the size of $G_{d_{=_i}}(n)$ is $\mathcal{O}(n)$. □