Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 344 (2019) 151–167

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Some Applications of the Formalization of the Pumping Lemma for Context-Free Languages

Marcus V. M. Ramos[1]

Colegiado de Engenharia de Computação
UNIVASF
Juazeiro, Bahia, Brazil

José Carlos Bacelar Almeida[2]

HASLab - INESC TEC
Universidade do Minho
Braga, Portugal

Nelma Moreira[3]

Departamento de Ciência de Computadores
Faculdade de Ciências
Porto, Portugal

Ruy J. G. B. de Queiroz[4]

Centro de Informática
UFPE
Recife, Pernambuco, Brazil

Abstract

Context-free languages are highly important in computer language processing technology as well as in formal language theory. The Pumping Lemma for Context-Free Languages states a property that is valid for all context-free languages, which makes it a tool for showing the existence of non-context-free languages. This paper presents a formalization, extending the previously formalized Lemma, of the fact that several well-known languages are not context-free. Moreover, we build on those results to construct a formal proof of the well-known property that context-free languages are not closed under intersection. All the formalization has been mechanized in the Coq proof assistant.

Keywords: non-context-free languages, closure, intersection, pumping lemma, formalization, Coq

[1] Email:marcus.ramos@univasf.edu.br
[2] Email:jba@di.uminho.pt
[3] Email:nam@dcc.fc.up.pt
[4] Email:ruy@cin.ufpe.br

https://doi.org/10.1016/j.entcs.2019.07.010

1 Introduction

A context-free grammar G is a four-tuple (V, Σ, P, S) where V is the vocabulary (a finite set consisting of terminal and non-terminal symbols), Σ is the set of terminal symbols or alphabet (thus $N = V \backslash \Sigma$ is the set of non-terminal symbols), P is the (finite) set of rules of the form $X \to \beta$, where $X \in N$ and $\beta \in V^*$, and S is the start symbol of the grammar, $S \in N$. A language is a set of words (also called "sentences") defined over an alphabet. The language L generated by a context-free grammar G is the (finite or infinite) set of words that can be obtained from the start symbol of the grammar by repeated use of its rules: $L(G) = \{w \mid S \Rightarrow^* w \text{ and } w \in \Sigma^*\}$. A language is context-free (CFL for short) if there exists a context-free grammar that generates it. Otherwise, the language is not context-free.

To prove that a language is not context-free requires, thus, to prove that there is no context-free grammar that generates it (a similar argument can be formulated about the nonexistence of pushdown automata that accept the language, but this is not in the scope of this work). This task can be simplified, however, by exploring a property that is observed by all context-free languages. This is accomplished by means of the Pumping Lemma for Context-Free Languages, which can be used to prove that a given language is not context-free. Proving that an arbitrary language is context-free is, however, an undecidable property [3].

The Pumping Lemma for CFLs was stated and proved for the first time by Bar-Hillel, Perles and Shamir in 1961 [1]. In what follows, it will be referred simply as "Pumping Lemma".

The main objectives of this paper are:

(i) Derive formal proofs that some well-known, classic languages, are not context-free. For this, we use the formalization of the Pumping Lemma previously obtained by the authors [7] in the Coq proof assistant [2]. For each of these languages, we discuss the formalization of their non-context-freeness and make hopefully useful considerations about the proof construction process and the complexity of the corresponding formal and text proofs;

(ii) Develop a formal proof of the fact that the class of the context-free languages is not closed under the intersection operation. For that, we follow the classical proof that uses a counter-example, which in our case is one of the languages proved not to be context-free in the previous objective.

The results presented here are important for various reasons. First, they are applications of the previous formalization of the Pumping Lemma. Second, they are the first ever languages to be proved not to be context-free using a computerized theorem prover. As far as the authors are aware of, all proofs of any language claimed to be non-context-free published until now are text (pen and paper) ones, as can be found in textbooks, papers and lectures on the subject. Thus, the present work brings mathematical formalization into a new area of application. Third, they give rise to interesting considerations about building formal proofs from text proofs. Fourth, they can be very useful in teaching the theory of context-free languages within a logic and formal background, with the help of interactive theorem provers and Coq

in particular. Besides this, we extend our previous results on the formalization of closure properties for context-free languages (closure under union, concatenation and Kleene star, [9]) with a new result with respect to language intersection.

The work presented here is part of a long-term project aimed at the formalization of context-free languages and grammars. It started with the formalization of closure properties for context-free grammars [9], evolved later into the formalization of context-free grammar simplification [10] and then into the Chomsky normalization of context-free grammars. Formalization of simplification enabled Chomsky normalization, which in turn enabled the formalization of the Pumping Lemma and then the results presented here. The whole work is described in detail in [5], and more information can be found in [8]. Previous results will not be discussed here and can be retrieved from the references.

In order to follow this paper, the reader is required to have basic knowledge of Coq and of context-free language theory. Background on context-free language theory can be found in [12], [3] or [11], among others.

The statement and applications of the Pumping Lemma for CFLs are presented in Section 2. The approach that we have adopted towards the formalization of languages in general is discussed in Section 3. Then, in Sections 3.1, 3.2 and 3.3, respectively, we present formalizations of three different and well-known languages using the approach discussed before. Besides that, each of these sections contains a text proof of the fact that the language is not context-free, a discussion about the formalization of such a proof and considerations about the complexity of the text and formal proofs. In Section 4 we describe our formalization of the non-closure of context-free languages under intersection, a result that is built on top of formalization described in 3.3. Final conclusions are presented in Section 5.

The definitions and proof scripts discussed in this paper were written in plain Coq and are available for download at [6]. Statements about prime numbers, required in Section 3.2, were written in SSReflect (a Coq plugin) and adapted from existing proofs in the Mathematical Components library [4].

2 Pumping Lemma for Context-Free Languages

The Pumping Lemma states that, for every context-free language and for every sentence of such a language that has a minimum length, it is possible to obtain an infinite number of new sentences that must also belong to the language. This minimum length depends on the definition of the language.

Let $\mathcal{L}$ be a context-free language defined over alphabet Σ. Then there is a number n, depending only on $\mathcal{L}$, such that for every sentence $\alpha \in \mathcal{L}$, if $|\alpha| \geq n$, then all of the following are true ($|w|$ denotes the length of the word w):

- $\exists\, u, v, w, x, y.(\alpha = uvwxy)$;
- $|vx| \geq 1$;
- $|vwx| \leq n$;
- $\forall i.(uv^i wx^i y \in \mathcal{L})$

A more concise, yet more difficult to read, representation of this statement is ("cfl" is a predicate that asserts that a language is context-free, see Section 3):

$$\forall \mathcal{L}.(\text{cfl } \mathcal{L}) \rightarrow \exists n.\forall \alpha.$$
$$\Big(\big((\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n)\big) \rightarrow \exists\, u, v, w, x, y.\Big((\alpha = uvwxy) \wedge (|vx| \geq 1)$$
$$\wedge (|vwx| \leq n) \wedge (\forall i.uv^i wx^i y \in \mathcal{L})\Big)\Big)$$

A typical use of the Pumping Lemma is to show that a given language is not context-free by using the contrapositive of the statement of the lemma. The informal proof proceeds by contraposition: the language is assumed to be context-free, and this leads to a contradiction from which one concludes that the language in question can not be context-free.

For details about our previous formalization of the Pumping Lemma, please refer to [7].

The non-context-free languages formalized in this work are:

(i) *square*: $\{w \in \{a\}^* \mid \exists\, i, |w| = i^2, i \geq 0\}$,

(ii) *prime*: $\{w \in \{a\}^* \mid |w|$ is a prime number$\}$,

(iii) *anbncn*: $\{w \in \{a, b, c\}^* \mid \exists\, i, w = a^i b^i c^i, i \geq 0\}$.

For each of these, we will refer to the text proofs of their non-context-freeness using the Pumping Lemma, as well as to the corresponding formalization in Sections 3.1, 3.2 and 3.3.

The Pumping Lemma does not characterize the CFLs, however, since it is also verified by some non-CFLs [3]. Besides that, the authors are not aware of any independent characterization of the class of languages that satisfy it.

3 Languages

In order to formally prove that a language is not context-free, we first need to have a formal definition of such a language. The definitions of the languages *square*, *prime* and *anbncn* introduced before are presented in the next three sections, however they all share the same and more fundamental definitions discussed in this section.

A language is defined as a predicate that maps a sentence (a list of terminal symbols) to a proposition (`Prop`):

```
Definition lang (terminal: Type):= list terminal → Prop.
```

Two languages are equal if they have the same sentences:

```
Definition lang_eq (l k: lang) :=
∀ w, l w ↔ k w.
```

Finally, a language is context-free if it is generated by some context-free grammar (for the definition of a context-free grammar in Coq, and other related definitions, please refer to [8]). The following definition is a predicate that represents this

property:

```
Definition cfl (terminal: Type) (l: lang terminal): Prop:=
∃ non_terminal: Type,
∃ g: cfg non_terminal terminal,
   lang_eq l (lang_of_g g).
```

where `lang_of_g` represents the language generated by grammar `g`:

```
Definition lang_of_g (g: cfg non_terminal terminal): lang :=
fun w: sentence ⇒ produces g w.
```

Thus, the definition of a new language comprises, essentially, the definition of a new type `terminal` whose constructors are the elements of Σ and of a membership predicate that asserts whether an arbitrary list of terminal symbols is a word of the language or not. This predicate, in our case, will carry the name of the language being defined. Examples of this are presented in the next three sections.

3.1 Language square

Language *square* ($\{a^i \mid i$ is the square of some number$\}$) is defined over a single symbol alphabet and contains words whose length corresponds to the square of some natural number. Thus, it contains words such as ϵ, a, $aaaa$, $aaaaaaaaa$ etc.

Text proof

To prove that *square* is not context-free, suppose that it is context-free and consider the word a^m, where $m = n^2$ and n is the constant of the Pumping Lemma. It is easy to observe that $a^m \in$ *square* and that $|a^m| \geq n$. Thus, the Pumping Lemma can be applied. Then, $a^m = uvwxy$ for some u, v, w, x and y, with $|uvwxy| = n^2$, $1 \leq |vwx| \leq n$ and $uv^iwx^iy \in$ *square*, $\forall i \geq 0$. But take $i = 2$. Then, $|uv^2wx^2y| = |uvwxy| + |vx|$. Since $|uvwxy| = n^2$ and $1 \leq |vx| \leq n$, we have $n^2 < |uv^2wx^2y| \leq n^2+n$, which is the same as $n^2 < |uv^2wx^2y| < (n+1)^2$. However, there is no such a number that is the square of another number and lies between the squares of two consecutive numbers. Thus, the hypothesis is false and *square* is not context-free.

Formalization of the definition

The language *square* is defined in our formalization as follows:

```
Inductive terminal: Type:=
| a.

Definition square: lang terminal:=
fun (s: list terminal) ⇒
∃ i: nat,
length s = i*i.
```

The type `terminal` has a single constructor `a`, which corresponds to the single element of the alphabet. Definition `square` is a predicate on lists of terminals, expressing the property that the length of the list is a square of some number. It embeds the property that all words of the language must satisfy, namely that the length of the word must be the square of some number.

Formalization of the statement

The statement of this lemma simply says that the predicate `square` can not represent a context-free language because it does not satisfy the predicate `cfl`:

```
Lemma not_cfl_square: ~ cfl square.
```

Formalization of the proof

The formal proof follows closely the argument and steps of the text proof. Two simple auxiliary lemmas had to be proved, however. The first asserts the existence of an infinite number of words in *square*:

$$\forall n.\exists w.(|w| = n^2) \wedge (w \in square)$$

while the second asserts the key property of natural numbers used in the proof[5]:

$$\forall i, j, k.(j \geq 1) \wedge (j \leq i) \rightarrow \neg(i^2 + j = k^2)$$

Comparison of the formal and text proofs

While the text proof is less than 10 lines long, the formal proof script is approximately 200 lines long. Despite the expansion factor being significant, the proof is simple and very readable. The size, in this case, can be justified by the style adopted in the writing of the script (with only one tactic per line, for example) and the details that are inherent to the proof. The elegance of this formalization stimulated the authors to pursue the formalization of the next section.

3.2 Language prime

Language *prime* ($\{a^i \mid i$ is a prime number$\}$) is defined over a single symbol alphabet and contains words whose length is a prime number. Thus, it contains words such as *aa*, *aaa*, *aaaaa*, *aaaaaaa* etc.

Text proof

To prove that *prime* is not context-free, suppose that it is context-free and consider the word a^m, where $m \geq n + 2$ is a prime number and n is the constant of the Pumping Lemma. It is easy to observe that $a^m \in prime$ and that $|a^m| \geq n$. Thus, the Pumping Lemma can be applied. Then, $a^m = uvwxy$ for some u, v, w, y and y, with $|uvwxy| = m$, $1 \leq |vwx| \leq n$ and $uv^iwx^iy \in prime, \forall i \geq 0$. But make $i = |uwy|$. Then, $|uv^{|uwy|}wx^{|uwy|}y| = |uwy| + |uwy| * |vx| = |uwy| * (1 + |vx|)$. Since $|uvwxy| \geq n + 2$ and $n \geq |vx| \geq 1$, we have that $|uwy| \geq 2$. Also, that $1 + |vx| \geq 2$, since $|vx| \geq 1$. Thus, the length of the new word is a composite number and not a prime number. The hypothesis is false and *prime* is not context-free.

[5] The scripts of this section can be found in file `pumping_square.v`.

Formalization of the definition

The language *prime* is defined in our formalization as follows:

```
Inductive terminal: Type:=
| a.

Definition prime_lang: lang terminal:=
fun (s: list terminal) ⇒
∃ i: nat,
is_prime i ∧
length s = i.
```

The predicate `is_prime` is the primality predicate.

As in the previous case, the type `terminal` has a single constructor `a`. Definition `prime` is a predicate on lists of terminals, expressing the property that the length of the list is a prime number. It embeds the property that all words of the language must satisfy, namely that the length of the word must be a prime number.

Formalization of the statement

As before, the statement of this lemma says that the predicate `prime_lang` can not represent a context-free language because it does not satisfy the predicate `cfl`:

```
Lemma not_cfl_prime: ∼cfl prime_lang.
```

Formalization of the proof

Similar to the previous case, the formal proof follows closely the argument and steps of the text proof. Three auxiliary lemmas had to be proved, however. The first asserts the existence of an infinite number of words in *prime*:

$$\forall n.\exists w.(|w| \geq n) \wedge (\text{prime } |w|)$$

while the second and third assert key properties of prime numbers used in the proof:

$$\forall i.\exists j.(j \geq i) \wedge (\text{prime } j)$$
$$\forall n, p, q.(n = p * q) \wedge (p \geq 2) \wedge (q \geq 2) \rightarrow \neg(\text{ prime } n)$$

The proof of the two lemmas on prime numbers was accomplished by means of previously existing proofs in the Mathematical Components library [4] [6] .

Comparison of the formal and text proofs

Also in this case, the text proof is less than 10 lines long, while the formal proof script is approximately 200 lines long, elegant, simple and readable. This and the previous result led the authors to work on the formalization of the next section.

3.3 Language anbncn

Language *anbncn* ($\{a^i b^i c^i \mid i \geq 0\}$) is defined over a three symbol alphabet (a, b and c) and contains words that start with some number of as, followed by the same

[6] The scripts of this section can be found in file `pumping_prime.v`.

number of bs and then by the same number of cs. Thus, it contains words such as ϵ, abc, $aabbcc$, $aaabbbccc$ etc.

Text proof

To prove that $anbncn$ is not context-free, suppose that it is context-free and consider the word $a^n b^n c^n$, where n is the constant of the Pumping Lemma. It is easy to observe that $a^n b^n c^n \in anbncn$ and that $|a^n b^n c^n| \geq n$. Thus, the Pumping Lemma can be applied. Then, $a^n b^n c^n = uvwxy$ for some u, v, w, x and y, with $|uvwxy| = 3n$, $1 \leq |vwx| \leq n$ and $uv^i wx^i y \in anbncn, \forall i \geq 0$. It can be observed that vwx, due to its length limitation, contains only one or two different kind of symbols. If it contains only one kind of symbol, then v and x are also built out of a single symbol and the pumping of v and x will change the number of a single symbol, while the number of the other two remain unchanged. Thus, the new word can not belong to $anbncn$. If it contains two different kinds of symbols, then v and x might contain one or two different kinds of symbols each. If both contain only one kind of symbol, pumping will change the number of at most two symbols, while the third will remain unchanged. If v or x contain two different kinds of symbols, pumping will lead to a word where the order is not respected (first as, then bs then cs). In all cases, the new word does not belong to $anbncn$. Thus, the hypothesis is false and $anbncn$ is not context-free.

Formalization of the definition

The language $anbncn$ is defined in our formalization as follows:

```
Inductive terminal: Type:=
| a
| b
| c.

Definition anbncn: lang terminal:=
fun (s: list terminal) ⇒
∃ x y z: list terminal,
∃ i: nat,
s = x ++y ++z ∧
    length x = i ∧ na x = i ∧ length y = i ∧
    nb y = i ∧ length z = i ∧ nc z = i.
```

The functions `na`, `nb` and `nc` evaluate, respectively, to the number of symbols `a`, `b` and `c` in the argument (a list of terminal symbols).

The type `terminal` has three constructors `a`, `b` and `c`. Definition `anbncn` is a predicate on lists of terminals, expressing the property that the list is built by the same number of each of the symbols `a`, `b` and `c`, in this order. It embeds the property that all words of the language must satisfy, as described above.

Formalization of the statement

Similar to previous cases, the statement of this lemma says that the predicate `anbncn` can not represent a context-free language because it does not satisfy the predicate `cfl`:

```
Lemma not_cfl_anbncn: ∼ cfl anbncn.
```

Formalization of the proof

Differently from the two previous cases, the proof that *anbncn* is not context-free is much longer and more complex. It is accomplished by means of extensive case analysis on the components of the $uvwxy$ word, which result in various cases to be considered. To start with, we observe that, since vwx is part of $uvwxy$, and since $|vwx| \leq n$, then vwx must contain only symbols a followed by symbols b, or only symbols b followed by symbols c. Observe also that, according to the Pumping Lemma, $vx \neq \epsilon$, which means that v and x can not be empty simultaneously. An extensive analysis of the possibilities for v and x under these circunstances is presented below: [7]

$$
\begin{cases}
vwx \in a^*b^* \begin{cases}
(v \neq \epsilon) \wedge (x = \epsilon) \begin{cases}
|v|_a \neq 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (1) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (2) \\
|v|_a \neq 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (3)
\end{cases} \\
(v = \epsilon) \wedge (x \neq \epsilon) \begin{cases}
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a \neq 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (4) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (5) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a \neq 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (6)
\end{cases} \\
(v = \epsilon) \wedge (x = \epsilon) \begin{cases}
\text{can not occur, since } |vx| \geq 1 & (7)
\end{cases} \\
(v \neq \epsilon) \wedge (x \neq \epsilon) \begin{cases}
|v|_a \neq 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a \neq 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (8) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (9) \\
|v|_a \neq 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (10) \\
|v|_a \neq 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a \neq 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (11) \\
|v|_a \neq 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (12)
\end{cases}
\end{cases} \\
vwx \in b^*c^* \begin{cases}
(v \neq \epsilon) \wedge (x = \epsilon) \begin{cases}
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (13) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c \neq 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (14) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c \neq 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c = 0 & (15)
\end{cases} \\
(v = \epsilon) \wedge (x \neq \epsilon) \begin{cases}
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (16) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c \neq 0 & (17) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c \neq 0 & (18)
\end{cases} \\
(v = \epsilon) \wedge (x = \epsilon) \begin{cases}
\text{can not occur, since } |vx| \geq 1 & (19)
\end{cases} \\
(v \neq \epsilon) \wedge (x \neq \epsilon) \begin{cases}
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c = 0 & (20) \\
|v|_a = 0 \ \wedge \ |v|_b = 0 \ \wedge \ |v|_c \neq 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c \neq 0 & (21) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c \neq 0 & (22) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c = 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b \neq 0 \ \wedge \ |x|_c \neq 0 & (23) \\
|v|_a = 0 \ \wedge \ |v|_b \neq 0 \ \wedge \ |v|_c \neq 0 \ \wedge \ |x|_a = 0 \ \wedge \ |x|_b = 0 \ \wedge \ |x|_c \neq 0 & (24)
\end{cases}
\end{cases}
$$

For each of the two initial cases ($vwx \in a^*b^*$ and $vwx \in b^*c^*$), we consider other four:

- $(v \neq \epsilon) \wedge (x = \epsilon)$, or
- $(v = \epsilon) \wedge (x \neq \epsilon)$, or
- $(v = \epsilon) \wedge (x = \epsilon)$, or
- $(v \neq \epsilon) \wedge (x \neq \epsilon)$.

Condition $(v = \epsilon) \wedge (x = \epsilon)$, corresponding to cases (7) and (19) above, can not of course happen since we know that $vx \neq \epsilon$ and for this reason they are dropped out. Next, for each of the subcases, it is possible to make statements about the number of each kind of symbol in both v and x (whether they are zero or not zero).

Now observe that, in all 22 valid cases above, at most two numbers are not zero

[7] $|v|_a$ stands for `na v`, that is, the number of symbols a contained in the word v. Similarly, $|v|_b$ stands for `nb v` and $|v|_c$ stands for `nc v`.

while at least one number is zero. This means that, when v and x are pumped into $uvwxy$, it is possible to conclude that at most two kinds of symbols are pumped (those whose number is not zero) while at least one kind of symbol is not pumped (those whose number is zero). Thus, in all cases, pumping v and x changes the number of at most two different kinds of symbols, while the number of at least one kind of symbol remains unchanged. As a consequence, pumping produces words that do not belong to the language in all cases and the language *anbncn* is not context-free.

As an example, consider case (1) and take $i = 2$ (corresponding to $uvvwxxy$). The pumping of v and x increases the number of symbols a but surely does not change the number of symbols b c. Thus, $uvvwxxy$ necessarily has more symbols a than symbols b and c and can not belong to *anbncn*. Consider now case (12) and take $i = 2$ again. The pumping of v and x increases the number of symbols a and b but surely does not change the number of symbols c. Thus, $uvvwxxy$ necessarily has more symbols a than symbols c and can not belong to *anbncn*. The same happens if the number of symbols b is compared to the number of symbols c. The conclusions are similar for all 22 cases, and show that the new word $uvvwxxy$ can not belong to *anbncn* in any situation.

Fortunately, the cases listed above do not have to be considered individually in the formalization. Instead, we first prove that $vwx \in a^*b^*$ or $vwx \in b^*c^*$. Then, for each case, it is enough to prove that either $v \neq \epsilon$ or $x \neq \epsilon$. If $v \neq \epsilon$, this means that v contains at most two different kinds of symbols (with at least one symbol in it). The same happens if $x \neq \epsilon$. Thus, making $i = 2$ results in $uvvwxxy$ that contains different numbers of as, bs and cs.

One of the key issues in this formalization is to prove the rather intuitive result about the structure of vwx in comparison to $uvwxy$, when considered its maximum length n and that $uvxwy = a^nb^nc^n$: that vwx belongs to either a^*b^* or b^*c^*.

This is accomplished by means of three auxiliary lemmas, presented next. The first states that if s is a subword of $a^nb^nc^n$ and $|s| \leq n$, then s must be a subword of either a^nb^n or b^nc^n. This proof alone has more than 400 lines of extensive case analysis on the structure of x, y, z and s, and has the following statement:

```
Lemma sublist_or:
∀ x y z s: list terminal,
∀ i: nat,
length x = i →
na x = i →
length y = i →
nb y = i →
length z = i →
nc z = i →
sublist (x++y++z) s →
length s ≥ 1 →
length s ≤ i →
sublist (x++y) s ∨ sublist (y++z) s.
```

The proof of the second lemma, with approximately 180 lines, also has an intuitive statement: that if s is a subword of xy, where x contains only symbols a (zero or more) and y contains only symbols b (zero or more), then s must have only symbols a (zero or more) followed by symbols b (zero or more):

```
Lemma sublist_only_a_b:
∀ s1 s2 s: list terminal,
only_a s1 →
only_b s2 →
sublist (s1 ++s2) s →
only_a_b s.
```

The third lemma is similar to the second, and refers to subwords of yz, where y contains only symbols b and z contains only symbols c:

```
Lemma sublist_only_b_c:
∀ s1 s2 s: list terminal,
only_b s1 →
only_c s2 →
sublist (s1 ++s2) s →
only_b_c s.
```

The whole formalization of this section is approximately 2,200 lines of script long [8].

Comparison of the formal and text proofs

It is interesting to note that the size of the text proof of *anbncn* (15 lines) does not differ much from the size of the text proofs of previous cases (10 lines each). However, the size of the formal proof is bigger (more than 10 times bigger) than the corresponding ones, which means that the formal proof of *anbncn* is more than 100 times longer than the corresponding text proof. This is probably due to the nature of our third language, which introduced many combinatorial problems in the way to the final solution. These combinatorial problems have simple and intuitive statements which, however, led to many different cases to be considered, many lines of script to be written and many different lemmas with similar statements and proofs.

While most of the lemmas used in this proof are short and repetitive (which can surely be grouped and reduced in number by using proper parametrization), the three lemmas stated above comprise almost 800 lines corresponding to approximately 40% of the whole formalization. This means that most of the effort put in this formalization was used to prove results about the structure of a given substring in comparison to the structure of another string. Since the authors are not aware of any Coq libraries that could handle this in more efficient way, or simply handle it at all, we consider that the lack of such a library increases considerably the complexity of some formalizations (such as the present one). Also, that it might be the case that a new library developed with this specific purpose be benefitial for similar applications. The convenience of having such a library could be an opportunity and an important outcome of our present work.

Another reason that may help understand the size and complexity of the formal proof in comparison to the text proof is the fact that the later is, in a certain sense, oversimplified. Informal statements may hide a number of different cases that must be considered explicitly when developing a formal proof. This is the case, for example, for most of the text proof presented before for language *anbncn*. Some examples of it are:

[8] The scripts of this section can be found in file `pumping_anbncn.v`.

- "due to its length limitation, contains only one or two different kind of symbols";
- "if it contains only one kind of symbol, then ... are also built out of a single symbol and the pumping of ... will change the number of a single symbol, while the number of the other two remain unchanged";
- "if it contains two different kinds of symbols, then ...might contain one or two different kinds of symbols each";
- "if both contain only one kind of symbol, pumping will change the number of at most two symbols, while the third will remain unchanged";
- "if ... contain two different kinds of symbols, pumping will lead to a word where the order is not respected".

Thus, it is very possibile that if the text proof were more detailed and explanatory, then the difference in size and complexity between it and the corresponding formal proof would not be considerable.

4 Intersection

To prove that the class of the context-free languages is not closed under intersection, it is sufficient to present two context-free languages whose intersection is not a context-free language. For that purpose we use the language *anbncn*, previously proved not to be context-free in Section 3.3.

Text proof

Let $L_1 = \{a^n b^n c^m \,|\, n \geq 0 \wedge m \geq 0\}$ and $L_2 = \{a^m b^n c^n \,|\, n \geq 0 \wedge m \geq 0\}$. Note that $L_1 \cap L_2 = anbncn$, which is known as a non-context-free language. Thus, we only have to prove that both L_1 and L_2 are context-free. For that, take $G_1 = (\{S, X, Y, a, b, c\}, \{a, b, c\}, \{S \to XY, X \to aXb, X \to \epsilon, Y \to cY, Y \to \epsilon\})$ and $G_2 = (\{S, X, Y, a, b, c\}, \{a, b, c\}, \{S \to XY, X \to aX, X \to \epsilon, Y \to bYc, Y \to \epsilon\})$. Both G_1 and G_2 are context-free grammars that generate respectively, L_1 and L_2. Thus, L_1 and L_2 are context-free languages and the class of the context-free languages in not closed under intersection.

Formalization of the proof

We initially represent formally the language intersection operation. This is accomplished by means of an inductive definition which states that a word belongs to the intersection of two languages if and only if it belongs to both languages. This definition is similar to other definitions used previously to represent the operations of union, concatenation and Kleene star of context-free languages [9]:

```
Inductive l_int (l1 l2: lang terminal): lang terminal:=
| l_int_c: ∀ s: list terminal, l1 s → l2 s → l_int l1 l2 s.
```

We build the proof of this section on top of the previous result of Section 3.3, that the language *anbncn* is not context-free. Thus, all we have to formalize is:

(i) L_1 is a context-free language;

(ii) L_2 is a context-free language;

(iii) $L_1 \cap L_2 = anbncn$

There are many details involved in these simple statements, however.

The strategy used to prove (i) is to prove first that $L_1 = L_{11} \cdot L_{12}$ with $L_{11} = \{a^n b^n \,|\, n \geq 0\}$ and $L_{12} = \{c^m \,|\, m \geq 0\}$. That is, that L_1 can be expressed as the concatenation of two other simpler languages (L_{11} and L_{12}). The reason for that is to simplify the proof that L_1 is context-free. Then, we use use a previously formalized result [9], that the the class of the context-free languages is closed under concatenation, to show that L_1 is context-free.

Languages L_1, L_{11} and L_{12} are formalized in Coq as follows:

```
Definition anbncm: lang terminal:=
fun (s: list terminal) ⇒
∃ x y z: list terminal,
∃ i: nat,
s = x ++y ++z ∧ length x = i ∧ na x = i ∧
length y = i ∧ nb y = i ∧ length z = nc z.

Inductive anbn: list terminal → Prop:=
| anbn_1: anbn []
| anbn_2: ∀ w: list terminal, anbn w → anbn ([a]++w++[b]).

Inductive cm: list terminal → Prop:=
| cm_1: cm []
| cm_2: ∀ w: list terminal, cm w → cm ([c]++w).
```

The proof of step (i) starts by introducing context-free grammars that generate L_{11} and L_{12}, and then by proving that these grammars indeed generate the corresponding languages (lemmas `cfl_anbn` and `cfl_cm`). Finally, we prove that the concatenation of L_{11} and L_{12} results in L_1, which is accomplished by lemma `cat_eq_anbncm`. The last step of (i) is obtained through lemma `cfl_anbncm`, which is a simple application of a previous result (lemma `l_cat_is_cfl`, see [5]) and states that the language $anbncm$ is context-free as required.

For step (ii), that is, to prove that the language L_2 is context-free, we first note that $L_2 = L_{21} \cdot L_{22}$ with $L_{21} = \{a^m \,|\, m \geq 0\}$ and $L_{22} = \{b^n c^n \,|\, n \geq 0\}$. However, a different approach was adopted to obtain the proofs that L_{21} and L_{22} are context-free. While an approach similar to the one used to prove that L_1 is context-free could have been used (simply by changing as, bs and cs by, respectively, bs, cs and as in the direct proof scripts), we decided to formalize first the substitution and then use the result in languages that we previously proved to be context-free.

Thus, we prove that the class of the context-free languages is closed under alphabet substitution (as long as this substitution is represented by a bijection between the original and the new alphabets, which can be the same or not). We define the substitution of the alphabet in a language in Coq by:

```
Definition change_alphabet_in_language
(t1 t2: Type) (l1: lang t1) (f: t1 → t2): lang t2:=
fun (w2: list t2) ⇒
∃ w1: list t1,
l1 w1 ∧ w2 = map f w1.
```

and the main result is:

```
Lemma change_alphabet_in_language_is_cfl:
```

```
∀ t1 t2: Type,
∀ l1: lang t1,
∀ f: t1 → t2,
cfl l1 ∧ bijective f →
cfl (change_alphabet_in_language l1 f).
```

Essentially, this lemma states that every context-free language that has its alphabet changed remains a context-free language. So, we define the bijection that provides the necessary mappings:

```
Definition f (t1: terminal): terminal:=
match t1 with
| a ⇒ b
| b ⇒ c
| c ⇒ a
end.
```

This way, using the bijection `f`, we can easily proof that $\{b^n c^n \mid n \geq 0\}$ is context-free from the previous proof that $\{a^n b^n \mid n \geq 0\}$ is context-free, and the same for $\{c^m, \mid m \geq 0\}$ and $\{a^m, \mid m \geq 0\}$. Then, we proceed in a similar way as we did before in order to prove that the concatenation of $\{a^m, \mid m \geq 0\}$ and $\{b^n c^n \mid n \geq 0\}$ results in L_2, and also that L_2 is context-free.

For step (iii), we have first to prove that the intersection of L_1 and L_2 indeed results in *anbncn*, which is obtained by means of lemma `int_eq_anbncn`. Finally, we proceed to our main theorem, the one that states that context-free languages are not closed under intersection:

```
Theorem cfl_not_closed_intersection:
~∀ l1 l2: lang terminal,
cfl l1 ∧ cfl l2 → cfl (l_int l1 l2).
```

With the previous results (including lemma `not_cfl_anbncn` from Section 3.3), the proof is straigtfoward and only a few lines long. It is obtained by contradiction, since we have both a prove that *anbncn* is context-free (from the hypotheses) and that *anbncn* is not context-free (from Section 3.3) [9] .

Comparison of the formal and text proofs

While the text proof is very straightforward and only a few lines long, the formal proof is longer and plenty of details that range from the proof that a simple language is context-free to the proof that a given language can result from the combination of other simpler languages by means of the use of the appropriate operations. The proofs are generally not short and use inductive arguments. The proof scripts for this part of the formalization are approximately 1,500 lines long (for `intersection.v`) and 500 lines long (for `bijection.v`).

5 Conclusions

Languages *square* and *prime* were easily proven not to be context-free, by straighforward application of the Pumping Lemma. On the other hand, language *anbncn* was harder to get to the same result.

[9] The scripts of this section can be found in files `intersection.v` and `bijection.v`.

Besides these three languages, a fourth language was considered as well. The tentative proof that the language *anbnanbn*, defined as $\{a^n b^n a^n b^n \mid n \geq 0\}$ is not context-free was, however, much harder and bigger than that for the language *anbncn* (at least using our current strategy, that relies on the application of the Pumping Lemma).

In a first proof sketch, we could prove that $(vwx \in a^*b^*)$ or $(vwx \in b^*a^*)$. Next, for the first case, we could prove that $(v \in a^*b^*) \wedge (x \in b^*)$ or $(v \in a^*) \wedge (x \in a^*b^*)$. Similarly, for the second case, we could prove that $(v \in b^*a^*) \wedge (x \in a^*)$ or $(v \in b^*) \wedge (x \in b^*a^*)$. For the rest, we could consider whether or not v and x are empty and make conjectures about v and x when they are made of two kinds symbols by splitting them into two different substrings, each of them made of a single kind of symbol. This sketch splits in 28 cases which, in turn, can be handled by three different proof strategies.

A second proof sketch could be to proceed as outlined in the proof of *anbncn*: first, to prove that $(vwx \in a^*b^*)$ or $(vwx \in b^*a^*)$. Next, to prove that at least one of v and x must be non-empty. Finally, to make conjectures about the number of as and bs in each of v and x (whether zero or not zero). This sketch leads to 22 cases that can be proved by three different strategies. For space reasons, the tables that summarize the cases in these two skectches are not included.

We tried to formalize a proof for *anbnanbn* based on the first attempt described above, but the results were not encouraging. The proof script grew very fast and had more than 4,000 lines when it was interrupted, still incomplete. The size and complexity of the script became so big, with so many cases to be considered and lemmas yet to be proved, that we decided not to go on for the time being. We did not try to formalize the second sketch, which has a smaller number of cases to be considered (22 *versus* 28).

For the sake of completeness, we should also make some considerations about a fifth classical non-context-free language, which we shall call *ww* and which is defined as the set of words ww over some alphabet (for example, $\{a, b\}$), such that $w \in \{a, b\}^*$. To prove that *ww* is not context-free it is sufficient to take word $a^n b^n a^n b^n$ and show that pumping over this word generates words that do not belong to *ww*. For that purpose, we could use a strategy similar to the one used for language *anbnanbn*, with the same number of cases to be considered. Thus, the complexixity of the formalization for *ww* should be similar to that of the formalization of *anbnanbn*. We have not tried to formalize *ww*.

It is worth to note that, for all five languages considered so far (*square*, *prime*, *anbncn*, *anbnanbn* and *ww*), their definition and the text proofs that they are not context-free are concise and elegant (the text proofs for *anbnanbn* and *ww*, although not presented here, are also straightforward). For languages *square* and *prime*, it is also possible to state that the corresponding formal proofs are straighforward to build and are easy to read. For languages *anbncn*, *anbnanbn* and *ww*, however, the situation is completely different. So, what explains the difference between these two groups of languages? As mentioned in the case of language *anbncn*, explanations may come both from the classical text proofs and the need to reason about the

structure of substrings, a feature that (as far as the authors are aware of) is not natively available in Coq nor in any of its standard libraries.

First of all, the size and complexity of the formal proofs for *anbncn*, *anbnanbn* and *ww* involve combinatory aspects needed in intermediate reasoning steps required to build the final proof, and this led to large and fully detailed scripts. If this is a consequence of some characteristic (or lack of feature) of Coq or its libraries, this is a matter still to be investigated. Indeed, it sounds very much like the libraries of Coq are not so well developed in respect to the combinatorics of strings, as it is in other areas, such as sets, lists and algebra to mention a few. Thus, this might suggest that some effort is required in this area in order to ease the development of proofs such as the ones that we have discussed in Section 3. What is needed are lemmas that prove that strings with some property that are substrings of another string with another property can get proper descriptions. An example is a lemma that proves that any string with maximum length n that is a substring of string $a^n b^n c^n$ belongs to $a^*b^*|b^*c^*$. Similarly for any string with the same size n that is a substring of $a^n b^n a^n b^n$, in this case being described by $a^*b^*|b^*a^*$.

Note, however, that this need was not present in the formal proofs developed for *square* and *prime*, which involve basically the length of the generated strings.

Second, we argue that perhaps the classical text proofs of *anbncn*, *anbnanbn* and *ww* are not as clear and understandable as they seem or claim to be. Indeed, the text proof that *anbncn* is not context-free is only 10 lines long, but hides in it 24 different cases. For the language *anbnanbn* the situation is even worse: 28 (or 22) different cases are hidden in the text proof, which is about 10 lines long as well. Is everyone capable of seeing so many cases in so few lines of english text? Aren't these text proofs oversimplified and relying too much on the common sense and perceptions of the reader (student, teacher, professional, whatever)?

The proof that context-free languages are not closed under intersection is a nice result that derives almost directly from one of the previous results. It demanded, however, some effort in order to adequately cope with the many details involved in proofs that usually do not take too much time or effort from the reader of a text proof (such as to convince himself that a given grammar generates a given language). Once again, we see here the formalization making all the details of a text proof explicit. This result, along with the other results previously obtained about the formalization of closure under union, concatenation and Kleene star [9], represent a good set of formalized results of closure properties for the class of the context-free languages.

The results obtained so far are interesting in their own. They correspond to important applications of an important property of context-free languages which can be used for different purposes. These include to guarantee the correctness of text proofs that may hide too much information, and thus escape from the full understanding of the reader, and to better understand the nature of the corresponding proofs. Also, these formal proofs are original and valuable material for courses in formal language theory and/or formal reasoning using interactive proof assistants.

Future works include improvements in the source code of this formalization via

simplifications and generalizations. As an example, files `pumping_anbncn.v` and `intersection.v` contain different lemmas and functions that can be combined into a single lemma or function. One case are functions `na`, `nb` and `nc` that count the occurrences of, respectively, symbols `a`, `b` and `c` in a list and can be grouped into a single function that counts the occurrences of a parametrized symbol. Another case are lemmas `cfl_ambm` and `cfl_bmcm` that can be merged into a single lemma where the terminal symbols are parameters of the lemma. Besides that, we plan to extend our work on the formalization of context-free language theory with the formalization of the closure of the class of the context-free languages under intersection with regular languages, of the non-closure of the same class under complementation and of Ogden's Lemma, a stronger version of the Pumping Lemma for Context-Free Languages [3].

References

[1] Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.

[2] The Coq Development Team. *The Coq Reference Manual, version 8.8.0*, 2018. Available electronically at https://coq.inria.fr/distrib/current/refman/.

[3] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages and Computation.* Addison-Wesley Publishing Co., Inc., 1979.

[4] Inria – Microsoft Research Joint Centre. SSReflect and the Mathematical Components library, 2018. http://math-comp.github.io/math-comp/, accessed August 14th, 2018.

[5] Marcus Vinícius Midena Ramos. *Formalization of Context-Free Language Theory.* PhD thesis, Centro de Informática - UFPE, 2016. http://www.marcusramos.com.br/univasf/tese.pdf, 2016.

[6] Marcus Vinícius Midena Ramos. Source files, 2016. https://github.com/mvmramos/intersection, 2018.

[7] Marcus Vinícius Midena Ramos, José Carlos Bacelar Almeida, Nelma Moreira, and Ruy José Guerra Barretto de Queiroz. Formalization of the Pumping Lemma for Context-Free Languages. *Journal of Formalized Reasoning*, 9(2):53–68, 2016. https://jfr.unibo.it/article/view/5595.

[8] Marcus Vinícius Midena Ramos, José Carlos Bacelar Almeida, Nelma Moreira, and Ruy José Guerra Barretto de Queiroz. On the Formalization of Some Results of Context-Free Language Theory. In Jouko Väänänen, Åsa Hirvonen, and Ruy de Queiroz, editors, *23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016, Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2016. http://dx.doi.org/10.1007/978-3-662-52921-8_21.

[9] Marcus Vinícius Midena Ramos and Ruy José Guerra Barretto de Queiroz. Formalization of closure properties for context-free grammars. *CoRR*, abs/1506.03428, 2014. http://arxiv.org/abs/1506.03428.

[10] Marcus
Vinícius Midena Ramos and Ruy José Guerra Barretto de Queiroz. Formalization of simplification for context-free grammars. *CoRR*, abs/1509.02032, 2015. http://arxiv.org/abs/1509.02032.

[11] Marcus Vinícius Midena Ramos, João José Neto, and Italo Santiago Vega. *Linguagens Formais: Teoria Modelagem e Implementação.* Bookman, 2009.

[12] Thomas A. Sudkamp. *Languages and Machines.* Addison-Wesley, 3rd edition, 2006.