Formalization of the Pumping Lemma for Context-Free Languages

M. V. M. RAMOS
Colegiado de Engenharia de Computação, UNIVASF, Brasil
R. J. G. B. DE QUEIROZ
Centro de Informática, UFPE, Brasil
N. MOREIRA
Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto, Portugal
J. C. B. ALMEIDA
HASLab - INESC TEC, Universidade do Minho, Portugal

Context-free languages are highly important in computer language processing technology as well as in formal language theory. The Pumping Lemma is a property that is valid for all context-free languages, and is used to show the existence of non context-free languages. This paper presents a formalization, using the Coq proof assistant, of the Pumping Lemma for context-free languages.

1. INTRODUCTION

The formalization of context-free language (CFL) theory is key to certification of compilers and programs, as well as to development of new languages and tools for certified programming.

Context-free language theory formalization is a relatively new area of research, with some results already obtained with a diversity of proof assistants, including Coq, HOL4 and Agda. Most of the effort started in 2010 and has been devoted to the certification and validation of parser generators. Examples of this are the works of Koprowski and Binsztok (using Coq, [KB10]), Ridge (using HOL4, [Rid11]), Jourdan, Pottier and Leroy (using Coq, [JPL12]) and, more recently, Firsov and Uustalu (in Agda, [FU14]).

On the more theoretical side, on which the present work should be considered, Norrish and Barthwal published on general context-free language theory formalization using the using HOL4 proof assistant [Bar10, BN10a, BN10b, BN14], including the Pumping Lemma, normal forms for grammars, pushdown automata and closure properties. Recently, Firsov and Uustalu proved the existence of a Chomsky Normal Form grammar for every context-free grammar, using the Agda proof assistant [FU15]. A special case of the Pumping Lemma, namely the Pumping Lemma for regular languages, is included in a comprehensive work on the formalization of regular languages [DKS] using SSRreflect, an extension of Coq.

We aim at formalizing a substantial part of context-free language theory in the Coq proof assistant, making it possible to reason about it in a fully checked environment, with all the related advantages. Initially, however, the focus has been restricted to context-free grammars and associated results. Pushdown automata and their relation to context-free grammars are not considered at this point.

The work, that started with the formalization of closure properties for context-free grammars [RdQ14], evolved later into the formalization of context-free grammar simplification [RdQ15] and then into the Chomsky normalization of context-free grammars. Formalization of simplification enabled Chomsky normalization, which in turn enabled the present formal proof of the Pumping Lemma. The whole work is described in detail in [Rama].

In order to follow this paper, the reader is required to have basic knowledge of Coq and of context-free language theory. For the beginner, the recommended starting point for Coq is the book by Bertot and Castéran [BC04]. Background on context-free language theory can be found in [Sud06], [HU79] or [RNV09], among others. Previous results, which were used in the formalization of the Pumping Lemma, will not be discussed here and can be retrieved from the above references.

The statement and applications of the Pumping Lemma for CFLs (or Pumping Lemma for short) are presented in Section 2. A typical informal proof, which served as the basis for the present formalization, is described in Section 3. Section 4 introduces results obtained previously and which are required for this work. The formalization is then described in Section 5, where the definitions and auxiliary lemmas used are discussed in some detail, as well as the Pumping Lemma itself. A brief comparison with the work of Barthwal is included in Section 6. Final conclusions are presented in Section 7.

As far as the authors are aware of, this is the second ever formalization of the Pumping Lemma for context-free languages (the first, in HOL4, is due to Barthwal, see [Bar10]) and the first ever with the Coq proof assistant. All the definitions and proof scripts discussed in this paper were written in plain Coq and are available for download at [Ramb].

2. STATEMENT AND APPLICATION

A language is a set of sentences defined over an alphabet. A context-free grammar is a grammar whose rules have the form $X \rightarrow \beta$, where X is a non-terminal symbol and β is a sequence (possibly empty) of terminal and non-terminal symbols. A context-free language is a language that is generated by some context-free grammar. The Pumping Lemma for CFLs was stated and proved for the first time by Bar-Hillel, Perles and Shamir in 1961 [BHPS61]. In what follows, it will be referred simply as "Pumping Lemma".

The Pumping Lemma does not characterize the CFLs, however, since it is also verified by some non CFLs ([HU79]). Besides that, the authors are not aware of any independent characterization of the class of languages that satisfy it. The Pumping Lemma states that, for every context-free language and for every sentence of such a language that has a certain minimum length, it is possible to obtain an infinite number of new sentences that must also belong to the language. This minimum length depends only on the language defined. In other words (let $\mathcal{L}$ be defined over alphabet Σ):

$$\forall \mathcal{L}, (\text{cfl } \mathcal{L}) \rightarrow \exists n \mid \forall \alpha, (\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n) \rightarrow$$

$$\exists u, v, w, x, y \in \Sigma^* \mid (\alpha = uvwxy) \wedge (|vx| \geq 1) \wedge (|vwx| \leq n) \wedge$$

$$\forall i, uv^iwx^iy \in \mathcal{L}$$

A typical use of the Pumping Lemma is to show that a certain language is not context-free by using the contrapositive of the statement of the lemma. The 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.

As an example, consider the language $\mathcal{L} = \{a^ib^ic^i \mid i \geq 1\}$. This language is defined over the alphabet $\{a, b, c\}$ and includes sentences such as $abc, aabbcc$ and $aaabbbccc$.

Should $\mathcal{L}$ be context-free, then the Pumping Lemma should hold for it. Consider n to be the constant of the Pumping Lemma and let's choose the sentence $\alpha = a^nb^nc^n$. Clearly, $\alpha \in \mathcal{L}$ and $|\alpha| = 3n \geq n$. Thus, exists u, v, x, w, y such that $\alpha = uvwxy, |vx| \geq 1, |vwx| \leq n$ and $\forall i \geq 0, uv^iwx^iy \in \mathcal{L}$.

However, it is easy to observe that, due to its length limitation, the sentence vwx should contain only one or two different symbols (namely, vwx should belong to either a^*, b^*, c^*, a^*b^* or b^*c^*). This implies that the repetition of v and x in uv^iwx^iy should increase (or decrease) the number of at most two different symbols while keeping the number of the other symbol(s) unchanged. As a result, the new sentence can not belong to the language and this proves that the initial hypothesis is not true. Thus, $\mathcal{L}$ is not a context-free language.

3. INFORMAL PROOF

In short, the Pumping Lemma derives from the fact that the number of non-terminal symbols in any context-free grammar G that generates $\mathcal{L}$ is finite. The classical proof considers that G is in the Chomsky Normal Form (CNF, a form in which the rules of the grammar have at most two symbols in the right-hand side), which means that derivation trees have the simpler form of binary trees. Then, if the sentence has a certain minimum length, the derivation tree for this sentence should have two or more instances of the same non-terminal symbol in some path that starts in the root of the tree and has maximal length. Finally, the context-free character of G guarantees that the subtrees related to these duplicated non-terminal symbols can be cut and pasted in such a way that an infinite number of new derivation trees are obtained, each of which is related to a new sentence of the language.

The proof comprises the following steps (more details can be found in [Sud06], [HU79] or [RNV09]):

(1) Since $\mathcal{L}$ is declared to be a context-free language (predicate `cfl`), then there exists a context-free grammar G such that $L(G) = \mathcal{L}$;
(2) Obtain G' such that G' is in Chomsky Normal Form and $L(G') = L(G)$;
(3) Take n as 2^k, where k is the number of non-terminal symbols in G';
(4) Consider an arbitrary sentence α such that $\alpha \in \mathcal{L}$ and $|\alpha| \geq n$;
(5) Obtain a derivation tree t that represents the derivation of α in G';
(6) Take a path that starts in the root of t and whose length is the height of t plus 1 (maximum length);
(7) Then, the height of t should be greater than or equal to $k + 1$;

(8) This means that the selected path has at least $k+2$ symbols, being at least $k+1$ non-terminals and one (the last) terminal symbol;

(9) Since G' has only k non-terminal symbols, this means that this path has at least one non-terminal symbol that appears at least two times in it;

(10) Call the duplicated symbol d and the corresponding subtrees t_1 and t_2 (note that t_2 is a subtree of t_1 and t_1 is a subtree of t);

(11) It is then possible to prove that the height of t_1 is greater than or equal to 2, and less than or equal to 2^k;

(12) Also, that the height of t_2 is greater than or equal to 1 and less than or equal to 2^{k-1};

(13) This implies that the frontier of t can be split into five parts: u, v, w, x, y, where w is the frontier of t_2 and vwx is the frontier of t_1;

(14) As a consequence of the heights of the corresponding subtrees, it can be shown that $|vx| \geq 1$ and $|vwx| \leq n$;

(15) If t_1 is removed from t, and t_2 is inserted in its place, then we have a new tree t^0 that represents the derivation of string $uv^0wx^0y = uwy$;

(16) If, instead, t_1 is inserted in the place where t_2 lies originally, then we have a tree t^2 that represents the derivation of string uv^2wx^2y;

(17) Repetition of the previous step generates all trees t^i that represent the derivation of the string uv^iwx^iy, $\forall i \geq 2$.

This proof is used in 6 out of 13 sources researched. In other 5 sources, the proof is based on generic derivation trees (not necessarily binary), where the grammar is not required to be in CNF. In such cases, it is enough to take $n = m^k$, where m is the length of the longest right-hand side among all rules of the grammar, and k is the number of non-terminal symbols in the grammar. The two other cases derive the proof of the Pumping Lemma, respectively, from the proof of the stronger Ogden's Lemma and from pushdown automata instead of context-free grammars. For the sources and more information on these informal proofs, see [Rama].

Since 11 out of 13 proofs considered use grammars and derivation trees and, of these, 6 use CNF grammars and binary trees (including the authors of the original proof), this strategy was considered as the best choice for the present work. Besides that, binary trees can be easily represented in Coq as simple inductive types, where generic trees require mutually inductive types, which increases the complexity of related proofs. Thus, for all these reasons we have adopted the proof strategy that uses CNF grammars and binary trees.

4. BACKGROUND

Formalization in the Coq proof assistant requires the formalization of the existence of CNF for context-free grammars. This, in turn, demands the formalization of context-free grammar simplification (useless and inaccessible symbol elimination and unit and empty rules elimination). For details on how these have been accomplished, please refer to [RdQ15] and [Rama].

The Chomsky Normal Form (CNF) theorem asserts that

$$\forall G = (V, \Sigma, P, S),\ \exists G' = (V', \Sigma, P', S') \mid L(G) = L(G') \wedge$$

$$\forall (\alpha \rightarrow_{G'} \beta) \in P', (\beta \in \Sigma) \vee (\beta \in N \cdot N)$$

where $N = V \backslash \Sigma$ and $N' = V' \backslash \Sigma$. That is, every context-free grammar can be converted to an equivalent one whose rules have only one terminal symbol or two non-terminal symbols in the right-hand side. Naturally, this is valid only if G does not generate the empty string. If this is the case, then the grammar that has this format, plus a single rule $S' \rightarrow_{G'} \epsilon$, is also considered to be in the Chomsky Normal Form, and generates the original language, including the empty string. It can also be assured that in either case the start symbol of G' does not appear on the right-hand side of any rule of G'.

The CNF theorem has been stated in our formalization in Coq as:

```
Theorem g_cnf:
∀ g: cfg non_terminal terminal,
   (produces_empty g ∨ ∼produces_empty g) ∧
   (produces_non_empty g ∨ ∼produces_non_empty g) →
   ∃ g': cfg non_terminal' terminal,
      g_equiv g' g ∧ (is_cnf g' ∨ is_cnf_with_empty_rule g').
```

Context-free grammars are represented by record `cfg` in a way that resembles the classical definition:

```
Notation sf := (list (non_terminal + terminal)).
Record cfg: Type:= {
   start_symbol: non_terminal;
   rules: non_terminal → sf → Prop;
   t_eqdec: ∀ (x y:terminal), {x=y}+{x≠ y};
   nt_eqdec: ∀ (x y:non_terminal), {x=y}+{x≠ y};
   rules_finite:
      ∃ n: nat,
      ∃ ntl: nlist,
      ∃ tl: tlist,
         rules_finite_def start_symbol rules n ntl tl
}.
```

This definition is parametrized by the types of terminal and non-terminal symbols, both required to have a decidable equality (fields `t_eqdec` and `nt_eqdec` respectively). Definition `sf` represents the type of sentential forms that can be generated with the corresponding grammar.

The set of grammar rules is captured by the corresponding characteristic predicate `rules`. The predicate `rules_finite_def` ensures that the set of rules of the grammar is finite by proving that the length of right-hand side of every rule is equal to or less than a given value, and also that both left and right-hand side of the rules are built from finite sets of, respectively, non-terminal and terminal symbols (represented here by lists). This is a direct consequence of the fact that standard Coq types are used to represent the sets of terminal and non-terminal symbols in the formalization. The same result, however, could have been obtained if lists were used instead of `Prop` to represent the rules of the grammar, as discussed before. In this case, the predicate `rules_finite_def` would not be necessary. As it is, it is necessary to explicitly prove that this predicate holds for every grammar that is used as an argument of a theorem. Besides that, the formalization itself ensures that the predicate holds for new grammars constructed in it (for example, when a CNF grammar is constructed from another grammar).

The decision of representing grammar rules as propositions (that is, inhabitants of the sort `Prop`) has the consequence that it prevents direct extraction of executable code from the formalization. It does not directly affect the formalization of the Pumping Lemma, however it prevents obtaining certified algorithms for other results included in this work, such as grammar union, concatenation and closure, as well as grammar simplification and Chomsky Normal Form.

To achieve this, an alternative would be to represent the set P of rules as a member of type `list (non_terminal * sf)` instead (that is, an inhabitant of the sort `Set`). In this case, P would be represented as a list of rules, and each rule as a pair where the first element is a non-terminal symbol and the second is a list of terminal and non-terminal symbols. Then, the inductive definitions used to construct new grammars (for previous results of this work) would have to be replaced by functions that construct the new grammars from the previous one. Finally, the desired lemmas and theorems would have to be proved on top of these functions. After that, the extraction of certified programs from these functions could be obtained directly using Coq's appropriate facilities.

All this, however, would have changed the declarative approach of the present work into an algorithmic one, by creating functions that generate new grammars with the desired properties. On the other hand, the purely logical approach adopted was considered more appealing, since it maps directly from the textbooks, and thus was selected as the choice for the present formalization.

The predicate `produces g s` asserts that context-free grammar `g` produces the list of terminals `s` as a sentence of the language. It is based on the more fundamental notion of *derivation*, present in the whole formalization and defined as:

```
Inductive derives (g: cfg): sf → sf → Prop :=
| derives_refl:
   ∀ s: sf,
     derives g s s
| derives_step:
   ∀ s1 s2 s3: sf,
   ∀ left: non_terminal,
   ∀ right: sf,
     derives g s1 (s2 ++inl left :: s3) →
     rules g left right →
     derives g s1 (s2 ++right ++s3).
```

The predicates used in theorem `g_cnf` assert that:

—a grammar produces the empty string:

```
Definition produces_empty
(g: cfg non_terminal terminal): Prop:=
produces g [].
```

—a grammar produces a non-empty string:

```
Definition produces_non_empty
(g: cfg non_terminal terminal): Prop:=
∃ s: sentence, produces g s ∧ s ≠ [].
```

—two grammars are equivalent:

```
Definition g_equiv
(non_terminal non_terminal' terminal: Type)
(g1: cfg non_terminal terminal)
(g2: cfg non_terminal' terminal): Prop:=
∀ s: sentence,
   produces g1 s ↔ produces g2 s.
```

—a rule is in the Chomsky Normal Form:

```
Definition is_cnf_rule
(left: non_terminal) (right: sf): Prop:=
(∃ s1 s2: non_terminal, right = [inl s1; inl s2]) ∨
(∃ t: terminal, right = [inr t]).
```

—a grammar is in the Chomsky Normal Form:

```
Definition is_cnf
(g: cfg non_terminal terminal): Prop:=
∀ left: non_terminal,
∀ right: sf,
   rules g left right → is_cnf_rule left right.
```

—a grammar is in the Chomsky Normal Form and has a single empty rule with the start symbol in the left-hand side:

```
Definition is_cnf_with_empty_rule
(g: cfg non_terminal terminal): Prop:=
∀ left: non_terminal,
∀ right: sf,
   rules g left right →
   (left = (start_symbol g) ∧ right = []) ∨ is_cnf_rule left right.
```

The predicates `produces_empty` and `produces_non_empty` used in Theorem `g_cnf` describe grammars that, respectively, generate the empty string and generate a non-empty string. Since these questions are decidable for every context-free grammar, the hypotheses that claim their decidability would not need to be included in the statement. However, the proof of these questions (among other decidable questions for context-free grammars) were not considered in the work developed so far, thus the related hypotheses have to be explicitly included. This is also a consequence of the fact that Coq uses a constructive logic, for which the law of the excluded middle (LEM) can not be generically accepted as being true without the corresponding proof.

5. FORMALIZATION

The formalization follows closely the steps described in Section 3. The (classical version of the) Pumping Lemma has been stated in Coq as follows (please refer and compare to the statement of Section 2):

```
Lemma pumping_lemma:
∀ l: lang terminal,
(contains_empty l ∨ ~contains_empty l) ∧
(contains_non_empty l ∨ ~contains_non_empty l) →
cfl l →
∃ n: nat,
  ∀ s: sentence,
    l s →
    length s ≥ n →
    ∃ u v w x y: sentence,
      s = u ++v ++w ++x ++y ∧
      length (v ++x) ≥ 1 ∧
      length (v ++w ++x) ≤ n ∧
      ∀ i: nat, l (u ++(iter v i) ++w ++(iter x i) ++y).
```

A language is defined as a function 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:

```
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.
```

Predicates `contains_empty` and `contains_non_empty` are language counterparts of the previously presented grammar predicates, respectively `produces_empty` and `produces_non_empty`. The same observations made in Section 4 for the presence of hypotheses concerning the decidability of the former questions apply here for the latter questions, and justify their inclusion in the statement of the Pumping Lemma.

Application `iter l i` on a list `l` and a natural number `i` yields a list $\mathtt{l}^{\mathtt{i}}$, that is, a list that corresponds to the concatenation of `l` to itself `i` times (`iter l 0` yelds the empty list for any argument).

Initially, the type `btree` (for binary trees) has been defined with the objective of representing derivation trees for strings generated by context-free grammars in Chomsky Normal Form:

```
Inductive btree (non_terminal terminal: Type): Type:=
| bnode_1: non_terminal → terminal → btree
| bnode_2: non_terminal → btree → btree → btree.
```

The constructors of `btree` relate to the two possible forms that the rules of a CNF grammar can assume (namely with one terminal symbol or two non-terminal symbols in the right-hand side).

The proof of the Pumping Lemma starts by finding a grammar G that generates the input language L (this is a direct consequence of the predicate `is_cfl` and corresponds to step 1 of Section 3). Next, we obtain a CNF grammar G' that is equivalent to G (step 2), using previous results. Then, G is substituted for G' and the value for n is defined as 2^k (step 3) where k is the length of the list of non-terminals of G' (which in turn is obtained from the predicate `rules_finite`). An arbitrary sentence α of $L(G')$ that satisfies the required minimum length n is considered (step 4).

Lemma `derives_g_cnf_equiv_btree` is then applied in order to obtain a `btree` t that represents the derivation of α in G' (step 5). This lemma is general enough in order to accept that the input grammar might either be a CNF grammar, or a CNF grammar with an empty rule. If this is the case, then we have to ensure that $\alpha \neq \epsilon$, which is true since by assumption $|\alpha| \geq 2^k$. The proof of `derives_g_cnf_equiv_btree` is reasonably long and uses induction on the number of derivation steps in G' in order to generate α:

```
Lemma derives_g_cnf_equiv_btree:
∀ g: cfg non_terminal' terminal,
∀ n: non_terminal',
∀ s: sentence,
  s ≠ [] →
  (is_cnf g ∨ is_cnf_with_empty_rule g) →
  start_symbol_not_in_rhs g →
  derives g [inl n] (map term_lift' s) →
  ∃ t: btree non_terminal' terminal,
    btree_cnf g t ∧
    broot t = n ∧
    bfrontier t = s.
```

The next step is to obtain a path (a sequence of non-terminal symbols ended by a terminal symbol) that has maximum length, that is, whose length is equal to the height of t plus 1 (steps 6 and 7). This is accomplished by means of the definition `bpath` (for binary path) and the lemma `btree_ex_bpath`:

```
Inductive bpath (bt: btree): sf → Prop:=
| bp_1:
  ∀ n: non_terminal,
  ∀ t: terminal,
    bt = (bnode_1 n t) → bpath bt [inl n; inr t]
| bp_l:
  ∀ n: non_terminal,
  ∀ bt1 bt2: btree,
  ∀ p1: sf,
    bt = bnode_2 n bt1 bt2 → bpath bt1 p1 → bpath bt ((inl n) :: p1)
| bp_r:
  ∀ n: non_terminal,
  ∀ bt1 bt2: btree,
  ∀ p2: sf,
    bt = bnode_2 n bt1 bt2 → bpath bt2 p2 → bpath bt ((inl n) :: p2).
```

```
Lemma btree_ex_bpath:
∀ bt: btree,
∀ ntl: list non_terminal,
   bheight bt ≥ length ntl + 1 →
   bnts bt ntl →
   ∃ z: sf,
      bpath bt z ∧
      length z = bheight bt + 1 ∧
      ∃ u r: sf,
      ∃ t: terminal,
         z = u ++r ++[inr t] ∧
         length u ≥ 0 ∧
         length r = length ntl + 1 ∧
         (∀  s: symbol, In s (u ++r) → In s (map inl ntl)).
```

The length of this path (which is $\geq k+2$) allows one to infer that it must contain at least one non-terminal symbol that appears at least twice in it (steps 8, 9 and 10). This result comes from the application of the lemma pigeon which represents a list version of the well-known pigeonhole principle (for any type A with decidable equality):

```
Lemma pigeon:
∀ A: Type,
∀ x y: list A,
(∀  e: A, In e x → In e y) →
   length x = length y + 1→
   ∃ d: A,
   ∃ x1 x2 x3: list A,
      x = x1 ++[d] ++x2 ++[d] ++x3.
```

Since a path is not unique in a tree, it is necessary to use another representation that can describe this path uniquely, which is done by the predicate bcode (for binary code) and the lemma bpath_ex_bcode:

```
Inductive bcode (bt: btree): list bool → Prop:=
| bcode_0:
   ∀ n: non_terminal,
   ∀ t: terminal,
      bt = (bnode_1 n t) → bcode bt []
| bcode_1:
   ∀ n: non_terminal,
   ∀ bt1 bt2: btree,
   ∀ c1: list bool,
      bt = bnode_2 n bt1 bt2 → bcode bt1 c1 → bcode bt (false :: c1)
| bcode_2:
   ∀ n: non_terminal,
   ∀ bt1 bt2: btree,
   ∀ c2: list bool,
      bt = bnode_2 n bt1 bt2 → bcode bt2 c2 → bcode bt (true :: c2).
```

The predicate bcode uses a sequence of Boolean values (false, true) to respectively select the left or right subtrees in a tree, and thus define a path in it.

```
Lemma bpath_ex_bcode:
∀ t: btree,
∀ p: sf,
   bpath t p →
   ∃ c: list bool,
      bcode t c ∧
      bpath_bcode t p c.
```

The predicate `bpath_bcode` merely ensures that `bcode c` is valid for `bpath p` in tree `t`. Once the path has been identified with a repeated non-terminal symbol, and a corresponding `bcode` has been assigned to it, lemma `bcode_split` is applied twice in order to obtain the two subtrees t_1 and t_2 that are associated respectively to the first and second repeated non-terminals of t. This lemma, which is key in the formalization, has a statement with a number of hypotheses and conclusions which provide useful information on the newly identified subtree. Among them, its height and its frontier (the latter as the result of a call to function `btree_decompose`, presented next):

```
Lemma bcode_split:
∀ t: btree,
∀ p1 p2: sf,
∀ c: list bool,
   bpath_bcode t (p1 ++p2) c →
   length p1 > 0 →
   length p2 > 1 →
   bheight t = length p1 + length p2 − 1 →
   ∃ c1 c2: list bool,
      c = c1 ++c2 ∧
      length c1 = length p1 ∧
      ∃ t2: btree,
      ∃ x y: sentence,
         bpath_bcode t2 p2 c2 ∧
         btree_decompose t c1 = Some (x, t2, y) ∧
         bheight t2 = length p2 − 1.
```

Function `btree_decompose` takes as arguments a `btree` and a sequence of Boolean values, and returns a triple consisting of the subtree of the first argument located at the corresponding position specified by the second argument, and the two sentences to the left and right of it. It is used to enable reasoning on the frontiers of the subtrees obtained before:

```
Fixpoint btree_decompose (bt: btree) (c: list bool):
option (sentence ∗ btree ∗ sentence):=
match bt, c with
| bnode_1 n t, [] ⇒
          Some ([],  bt,  [])
| bnode_1 n t, _ ⇒
          None
| bnode_2 n bt1 bt2, [] ⇒
          Some ([],  bt,  [])
| bnode_2 n bt1 bt2, false :: c ⇒
          match btree_decompose bt1 c with
          | None ⇒ None
          | Some (l, bt, r) ⇒ Some (l, bt, r ++bfrontier bt2)
          end
| bnode_2 n bt1 bt2, true :: c ⇒
          match btree_decompose bt2 c with
          | None ⇒ None
          | Some (l, bt, r) ⇒ Some (bfrontier bt1 ++l, bt, r)
          end
end.
```

From this information it is then possible to extract most of the results needed to prove the goal (steps 11, 12, 13 and 14), except for the pumping condition. This is obtained by an auxiliary lemma `pumping_aux`, which takes as hypothesis the fact that a tree t_1 (with frontier vwx) has a subtree t_2 (with frontier w), both with the same roots, and asserts the existence of an infinite number of new trees obtained by repeated substitution of t_2 by t_1 or simply t_1 by t_2, with respectively frontiers $v^i w x^i, i \geq 1$ and w, or simply $v^i w x^i, i \geq 0$:

```
Lemma pumping_aux:
∀ g: cfg _ _,
∀ t1 t2: btree (non_terminal' non_terminal terminal) _,
∀ n: _,
∀ c1 c2: list bool,
∀ v x: sentence,
   btree_decompose t1 c1 = Some (v, t2, x) →
   btree_cnf g t1 →
   broot t1 = n →
   bcode t1 (c1 ++c2) →
   c1 ≠ [] →
   broot t2 = n →
   bcode t2 c2 →
   (∀ i: nat,
      ∃ t': btree _ _,
         btree_cnf g t' ∧
         broot t' = n ∧
         btree_decompose t' (iter c1 i) = Some (iter v i, t2, iter x i) ∧
         bcode t' (iter c1 i ++c2) ∧
         get_nt_btree (iter c1 i) t' = Some n).
```

The proof continues by showing that each of these new trees can be combined with tree t obtained before, thus representing strings $uv^i wx^i y, i \geq 0$ as necessary (steps 15 and 16).

Finally, it must be proved that each of these trees is related to a derivation in G', which is accomplished by lemma `btree_equiv_produces_g_cnf`, the dual version of lemma `derives_g_cnf_equiv_btree` (step 17).

The Pumping Lemma has some 400 lines of Coq script, which adds to auxiliary lemmas and an extensive library of lemmas on binary trees and on the relation of binary trees to CNF grammars. The approach is constructive and requires, in the proof of the `pigeon` lemma (only), decidable equality. That was indeed the main reason why the definition of `cfg` enforces decidable equality on the types of the terminal and non-terminal symbols.

6. COMPARISON

Coq and HOL4 share many characteristics in common, in particular both are typed systems that use higher order logics. Coq, however, is based on the very powerful Calculus of Inductive Constructions (and thus supports constructive logic) and includes dependent types, a feature that is not offered by HOL4. HOL4 uses classical higher-order logic with axioms of infinity, extensionality and choice, and is based on simply typed lambda-calculus with polymorphic type variables. Both systems follow the LCF approach, a set of ideas about the design of proof assistants with the objective of granting a high level of confidence in their operation [Zam97, Wie06].

As a preparation to the formalization of the Pumping Lemma, Barthwal ([Bar10]) discusses the importance of generic parse trees (not necessarily binary) and formalizes its definition and some properties. Also, the relationship between valid parse trees and derivations in a grammar, and the notion of subtrees along with some functions. The strategy adopted in the proof is the same as ours. She uses the classical statement, however considering that the grammar is already in CNF, which means that the proof is valid only for grammars that do not generate the empty string. Basically, Barthwal proceeds by showing that sufficiently long sentences have parse trees with a repeated non-terminal in some path, and then by showing that a subtree can be "pumped" in such a way that the statement can be proved. Barthwal ends with some brief considerations about the complexity of the formalization in comparison to the informal proof. Still, the proof presented in the thesis is concise, using a mixture of informal arguments and HOL4 lemmas previously introduced.

In comparison, our statement refers to any context-free language (including those with the empty string) and we have also proved an alternative version with an additional clause ($|uy| \geq 1$, which is not present in the classical statement) and a smaller value of n, as in the original proof (for details, please refer to [Rama]).

7. CONCLUSIONS

The formalization of the Pumping Lemma for context-free languages represents the culmination of an effort that started with closure properties for context-free grammars [RdQ14] and continued with simplification for context-free grammars [RdQ15] and the Chomsky Normal Form [Rama]. The whole formalization has 20.000+ lines of Coq script and was developed over a period of two years.

The Pumping Lemma for CFLs is a significant result in language theory in general and this is, as far as the authors are aware of, the second ever formalization of it, 55 years after is was stated and informally proved for the first time (this is, however, the first in Coq and the first constructive proof). It has to be seen against the backdrop of the important and well sought after goal of formalizing fundamental results in language theory, as well as formalizing mathematics in general.

The libraries developed to support this formalization will hopefully play an equally important role, as they include general results on context-free language theory that can be used or adapted to prove other results. The whole work can serve different purposes, including the continued formalization of language theory and the teaching of formal languages, formalization and Coq itself.

Short-term future work includes the formalization of decidable questions for context-free languages (emptiness, membership, existence of empty string and existence of non-empty string), in order to simplify the statement of some theorems (as explained in Sections 4 and 5). Medium and long-term future work include the formalization of the Greibach Normal Form Theorem, pushdown automata and Ogden's Lemma (a stronger version of the Pumping Lemma for CFLs).

References

[Bar10] Aditi Barthwal. *A formalisation of the theory of context-free languages in higher order logic*. PhD thesis, The Australian National University, 2010. https://openresearch-repository.anu.edu.au/handle/1885/16399.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004. http://dx.doi.org/10.1007/978-3-662-07964-5.

[BHPS61] 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.

[BN10a] Aditi Barthwal and Michael Norrish. A Formalisation of the Normal Forms of Context-Free Grammars in HOL4. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23–27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2010. http://dx.doi.org/10.1007/978-3-642-15205-4_11.

[BN10b] Aditi Barthwal and Michael Norrish. Mechanisation of PDA and Grammar Equivalence for Context-Free Languages. In Anuj Dawar and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information and Computation, 17th International Workshop, WoLLIC 2010*, volume 6188 of *Lecture Notes in Computer Science*, pages 125–135. Springer, 2010. http://dx.doi.org/10.1007/978-3-642-13824-9_11.

[BN14] Aditi Barthwal and Michael Norrish. A mechanisation of some context-free language theory in HOL4. *Journal of Computer and System Sciences*, 80(2):346–362, March 2014. http://dx.doi.org/10.1016/j.jcss.2013.05.003.

[DKS] Christian Doczkal, Jan-Oliver Kaiser, and Gert Smolka. A Constructive Theory of Regular Languages in Coq. In Georges Gonthier and Michael Norrish, editors, *Certified Programs and Proofs: Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11–13, 2013, Proceedings*, volume 8307 of *Lecture Notes in Computer Science*, pages 82–97. Springer. http://dx.doi.org/10.1007/978-3-319-03545-1_6.

[FU14] Denis Firsov and Tarmo Uustalu. Certified CYK parsing of context-free languages. *Journal of Logical and Algebraic Methods in Programming*, 83(5–6):459–468, 2014. http://dx.doi.org/10.1016/j.jlamp.2014.09.002.

[FU15] Denis Firsov and Tarmo Uustalu. Certified Normalization of Context-Free Grammars. In *Certified Programs and Proofs:*

Fourth International Conference, CPP 2015, Mumbai, India, January 13–14, 2015, Proceedings, pages 167–174. ACM, 2015. http://doi.acm.org/10.1145/2676724.2693177.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages and Computation*. Addison-Wesley Publishing Co., Inc., 1979.

[JPL12] Jacques-Henri Jourdan, François Pottier, and Xavier Leroy. Validating LR(1) Parsers. In Helmut Seidl, editor, *Programming Languages and Systems: 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2012. http://dx.doi.org/10.1007/978-3-642-28869-2_20.

[KB10] Adam Koprowski and Henri Binsztok. TRX: A Formally Verified Parser Interpreter. In Andrew D. Gordon, editor, *Programming Languages and Systems: 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS, 2010, Paphos, Cyprus, March 20–28, 2010. Proceedings*, volume 6012 of *Lecture Notes in Computer Science*, pages 345–365. Springer, 2010. http://dx.doi.org/10.1007/978-3-642-11957-6_19.

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

[Ramb] Marcus Vinícius Midena Ramos. Source files of [Rama]. https://github.com/mvmramos/pumping.

[RdQ14] M. V. M. Ramos and Ruy J. G. B. de Queiroz. Formalization of closure properties for context-free grammars. *CoRR*, abs/1506.03428, 2014. http://arxiv.org/abs/1506.03428.

[RdQ15] M. V. M. Ramos and Ruy J. G. B. de Queiroz. Formalization of simplification for context-free grammars. *CoRR*, abs/1509.02032, 2015. http://arxiv.org/abs/1509.02032.

[Rid11] Tom Ridge. Simple, Functional, Sound and Complete Parsing for All Context-Free Grammars. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs: First International Conference, CPP 2011, Kenting, Taiwan, December 7–9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2011. http://dx.doi.org/10.1007/978-3-642-25379-9_10.

[RNV09] Marcus Vinícius Midena Ramos, João José Neto, and Italo Santiago Vega. *Linguagens Formais: Teoria Modelagem e Implementação*. Bookman, 2009.

[Sud06] Thomas A. Sudkamp. *Languages and Machines*. Addison-Wesley, 3rd edition, 2006.

[Wie06] Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*. Springer, 2006. http://dx.doi.org/10.1007/11542384.

[Zam97] Vincent Zammit. A comparative study of Coq and HOL. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: 10th International Conference, TPHOLs '97, Murray Hill, NJ, USA, August 19–22, 1997 Proceedings*, volume 1275 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 1997. http://dx.doi.org/10.1007/BFb0028403.