Bar-Hillel Theorem Mechanization in Coq

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

Sergey Bozhko
Student
St.Petersburg State University
St.Petersburg, Russia
gkerfimf@gmail.com

Leyla Hatbullina
Position1
Department1
St.Petersburg Electrotechnical University
St.Petersburg, Saint-Petersburg
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 deep connection with different areas such as static code analysis, graph database querying, formal verification, and compressed data processing. One of the most frequent uses is to formulate a problem in terms of languages intersection. Fundamental result on languages intersection is the Bar-Hillel theorem on close of context-free languages over intersection with regular language. This theorem has a constructive proof and thus provide a formal base for algorithms for applications mentioned above and certified implementation of respective algorithms are required (for security reason for example). On the other hand, the Bar-Hillel theorem is a fundamental result of formal language theory and its mechanization is important itself. In this work we present mechanized proof of the Bar-Hillel theorem in Coq. We generalize results of Gert Smolka and Jana Hofmann and use them as base for our work.

Keywords Formal languages, Coq, Bar-Hillel, closure, intersection, regulalr language, context-free language

1 Introduction

Formal language theory has deep connection with different areas such as static code analysis [14, 18, 23, 24, 26–28], graph database querying [11–13, 29], formal verification [**?**], 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, 13]. 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 [24, 26, 27], graph analysis [10, 12, 29], context-free compressed data processing [15], and verification [**?**]. 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 [28], is undecidable while multiple context-free languages is closed under intersection with regular languages and emptiness problem for MCFLs is decidable [**?**]. 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

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

for future research in this area. 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 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 [**?**], and a number on certified parsers [**?**]. 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 [**?**]. 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 Smolka and !!! on context-free languages mechanization [**?**] 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 the CFL results of Smolka: terminals is abstract types....
- All code is published on GitHub: https://github.com/YaccConstructor/YC_in_Coq.

This work is organized as follows. In the section **??** 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 **??**. After that we describe our solution in the section **??**. 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 **??** and conclude with the discussion of the presented work and possible directions for future research in the section **??**.

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 $A_1, \ldots, A_n$ 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 $\{A_1 \ldots A_n\}$ where each A_i is recognized by a DFA with exactly one final state and $L_2 = A_1 \cup \ldots \cup A_n$
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 ε-rules.
5. Eliminate unit rules.

As far as Bar-Hillel theorem operates with arbitrary context-free languages, but the proof requires grammar in nCNF, it is necessary to implement a certified algorithm for conversion of arbitrary CFG to CNF. We wanted to reuse existing

proof of conversion of arbitrary context-free grammar to CNF. We chose Smolka version which proves conversion to CNF correctness in the following way.

CNF ! ! !

Listing 1. TODO

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 CNF grammar of the intersection of two languages. Final formulation of the obtained theorem can be found in the last subsection.

All code are published on GitHub [1].

4.1 Smolka's code generalization

In this section, we describe the exact steps taken to use the proof of TODO:Smolka's theorem in the proof of this article's theorem.

A substantial part of this proof relies on the work of TODO:Smolka. From this work(,?) 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, this proof had one major flaw that we needed to fix. One could define a terminal symbol as in inductive type over natural numbers[TODO].

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

Listing 2. TODO

That is how it was done in TODO:Smolka. 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 follows (???):

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

[1] https://github.com/YaccConstructor/YC_in_Coq

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

Listing 3. TODO

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 4. TODO, CHECK

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 Part ..: derivation and so on

In this section, we introduce the basic definitions used in the article.

We define a symbol is either a terminal or a nonterminal.

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

Listing 5. TODO

Next we define a word and a phrase as lists of terminals and symbols respectively.

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

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

Listing 6. TODO

introduce a predicate that defines whenever a phrase consists of only terminals. And if so, the phrase it can be safely converted to the corresponding word.

We inheriting the definition of CFG from [Smplka] paper. Rule is defined as a pair of a nonterminal and a list of symbols. Grammar is a list of rules.

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

Definition grammar := list rule.
```

Listing 7. TODO

An important step towards the definition of a language (?) governed (formed?)(?!) by a grammar is the definition of derivability. 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 8. TODO

Proof of TODO requires grammar to be in CNF. We used statement that every grammar in convertible into CNF from TODO:Smolka work.

... ...

4.3 General scheme of the proof

General scheme of our proof is based on constructive proof presented by [**?**]. 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 (TODO: del this?)
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 Part one: trivial case

(TODO: del?)

4.5 Part two: regular language 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 is well-defined(?).

We assume that regular language by definition 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.

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

Listing 9. TODO

Next we define a function that would evaluate 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 $[final\_state\_sw]$ ends in 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. The definitions of "accepts" and "dfa_language" vary slightly.

Similarly, we can define functions $s\_accepts$ and $s\_dfa\_language$ for sDFA. Since in this case, there is only one final state, to

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

Listing 11. TODO

define function *s_accepts* it is enough to check the state in which the automaton stopped with the finite state. The function *s_dfa_language* repeats the function *dfa_language*, except that the function must use *s_accepts* instead of accepts.

Now we canto define a function that converts an ordinary DFA into a set of DFAs with exactly one final state. 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 current 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. TODO

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

Listing 13. TODO

We prove theorem that the function of splitting preserves the 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 Part ..: 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 canâĂŹt simply apply induction over derivation structure. To tackle this problem we create custom induction principle for grammars in CNF.

As one might notice, TODO

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: add picture
TODO: add Lemma derivability_backward_step.

(TODO: lemma) More formally: 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 exist(TODO:s) 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 $w_1 + +w_2 = w$.

(TODO: fix) **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 $N- > prhinG$. 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| >= 2). On the other hand, if there is an intermediate phrase that was obtained by applying a rule, then the phrase has form N_1N_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: lemma) Let G be a grammar in CNF. And P be a predicate on nonterminals and phrases (i.e. $P : var \rightarrow phrase \rightarrow Prop$). Let's also assume that the following two hypotheses are satisfied: (1) for every terminal production (i.e. in the form $N \rightarrow a$) of grammar G, $P(r, [Tsr])$ 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 $P(N, w)$.

(TODO: fix) **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 Part ..: intersection

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 Function

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

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(in?) G.

Since G is a grammar in CNF, it has only two type of productions: (1) $N \rightarrow a$ and (2) $N \rightarrow N_1N_2$, where N, N_1, N_2 are nonterminals and a is a terminal.

For every production $N \rightarrow N_1N_2$ in G we generate a set of productions of the form $(from, N, to) \rightarrow (from, N_1, m)(m, N_2, to)$ 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 14. TODO

For every production of the form $N \rightarrow a$ we add a set of productions $(from, N, (dfa\_step(from, a))) \rightarrow a$ 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 15. TODO

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.

Note that at this point we do not have any manipulations with starting rules. Nevertheless(?), the hypothesis of the

```
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 16. TODO

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

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.

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 4.2. *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. TODO. In another case, it would be logical to use induction on the derivation structure in 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.

Let's apply chomsky induction principle with

$$P := funrphr => \forall(next : dfa\_rule)(fromto : DfaState),\ final\_statenextfrom(to_w ordphr) = to -> der(convert\_rulesGnext)(V(from, r, to))phr.$$

We will get the bla-bla, bla-bla, bla-bla-bla

Since a language is just a bla-bla-bla, we use the lemma above to prove bla-bla-bla

4.8 Part ..: union

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 nonintersecting 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 17. 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 Equivalence proof

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

Theorem 4.3. *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).*

Proof of theorem 4.3. 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

```
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 18. TODO

```
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 19. TODO

of the initial language.
2. If w belongs to one of the initial language, then w belongs to the union language.
The fact that $(st, grammar) \in grammars$ implies that there exist $gr1$ and $gr2$ such that: $gr1 + +(st, grammar) :: gr2 = grammars$.

Proof. This proved through induction over l. assume l = h :: t, then either word accepted by h or tail. If word accepted by h If word accepted by l. We just proving that adding one more language to union preserves word derivability. Which is equivalent to proving that adding new rules to grammar preserves word derivability

2. If we have derivation for some word in new grammar lanager we can provide derivate in for some language from union.

Proof. Here we converting derivability procedure for language union into derivability procedure of one of language. Then we proving that in derivation we can use rules from only one language at time. Finally we converting derivation by simple relabelling back all nonterminals.

4.9 Part N: taking all parts together

TODO: add some text

Theorem 4.4. *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 TODO is possible, then for any DFA dfa which accepts Terminal and any grammar G, there exists the grammar of intersection $L(DFA)$ and G.*

Proof.

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.

Huge amount of work was done by Ruy de Queiroz who formalize different parts of formal language theory, such as pumping lemma [22], context-free grammar simplification [19] and closure properties (union, Klenee star, but without intersection with regular language) [21] in Coq. All these results are summarized in [20].

Another part of formal languages formalization in Coq by Gert Smolka et.al. [5, 6]. Regular and Context-Free languages.

Also exist some works by Denis Firsov who implements in Agda some parts of formal language theory and parsing algorithms: CYK [8], Chomsky Normal Form [9], etc [7].

Certified parsers and parser generators.

In HOL4.

In the graph database area there exist work on certified Regular Datalog querying in Coq [4]. This work is inspiration for certified CFPQs.

6 Conclusion

We present mechanized in Coq proof of Bar-Hillel theorem on closure of context-free languages under intersection with regular one. 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 Smolka and !!! : 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.

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 [**?** **?**] or even on GLL [25] parsing algorithm [10]. Final target here is certified algorithm for context-free constrained path querying for graph database.

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 [16, 17]. It may be useful for compressed data processing [**?**].

Finally, thmthnk about integration with other results, generalization and zero-costs reusing (Firsov).

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] Angela Bonifati, Stefania Dumbrava, and Emilio Jesús Gallego Arias. 2018. Certified Graph View Maintenance with Regular Datalog. *arXiv preprint arXiv:1804.10565* (2018).

[5] 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.

[6] Christian Doczkal and Gert Smolkab. 2017. Regular Language Representations in the Constructive Type Theory of Coq. (2017).

[7] Denis Firsov. 2016. Certification of Context-Free Grammar Algorithms. (2016).

[8] 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.

[9] 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.

[10] Semyon Grigorev and Anastasiya Ragozina. 2016. Context-Free Path Querying with Structural Representation of Result. *arXiv preprint arXiv:1612.08872* (2016).

[11] J. Hellings. 2014. Conjunctive context-free path queries. (2014).

[12] Jelle Hellings. 2015. Querying for Paths in Graphs using Context-Free Path Queries. *arXiv preprint arXiv:1502.02242* (2015).

[13] André Koschmieder and Ulf Leser. 2012. Regular path queries on large graphs. In *International Conference on Scientific and Statistical Database Management*. Springer, 177–194.

[14] 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.

[15] 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

[16] 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.

[17] 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.

[18] Polyvios Pratikakis, Jeffrey S Foster, and Michael Hicks. 2006. Existential label flow inference via CFL reachability. In *International Static Analysis Symposium*. Springer, 88–106.

[19] Marcus VM Ramos and Ruy JGB de Queiroz. 2015. Formalization of simplification for context-free grammars. *arXiv preprint arXiv:1509.02032* (2015).

[20] 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.

[21] 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

[22] 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

[23] Jakob Rehof and Manuel Fähndrich. 2001. Type-base flow analysis: from polymorphic subtyping to CFL-reachability. *ACM SIGPLAN Notices* 36, 3 (2001), 54–66.

[24] 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

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

[26] 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

[27] 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

[28] 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

[29] 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.