13th LSFA

Logical and Semantic Frameworks, with Applications

UFC, Fortaleza, CE

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

Marcus Vinícius Midena Ramos
(UNIVASF, Petrolina, PE)
José Carlos Bacelar Almeida
(HASLab - INESC TEC, Universidade do Minho, Braga, Portugal)
Nelma Moreira
(Departamento de Ciência de Computadores, Faculdade de Ciências, Porto, Portugal)
Ruy J. G. B. de Queiroz
(UFPE, Recife, PE)

September 26th, 2018

Scope

Formalize of a substantial part of context-free language theory in the Coq proof assistant.

- Formalization is the process of writing proofs such that they have a precise meaning over a simple and well-defined calculus whose rules can be automatically checked by a machine;
- Context-free language theory is fundamental in the representation and study of artificial languages, specially programming languages, and in the construction of their processors (compilers and interpreters);
- The formalization of context-free language theory is a key to the certification of compilers and programs, as well as to the development of new languages and tools for certified programming.

Overview

- Part of Formal Language Theory (Chomsky Hierarchy):
  - Regular Languages;
  - Context-Free Languages;
  - Context-Sensitive Languages;
  - Recursively Enumerable Languages.
- Developed from mid 1950s to late 1970s;
- Since then, mostly text proofs and almost no formalization;
- Relevant to the representation, study and implementation of artificial languages;

Steps

Main results since 2013:

1. Closure properties (9th LSFA 2014):
   - Union;
   - Concatenation;
   - Kleene star.
2. Grammar simplification (10th LSFA 2015):
   - Elimination of empty rules;
   - Elimination of unit;
   - Elimination of useless symbols;
   - Elimination of inaccessible symbols.
3. Chomsky Normal Form;
4. Pumping Lemma (JFR, 2016)
5. Languages that are not context-free (13th LSFA, 2018)

Context-Free Grammar

$G = (V, \Sigma, P, S)$, where:

- V is the vocabulary of G;
- Σ is the set of terminal symbols;
- $N = V \setminus \Sigma$ is the set of non-terminal symbols;
- P is the set of rules $\alpha \to \beta$, with $\alpha \in N$ and $\beta \in V^*$;
- $S \in N$ is the start symbol.

```
Record cfg (non_terminal terminal : Type): Type:= {
start_symbol: non_terminal;
rules: non_terminal → list (non_terminal + terminal) → Prop;
rules_finite:
   ∃ n: nat,
   ∃ ntl: nlist,
   ∃ tl: tlist,
   rules_finite_def start_symbol rules n ntl tl }.
```

Context-Free Grammar

Making sure that `cfg` represents a context-free grammar:

- General types might have an infinite number of elements;
- We must check that the rules of the grammar are built from finite sets of terminal and non-terminal symbols;
- We must also check that the set of rules is finite;
- The predicate `rules_finite_def` is used to make sure that these conditions are satisfied for every grammar in the formalization, either user-defined or constructed;
- A list of non-terminal symbols (`ntl`), a list of terminal symbols (`tl`) and an upper bound on the length of the right-hand side of the rules (`n`) must be supplied.

Example

$G = (\{S', A, B, a, b\}, \{a, b\}, \{S' \to aS', S' \to b\}, S')$ generates the language a^*b.

```
Inductive nt1: Type:= | S' | A | B.
Inductive t1: Type:= | a | b.
Inductive rs1: nt1 → list (nt1 + t1) → Prop:=
  r1: rs1 S' [inr a; inl S']
| r2: rs1 S' [inr b].

Definition g1: cfg nt1 t1:= {|
start_symbol:= S';
rules:= rs1;
rules_finite:= rs1_finite |}.
```

Derivation

Substitution process:

s_1 *derives* s_2 by application of zero or more rules: $s_1 \Rightarrow^* s_2$.

```
Inductive derives
  (non_terminal terminal : Type)
  (g : cfg non_terminal terminal)
  : 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)
```

Derivation

- Predicate `generates`: a derivation that begins with the start symbol of the grammar;
- Predicate `produces`: a derivation that begins with the start symbol of the grammar and ends with a sentence.

$$\underbrace{\underbrace{S \Rightarrow \alpha_1 \Rightarrow \overbrace{\alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1}}^{\texttt{derives}}}_{\texttt{generates}} \Rightarrow \alpha_n \Rightarrow \omega}_{\texttt{produces}}$$

Example

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aab$

```
Lemma produces_g1_aab:
produces g1 [a; a; b].
Proof.
unfold produces.
unfold generates.
simpl.
apply derives_step with (s2:=[inr a; inr a])(left:=S')(right:=[inr b]).
apply derives_step with (s2:=[inr a])(left:=S')(right:=[inr a;inl S']).
apply derives_start with (left:=S')(right:=[inr a;inl S']).
apply r11.
apply r11.
apply r12.
Qed.
```

Grammar Equivalence

$g_1 \equiv g_2$
if they generate the same language, that is,
$\forall s, (S_1 \Rightarrow^*_{g_1} s) \leftrightarrow (S_2 \Rightarrow^*_{g_2} s)$

```
Definition g_equiv
(non_terminal1 non_terminal2 terminal : Type)
(g1: cfg non_terminal1 terminal)
(g2: cfg non_terminal2 terminal): Prop:=
∀ s: list terminal,
produces g1 s ↔ produces g2 s.
```

Context-Free Language

- A *language* is a set of strings over a given alphabet;
- A *context-free language* is a language that is generated by some context-free grammar: $L(G) = \{w \mid S \Rightarrow_g^* w\}$.

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

Definition lang_of_g (g: cfg): lang :=
fun w: list terminal ⇒ produces g w.

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

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

Objectives

- 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 in the Coq proof assistant. 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;
- 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.

Pumping Lemma

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 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.

Languages

Classical languages considered:

1. *square*: $\{w \in \{a\}^* \mid \exists\, i, |w| = i^2, i \geq 0\}$,
2. *prime*: $\{w \in \{a\}^* \mid |w|$ is a prime number$\}$,
3. *anbncn*: $\{a^i b^i c^i \mid i \geq 0\}$.

For each:

- Text proof;
- Formal proof;
- Comparison.

Besides these languages, we also discuss:

4. *anbnanbn*: $\{a^n b^n a^n b^n \mid n \geq 0\}$
5. *ww*: $\{ww \mid w \in \{a, b\}^*\}$

Languages

Results:

1. *square*: straightforward;
   (∼20x expansion factor)
2. *prime*: straightforward;
   (∼20x expansion factor)
3. *anbncn*: much harder (why?);
   (∼100x expansion factor)
4. *anbnanbn*: much much harder (incomplete);
   (∼200x expansion factor)
5. *ww*: also incomplete.

Let's take a look at *anbncn*.

Language anbncn

Text proof

- 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;
- $a^n b^n c^n \in$ *anbncn* and $|a^n b^n c^n| \geq n$. Thus, the Pumping Lemma can be applied;
- $a^n b^n c^n = uvwxy$ for some u, v, w, x and y, with $|uvwxy| = 3n$, $1 \leq |vwx| \leq n$ and $uv^i wx^i y \in$ *anbncn*, $\forall i \geq 0$;
- 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*;

Language anbncn

Text proof

- 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*;
- Hypothesis is false and *anbncn* is not context-free.

Language anbncn

Formalization

```
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.
```

Language anbncn

Formal proof

Steps:

- We have to reason about vwx (either $vwx \in a^*b^*$ or $vwx \in b^*c^*$);
- v and x might be empty (but not both);
- We conclude about $|v|_a, |v|_b, |v|_c, |x|_a, |x|_b$ and $|x|_c$ (whether each is $= 0$ or $\neq 0$);
- 24 cases must be considered;
- 2 are discarded;
- 22 cases must be used to show that uv^2wx^2y can not belong to *anbncn*;
- The formalization is long and tedious;
- Some simplification can be pursued.

Language anbncn

Formal proof

$$
\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}
$$

Intersection

Context-free languages are not closed under intersection:

1. Formalize $L_1 = \{a^n b^n c^m \mid n \geq 0 \wedge m \geq 0\}$;
2. Prove L_1 is context-free;
3. Formalize $L_2 = \{a^m b^n c^n \mid n \geq 0 \wedge m \geq 0\}$;
4. Prove L_2 is context-free;
5. Formalize language intersection;
6. Prove $L_1 \cap L_2 = anbncn$;
7. Recall $anbncn$ is previously proved not to be context-free;

Results

Languages:

- *square*: straightforward to build and easy to read;
- *prime*: straightforward to build and easy to read;
- *anbncn*: long and complex with extensive case analysis;
- *anbnanbn*: longer and more complex;
  Take $a^n b^n a^n b^n$, guess about v and x and pump them;
- *ww*: similar to *anbnanbn*;
  Take $a^n b^n a^n b^n$, guess about v and x and pump them.

Why?

Intersection:

- straightforward to build and easy to read.

Results

- Size and complexity in proofs with case analysis only;
- That is, only with languages *anbncn*, *anbnanbn* and *ww*;
- Does not occur with languages *square* and *prime* and intersection;
- Possible reasons:
  - (*) Proof writing style (one tactic per line; no proof searching tactic);
  - (*) Parametrization can reduce the number of functions and lemmas;
  - (**) Extensive case analysis with no native support;
  - (**) Text proofs hide many details that have to be explicitly stated.

Case analysis

Case analysis with lists (examples from *anbncn* and *anbnanbn*):

- Any string that is a substring of a^*b^* belongs to a^*b^*;
- Any string with maximum length n that is a substring of $a^nb^nc^n$ belongs to $a^*b^* \mid b^*c^*$;
- Any string with maximum length n that is a substring of $a^nb^na^nb^n$ belongs to $a^*b^* \mid b^*a^*$.

Text proofs

Text proofs hide details (examples from *anbncn*):

- "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".

Conclusions

Possible consequences:

- Future support (new libraries) for combinatorics of strings (lists of symbols) in Coq might help and simplify the formalization;
- More detailed (and thus self-explanatory) text proofs must be considered.

To be used in undergraduate classes as case study.

Final remarks

- All scripts are publicly available and can be executed in Coq 8.8.1 (July 2018);
- Application of a previously formalized lemma, fundamental to the language class;
- First known formal proofs that some classical languages are not context-free;
- Insights involving text and formal proofs;
- Helpful in teaching formal languages in a formal framework;
- Adds a new closure result to previous results concerning the class of the context-free languages.