# Bar-Hillel Theorem Mechanization in Coq

## Closure of Context-Free Languages Under Intersection With Regular Languages is Mechanized in Coq


Sergey Bozhko
Doctoral Student
Max Planck Institute for Software Systems (MPI-SWS)
SaarbrÃijcken, Germany
sbozhko@mpi-sws.com

Leyla Khatbullina
Student
Department of Computer Science and Technology
St.Petersburg Electrotechnical University "LETI"
St.Petersburg, Russia
leila.xr@gmail.com

Semyon Grigorev
Associate Professor
St.Petersburg State University
St.Petersburg, Russia
s.v.grigoriev@spbu.ru
Researcher
JetBrains Research
St.Petersburg, Russia
semen.grigorev@jetbrains.com



## Abstract

Formal language theory has a deep connection with such areas as static code analysis, graph database querying, formal verification, and compressed data processing. Many application problems can be formulated in terms of languages intersection. The Bar-Hillel theorem states that context-free languages are closed under intersection with a regular set. This theorem has a constructive proof and thus provides a formal justification of correctness of the algorithms for applications mentioned above. Mechanization of the Bar-Hillel theorem, therefore, is both a fundamental result of formal language theory and a basis for the certified implementation of the algorithms for applications. In this work, we present the mechanized proof of the Bar-Hillel theorem in Coq. We generalize results of Gert Smolka and Jana Hofmann and use them as the base for our work.



***Keywords*** Formal languages, Coq, Bar-Hillel Theorem, Closure, Intersection, Regular Language, Context-free Language


## 1 Introduction

Formal language theory has deep connection with different areas such as static code analysis [30, 36, 41, 42, 45–47], graph database querying [22, 23, 27, 48], formal verification [11, 15], and others. One of the most frequent uses is to formulate a problem in terms of languages intersection. In verification, one language can serve as a model of a program and another language describe undesirable behaviors. When intersection of these two languages is not empty, one can conclude that the program is incorrect. Usually, the only concern is the decidability of the languages intersection emptiness problem. But in some cases a constructive representation of the intersection may prove useful. This is the case, for example, when the intersection of the languages models graph querying: a language produced by intersection is a query result and to be able to process it, one needs the appropriate representation of the intersection result.

Let us consider several applications starting with the user input validation. The problem is to check if the input provided by the user is correct with respect to some validation template such as a regular expression for e-mail validation. User input can be represented as a one word language. The intersection of such a language with the language specifying the validation template is either empty or contains the only string: the user input. If the intersection is empty, then the input should be rejected.

Checking that a program is syntactically correct is another example. The AST for the program (or lack thereof) is just a constructive representation of the intersection of the one word language (the program) and the programming language itself.

Graph database regular querying serves as an example of the intersection of two regular languages [1, 2, 27]. Next and one of the most comprehensive cases with decidable emptiness problem is an intersection of a regular language with a context-free language. This case is relevant for program analysis [42, 45, 46], graph analysis [20, 23, 48], context-free compressed data processing [31], and other areas. The constructive intersection representation in these applications is helpful for further analysis.

Intersection of some classes of languages is not generally decidable. For example, intersection of the linear conjunctive and the regular languages, used in the static code analysis [47], is undecidable while multiple context-free languages (MCFL) is closed under intersection with regular languages and emptiness problem for MCFLs is decidable [44]. Is it possible to express any useful properties in terms of regular and multiple context-free languages intersection? This question is beyond the scope of this paper but provides a good reason for future research in this area. Moreover, the history about pumping lemma for MCFG shows necessity to


*Conference'17, July 2017, Washington, DC, USA*
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

mechanize formal language theory. In this paper we focus on the intersection of regular and context-free languages.

Some applications mentioned above require certifications. For verification this requirement is evident. For databases it is necessary to reason about security aspects and, thus, we should create certified solutions for query executing. Certified parsing may be critical for secure data loading (for example in Web), as well as certified regular expressions for input validation. As a result, there is a big number of papers focusing on regular expressions mechanization and certification [17], and a number on certified parsers [5, 18, 21]. On the other hand, mechanization (formalization) is important by itself as theoretical results mechanization and verification, and there is a lot of work done on formal languages theory mechanization [4, 19, 38]. Also it is desirable to have a base to reason about parsing algorithms and other problems on languages intersection.

Context-free languages are closed under intersection with regular languages. It is stated as the Bar-Hillel theorem [3] which provides a constructive proof and a construction for the resulting language description. We believe that the mechanization of the Bar-Hillel theorem is a good starting point for certified application development and since it is one of fundamental theorems, it is an important part of formal language theory mechanization. And this work aims to provide such mechanization in Coq.

Our current work is a first step: we provide mechanization of theoretical results on context-free and regular languages intersection. We choose the result of Gert Smolka and Jana Hofmann on context-free languages mechanization [24] as a base for our work. The main contribution of this paper may be summarized as follows.

- We provide the constructive proof of the Bar-Hillel theorem in Coq.
- We generalize results of Gert Smolka: terminal alphabet in context-free grammar definition changed from Nat to generic and all relative stuff also modified to works with updated definition.
- All code is published on GitHub: https://github.com/YaccConstructor/YC_in_Coq.

This work is organized as follows. In the section 2 we formulate Bar-Hillel theorem and provide the sketch of its proof. The next part is a brief discussion of the Chomsky normal form in section 3. After that we describe our solution in the section 4. This description is split into steps with respect to provided sketch and contains basic definitions, Smolka results generalization, handling of trivial cases, and steps summarization as a final proof. Finally, we discuss related works in the section 5 and conclude with the discussion of the presented work and possible directions for future research in the section 6.

## 2 Bar-Hillel Theorem

In this section we provide the Bar-Hillel theorem and sketch the proof which we use as base of our work. Also we provide some additional lemmas which are used in the proof of the main theorem.

**Lemma 2.1.** *If $L$ is a context free language and $\varepsilon \notin L$ then there is a grammar in Chomsky Normal Form that generates $L$.*

**Lemma 2.2.** *If $L \neq \varnothing$ and $L$ is regular then $L$ is the union of regular language $A_1, \dots, A_n$ where each $A_i$ is accepted by a DFA with exactly one final state.*

**Theorem 2.3** (Bar-Hillel Theorem)**.** *If $L_1$ is a context free language and $L_2$ is a regular language then $L_1 \cap L_2$ is context free.*

Sketch of the proof.

1. By lemma 2.1 we can assume that there is a context-free grammar $G_{CNF}$ in Chomsky normal form, such that $L(G_{CNF}) = L_1$
2. By lemma 2.2 we can assume that there is a set of regular languages $\{A_1 \dots A_n\}$ where each $A_i$ is recognized by a DFA with exactly one final state and $L_2 = A_1 \cup \dots \cup A_n$
3. For each $A_i$ we can explicitly define a (?) grammar of the intersection: $L(G_{CNF}) \cap A_i$
4. Finally, we join them together with the (?) operation of union

## 3 The Chomsky Normal Form

The important part of our proof is that any context-free language can be described with grammar in Chomsky Normal Form (CNF) or, equally, any context-free grammar can be converted to the grammar in CNF which specifies the same language. Let us recall the definition of CNF and the algorithm for CFG to CNF conversion.

**Definition 3.1** (Chomsky Normal Form)**.** Context-free grammar is in CNF if:

- start nonterminal does not occur in the right-hand side of rules,
- all rules are of the form: $N_i \to t_i$, $N_i \to N_j N_k$ or $S \to \varepsilon$ where $N_i, N_j, N_k$ are nonterminals, $t_i$ is a terminal and $S$ is the start nonterminal.

Transformation algorithm has the following steps.

1. Eliminate the start symbol from the right-hand sides of the rules.
2. Eliminate rules with nonsolary terminals.
3. Eliminate rules which right-hand side contains more than two nonterminals.
4. Delete $\varepsilon$-rules.
5. Eliminate unit rules.

As far as Bar-Hillel theorem operates with arbitrary context-free languages, but the selected proof requires grammar in CNF, it is necessary to implement a certified algorithm for conversion of arbitrary CFG to CNF. We wanted to reuse existing mechanized proof of conversion of arbitrary context-free grammar to CNF. We choose one provided in Smolka’s work and discussed in context of our work in the section 4.1.

# 4 Bar-Hillel Theorem Mechanization in Coq

In this section we describe in detail all the fundamental parts of the proof. Also in this section, we briefly describe motivation to use the chosen definitions. In addition, we discuss the advantages and disadvantages of using of third-party proofs.

Overall goal of this section is to provide step-by-step algorithm of constructing the context-free (CF) grammar of the intersection of two languages. Final formulation of the obtained theorem can be found in the last subsection.

## 4.1 Smolka’s Results Generalization

In this section, we describe the exact steps taken to use the results of Gert Smolka and Jana Hofmann [24] on context-free languages mechanization in Coq in the proof of this article’s theorem.

A substantial part of this proof relies on the work of [24][1] from which many definitions and theorems were taken. Namely, the definition of a grammar, definitions of a derivation in grammar, some auxiliary lemmas about the decidability of properties of grammar/derivation, we also use the theorem that states that there always exists the transformation from context-free grammar to grammar in Chomsky Normal Form (CNF).

However, the proof about existence of transformation to CNF had one major flaw that we needed to fix. One could define a terminal symbol as in inductive type over natural numbers (Lst.1).

```
Inductive ter : Type := | T : nat -> ter.
```

**Listing 1.** The original Smolka’s definition of terminals

That is how it was done in [24]. However for purposes of our proof, we need to consider nonterminals over the alphabet of triples. Therefore, it was decided to add polymorphism over the target alphabet. Namely, let $Tt$ and $Vt$ be types with decidable relation of equality, then we can define the types of terminal and nonterminal over alphabets $Tt$ and $Vt$ respectively as presented in 2.

[1]Gert Smolka, Jana Hofmann, Verified Algorithms for Context-Free Grammars in Coq. Related sources in Coq: https://www.ps.uni-saarland.de/~hofmann/bachelor/coq_src.zip. Documentation: https://www.ps.uni-saarland.de/~hofmann/bachelor/coq/toc.html. Access date: 10.10.2018.

```
Inductive ter : Type := | T : Tt -> ter.
Inductive var : Type := | V : Vt -> var.
```

**Listing 2.** The new polymorphic definitions of terminals and nonterminals

The proof of Smolka has a clear structure, therefore only part of the proof where the use of natural numbers was essential has become incorrect. One of the grammar transformations (namely deletion of long rules) requires the creation of many new non-terminals. In the original proof for this purpose, the maximum over non-terminals included in the grammar was used. However, it is impossible for an arbitrary type.

To tackle this problem we introduce an additional assumption on alphabet types for terminals and nonterminals. We require an existence of the bijection between natural numbers and alphabet of terminals as well as nonterminals.

Another difficulty is that the original work defines grammar as a list of rules (without a distinct starting nonterminal). Thus, in order to define the language that is defined by a grammar, one needs to specify the grammar and a starting terminal. This leads to the fact that the theorem about the equivalence of a CF grammar and the corresponding CNF grammar isn’t formulated in the most general way, namely it guarantees equivalence only for non-empty words.

```
Lemma language_normal_form
    (G:grammar) (A: var) (u: word):
  u <> [] ->
  (language G A u <->
     language (normalize G) A u).
```

**Listing 3.** The equivalence of languages specified by context-free grammar and by transformed grammar in CNF

Changes in the definition of grammar or language would lead to significant code corrections. However, the question of whether the empty word is derivable is decidable for both the CF grammar and the DFA. Therefore, it is possible to simply consider two cases (1) when the empty word is derivable in the grammar and (2) when the empty word is not derivable.

## 4.2 Basic Definitions

In this section, we introduce the basic definitions used in the article, such as alphabets, context-free grammar, and derivation.

We define a symbol is either a terminal or a nonterminal (4).

Next we define a word and a phrase as lists of terminals and symbols respectively (5). One can think that word is an element of the language defined by grammar, and phrase is an intermediate result of derivation. Also, phrase is a right side of derivation rule.

```
Inductive symbol : Type :=
  | Ts : ter -> symbol
  | Vs : var -> symbol.
```

**Listing 4.** Definition of symbol (union of terminals and non-terminals)

```
Definition word := list ter.
Definition phrase := list symbol.
```

**Listing 5.** Definitions of word and phrase.

The notion of nonterminal doesn't make sense for DFA, but in order to construct derivation in grammar we need to use nonterminal in intermediate states. For phrases, we introduce a predicate that defines whenever a phrase consists of only terminals. And if so, the phrase can be safely converted to the corresponding word.

We inheriting the definition of CFG from [24]. Rule is defined as a pair of a nonterminal and a phrase, and grammar is a list of rules (6). Note, that this definition of grammar is not include start nonterminal, and it is sufficient difference from classical definition, because such defined grammar is not specified language.

```
Inductive rule : Type :=
| R : var -> phrase -> rule.

Definition grammar := list rule.
```

**Listing 6.** Context-free rule and grammar definition

An important step towards the definition of a language specified by a grammar is the definition of derivability (7). Having $der(G, A, p)$ — means that phrase $p$ is derivable in grammar $G$ starting from nonterminal $A$.

```
Inductive der (G : grammar)
              (A : var) : phrase -> Prop :=
| vDer : der G A [Vs A]
| rDer l : (R A l) el G -> der G A l
| replN B u w v :
    der G A (u ++ [Vs B] ++ w) ->
    der G B v -> der G A (u ++ v ++ w).
```

**Listing 7.** Derivability definition. Informally it is a recognizer of the language specified by grammar $G$ and start nonterminal $A$

Proof of [8] requires grammar to be in CNF. We used statement that every grammar in convertible into CNF from [24].

In the end, we recall the definition of language. We say that a phrase (not a word) $w$ belongs to the language generated by a grammar $G$ from a non-terminal $A$, if $w$ is derivable from nonterminal $A$ in grammar $G$ and $w$ consists only of terminals.

```
Definition language
         (G : grammar) (A : var) (w : phrase) :=
  der G A w /\ terminal w.
```

**Listing 8.** Definition of language

### 4.3 General Scheme of the Proof

General scheme of our proof is based on constructive proof presented in [8]. This proof does not use push-down automata explicitly and operates by grammars, so it looks pretty simple to be mechanized. In the following subsections the main steps of the proof are presented. Overall, we will adhere to the following plan.

1. First we consider trivial case, when DFA has no state
2. Every CF language can be converted to CNF
3. Every DFA can be presented as an union of DFAs with single final state
4. Intersecting grammar in CNF with DFA with one final state
5. Proving than union of CF languages is CF language

### 4.4 Trivial Cases

First, we consider the case when the DFA does not have any state, that is, the number of states is equal to zero. In this case, we can immediately derive a contradiction. By definition, for any DFA an initial state is known. It means that there is at least one state, which contradicts the fact that the number of states is equal to zero.

In addition, it is worth mention, that in the proof [8] cases when in grammar an empty word is derivable or a DFA sets an empty language are discarded as trivial. It is assumed that the proof for this cases one can carry out himself. In our proof, we do not consider these cases as separate ones. Consideration of these cases is included in the corresponding theorems in general formulation.

### 4.5 Regular Languages and Automata

In this section we describe definitions of DFA and DFA with exactly one final state, we also present function that converts any DFA to a set of DFA with one final state and lemma that states this split preserves language in some sense.

We assume that regular language is described by DFA. As the definition of an DFA, we have chosen a general definition, which does not impose any restrictions on the type of input symbols and the number of states. Thus, in our case, the DFA is a 5-tuple, (1) a state type, (2) a type of input symbols, (3) a start state, (4) a transition function, and (5) a list of final states (Lst.9).

```
Context {State T: Type}.
Record dfa: Type :=
  mkDfa {
    start: State;
    final: list State;
    next: State -> (@ter T) -> State;
  }.
```

**Listing 9.** Definition of deterministic finite automaton

Next we define a function that evaluates the final state of the automaton if it starts from state $s$ and receives a word $w$.

```
Fixpoint final_state
           (next_d: dfa_rule)
           (s: State)
           (w: word): State :=
  match w with
  | nil => s
  | h :: t => final_state next_d
                           (next_d s h)
                           t
  end.
```

**Listing 10.** TODO

We say that the automaton accepts a word $w$ being in state $s$ if the function [*final_state s w*] returns one of the final states. Finally, we say that an automaton accepts a word $w$, if the DFA starts from the initial state and ends in one of the final states.

In order to define the DFA with exactly one final state, it is necessary to replace the list of final states by one final state in the definition of an ordinary DFA. Related definitions such as *accepts* and *dfa_language* should be modified slightly.

```
Record s_dfa : Type :=
  s_mkDfa {
    s_start: State;
    s_final: State;
    s_next: State -> (@ter T) -> State;
  }.
```

**Listing 11.** Definition of DFA with exactly one final states

Similarly, we can define functions *s_accepts* and *s_dfa_language* for DFA with one final state. Since in this case there is only one final state in order to define function *s_accepts* it is enough to check the state in which the automaton stopped with the finite state. Function *s_dfa_language* repeats function *dfa_language* except that the function for a DFA with one final state should use *s_accepts* instead of *accepts*.

Now we can to define a function that converts an ordinary DFA into a set of DFAs with exactly one final state (Lst.12).

Let $d$ be a DFA. Then the list of its final states is known. For each such state, one can construct a copy of the original DFA, but with one selected final state.

```
Fixpoint split_dfa_list
    (st_d : State)
    (next_d : dfa_rule)
    (f_list : list State): list (s_dfa) :=
  match f_list with
  | nil => nil
  | h :: t => (s_mkDfa st_d h next_d)
                :: split_dfa_list st_d next_d t
  end.

Definition split_dfa (d: dfa) :=
  split_dfa_list (start d) (next d) (final d).
```

**Listing 12.** Split DFA into set of DFAs with exactly one final state

Now we should prove theorem that the function of splitting preserves the language (Lst.13).

```
Lemma correct_split:
  forall dfa w,
    dfa_language dfa w <->
    exists sdfa,
      In sdfa (split_dfa dfa) /\
      s_dfa_language sdfa w.
```

**Listing 13.** Splitting of DFA into DFAs with exactly one final state preserves language

**Theorem 4.1.** *Let dfa be an arbitrary DFA and w be a word. Then the fact that dfa accepts w implies that there exists a single-state DFA s_dfa, such that s_dfa* $\in$ *split_dfa(dfa). And vice versa, for any s_dfa* $\in$ *split_dfa(dfa) the fact that s_dfa accepts a word w implies that dfa also accepts w.*

**Proof.** Let us divide the proof into two parts.

1. Suppose *dfa* accepts $w$. Then we prove that there exists a single-state DFA *s_dfa*, such that *s_dfa* $\in$ *split_dfa(dfa)*. Let *finals* be the set of final states of *dfa*. We carry out the proof by induction on *finals*.
   Base step: *finals = [::]*. Trivial by contradiction (DFA with no final state cannot accept a word).
   Induction step: *finals = a::old_finals* and the statement holds for *old_finals*. Since *dfa* accepts $w$, it either ends up in $a$, or in one of the state from *old_finals*. If *dfa* is ends up in $a$, then we simply choose an automaton with the final state that is equal to $a$. Such an automaton exists, since now the list of final states also contains $a$. On the other hand, if *dfa* is ends up in one of

the state from *old_finals*, then we can apply induction hypothesis.

2. Similarly for the opposite direction. Assume that there exists an automaton with exactly one final state from *split_dfa(dfa)* that accepts $w$. Then we prove that *dfa* also accepts $w$. Let *finals* be the set of final states of *dfa*. We carry out the proof by induction on *finals*.
   Base step: *finals = [::]*. Trivial by contradiction.
   Induction step: *finals = a::old_finals* and the statement holds for *old_finals*. We know that one of the DFAs form *split_dfa(dfa* accepts $w$, its final state either is equal to $a$, or lies in *old_finals*. If the final state is equal to $a$, then *dfa* also ends up in state $a$. On the other hand, if final state lies in *old_finals*, then we can apply induction hypothesis.

### 4.6 Chomsky Induction

In this section, we introduce the notion of Chomsky induction.

Naturally many statements about properties of language's words can be proved by induction over derivation structure. Unfortunately, grammar can derive phrase us an intermediate step, but DFA supposed to work only with words, so we can not simply apply induction over derivation structure. To tackle this problem we create custom induction principle for grammars in CNF.

As one might have noticed the current definition of derivability does not imply the ability to "reverse" the derivation back. That is, from the fact if a phrase $w$ is derived from a non-terminal $A$ in a grammar $G$ does not follow anything about the rules of the grammar or properties of this derivation. Because of this, we introduce an additional assumption on derivations that is similar in some sense to the syntactic analysis of words. Namely, we assume that if phrase $w$ is derived from nonterminal $A$ in grammar $G$, then either there is a rule $A \to w \in G$ or there is an RHS *rhs* such that $A \to rhs \in G$ and $w$ is derivable from *rhs*.

```
Definition syntactic_analysis_is_possible :=
  forall (G : grammar) (A : var) (w : phrase),
  der G A w ->
   (R A w \in G) \/
   (exists rhs, R A rhs \in G /\ derf G rhs w).
```

**Listing 14.** If word in language then we can recostruct its derivation

The main point is that if we have a grammar in CNF, we can always divide the word into two parts, each of which is derived only from one nonterminal. Note that if we naively take a step back, we can get nonterminal in the middle of the word. Such a situation will not make any sense for DFA.

With induction we always work with subtrees that describes some part of word. Here is a picture of subtree describing intuition behind the Chomsky induction.

(**TODO**: is it okay to use png?) **TODO**: add picture

(**TODO**: should be lemma) More formally:

**Theorem 4.2.** *Let G be a grammar in CNF. Consider an arbitrary nonterminal $N \in G$ and phrase which consists only on terminals $w$. If $w$ is derivable from $N$ and $|w| \geq 2$, then there exists two nonterminals $N_1, N_2$ and subphrases of $w$ — $w_1, w_2$ such that: $N \to N_1 N_2 \in G$, $der(N_1, w_1)$, $der(N_2, w_2)$, $|w_1| \geq 1$, $|w_2| \geq 1$ and $w_1 + +w_2 = w$.*

**Proof.** The proof heavily uses the fact that grammar $G$ is in Chomsky Normal Form. We apply the hypothesis "syntactic analysis is possible". After application, we get the fact that word $w$ is either an RHS of a rule of grammar $G$, or there is a phrase *phr*, such that (1) word $w$ is derivable from phrase *phr* and (2) there exists a non-terminal $N$ such that $N- > prh \in G$.

The first case we finish with the proof by contradiction since the grammar is in CNF and there might be only a single terminal in an RHS (by assumption we have $|w| \geq 2$). On the other hand, if there is an intermediate phrase that was obtained by applying a rule, then the phrase has form $N_1 N_2$, since it also derived by rule in normal form. Finally, now we need to prove that both of this nonterminals has a non-empty contribution to word $w$. This is also true since it is impossible to derive empty word in CNF grammar (see ...).

(**TODO**: should be lemma)

**Theorem 4.3.** *Let G be a grammar in CNF. And P be a predicate on nonterminals and phrases (i.e. $P : var \to phrase \to Prop$). Let's also assume that the following two hypotheses are satisfied: (1) for every terminal production (i.e. in the form $N \to a$) of grammar G, $P(r, [Ts\ r])$ holds and (2) for every $N, N_1, N_2 \in G$ and two phrases that consist only of terminals $w_1, w_2$, if $P(N_1, w_1)$, $P(N_2, w_2)$, $der(G, N_1, w_1)$ and $der(G, N_2, w_2)$ then $P(N, w_1 + +w_2)$. Then for any nonterminal N and any phrase consisting only of terminals w, the fact that w is derivable from N implies $P(N, w)$.*

**Proof.** Let $n$ be an upper bound of the length of word $w$. We carry out the proof by induction on $n$.

*Base case:* $n = 0$. Proof by contradiction. $|w| \leq 0$ implies that $w$ is empty. But an empty word cannot be derived in CNF grammar (see ...).

*Induction step:* $|w| \leq n + 1$. This fact is equivalent to the following: $|w| = n + 1$ or $|w| < n$. In case of $|w| < n$ we use the induction hypothesis. Next we consider two new cases, either $|w| = 1$, or $1 < |w| = n + 1$. In the first case, it is clear that this is possible only if there is a production $N- > w$, which means you can apply assumption (1). If the word is longer than 1, then we apply the previous lemma, after that we can conclude that $\exists w1w2, w = w1 + +w2$. After that, one need to apply assumption (2). We subgoals that are guaranteed by the lemma .... And for shorter words, we apply the induction hypothesis.

TODO: add some text

### 4.7 Intersection of CFG and Automaton

Since we already have lemmas about the transformation of a grammar to CNF and the transformation a DFA to a DFA with exactly one state, further we assume that we have (1) DFA with exactly one final state — *dfa* and (2) grammar in CNF — *G*. In this section, we describe the proof of the lemma that states that for any grammar in CNF and any automaton with exactly one state there is the intersection grammar.

#### 4.7.1 Construction of Intersection

Next we present adaptation of the algorithm given in [8].

Let $G_{INT}$ be the grammar of intersection. In $G_{INT}$ nonterminals presented as triples $(from \times var \times to)$ where *from* and *to* are states of *dfa*, and *var* is a nonterminal of *G*.

Since *G* is a grammar in CNF, it has only two type of productions: (1) $N \to a$ and (2) $N \to N_1 N_2$, where $N, N_1, N_2$ are nonterminals and *a* is a terminal.

For every production $N \to N_1 N_2$ in *G* we generate a set of productions of the form

$$(from, N, to) \to (from, N_1, m)(m, N_2, to)$$

where: $from, m, to$ — goes through all *dfa* states.

```
Definition convert_nonterm_rule_2
  (r r1 r2: _)
  (state1 state2 : _) :=
  map (fun s3 => R (V (s1, r, s3))
                  [Vs (V (s1, r1, s2));
                   Vs (V (s2, r2, s3))])
    list_of_states.

Definition convert_nonterm_rule_1
             (r r1 r2: _)
             (s1 : _) :=
  flat_map (convert_nonterm_rule_2 r r1 r2 s1)
           list_of_states.

Definition convert_nonterm_rule (r r1 r2: _) :=
  flat_map (convert_nonterm_rule_1 r r1 r2)
           list_of_states.
```

**Listing 15.** Grammar rules convertions for nonterminal rules

For every production of the form $N \to a$ we add a set of productions

$$(from, N, (dfa\_step(from, a))) \to a$$

where *from* — goes through all *dfa* states and *dfa_step (from, a)* is the state in which the *dfa* appears after receiving terminal *a* in state *from*.

```
Definition convert_terminal_rule
             (next: _)
             (r: _)
             (t: _): list TripleRule :=
  map (fun s1 => R (V (s1, r, next s1 t))
                   [Ts t])
      list_of_states.
```

**Listing 16.** Grammar rules convertion for terminal rule

Next we join the functions above to get a generic function that works for both types of productions. Note that since the grammar is in CNF, the third alternative can never be the case.

```
Definition convert_rule (next: _) (r: _ ) :=
  match r with
  | R r [Vs r1; Vs r2] =>
      convert_nonterm_rule r r1 r2
  | R r [Ts t] =>
      convert_terminal_rule next r t
  | _  => []   (* Never called *)
  end.

Definition convert_rules
  (rules: list rule) (next: _): list rule :=
  flat_map (convert_rule next) rules.

(* Maps grammar and s_dfa
   to grammar over triples *)
Definition convert_grammar grammar s_dfa :=
  convert_rules grammar (s_next s_dfa).
```

**Listing 17.** Grammar convertion by using rules convertions

Note that at this point we do not have any manipulations with starting rules. Nevertheless, the hypothesis of the uniqueness of the final state of the DFA, will help us unambiguously introduce the starting nonterminal of the grammar of intersection.

#### 4.7.2 Correctness of Intersection

In this subsection we present a high-level description of the proof about correctness of the intersection function.

In the interest of clarity of exposition, we skip some auxiliary lemmas, such as (TODO:fix) "we can get the initial grammar from the grammar of intersection by projecting the triples back to terminals/nonterminals ". Also note that the grammar after the conversion remains in CFN. Since the transformation of rules does not change the structure of the rules, but only replaces one(??!!) terminals and nonterminals with others.

(**TODO**: move to ...?) The starting nonterminal for the intersection grammar is the following nonterminal: *(start, S, final)*. Where: *start* — the start state of DFA, $S$ — the start symbol of initial grammar, and *final* — the final state of DFA.

Next we prove the two main lemmas. Namely, the derivability in the initial grammar and the *s_dfa* implies the derivability in the grammar of intersection. And the other way around, the derivability in the grammar of intersection implies the derivability in the initial grammar and the *s_dfa*.

Let $G$ be a grammar in CNF. In order to use Chomsky Induction we also assume that syntactic analysis is possible.

**Theorem 4.4.** *Let s_dfa be an arbitrary DFA, let r be a nonterminal of grammar G, let from and to be two states of the DFA. We also pick an arbitrary word — w. If in grammar G it is possible to derive w out of r and starting from the state from when w is received, the s_dfa ends up in state to, then word w is also derivable in grammar (convert_rules G next) from the nonterminal (V (from, r, to)).*

**Proof.** It would be logical to use induction on the derivation structure in grammar $G$. But as it was discussed earlier, this is not the case, otherwise we will get a phrase (list of terminals and nonterminals) instead of a word. Therefore we should use another way to use induction, and for grammar in Chomsky normal form it is possible as we show in the section **??**. Roughly speaking, we can split the word into two subwords, such that each of them can be derived from some nonterminal and these two nonterminals is a RHS of some rule from the given grammar.

Let's apply chomsky induction principle with the following predicate $P$:

$$
\begin{aligned}
&P := \lambda\ r\ phr \Rightarrow \\
&\qquad \forall (next : dfa\_rule)(from\ to : DfaState), \\
&\qquad \mathit{final\_state\ next\ from\ (to\_word\ phr) = to} \rightarrow \\
&\qquad \mathit{der\ (convert\_rules\ G\ next)\ (V\ (from,\ r,\ to))\ phr.}
\end{aligned}
$$

Basically, predicate $P$ is the property that we are trying to prove in the theorem. Chomsky Induction has 3 assumptions. (1) The phrase to which $P$ is applied should consist of only non-terminals. We consider only words in this theorem, therefore after conversion of the word to the phrase, no terminals can appear in it. So, we do not violate this assumption. Moreover, there is a base of induction (2) in the form of a property for a terminal rule and (3) an induction step for a non-terminal rule. Both statements can be proved by induction on the number of rules in the grammar $G$ in combination with a simple calculation of the functions *convert_terminal_rule* and *convert_nonterm_rule* for terminal and non-terminal rules, respectively.

On the other side, now we need to prove the theorems of the form "if it is derivable in the grammar of triples, then it is derivable in the automaton and in ordinary grammar"!!!

We start with the DFA.

**Theorem 4.5.** *Let from and to be states of the automaton, var be an arbitrary non-terminal grammar of G. We prove that If a word w is derived from the non-terminal (from, var, to) in the grammar (convert_rules G), then the automaton, starting from the state from at the input w stops in state to.*

**Proof.** Like last time, we use the principle of Chomsky Induction. We apply the induction with the following parameter $P$:

$$
\begin{aligned}
&P := \lambda\ tr\_non\ phr \Rightarrow \\
&\qquad \mathit{final\_state\ next\ \ (first3\ \ tr\_non)\ (to\_word\ \ phr) =} \\
&\qquad \mathit{third3\ r.}
\end{aligned}
$$

Note that in this case, one need to use projections from triple-non-terminals to "single" non-terminals. Induction is carried out in the grammar of triples, but the property operates with the automaton. However, this property can also be expressed in terms of triples-non-terminals. As last time, $P$ is the statement we want to prove. After applying the induction principle, it remains only to prove the fidelity of the assumptions. First of all, since $w$ is a word, converting it into a phrase does not add any non-terminal. Next, one need to show that the grammar *convert_rules G* is in CNF. It is easy to see, since $G$ in CNF and transformation *convert_rules* maps non-terminal rules to (TODO: "triple"?) "triple" non-terminal rules and terminal rules to "triple" terminal rules. Finally, one need to show that both assumption of the induction principle hold. In this case it would be:

$$
\begin{gathered}
r \rightarrow t \in convert\_rules\, G \implies \\
next\ (fst3\ (unVar\, r))\ t = thi3(unVar\, r)
\end{gathered}
$$

and

$r \rightarrow [r1; r2] \in convert\_rules\, G- >$

$\ldots- >$

$final\_state\ next(fst3(unVarr))(to_word(w1 + +w2)) = thi3(unVarr)$

In both cases, proof can be done by an "inverse" calculation of functions *convert_terminal_rule* and *convert_nonterm_rule*. That is, by inversing the assumption

$$r \rightarrow [r1; r2] \in convert\_rules\, G$$

in the assumptions, we gradually come to the conclusion that the only possible option is that the input satisfies the property of the goal. Then it remains only to simplify assumptions and conclusion.

Further we prove the theorem for grammar.

**Theorem 4.6.** *Let from and to be the states of the automaton, let var be an arbitrary non-terminal of grammar G. We prove* *that if a word w is derivable from non-terminal (from, var, to) in the grammar (convert_rules G), then w is also derivable in grammar G from nonterminal var.*

**Proof.** We again prove the theorem using Chomsky induction with the following predicate P:

$$P := \lambda\, r\; phr \Rightarrow der\; G\; (snd3\; (unVar\; r))\; (phr).$$

Again, note that induction is carried out in the grammar of triples, but the property is talking about the "unit" grammar, therefore we use projections from triples to non-triples. Here we can use the same idea of "inversing" of functions *convert_terminal_rule* and *convert_nonterm_rule*. Again, by inversing the induction hypothesis, we gradually come to the conclusion that the only possible option is that the input satisfies the property of the goal.

Well, in the end one need to combine both theorems to get full equivalence. On this, the correctness of the intersection is proved.

### 4.8 Union of Languages

After the previous step, we have a list of grammars of CF languages, in this section, we provide a function by which we construct a grammar of the union of languages.

For this, we need nonterminals from every language to be from different non-intersecting sets. To achieve this we add labels to nonterminals. Thus, each grammar of the union would have its own unique ID number, all nonterminals within one grammar will have the same ID which coincides with the ID of a grammar. In addition, it is necessary to introduce a new starting nonterminal of the union.

```
Inductive labeled_Vt : Type :=
| start : labeled_Vt
| lV : nat -> Vt -> labeled_Vt.

Definition label_var (label: nat)
                     (v: @var Vt): @var
                     labeled_Vt :=
  V (lV label v).
```

**Listing 18.** TODO

Construction of new grammar is quite simple. The function that constructs the union grammar takes a list of grammars, then, it (1) splits the list into head $[h]$ and tail $[tl]$, (2) labels $[length\;\; tl]$ to $h$, (3) adds a new rule from the start nonterminal of the union to the start nonterminal of the grammar $[h]$, finally (4) the function is recursively called on the tail $[tl]$ of the list.

#### 4.8.1 Proof of Languages Equivalence

In this section, we prove that function *grammar_union* constructs a correct grammar of union language indeed. Namely, we prove the following theorem.

```
Definition label_grammar label grammar := ...

Definition label_grammar_and_add_start_rule
            label
            grammar :=
  let '(st, gr) := grammar in
  (R (V start) [Vs (V (lV label st))])
     :: label_grammar label gr.

Fixpoint grammar_union
    (grammars : seq (@var Vt * (@grammar Tt Vt)))
      : @grammar
    Tt
    labeled_Vt :=
  match grammars with
  |  [] => []
  |  (g::t) =>
       label_grammar_and_add_start_rule
         (length t)
         g ++ (grammar_union t)
  end.
```

**Listing 19.** TODO

**Theorem 4.7.** *Let grammars be a sequence of pairs of starting nonterminals and grammars. Then for any word w, the fact that w belongs to language of union is equivalent to the fact that there exists a grammar $(st, gr) \in grammars$ such that w belongs to language generated by $(st, gr)$.*

```
Variable grammars: seq (var * grammar).

Theorem correct_union:
  forall word,
    language (grammar_union grammars)
      (V (start Vt)) (to_phrase word) <->
    exists s_l,
      language (snd s_l) (fst s_l)
        (to_phrase word) /\
      In s_l grammars.
```

**Listing 20.** Theorem on languages equivalence

**Proof of theorem 20.** Since the statement is formulated as an equivalence, we divide the proof into two parts.

1. If $w$ belongs to the union language, then $w$ belongs to one of the initial language.
   From an auxiliary lemma, we know that either (1) the phrase is equal to the starting nonterminal or (2) there exists a grammar $G$ from the grammars-list such that TODO Let us prove that this is the grammar we are interested in. Since we consider the word, it cannot be a starting non-terminal. So this might be only the second

case. According to another lemma, if the output doesn't start from the starting non-terminal, then it cannot appear in this derivation. So all the rules that use the starting non-terminal can be safely (for this derivation) removed from the grammar. There is a lemma that says, for a derivation that starts from a nonterminal labeled with $x$, can not contain any nonterminals with a label other than x. Therefore, for this derivation, one can ignore the rules with other(?) labels. These two grammars are identical, but one of them is labeled and the other is not. It is clear that if there exists a bijection between nonterminals the set of derivable words doesn't change.

2. If $w$ belongs to one of the initial language, then $w$ belongs to the union language.
   In this case, one explicitly specify the corresponding derivation in the union-grammar. The labeling function is arranged in such a way that knowing the place of a certain grammar in the list of grammars, one can calculate the exact number that will be assigned to this grammar as a label. After that proof can be finished in two steps.
   a. One need to apply the rule from the starting nonterminal of the union-grammar to the starting nonterminal of the initial grammar.
   b. One should use the fact that derivation in initial grammar and labeled grammar are equivalent.

### 4.9 Taking All Parts Together

**Theorem 4.8.** *For any two decidable types Terminal and Nonterminal for type of terminals and nonterminals correspondingly. If there exists bijection from Nonterminal to $\mathbb{N}$ and syntactic analysis in the sense of definition 14 is possible, then for any DFA dfa that accepts Terminal and any grammar G, there exists the grammar of intersection $G_{INT}$.*

Proof. Let $NG$ be the grammar in CNF obtained after applying the algorithm of [24]. Let *sdfas* be the list of DFAs with exactly one final state obtained after splitting *dfa*. Since we now have $NG$ in CNF and the list of DFAs with one state, we can compute a list of their intersections. I.e. we intersect each of the *sdfa* of the list with grammar $NG$. Next, we find the union of the languages.

Next, we divide the proof into two branches. We check whether the empty word is derivable in *dfa* and in grammar $G$. (1) If so, we add one more rule to the language of the union ($S \to \varepsilon$). (2) If not, we add nothing.

Now for the cases (1) and (2) we will prove that $G_{INT}$ is the grammar of the intersection indeed. That is, if a word is derivable in *dfa* and $G$, then it must also be derivable in $G_{INT}$. And vice versa.

For branch (1) we carry out the proof in 2 stages.

a Consider the case when $w$ is an empty word. By assumption, we already know that the empty word is derivable in the DFA and in the grammar. But we also know that we have added a rule from the starting terminal to the empty word to the grammar of intersection. So, if $w$ is an empty word it derivable in both cases.

b Let's now prove for the case when $w$ is a non-empty word. We consistently modify the premises and conclusion using theorems about equivalences. First we prove that the fact that $w$ derivable in $G$ and *dfa* implies the fact that $w$ is also derivable in $G_{INT}$. We can safely remove the rule $S \to \varepsilon$, since we apply unification only to grammars in CNF (any grammar of intersection is in CNF), the epsilon rule cannot be used anywhere except the initial step. We know, since the word is accepted by *dfa*, then there is a DFA with one final state *sdfa*, which also accepts this word. So, we can safely replace *dfa* with *sdfa*. For grammar $G$ there is an equivalent grammar $NG$ in CNF. And since word $w$ is not empty, we maintain equivalence. Let $INT$ be a grammar of the intersection of *sdfa* and $NG$. We can use theorem TODO to prove that it is a grammar of intersection of *sdfa* and $G$ indeed. But by the construction, such a grammar belongs to the union of languages $G_{INT}$. this finishes inclusion of $G$ and *sdfa* to $G_{INT}$. In the other direction: the fact that $w$ is derivable in $G_{INT}$ implies that $w$ is derivable in grammar $G$ and is accepted by DFA *dfa*.

Grammar $G_{INT}$ consists of a union of the empty language and list languages of the intersection of some sDFA and grammar in CNF. We can safely remove an empty grammar since $w$ is not an empty word and any grammar in the list of languages of intersection is in CNF. We know that since the word is accepted by $G_{INT}$ grammar, there is a grammar from the union of grammars in which this word is derivable. But this grammar is a grammar of intersection of some sDFA and grammar in CNF. But now we can use TODO in order to prove equivalence.

The second case is when the empty word is not derivable either in $G$ or in *dfa*. For an empty word, one needs to prove that it is not derivable in the grammar of the intersection. And indeed. None of the grammar from the union has the rule $S \to \varepsilon$. Next, one have to repeat what is discussed above, but without the additional steps about the empty language.

## 5 Related Works

There is a big number of works in mechanization of different parts of formal languages theory and certified implementations of parsing algorithms and algorithms for graph data base querying. These works use different tools, such s Coq, Agda, Isabelle/HOL, and aimed to different problems such as theory mechanization or executable algorithm certification. We discuss only small part which is close enough to the scope of this work.

### 5.1 Formal Language Theory in Coq

Huge amount of work was done by Ruy de Queiroz who formalize different parts of formal language theory, such as pumping lemma [40], context-free grammar simplification [37] and closure properties [39] in Coq. The work on closure properties contains mechanization of such properties as closure under union, Klenee star, but it does not contains mechanization of intersection with regular language. All these results are summarized in [38].

Gert Smolka et.al. also provide big set of works on regular and context-free languages formalization in Coq [13, 14, 24, 26]. The work [24] contains certified transformation of arbitrary context-free grammar to Chomsky normal form which is required for our proof of the Bar-Hillel theorem. Initially we hope to use these both parts because Bar-Hillel theorem is about both context-free and regular languages, and it was the reason to choose results of Gert Smolka as base for our work. But works on regular languages and on context-free languages are independent and we face with problems of reusing and integration and in current proof we use only results on context-free languages.

### 5.2 Formal Language Theory in Other Languages

In the parallel with works in Coq there exist works on formal languages mechanization in other languages and tools such as Agda or Isabelle/HOL.

First part is works of Denis Firsov who implements in Agda some parts of formal language theory and parsing algorithms. CYK parsing algorithm [16, 18] and Chomsky Normal Form [19], and some results on regular languages [17].

Another part is formal language theory mechanization in Isabelle/HOL [4, 6, 7] by Aditi Bartwall and Michael Norrish. This work contains basic definitions and big number of theoretical results, such as Chomsky normal form and Greibach normal form for context-free grammars. As an application of mechanized theory authors provide certified implementation of SLR parsing algorithm [5].

### 5.3 Certified Algorithms

Additionally we want to mention some works on certified applied algorithms based on formal language theory. Certification are required in different areas for various reasons and it is a reason to work on theory mechanization.

The first area where languages intersection may be applied is language constrained path querying in structured data (for example in graphs or XML). There exist works on certification of core of XQuery [12]. XQuery is a W3C standard for path querying in XML, extended for graph querying. Another result is a work on certified Regular Datalog querying in Coq [10]. Inspired by these results, our work may be a base for certified context-free path querying algorithm.

Another area which is growth fast is certified parsers and parser generators based on different algorithms for different language classes [9, 21, 25, 29].

## 6 Conclusion

We present mechanized in Coq proof of the Bar-Hillel theorem — the fundamental theorem on closure of context-free languages under intersection with regular set. By this we increase mechanized part of formal language theory and provide a base for reasoning about many applicative algorithms which are based on languages intersection. Also we generalize results of Gert Smolka and Jana Hofmann: generalized terminal alphabet. It makes previously existing results more flexible and ease for reusing. All results are published at GitHub and equipped with automatically generated documentation.

The first open question, and seams that very important question, is integration of our results with other results on formal languages theory mechanization in Coq. There are two independent sets of results in this area: works of Ruy de Queiroz and works of Gert Smolka. We use part of Smolka's results in our work, but even here we do not use existing results on regular languages. We think that theory mechanization should be unified and results should be generalized. We think that these and other related questions should be discussed in community.

One of direction of future research is mechanization of practical algorithms which are just implementation of Bar-Hillel theorem. For example, context-free path querying algorithm, based on CYK [23, 48] or even on GLL [43] parsing algorithm [20]. Final target here is certified algorithm for context-free constrained path querying for graph databases.

Yet another direction is mechanization of other problems on language intersection which can be useful for applications. For example, intersection of two context-free grammars one of which describes finite language [32, 34]. It may be useful for compressed data processing [28] or speech recognition [33, 35]. And, of course, all these works should be done on the common base of mechanized theoretical results.

## Acknowledgments

We are grateful to Ekaterina Verbitskaia for discussion, careful reading, and pointing out some mistakes. The research was supported by the Russian Science Foundation grant No. 18-11-00100 and by the grant from JetBrains Research.

## References

[1] Serge Abiteboul and Victor Vianu. 1999. Regular Path Queries with Constraints. *J. Comput. System Sci.* 58, 3 (1999), 428 – 452. https://doi.org/10.1006/jcss.1999.1627

[2] Faisal Alkhateeb. 2008. *Querying RDF(S) with Regular Expressions*. Theses. Université Joseph-Fourier - Grenoble I. https://tel.archives-ouvertes.fr/tel-00293206

[3] Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. *Sprachtypologie und Universalienforschung* 14 (1961), 143–172.
[4] Aditi Barthwal. 2010. *A formalisation of the theory of context-free languages in higher order logic.* Ph.D. Dissertation. College of Engineering & Computer Science, The Australian National University.
[5] Aditi Barthwal and Michael Norrish. 2009. Verified, Executable Parsing. In *Programming Languages and Systems*, Giuseppe Castagna (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 160–174.
[6] Aditi Barthwal and Michael Norrish. 2010. A formalisation of the normal forms of context-free grammars in HOL4. In *International Workshop on Computer Science Logic*. Springer, 95–109.
[7] Aditi Barthwal and Michael Norrish. 2010. Mechanisation of PDA and Grammar Equivalence for Context-Free Languages. In *Logic, Language, Information and Computation*, Anuj Dawar and Ruy de Queiroz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 125–135.
[8] Richard Beigel and William Gasarch. [n. d.]. A Proof that if $L = L_1 \cap L_2$ where $L_1$ is CFL and $L_2$ is Regular then $L$ is Context Free Which Does Not use PDAâĂŹs. ([n. d.]), 3.
[9] Jean-Philippe Bernardy and Patrik Jansson. 2016. Certified Context-Free Parsing: A formalisation of Valiant's Algorithm in Agda. *arXiv preprint arXiv:1601.07724* (2016).
[10] Angela Bonifati, Stefania Dumbrava, and Emilio Jesús Gallego Arias. 2018. Certified Graph View Maintenance with Regular Datalog. *arXiv preprint arXiv:1804.10565* (2018).
[11] Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Concurrency Theory*, Antoni Mazurkiewicz and Józef Winkowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 135–150.
[12] James Cheney and Christian Urban. 2011. Mechanizing the Metatheory of mini-XQuery. In *Certified Programs and Proofs*, Jean-Pierre Jouannaud and Zhong Shao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 280–295.
[13] Christian Doczkal, Jan-Oliver Kaiser, and Gert Smolka. 2013. A constructive theory of regular languages in Coq. In *International Conference on Certified Programs and Proofs*. Springer, 82–97.
[14] Christian Doczkal and Gert Smolka. 2018. Regular Language Representations in the Constructive Type Theory of Coq. *Journal of Automated Reasoning* 61, 1 (01 Jun 2018), 521–553. https://doi.org/10.1007/s10817-018-9460-x
[15] Michael Emmi and Rupak Majumdar. 2006. Decision Problems for the Verification of Real-Time Software. In *Hybrid Systems: Computation and Control*, João P. Hespanha and Ashish Tiwari (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 200–211.
[16] Denis Firsov. 2016. Certification of Context-Free Grammar Algorithms. (2016).
[17] Denis Firsov and Tarmo Uustalu. 2013. Certified Parsing of Regular Languages. In *Certified Programs and Proofs*, Georges Gonthier and Michael Norrish (Eds.). Springer International Publishing, Cham, 98–113.
[18] Denis Firsov and Tarmo Uustalu. 2014. Certified CYK parsing of context-free languages. *Journal of Logical and Algebraic Methods in Programming* 83, 5-6 (2014), 459–468.
[19] Denis Firsov and Tarmo Uustalu. 2015. Certified normalization of context-free grammars. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*. ACM, 167–174.
[20] Semyon Grigorev and Anastasiya Ragozina. 2016. Context-Free Path Querying with Structural Representation of Result. *arXiv preprint arXiv:1612.08872* (2016).
[21] Jason Gross and Adam Chlipala. 2015. Parsing Parses A Pearl of ( Dependently Typed ) Programming and Proof. 11.
[22] J. Hellings. 2014. Conjunctive context-free path queries. (2014).
[23] Jelle Hellings. 2015. Querying for Paths in Graphs using Context-Free Path Queries. *arXiv preprint arXiv:1502.02242* (2015).
[24] Jana Hofmann. 2016. Verified Algorithms for Context-Free Grammars in Coq.
[25] Jacques-Henri Jourdan, François Pottier, and Xavier Leroy. 2012. Validating LR(1) Parsers. In *Programming Languages and Systems*, Helmut Seidl (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 397–416.
[26] Jan-Oliver Kaiser. 2012. *Constructive Formalization of Regular Languages.* Ph.D. Dissertation. Saarland University.
[27] André Koschmieder and Ulf Leser. 2012. Regular path queries on large graphs. In *International Conference on Scientific and Statistical Database Management*. Springer, 177–194.
[28] Markus Lohrey. 2012. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology* 4 (2012), 241–299.
[29] Raul Lopes, Rodrigo Ribeiro, and Carlos Camarão. 2016. Certified Derivative-Based Parsing of Regular Expressions. In *Programming Languages*, Fernando Castor and Yu David Liu (Eds.). Springer International Publishing, Cham, 95–109.
[30] Yi Lu, Lei Shang, Xinwei Xie, and Jingling Xue. 2013. An incremental points-to analysis with CFL-reachability. In *International Conference on Compiler Construction*. Springer, 61–81.
[31] Sebastian Maneth and Fabian Peternek. 2018. Grammar-based graph compression. *Information Systems* 76 (2018), 19 – 45. https://doi.org/10.1016/j.is.2018.03.002
[32] Mark-Jan Nederhof and Giorgio Satta. 2002. Parsing non-recursive context-free grammars. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 112–119.
[33] Mark-Jan Nederhof and Giorgio Satta. 2002. Parsing Non-recursive Context-free Grammars. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 112–119. https://doi.org/10.3115/1073083.1073104
[34] Mark-Jan Nederhof and Giorgio Satta. 2004. The language intersection problem for non-recursive context-free grammars. *Information and Computation* 192, 2 (2004), 172–184.
[35] Mark-Jan Nederhof and Giorgio Satta. 2004. The language intersection problem for non-recursive context-free grammars. *Information and Computation* 192, 2 (2004), 172 – 184. https://doi.org/10.1016/j.ic.2004.03.004
[36] Polyvios Pratikakis, Jeffrey S Foster, and Michael Hicks. 2006. Existential label flow inference via CFL reachability. In *International Static Analysis Symposium*. Springer, 88–106.
[37] Marcus VM Ramos and Ruy JGB de Queiroz. 2015. Formalization of simplification for context-free grammars. *arXiv preprint arXiv:1509.02032* (2015).
[38] Marcus Vinícius Midena Ramos, Ruy JGB de Queiroz, Nelma Moreira, and José Carlos Bacelar Almeida. 2016. On the Formalization of Some Results of Context-Free Language Theory. In *International Workshop on Logic, Language, Information, and Computation*. Springer, 338–357.
[39] Marcus Vinícius Midena Ramos and Ruy J. G. B. de Queiroz. 2015. Formalization of closure properties for context-free grammars. *CoRR* abs/1506.03428 (2015). arXiv:1506.03428 http://arxiv.org/abs/1506.03428
[40] Marcus Vinícius Midena Ramos, Ruy J. G. B. de Queiroz, Nelma Moreira, and José Carlos Bacelar Almeida. 2015. Formalization of the pumping lemma for context-free languages. *CoRR* abs/1510.04748 (2015). arXiv:1510.04748 http://arxiv.org/abs/1510.04748
[41] Jakob Rehof and Manuel Fähndrich. 2001. Type-base flow analysis: from polymorphic subtyping to CFL-reachability. *ACM SIGPLAN Notices* 36, 3 (2001), 54–66.
[42] Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *Proceedings*

of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '95)*. ACM, New York, NY, USA, 49–61. https://doi.org/10.1145/199448.199462

[43] Elizabeth Scott and Adrian Johnstone. 2010. GLL parsing. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 177–189.

[44] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88, 2 (1991), 191 – 229. https://doi.org/10.1016/0304-3975(91)90374-B

[45] Dimitrios Vardoulakis and Olin Shivers. 2010. CFA2: A Context-free Approach to Control-flow Analysis. In *Proceedings of the 19th European Conference on Programming Languages and Systems (ESOP'10)*. Springer-Verlag, Berlin, Heidelberg, 570–589. https://doi.org/10.1007/978-3-642-11957-6_30

[46] Dacong Yan, Guoqing Xu, and Atanas Rountev. 2011. Demand-driven Context-sensitive Alias Analysis for Java. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11)*. ACM, New York, NY, USA, 155–165. https://doi.org/10.1145/2001420.2001440

[47] Qirun Zhang and Zhendong Su. 2017. Context-sensitive Data-dependence Analysis via Linear Conjunctive Language Reachability. *SIGPLAN Not.* 52, 1 (Jan. 2017), 344–358. https://doi.org/10.1145/3093333.3009848

[48] Xiaowang Zhang, Zhiyong Feng, Xin Wang, Guozheng Rao, and Wenrui Wu. 2016. Context-Free Path Queries on RDF Graphs. In *The Semantic Web – ISWC 2016*, Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil (Eds.). Springer International Publishing, Cham, 632–648.