WoLLIC 2019

Bar-Hillel Theorem Mechanization in Coq

Sergey Bozhko, Leyla Khatbullina, **Semyon Grigorev**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

July 05, 2019

Automated Theorem Proving

- Yet another attemt to automate proof correctness checking
- In some systems — a way to create correct by construction algorithms
  - Coq

Formal Language Theory Mechanization

- Nontrivial proofs checking
- Correctness of algorithms

The Bar-Hillel Theorem

Theorem (Bar-Hillel)

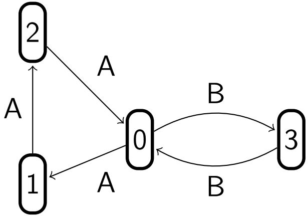
If L_1 is a context-free language and L_2 is a regular language, then $L_1 \cap L_2$ is context-free.

Context-Free Path Quierying (CFPQ)

Navigation through a edge-labelled graph

Context-Free Path Quierying (CFPQ)

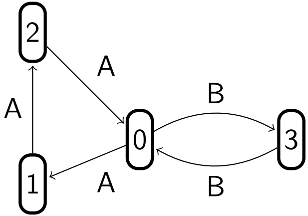
Navigation through a edge-labelled graph



Context-Free Path Quierying (CFPQ)

Navigation through a edge-labelled graph

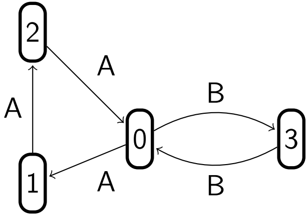
- Whether exist paths in graph, such that they looks like well-balanced sequences over `A` and `B`?



Context-Free Path Quierying (CFPQ)

Navigation through a edge-labelled graph

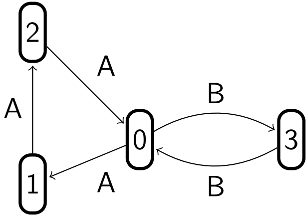
- Whether exist paths in graph, such that they looks like well-balanced sequences over `A` and `B`?
- Find all paths, such that thay form a world in the Dyck language over `A` and `B`



Context-Free Path Quierying (CFPQ)

Navigation through a edge-labelled graph

- Whether exist paths in graph, such that they looks like well-balanced sequences over `A` and `B`?
- Find all paths, such that thay form a world in the Dyck language over `A` and `B`



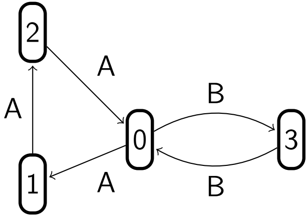
Paths filter (query):

$$s \to A\ s\ B\ s \mid \varepsilon$$

Context-Free Path Quierying (CFPQ)

Navigation through a edge-labelled graph

- Whether exist paths in graph, such that they looks like well-balanced sequences over `A` and `B`?
- Find all paths, such that thay form a world in the Dyck language over `A` and `B`



Paths filter (query):

$$s \to A\ s\ B\ s \mid \varepsilon$$

Answer:

- $2 \xrightarrow{A} 0 \xrightarrow{B} 3$
- $1 \xrightarrow{A} 2 \xrightarrow{A} 0 \xrightarrow{B} 3 \xrightarrow{B} 0$
- $\ldots$

CFPQ: Formal View

- $\mathbb{G} = (\Sigma, N, P, S)$ — context-free grammar
  - $L(\mathbb{G}) = \{w \mid S \Rightarrow^* w\}$

CFPQ: Formal View

- $\mathbb{G} = (\Sigma, N, P, S)$ — context-free grammar
  - $L(\mathbb{G}) = \{w \mid S \Rightarrow^* w\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$

CFPQ: Formal View

- $\mathbb{G} = (\Sigma, N, P, S)$ — context-free grammar
  - $L(\mathbb{G}) = \{w \mid S \Rightarrow^* w\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$

CFPQ: Formal View

- $\mathbb{G} = (\Sigma, N, P, S)$ — context-free grammar
  - $L(\mathbb{G}) = \{w \mid S \Rightarrow^* w\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$
- $R = \{(n, m) \mid \exists n \pi m$, such that $\omega(\pi) \in L(\mathbb{G})\}$

CFPQ: Formal View

- $\mathbb{G} = (\Sigma, N, P, S)$ — context-free grammar
  - $L(\mathbb{G}) = \{w \mid S \Rightarrow^* w\}$
- $G = (V, E, L)$ — directed graph
  - $v \xrightarrow{l} u \in E$
  - $L \subseteq \Sigma$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$
- $R = \{(n, m) \mid \exists n \pi m$, such that $\omega(\pi) \in L(\mathbb{G})\}$
- $P = \{\pi \mid \pi$ is a path in G, such that $\omega(\pi) \in L(\mathbb{G})\}$

Applications of CFPQ

- Graph data base querying
  - Yannacacis !!!
  - Static code analysis

Applications of CFPQ

- Graph data base querying
  - Yannacacis !!!
  - Static code analysis
- Static code analysis
  - Reps !!!
  - Static code analysis

Sketch of the Proof

1. Assume that there is a contextfree grammar $\mathbb{G}_{CNF}$ in Chomsky normal form, such that $L(\mathbb{G}_{CNF}) = L_1$

Sketch of the Proof

1. Assume that there is a contextfree grammar $\mathbb{G}_{CNF}$ in Chomsky normal form, such that $L(\mathbb{G}_{CNF}) = L_1$
2. Assume that there is a set of regular languages $\{A_1 \dots A_n\}$ where each A_i is recognized by a DFA with precisely one final state and $L_2 = A_1 \cup \ldots \cup A_n$

Sketch of the Proof

1. Assume that there is a contextfree grammar $\mathbb{G}_{CNF}$ in Chomsky normal form, such that $L(\mathbb{G}_{CNF}) = L_1$
2. Assume that there is a set of regular languages $\{A_1 \dots A_n\}$ where each A_i is recognized by a DFA with precisely one final state and $L_2 = A_1 \cup \dots \cup A_n$
   - If $L \neq \varnothing$ and L is regular then L is the union of regular language $A_1, \dots, A_n$ where each A_i is accepted by a DFA with exactly one final state

Sketch of the Proof

1. Assume that there is a contextfree grammar $\mathbb{G}_{CNF}$ in Chomsky normal form, such that $L(\mathbb{G}_{CNF}) = L_1$
2. Assume that there is a set of regular languages $\{A_1 \dots A_n\}$ where each A_i is recognized by a DFA with precisely one final state and $L_2 = A_1 \cup \dots \cup A_n$
   - If $L \neq \varnothing$ and L is regular then L is the union of regular language $A_1, \dots, A_n$ where each A_i is accepted by a DFA with exactly one final state
3. For each A_i we can explicitly define a grammar of the intersection: $L(\mathbb{G}_{CNF}) \cap A_i$

Sketch of the Proof

1. Assume that there is a contextfree grammar $\mathbb{G}_{CNF}$ in Chomsky normal form, such that $L(\mathbb{G}_{CNF}) = L_1$
2. Assume that there is a set of regular languages $\{A_1 \dots A_n\}$ where each A_i is recognized by a DFA with precisely one final state and $L_2 = A_1 \cup \dots \cup A_n$
   - If $L \neq \varnothing$ and L is regular then L is the union of regular language $A_1, \dots, A_n$ where each A_i is accepted by a DFA with exactly one final state
3. For each A_i we can explicitly define a grammar of the intersection: $L(\mathbb{G}_{CNF}) \cap A_i$
4. Finally, join them together with the operation of the union

Hofmann's Results Generalization

Jana Hofmann provides mechanization of the part of CFL in the Coq

- Basic definitions: terminal, nonterminal, grammar, word, ...

Hofmann's Results Generalization

Jana Hofmann provides mechanization of the part of CFL in the Coq

- Basic definitions: terminal, nonterminal, grammar, word, ...
- **Context-Free grammar to the Chomsky Normal Form convertion**

Hofmann's Results Generalization

Jana Hofmann provides mechanization of the part of CFL in the Coq

- Basic definitions: terminal, nonterminal, grammar, word, ...
- **Context-Free grammar to the Chomsky Normal Form convertion**

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

Jana Hofmann

Hofmann's Results Generalization

Jana Hofmann provides mechanization of the part of CFL in the Coq

- Basic definitions: terminal, nonterminal, grammar, word, ...
- **Context-Free grammar to the Chomsky Normal Form convertion**

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

Jana Hofmann

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

We need an arbitrary type for terminals and nontermianls!

Hofmann's Results Generalization

Jana Hofmann provides mechanization of the part of CFL in the Coq

- Basic definitions: terminal, nonterminal, grammar, word, ...
- **Context-Free grammar to the Chomsky Normal Form convertion**

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

Jana Hofmann

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

We need an arbitrary type for terminals and nontermianls!

And now we should carefully rewrite all existing stuff ...

DFA Splitting

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

DFA Splitting

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

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

Chomsky Induction

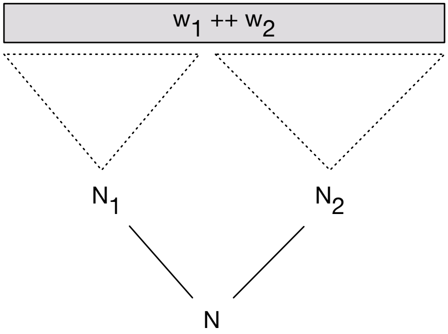
Lemma

Let $\mathbb{G}$ *be a grammar in CNF. Consider an arbitrary nonterminal* $N \in \mathbb{G}$ *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:* $N \to N_1 N_2 \in \mathbb{G}$*,* $der(\mathbb{G}, N_1, w_1)$*,* $der(\mathbb{G}, N_2, w_2)$*,* $|w_1| \geq 1$*,* $|w_2| \geq 1$ *and* $w_1 \mathbin{++} w_2 = w$*.*

Chomsky Induction

Lemma

Let $\mathbb{G}$ be a grammar in CNF. Consider an arbitrary nonterminal $N \in \mathbb{G}$ 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: $N \to N_1 N_2 \in \mathbb{G}$, $der(\mathbb{G}, N_1, w_1)$, $der(\mathbb{G}, N_2, w_2)$, $|w_1| \geq 1$, $|w_2| \geq 1$ and $w_1 ++ w_2 = w$.



Chomsky Induction in Coq

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

Languges Union

```
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.
```

The Final Theorem

Theorem

*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 is possible (in the sense of our definition), then for any DFA **dfa** and any context-free grammar $\mathbb{G}$, there exists the context-free grammar $\mathbb{G}_{INT}$, such that $L(\mathbb{G}_{INT}) = L(\mathbb{G}) \cap L(dfa)$.*

The Final Theorem in Coq

```
Theorem grammar_of_intersection_exists:
  exists
   (NewNonterminal: Type)
   (IntersectionGrammar: @grammar Terminal NewNonterminal)
   St,
  forall word,
    dfa_language dfa word /\ language G S (to_phrase word)
    <->
    language IntersectionGrammar St (to_phrase word).
```

Conclusion

- We present mechanized in Coq proof of the Bar-Hillel theorem on the closure of context-free languages under intersection with the regular languages
- We generalize the results of Jana Hofmann and Gert Smolka
  - The definition of the terminal and nonterminal alphabets in context-free grammar were made generic
  - All related definitions and theorems were adjusted to work with the updated definition
- All results are published at GitHub and are equipped with automatically generated documentation

Future work

- Ruy J. G. B. de Queiroz vs Jana Hifmann
  - We use results of Jana Hofman
  - Results of Ruy J. G. B. de Queiroz looks more mature
  - Is it possible to create one "true" solution in this area?
    - Wether our grammar-based proof is always better then PDA-based one?

Future work

- Ruy J. G. B. de Queiroz vs Jana Hifmann
  - We use results of Jana Hofman
  - Results of Ruy J. G. B. de Queiroz looks more mature
  - Is it possible to create one "true" solution in this area?
    - ⋆ Wether our grammar-based proof is always better then PDA-based one?
- Mechanization of practical algorithms which are just implementation of the Bar-Hillel theorem
  - Context-free path querying algorithm, based on CYK or even on GLL parsing algorithm
  - Certified algorithm for context-free constrained path querying for graph databases

Contact Information

- Semyon Grigorev:
  - s.v.grigoriev@spbu.ru
  - Semen.Grigorev@jetbrains.com
- Sergey Bozhko:
  - Max Planck Institute for Software Systems (MPI-SWS), Saarbrcken, Germany
  - sbozhko@mpi-sws.com
- Leyla Khatbullina:
  - St.Petersburg Electrotechnical University "LETI", St.Petersburg, Russia
  - leila.xr@gmail.com

- Sources: https://github.com/YaccConstructor/YC_in_Coq

Thanks!