Contribution Title*

First Author[1[0000−1111−2222−3333]], Second Author[2,3[1111−2222−3333−4444]], and Third Author[3[2222−−3333−4444−5555]]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

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: First keyword · Second keyword · Another keyword.

1 Introduction

Formal language theory has a deep connection with different areas such as static code analysis [39, 42, 43, 38, 29, **?**, **?**], graph database querying [21, **?**, 45, **?**], formal verification [14, **?**], 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 the 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.

* Supported by organization x.

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, 26, 2]. 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 [39, 42, 43], graph analysis [22, 45, 19], context-free compressed data processing [30], and other areas. The constructive intersection representation in these applications is helpful for further analysis.

The intersection of some classes of languages is not generally decidable. For example, the intersection of the linear conjunctive and the regular languages, used in the static code analysis [44], is undecidable while multiple context-free languages (MCFL) is closed under intersection with regular languages and emptiness problem for MCFLs is decidable [41]. 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 of pumping lemma for MCFG shows the necessity to 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 significant number of papers focusing on regular expressions mechanization and certification [16], and a number on certified parsers [5, 17, 20]. 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 [18, 36, 4]. Also, it is desirable to have a base to reason about parsing algorithms and other problems of 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 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 the 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 the 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 [23] 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: alphabets (nonterminals and terminals) in context-free grammar definition are changed from Nat to generic, and all the code affected by this change is also modified to work with the updated definition.
- All code is published on GitHub: `https://github.com/YaccConstructor/YC_in_Coq`.

This work is organized as follows. In 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 section 4. This description is split into steps with respect to provided sketch and contains basic definitions, Smolka's results generalization, handling of trivial cases, and steps summarization as final proof. Finally, we discuss related works in section 5 and conclude with the discussion of the presented work and possible directions for future research in section 6.

2 Bar-Hillel Theorem

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

Lemma 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. *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 precisely one final state.*

Theorem 1 (Bar-Hillel). *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 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 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 precisely 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: $L(G_{\text{CNF}}) \cap A_i$
4. Finally, we join them together with the operation of the union

3 The Chomsky Normal Form

The important aspect of our proof is that any context-free language can be described with a 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 conversion of an arbitrary context-free (CF) grammar to CNF.

Definition 1 (Chomsky Normal Form). *A context-free grammar is in CNF if:*

- *the start nonterminal does not occur in the right-hand side of any rule,*
- *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 nonterminal from the right-hand sides of the rules.
2. Eliminate rules with nonsolitary 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 and the selected proof requires grammar in CNF, it is necessary to implement a certified algorithm for the conversion of an arbitrary CF grammar to CNF. We wanted to reuse existing mechanized proof for the conversion. We chose the one provided in Smolka's work and discussed it in the context of our work in section 4.1.

4 Bar-Hillel Theorem Mechanization in Coq

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

The overall goal of this section is to provide a step-by-step algorithm which constructs the context-free grammar of the intersection of two languages. The final formulation of the theorem can be found in the last subsection.

4.1 Smolka's Results Generalization

A substantial part of this proof relies on the work of Gert Smolka and Jana Hofmann [23][4] from which many definitions and theorems were taken. Namely,

[4] 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.

the definition of a grammar, the definitions of a derivation in grammar, some auxiliary lemmas about the decidability of properties of grammar and derivation. We also use the theorem that states that there always exists the transformation from a context-free grammar to a grammar in Chomsky Normal Form.

However, the proof of the existence of the transformation to CNF had one major flaw that we needed to fix: the representation of terminals and nonterminals. In the definition of the grammar, a terminal is an element of the set of terminals—the alphabet of terminals. It is sufficient to represent each terminal by a unique natural number—conceptually, the index of the terminal in the alphabet. This is how it is done in [23]:

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

The same observation is correct for nonterminals. Sometimes it is useful when the alphabet of nonterminals bears some structure. For the purposes of our proof, nonterminals are better represented as triples. We decided to make terminals and nonterminals to be polymorphic over the alphabet. We are only concerned that the representation of symbols is a type with decidable relation of equality. Namely, let Tt and Vt be such types, then we can define the types of terminals and nonterminals over Tt and Vt respectively as presented in Lst. 1.

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

Listing 1: The new polymorphic definitions of terminals and nonterminals

Fortunately, the proof of Smolka has a clear structure, and there was only one aspect of the proof where the use of natural numbers was essential. The grammar transformation which eliminates long rules creates new nonterminals. In the original proof, it was done by taking the maximum of the nonterminals included in the grammar. It is not possible to use the same mechanism for an arbitrary type.

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

Another difficulty is that the original work defines grammar as a list of rules and does not specify the start nonterminal. Thus, in order to define the language described by a grammar, one needs to specify the start terminal explicitly. It leads to the fact that the theorem about the equivalence of a CF grammar and the corresponding CNF grammar is not formulated in the most general way, namely, it guarantees equivalence only for non-empty words.

The predicate "is grammar in CNF" as defined in [23] does not treat the case when the empty word is in the language. That is, with respect to the definition in [23], a grammar cannot have epsilon rules at all.

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

Listing 2: The equivalence of languages specified by the context-free grammar and the transformed grammar in CNF

The question of whether the empty word is derivable is decidable for both the CF grammar and the DFA. Therefore, there is no need to adjust the definition of the grammar (and subsequently all proofs). It is possible just to consider two cases (1) when the empty word is derivable in the grammar (and acceptable by DFA) and (2) when the empty word is not derivable. We use this feature of CNF definition to prove some of the lemmas presented in this paper.

4.2 Basic Definitions

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

We define a symbol as either a terminal or a nonterminal (Lst.3).

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

Listing 3: Definition of symbol (union of terminals and nonterminals)

Next, we define a word and a phrase as lists of terminals and symbols respectively (Lst.4). One can think that word is an element of the language defined by the grammar, and a phrase is an intermediate result of derivation. Also, a right-hand side of any derivation rule is a phrase.

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

Listing 4: Definitions of word and phrase.

The notion of nonterminal does not make sense for DFA, but in order to construct the derivation in grammar, we need to use nonterminals in intermediate states. For phrases, we introduce a predicate that defines whenever a phrase consists of only terminals. If it is the case, the phrase can be safely converted to the word.

We inherit the definition of CFG from [23]. The rule is defined as a pair of a nonterminal and a phrase, and a grammar is a list of rules (Lst.5). Note, that this definition of a grammar does not include the start nonterminal, and thus does not specify the language by itself.

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

Definition grammar := list rule.
```

Listing 5: Context-free rule and grammar definition

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

Also, we use the proof of the fact that every grammar in convertible into CNF from [23] because this fact is important for our proof.

We define the language as follows. We say that a phrase (not a word) w belongs to the language generated by a grammar G from a nonterminal A, if w is derivable from nonterminal A in grammar G and w consists only of terminals.

4.3 General Scheme of the Proof

A general scheme of our proof is based on the constructive proof presented in [8]. This proof does not use push-down automata explicitly and operates with grammars, so it is pretty simple to mechanize it. Overall, we will adhere to the following plan.

1. We consider the trivial case when DFA has no states.
2. We state that every CF language can be converted to CNF.
3. We show that every DFA can be presented as a union of DFAs with the single final state.
4. We construct an intersection of grammar in CNF with DFA with one final state.
5. We prove that the union of CF languages is CF language.
6. We putting everything mentioned above together. Additionally, we handle the fact that the initial CF language may contain the ε word. By the definition which we reuse from [23], the grammar in CNF has no epsilon rules, but we still need to consider the case when the empty word is derivable in the grammar. We postpone this consideration to the last step. Only one of the following statements is true.
   (a) $\varepsilon \in L(G)$ and $\varepsilon \in L(dfa)$
   (b) $\neg\varepsilon \in L(G)$ or $\neg\varepsilon \in L(dfa)$

   So, we should just check emptiness of languages as a separated case.

4.4 Trivial Cases

First, we consider the case when the number of the DFA states is zero. In this case, we immediately derive a contradiction. By definition, any DFA has an initial state. It means that there is at least one state, which contradicts the assumption that the number of states is zero.

It is worth to mention, that in the proof [8] cases when the empty word is derivable in the grammar or a DFA specifies the empty language are discarded as trivial. It is assumed that one can carry out themselves the proof for these cases. In our proof, we include the trivial cases in the corresponding theorems.

4.5 Regular Languages and Automata

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

We assume that a regular language is described by a DFA. We do not impose any restrictions on the type of input symbols and the number of states in DFA. Thus, the DFA is a 5-tuple: (1) a type of states, (2) a type of input symbols, (3) a start state, (4) a transition function, and (5) a list of final states (Lst.**??**).

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

We say that the automaton accepts a word w being in state s if the function (*final_state* s w) returns a final state. Finally, we say that an automaton accepts a word w, if the DFA starts from the initial state and stops in a final state.

The definition of the DFA with exactly one final state differs from the definition of an ordinary DFA in that the list of final states is replaced by one final state. Related definitions such as *accepts* and *dfa_language* are slightly modified.

We define functions *s_accepts* and *s_dfa_language* for DFA with one final state in the same fashion. In the function *s_accepts*, it is enough to check for equality the state in which the automaton stopped with the finite state. Function *s_dfa_language* is the same as *dfa_language* except for that the function for a DFA with one final state should use *s_accepts* instead of *accepts*.

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

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

Theorem 2. *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 stops in a, or in one of the states from *old_finals*. If *dfa* stops in a, then we simply choose the 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* stops in one of the *old_finals* states, then we can apply the induction hypothesis.
2. Similarly in 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 from *split_dfa(dfa* accepts w, and its final state either is equal to a, or is in *old_finals*. If the final state is equal to a, then *dfa* also stops in the state a. On the other hand, if the final state is in *old_finals*, then we apply the induction hypothesis.

4.6 Chomsky Induction

Many statements about properties of words in a language can be proved by induction over derivation structure. Although a one can get a phrase as an intermediate step of derivation, DFA only works on words, so we can not simply apply induction over the derivation structure. To tackle this problem, we created a custom induction principle for grammars in CNF.

The current definition of derivability does not imply the ability to "reverse" the derivation back. That is, nothing about the rules of the grammar or properties of derivation follows from the fact that a phrase w is derived from a nonterminal A in a grammar G. Because of this, we introduce an additional assumption on derivations that is similar to the syntactic analysis of words. Namely, we assume that if the phrase w is derived from the nonterminal A in grammar G, then either there is a rule $A \to w \in G$ or there is a rule $A \to rhs \in G$ and w is derivable from rhs.

Any word derivable from a nonterminal A in the grammar in CNF is either a solitary terminal or can be split into two parts, each of which is derived from nonterminals B and C, when the derivation starts with the rule $A \to BC$ Note that if we naively take a step back, we can get a nonterminal which derives some substring in the middle of the word. Such a situation does not make any sense for DFA.

By using induction, we always deal with subtrees that describe a substring of the word. Graphic representation of the idea of the Chomsky induction fore case described above one can find in figure **??**.

To put it more formally:

Lemma 3. *Let G be a grammar in CNF. Consider an arbitrary nonterminal $N \in G$ and phrase which consists only of terminals w. If w is derivable from N and $|w| \geq 2$, then there exists two nonterminals N_1, N_2 and two phrases w_1, w_2 such that: $N \to N_1 N_2 \in G$, $der(G, N_1, w_1)$, $der(G, N_2, w_2)$, $|w_1| \geq 1$, $|w_2| \geq 1$ and $w_1 + +w_2 = w$.*

Proof. The proof heavily uses the fact that grammar G is in CNF in the sense of definition given in the work of Gert Smolka and Jana Hofmann. We apply the hypothesis "syntactic analysis is possible". After their application, we get the fact that word w is either a 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 nonterminal N such that $N \to prh \in G$.

The first case is proved by contradiction since the grammar is in CNF and there might be only a single terminal in a 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 is also derived by a rule in normal form. Finally, now we need to prove that both of this nonterminals has a non-empty contribution to the word w. This is also true since it is impossible to derive the empty word in CNF grammar (see 4.1).

Lemma 4. *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 4.1).

Induction step: $|w| \leq n+1$. This fact is equivalent to the following: $|w| = n+1$ or $|w| < n$. In the 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 $N \to w$, which means one can apply assumption (1). If the word is longer than 1, then we apply the previous lemma and conclude that $\exists w_1\ w_2, w = w_1 + +w_2$. After that, one needs to apply assumption (2). All the generated subgoals are also guaranteed by the lemma 3. For shorter words w_1 and w_2, we apply the induction hypothesis.

4.7 Intersection of CFG and Automaton

Since we already have lemmas about the transformation of a grammar to CNF and the transformation of a DFA to a DFA with exactly one state, further we assume that we only deal with (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 a grammar for an intersection of the languages.

Construction of Intersection We present the adaptation of the algorithm given in [8].

Let G_{INT} be the grammar of intersection. In G_{INT}, nonterminals are presented as triples $(\mathit{from} \times \mathit{var} \times \mathit{to})$ 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 types 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

$$(\mathit{from}, N, \mathit{to}) \to (\mathit{from}, N_1, m)(m, N_2, \mathit{to})$$

where: *from*, m, *to* enumerate all *dfa* states.

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

$$(\mathit{from}, N, (\mathit{dfa\_step}(\mathit{from}, a))) \to a$$

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

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 conduct any manipulations with the start nonterminal. Nevertheless, the hypothesis of the uniqueness of the final state of the DFA helps to define the start nonterminal of the grammar of intersection unambiguously. The start nonterminal for the intersection grammar is the following nonterminal: *(start, S, final)* where: *start*—the start state of DFA, S—the start nonterminal of the initial grammar, and *final*—the final state of DFA. Without the assumption that the DFA has only one final state it is not clear how to unequivocally define the start nonterminal over the alphabet of triples.

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

In the interest of clarity of exposition, we skip some auxiliary lemmas and facts like that we can get the initial grammar from the grammar of intersection by projecting the triples back to the corresponding terminals/nonterminals. Also note that grammar remains in CNF after the conversion, since the transformation of rules does not change the structure of them, but only replaces their terminals and nonterminals with attributed ones.

Next, we prove the following lemmas. First, the fact that a word can be derived in the initial grammar and is accepted by *s_dfa* implies it can be derived in the grammar of the intersection. And the other way around, the fact that a word can be derived in the grammar of the intersection implies that it is derived in the initial grammar and is accepted by *s_dfa*.

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

Theorem 3. *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 it is possible to derive w from r and the s_dfa starting from the state from finishes in the state to after consuming the word w, then the word w is also derivable in grammar (convert_rules G next) from the nonterminal (V (from, r, to)).*

Proof. It is tempting to use induction on the derivation structure in grammar G. But we cannot do it, otherwise, we get a phrase (list of terminals and nonterminals) instead of a word. Therefore we should use another way to employ induction, and for the grammar in Chomsky normal form it is possible, as we show in section 4.6. 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:

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

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 nonterminals. 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 nonterminal 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 nonterminal 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 accepted by the automaton and is derivable in the initial grammar".

We start with the DFA.

Theorem 4. *Let from and to be states of the automaton, var be an arbitrary nonterminal of G. We prove that if a word w is derived from the nonterminal (from, var, to) in the grammar (convert_rules G), then the automaton starting from the state from accepts the word w and stops in the state to.*

Proof. We use the principle of Chomsky Induction. We apply the induction with the following parameter P:

$$P := \lambda \ \mathit{tr\_non}\ \mathit{phr} \Rightarrow$$
$$\mathit{final\_state}\ \mathit{next}\ \ (fst_3\ \mathit{tr\_non})\ (\mathit{to\_word}\ \ \mathit{phr}) = thi_3\ r.$$

Note that in this case, one need to use projections from attributed nonterminals to simple nonterminals. Induction is carried out in the grammar over triples, but the property operates with the automaton. However, this property can also be expressed in terms of nonterminals-triples. P is the statement we want to prove. After applying the induction principle, it remains to prove only the fidelity of the assumptions. First of all, since w is a word, converting it to a phrase does not add any nonterminals. Next, one needs 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 symbols in the rules to the symbols over the alphabet of triples. Finally, one needs to show that both assumptions of the induction principle hold. In this case, it would be:

$$(N \to a) \in \mathit{convert\_rules}\ G \to$$
$$\mathit{next}(\ fst_3\ \ (\mathit{unVar}\ \ N))\ a = thi_3\ (\mathit{unVar}\ N)$$

and

$$(N \to [N_1; N_1]) \in \mathit{convert\_rules}\ G \to$$
$$\dots \to$$
$$\mathit{final\_state}\ \mathit{next}\ (fst_3(\mathit{unVar}\ N))(\mathit{to\_word}(w_1 + +w_2)) =$$
$$thi_3(\mathit{unVar}\ N)$$

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

$$(N \to [N_1; N_2]) \in \mathit{convert\_rules}\ G$$

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

Next, we prove the similar theorem for the grammar.

Theorem 5. *Let from and to be the states of the automaton, let var be an arbitrary nonterminal of grammar G. We prove that if a word w is derivable from the nonterminal (from, var, to) in the grammar (convert_rules G), then w is also derivable in the grammar G from the nonterminal var.*

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

$$P := \lambda\ r\ \mathit{phr} \Rightarrow \mathit{der}(G,\ \mathit{snd}_3\ \ (\mathit{unVar}\ r),\ \mathit{phr}).$$

Note that the induction is carried out in the grammar over triples, but the property is 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*. 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.

In the end, one needs to combine both theorems to get a full equivalence. By this, the correctness of the intersection is proved.

4.8 Union of Languages

During the previous step, we constructed a list of context-free grammars. In this section, we provide a function which constructs a grammar for the union of the languages.

First, we need to make sure the sets of nonterminals for each of the grammars under consideration have empty intersections. To achieve this, we label nonterminals. Each grammar of the union receives a unique ID number and all nonterminals within one grammar will have the same ID as the grammar. In addition, it is necessary to introduce a new start nonterminal of the union.

The function that constructs the union grammar (Lst.9) 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.

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

Theorem 6. *Let grammars be a sequence of pairs of starting nonterminals and grammars. Then for any word w, the fact that w belongs to the language of the union is equivalent to the fact that there exists a grammar $(st, gr) \in$ grammars such that w belongs to the language generated by (st, gr).*

Proof of theorem 6. 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 languages.
   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 the phrase is derivable from the labeled starting nonterminal of grammar G. Let us prove that this is the grammar we are interested in. Since we consider the word, it cannot be the start nonterminal. So this might be only the second case. According to another lemma, if the output does not start from the start nonterminal, then it cannot appear in this derivation. So

all the rules that use the start nonterminal can be safely (for this derivation) removed from the grammar. There is a lemma that says, that a derivation that starts from a nonterminal labeled by x cannot 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 does not change.

2. If w belongs to one of the initial languages, 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 needs to apply the rule from the start nonterminal of the union-grammar to the start nonterminal of the initial grammar.
   (b) One should use the fact that derivation in the initial grammar and the labeled grammar are equivalent.

4.9 Putting All Parts Together

Now we can put all previously described lemmas together to prove the main statement of this paper.

Theorem 7. *For any two decidable types Tt and Nt for types of terminals and nonterminals correspondingly. If there exists a bijection from Nt to $\mathbb{N}$ and syntactic analysis in the sense of definition* **??** *is possible, then for any DFA dfa that define language over Tt and any context-free grammar G, there exists the context-free grammar G_{INT} , such that $L(G_{INT}) = L(G) \cap L(dfa)$.*

Proof. Let NG be the grammar in CNF obtained after applying the algorithm of [23]. 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 the 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 accepted by the *dfa* and is derived in the 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 prove that G_{INT} is the grammar of the intersection. That is, if a word is accepted by the *dfa* and is derivable in 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 accepted by the DFA and can be derived in the grammar. But we also know that we have added a rule from the start nonterminal to the empty word to the grammar of the intersection. So, if w is an empty word it derivable in both cases.

b Let's now prove for the case when w is non-empty. We consistently modify the premises and the conclusion using theorems about equivalences. First, we prove that the fact that w derivable in G and is accepted by *dfa* implies the fact that w is also derivable in G_{INT}. We can safely remove the rule $S \to \varepsilon$, since we apply the unification only to grammars in CNF (any grammar of intersection is in CNF), the epsilon rule cannot be used anywhere except for the initial step. 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. Since word w is not empty, we maintain equivalence. Let INT be a grammar of the intersection of *sdfa* and NG. We can use theorems from section 4.7 to prove that it is a grammar of the intersection of *sdfa* and G. 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 the grammar G and is accepted by the DFA *dfa*.

Grammar G_{INT} consists of a union of the empty language and list languages of the intersection of some DFA with one final state 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 the 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 DFA with one final state and grammar in CNF. Now we can use theorems from section 4.7 to prove equivalence.

The second case is when the empty word is not derivable in G and is not accepted by *dfa*. For an empty word, one needs to prove that it is not derivable in the grammar of the intersection. None of the grammars from the union has the rule $S \to \varepsilon$. Next, one has 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 the mechanization of different parts of formal languages theory and certified implementations of parsing algorithms and algorithms for graph database querying. These works use various tools, such as Coq, Agda, Isabelle/HOL, and are aimed at different problems such as the theory mechanization or executable algorithm certification. We discuss only a small part which is close enough to the scope of this work.

5.1 Formal Language Theory in Coq

The massive amount of work was done by Ruy de Queiroz who formalized different parts of formal language theory, such as pumping lemma [35], context-free grammar simplification [37] and closure properties [34] in Coq. The work on closure properties contains mechanization of such properties as closure under union, Kleene star, but it does not contain mechanization of the intersection with a regular language. All these results are summarized in [36].

Gert Smolka et al. also provide a big set of works on regular and context-free languages formalization in Coq [13, 12, 25, 23]. The paper [23] describes the certified transformation of an arbitrary context-free grammar to the Chomsky normal form which is required for our proof of the Bar-Hillel theorem. Initially, we hoped to reuse these both parts because the Bar-Hillel theorem is about both context-free and regular languages, and it was the reason to choose results of Gert Smolka as the base for our work. But the works on regular and on context-free languages are independent, and we are faced with the problems of reusing and integration, so in the 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.

Firstly, there are works of Denis Firsov who implements some parts of the formal language theory and parsing algorithms in Agda. In particular, Firsov implements CYK parsing algorithm [17, 15] and Chomsky Normal Form [18], and some other results on regular languages [16].

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

5.3 Certified Algorithms

Additionally, we want to mention some works on certified applied algorithms based on the formal language theory. Certification is required in different areas, 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 the core of XQuery [11]. XQuery is a W3C standard for path querying in XML, extended for graph querying. Another result is 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 grows fast is certified parsers and parser generators based on different algorithms for different language classes [24, 9, 28, 20].

6 Conclusion

We present mechanized in Coq proof of the Bar-Hillel theorem, the fundamental theorem on the closure of context-free languages under intersection with the 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. We generalize the results of Gert Smolka and Jana Hofmann: the definition of the terminal and nonterminal alphabets in context-free grammar were made generic, and all related definitions and theorems were adjusted to work with the updated definition. It makes previously existing results more flexible and eases reusing. All results are published at GitHub and are equipped with automatically generated documentation.

The first open question is the 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 believe that theory mechanization should be unified and results should be generalized. We think that these and other related questions should be discussed in the community.

One direction for future research is mechanization of practical algorithms which are just implementation of the Bar-Hillel theorem. For example, context-free path querying algorithm, based on CYK [22, 45] or even on GLL [40] parsing algorithm [19]. Final target here is the certified algorithm for context-free constrained path querying for graph databases.

Another direction is mechanization of other problems on language intersection which can be useful for applications. For example, the intersection of two context-free grammars one of which describes finite language [31, **?**]. It may be useful for compressed data processing [27] or speech recognition [32, 33]. And we believe all these works should share the common base of mechanized theoretical results.

References

1. Abiteboul, S., Vianu, V.: Regular path queries with constraints. Journal of Computer and System Sciences **58**(3), 428 – 452 (1999). https://doi.org/https://doi.org/10.1006/jcss.1999.1627, `http://www.sciencedirect.com/science/article/pii/S0022000099916276`
2. Alkhateeb, F.: Querying RDF(S) with Regular Expressions. Theses, Université Joseph-Fourier - Grenoble I (Jun 2008), `https://tel.archives-ouvertes.fr/tel-00293206`
3. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Sprachtypologie und Universalienforschung **14**, 143–172 (1961)
4. Barthwal, A.: A formalisation of the theory of context-free languages in higher order logic. Ph.D. thesis, College of Engineering & Computer Science, The Australian National University (12 2010)

5. Barthwal, A., Norrish, M.: Verified, executable parsing. In: Castagna, G. (ed.) Programming Languages and Systems. pp. 160–174. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
6. Barthwal, A., Norrish, M.: A formalisation of the normal forms of context-free grammars in hol4. In: International Workshop on Computer Science Logic. pp. 95–109. Springer (2010)
7. Barthwal, A., Norrish, M.: Mechanisation of pda and grammar equivalence for context-free languages. In: Dawar, A., de Queiroz, R. (eds.) Logic, Language, Information and Computation. pp. 125–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
8. Beigel, R., Gasarch, W.: 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 pdas
9. Bernardy, J.P., Jansson, P.: Certified context-free parsing: A formalisation of valiant's algorithm in agda. arXiv preprint arXiv:1601.07724 (2016)
10. Bonifati, A., Dumbrava, S., Arias, E.J.G.: Certified graph view maintenance with regular datalog. arXiv preprint arXiv:1804.10565 (2018)
11. Cheney, J., Urban, C.: Mechanizing the metatheory of mini-xquery. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. pp. 280–295. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
12. Doczkal, C., Kaiser, J.O., Smolka, G.: A constructive theory of regular languages in coq. In: International Conference on Certified Programs and Proofs. pp. 82–97. Springer (2013)
13. Doczkal, C., Smolka, G.: Regular language representations in the constructive type theory of coq. Journal of Automated Reasoning **61**(1), 521–553 (Jun 2018). https://doi.org/10.1007/s10817-018-9460-x, `https://doi.org/10.1007/s10817-018-9460-x`
14. Emmi, M., Majumdar, R.: Decision problems for the verification of real-time software. In: Hespanha, J.P., Tiwari, A. (eds.) Hybrid Systems: Computation and Control. pp. 200–211. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
15. Firsov, D.: Certification of context-free grammar algorithms (2016)
16. Firsov, D., Uustalu, T.: Certified parsing of regular languages. In: Gonthier, G., Norrish, M. (eds.) Certified Programs and Proofs. pp. 98–113. Springer International Publishing, Cham (2013)
17. Firsov, D., Uustalu, T.: Certified cyk parsing of context-free languages. Journal of Logical and Algebraic Methods in Programming **83**(5-6), 459–468 (2014)
18. Firsov, D., Uustalu, T.: Certified normalization of context-free grammars. In: Proceedings of the 2015 Conference on Certified Programs and Proofs. pp. 167–174. ACM (2015)
19. Grigorev, S., Ragozina, A.: Context-free path querying with structural representation of result. arXiv preprint arXiv:1612.08872 (2016)
20. Gross, J., Chlipala, A.: Parsing parses a pearl of (dependently typed) programming and proof (2015)
21. Hellings, J.: Conjunctive context-free path queries (2014)
22. Hellings, J.: Querying for paths in graphs using context-free path queries. arXiv preprint arXiv:1502.02242 (2015)
23. Hofmann, J.: Verified algorithms for context-free grammars in coq (2016)
24. Jourdan, J.H., Pottier, F., Leroy, X.: Validating lr(1) parsers. In: Seidl, H. (ed.) Programming Languages and Systems. pp. 397–416. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
25. Kaiser, J.O.: Constructive Formalization of Regular Languages. Ph.D. thesis, Saarland University (2012)

26. Koschmieder, A., Leser, U.: Regular path queries on large graphs. In: International Conference on Scientific and Statistical Database Management. pp. 177–194. Springer (2012)
27. Lohrey, M.: Algorithmics on slp-compressed strings: A survey. Groups Complexity Cryptology **4**, 241–299 (2012)
28. Lopes, R., Ribeiro, R., Camarão, C.: Certified derivative-based parsing of regular expressions. In: Castor, F., Liu, Y.D. (eds.) Programming Languages. pp. 95–109. Springer International Publishing, Cham (2016)
29. Lu, Y., Shang, L., Xie, X., Xue, J.: An incremental points-to analysis with cfl-reachability. In: International Conference on Compiler Construction. pp. 61–81. Springer (2013)
30. Maneth, S., Peternek, F.: Grammar-based graph compression. Information Systems **76**, 19 – 45 (2018). https://doi.org/https://doi.org/10.1016/j.is.2018.03.002, `http://www.sciencedirect.com/science/article/pii/S0306437917301680`
31. Nederhof, M.J., Satta, G.: Parsing non-recursive context-free grammars. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. pp. 112–119. Association for Computational Linguistics (2002)
32. Nederhof, M.J., Satta, G.: Parsing non-recursive context-free grammars. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. pp. 112–119. ACL '02, Association for Computational Linguistics, Stroudsburg, PA, USA (2002). https://doi.org/10.3115/1073083.1073104, `https://doi.org/10.3115/1073083.1073104`
33. Nederhof, M.J., Satta, G.: The language intersection problem for non-recursive context-free grammars. Information and Computation **192**(2), 172 – 184 (2004). https://doi.org/https://doi.org/10.1016/j.ic.2004.03.004, `http://www.sciencedirect.com/science/article/pii/S0890540104000562`
34. Ramos, M.V.M., de Queiroz, R.J.G.B.: Formalization of closure properties for context-free grammars. CoRR **abs/1506.03428** (2015), `http://arxiv.org/abs/1506.03428`
35. Ramos, M.V.M., de Queiroz, R.J.G.B., Moreira, N., Almeida, J.C.B.: Formalization of the pumping lemma for context-free languages. CoRR **abs/1510.04748** (2015), `http://arxiv.org/abs/1510.04748`
36. Ramos, M.V.M., de Queiroz, R.J., Moreira, N., Almeida, J.C.B.: On the formalization of some results of context-free language theory. In: International Workshop on Logic, Language, Information, and Computation. pp. 338–357. Springer (2016)
37. Ramos, M.V., de Queiroz, R.J.: Formalization of simplification for context-free grammars. arXiv preprint arXiv:1509.02032 (2015)
38. Rehof, J., Fähndrich, M.: Type-base flow analysis: from polymorphic subtyping to cfl-reachability. ACM SIGPLAN Notices **36**(3), 54–66 (2001)
39. Reps, T., Horwitz, S., Sagiv, M.: Precise interprocedural dataflow analysis via graph reachability. In: Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 49–61. POPL '95, ACM, New York, NY, USA (1995). https://doi.org/10.1145/199448.199462, `http://doi.acm.org/10.1145/199448.199462`
40. Scott, E., Johnstone, A.: Gll parsing. Electronic Notes in Theoretical Computer Science **253**(7), 177–189 (2010)
41. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88**(2), 191 – 229 (1991). https://doi.org/https://doi.org/10.1016/0304-3975(91)90374-B, `http://www.sciencedirect.com/science/article/pii/030439759190374B`

42. Vardoulakis, D., Shivers, O.: Cfa2: A context-free approach to control-flow analysis. In: Proceedings of the 19th European Conference on Programming Languages and Systems. pp. 570–589. ESOP'10, Springer-Verlag, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11957-6_30, http://dx.doi.org/10.1007/978-3-642-11957-6_30
43. Yan, D., Xu, G., Rountev, A.: Demand-driven context-sensitive alias analysis for java. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis. pp. 155–165. ISSTA '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2001420.2001440, http://doi.acm.org/10.1145/2001420.2001440
44. Zhang, Q., Su, Z.: Context-sensitive data-dependence analysis via linear conjunctive language reachability. SIGPLAN Not. **52**(1), 344–358 (Jan 2017). https://doi.org/10.1145/3093333.3009848, http://doi.acm.org/10.1145/3093333.3009848
45. Zhang, X., Feng, Z., Wang, X., Rao, G., Wu, W.: Context-free path queries on rdf graphs. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) The Semantic Web – ISWC 2016. pp. 632–648. Springer International Publishing, Cham (2016)