Formalization of Context-Free Language Theory

Marcus Vinícius Midena Ramos

Ruy J. G. B. de Queiroz
(Supervisor)

January 18th, 2016

Pós-Graduação em Ciência da Computação

Introduction

Mathematical formalization

+

Context-free language theory

=

Formalization of context-free language theory

- Some experience with formal languages (informal);
- First contact with Coq;
- No (context-free) formalization in Coq;
- Formalization of the Pumping Lemma (in HOL4) discovered only in november 2015.

Introduction

Plan

- Formalize an important subset of context-free language theory;
- Use the Coq proof assistant (type theory).
- Study:
  - Logics and natural deduction;
  - Lambda calculus;
  - Type theory;
  - Mathematical formalization;
  - Interactive theorem proving.
- Build a set of libraries that could be used for:
  - Education;
  - Certified software construction.
- Create and develop a culture of mathematical formalization.

Introduction

History

Background:

- 2011-2012: classes on lambda calculus, set theory and logic;
- 2013-2015: self study of proof theory, type theory and Coq;

Formalization:

- July 2013 until April 2014: regular languages, Coq as a functional programming language;
- April 2014 until August 2015: context-free languages, focus on lemmas and theorems;

Introduction

History

Presentations:

- 02/2014: WTA/EPUSP/USP;
- 02/2014: Thesis proposal examination;
- 09/2014: LSFA'14;
- 07/2015: DCC/FC/UP;
- 08/2015: LSFA'15.

Thesis writing:

- September 2015 until December 2015.

Formally Checked Projects

Mathematical formalization is a mature activity:

- Use over the years;
- Diversity of proof assistants and underlying theories;
- Development of proof assistants technology;
- Size, complexity and importance of many different projects;
- Theoretical and technologically oriented;
- Academy and industry oriented;
- A clear trend;
- A point of no return.

Related Work

Formal languages

- Language and automata theory has been subject of formalization since the mid-1980s, when Kreitz used the Nuprl proof assistant to prove results about deterministic finite automata and the pumping lemma for regular languages;
- Since then, the theory of regular languages has been the subject of intense formalization by various researchers using many different proof assistants;
- The formalization of context-free language theory, on the other hand, is more recent and includes fewer accomplishments, mostly concentrated in certified parser generation.

Related Work

Regular languages

Doczkal, Kaiser and Smolka

A Constructive Theory of Regular Languages in Coq (CPP 2013)

- Follows the book by Kozen;
- Fairly complete formalization of regular languages theory;
- Coq/SSReflect;
- 1,400 lines.

Related Work

Context-free languages

Formalization results:

- Mostly from 2010;
- Mostly devoted to the certification and validation of parser generators;
- Few on theory.

Related Work

Context-free language theory

Norrish and Barthwal

A mechanization of some context-free language theory in HOL4 (WoLLIC 2010)

- Closure properties;
- The existence of normal forms for grammars (Chomsky and Greibach);
- Pumping Lemma for context-free languages;
- Pushdown automata;
- Classical (non-constructive) reasoning;
- Verified SLR parser generator with an executable version.

Related Work

Context-free language theory

Firsov and Uustalu

Certified normalization of context-free grammars (CPP 2015)

- Chomsky Normal Form (empty rules, unit rules and CNF);
- Constructive reasoning;
- Non-terminal and terminal sets must be decidable;
- Certified normalization function;
- Certified CYK parser.

Related Work

Summary

	Norrish & Barthwal2010	Firsov & Uustalu
Proof assistant	HOL4	Agda
Closure	✓	×
Simplification	✓	*only empty and unit rules*
CNF	✓	✓
GNF	✓	×
PDA	✓	×
PL	✓	×

Related Work

Motivation

- Until 2015, the only comprehensive work is the one by Norrish and Barthwal (HOL4 in 2010);
- The Pumping Lemma has not been published;
- Firsov and Uustalu add a more limited formalization (Agda in 2015);
- No formalization in Coq.
- Formalization of the PL in HOL4 discovered only in november 2015.

Objectives

Formally describe and prove:

- Closure properties of context-free languages and grammars;
- Context-free grammar simplification;
- Chomsky Normal Form (CNF);
- Pumping Lemma (PL) for context-free languages.

PL depends on CNF, which in turn depends on grammar simplification.

Steps

1. Selection of an underlying formal logic to express the theory and then a tool that supports it adequately;
2. Representation of the objects of the universe of discourse in this logic;
3. Implementation of a set of basic transformations and mappings over these objects;
4. Statement of the lemmas and theorems that describe the properties and the behaviour of these objects, and establish a consistent and complete theory;
5. Formal derivation of proofs of these lemmas and theorems, leading to proof objects that can confirm their validity.

Definitions

Main definitions in Coq:

- Symbols (including terminal and non-terminal);
- Sentential forms (strings of terminal and non-terminal symbols);
- Sentences (strings of terminal symbols);
- Context-free grammars;
- Derivations;
- Binary trees.

Sequence

1. General purpose libraries;
   - Basic lemmas on arithmetic, lists and logic;
   - Basic lemmas on context-free languages and grammars;
   - Basic lemmas on binary trees and their relation to CNF grammars;
2. Closure properties;
3. Grammar simplification $\rightarrow$ Chomsky Normal Form $\rightarrow$ Pumping Lemma.

Generic CFG Library

General results on context-free gramars and languages:

- 4,393 lines of Coq script, ~18.3% of the total;
- 105 lemmas and theorems;
- Alternative definitions for predicate `derives`;
- Supports the whole formalization;
- Some examples follow.

Generic CFG Library

- Derivation transitivity:
  $\forall g, s_1, s_2, s_3, (s_1 \Rightarrow_g^* s_2) \rightarrow (s_2 \Rightarrow_g^* s_3) \rightarrow (s_1 \Rightarrow_g^* s_3)$
- Context independence:
  $\forall g, s_1, s_2, s, s', (s_1 \Rightarrow_g^* s_2) \rightarrow (s \cdot s_1 \cdot s' \Rightarrow_g^* s \cdot s_2 \cdot s')$
- Concatenation:
  $\forall g, s_1, s_2, s_3, s_4, (s_1 \Rightarrow_g^* s_2) \rightarrow (s_3 \Rightarrow_g^* s_4) \rightarrow (s_1 \cdot s_3 \Rightarrow_g^* s_2 \cdot s_4)$
- Derivation independence:
  $\forall g, s_1, s_2, s_3, (s_1 \cdot s_2 \Rightarrow_g^* s_3) \rightarrow \exists s_1', s_2' \,|\, (s_3 = s_1' \cdot s_2') \wedge (s_1 \Rightarrow_g^* s_1') \wedge (s_2 \Rightarrow_g^* s_2')$

Generic CFG Library

Alternative definitions for predicate `derives`:

- Used to ease some proofs;
- Equivalence has been proved;
- Standard `derives` has been used in statements.

Method

Most of the work share a common objective: to construct a new grammar from an existing one (or two existing ones).

$$G \rightsquigarrow G'$$

$$\begin{aligned} P &\rightsquigarrow P' \\ \Sigma &\rightsquigarrow \Sigma' \\ V &\rightsquigarrow V' \\ S &\rightsquigarrow S' \end{aligned}$$

Method

This is the case of:

- Closure properties:
  - Union;
  - Concatenation;
  - Kleene star;
- Grammar simplification:
  - Elimination of empty rules;
  - Elimination of unit rules;
  - Elimination of useless symbols;
  - Elimination of inaccessible symbols;
- Chomsky Normal Form (CNF).

Thus, a common method to be used in all these cases has been devised.

Does not apply for the Pumping Lemma.

Method

For the first three objectives

1. Inductively define a new type of non-terminal symbols (if necessary):
   - Ensure that the start symbol of the grammar does not appear in the right-hand side of any rule;
   - New non-terminals are necessary;
   - Map previous;
   - Add new.
2. Inductively define the rules of the new grammar:
   - Exclude previous;
   - Map previous;
   - Add new (directly and/or indirectly).

Method

For the first three objectives

3. Define the new grammar:
   - Use the new non-terminal symbols and the new rules;
   - Start symbol (previous or new);
   - Prove the finiteness of the set of rules for this new grammar;
4. State and prove all the lemmas and theorems that will assert that the newly defined grammar has the desired properties;
5. Consolidate the results within the same scope and finally with the previously obtained results.

Closure Properties

Union

Given two arbitrary context-free grammars g_1 and g_2, the following definitions are used to construct g_3 such that:

$$L(g_3) = L(g_1) \cup L(g_2)$$

(that is, the language generated by g_3 is the union of the languages generated by g_1 and g_2).

Closure Properties

Union

$g_1,\ g_2 \rightsquigarrow g_3$

$$
\begin{array}{rcl}
P_1,\ P_2 & \underbrace{\rightsquigarrow}_{+2} & P_3 \\
\Sigma,\ \Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1,\ V_2 & \underbrace{\rightsquigarrow}_{+1} & V_3 \\
S_1,\ S_2 & \underbrace{\rightsquigarrow}_{new} & S_3
\end{array}
$$

Closure Properties

Union

- For the new set of non-terminals (`g_uni_nt`):
  - All the non-terminals of g_1;
  - All the non-terminals of g_2;
  - A fresh new non-terminal symbol (S_3).
- For the new set of rules (`g_uni_rules`):
  - All the rules of g_1;
  - All the rules of g_2;
  - Two new rules: $S_3 \rightarrow S_1$ and $S_3 \rightarrow S_2$.
- For the new grammar (`g_uni`):
  - The new set of non-terminals;
  - The new set of rules;
  - The new non-terminal (S_3) as the start symbol.

Closure Properties

Concatenation

Given two arbitrary context-free grammars g_1 and g_2, the following definitions are used to construct g_3 such that:

$$L(g_3) = L(g_1) \cdot L(g_2)$$

(that is, the language generated by g_3 is the concatenation of the languages generated by g_1 and g_2).

Closure Properties

Concatenation

$g_1,\ g_2 \rightsquigarrow g_3$

$$
\begin{array}{rcl}
P_1,\ P_2 & \underbrace{\rightsquigarrow}_{+1} & P_3 \\
\Sigma,\ \Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1,\ V_2 & \underbrace{\rightsquigarrow}_{+1} & V_3 \\
S_1,\ S_2 & \underbrace{\rightsquigarrow}_{new} & S_3
\end{array}
$$

Closure Properties

Concatenation

- For the new set of non-terminals (`g_cat_nt`):
  - All the non-terminals of g_1;
  - All the non-terminals of g_2;
  - A fresh new non-terminal symbol (S_3).
- For the new set of rules (`g_cat_rules`):
  - All the rules of g_1;
  - All the rules of g_2;
  - One new rule: $S_3 \rightarrow S_1 S_2$.
- For the new grammar (`g_cat`):
  - The new set of non-terminals;
  - The new set of rules;
  - The new non-terminal (S_3) as the start symbol.

Closure Properties

Kleene star

Given an arbitrary context-free grammar g_1, the following definitions are used to construct g_2 such that:

$$L(g_2) = (L(g_1))^*$$

(that is, the language generated by g_2 is the reflexive and transitive concatenation (Kleene star) of the language generated by g_1).

Closure Properties

Kleene star

$g_1 \rightsquigarrow g_2$

$$
\begin{array}{lcl}
P_1 & \underbrace{\rightsquigarrow}_{+2} & P_2 \\
\Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1 & \underbrace{\rightsquigarrow}_{+1} & V_2 \\
S_1 & \underbrace{\rightsquigarrow}_{new} & S_2
\end{array}
$$

Closure Properties

Kleene star

- For the new set of non-terminals (`g_clo_nt`):
  - All the non-terminals of g_1;
  - A fresh new non-terminal symbol (S_2).
- For the new set of rules (`g_clo_rules`):
  - All the rules of g_1;
  - Two new rules: $S_2 \rightarrow S_2 S_1$ and $S_2 \rightarrow \epsilon$.
- For the new grammar (`g_clo`):
  - The new set of non-terminals;
  - The new set of rules;
  - The new non-terminal (S_2) as the start symbol.

Closure Properties

Correctness and Completeness

Union (correctness)

Considering that g_3 is the union of g_1 and g_2 and S_3, S_1 and S_2 are, respectively, the start symbols of g_3, g_1 and g_2):

$$\forall g_1,\, g_2,\, s_1,\, s_2, (S_1 \Rightarrow^*_{g_1} s_1 \rightarrow S_3 \Rightarrow^*_{g_3} s_1) \wedge (S_2 \Rightarrow^*_{g_2} s_2 \rightarrow S_3 \Rightarrow^*_{g_3} s_2)$$

Union (completeness)

$$\forall s_3, (S_3 \Rightarrow^*_{g_3} s_3) \rightarrow (S_1 \Rightarrow^*_{g_1} s_3) \vee (S_2 \Rightarrow^*_{g_2} s_3)$$

Closure Properties

Correctness and Completeness

Concatenation (correctness)

Considering that g_3 is the concatenation of g_1 and g_2 and S_3, S_1 and S_2 are, respectively, the start symbols of g_3, g_1 and g_2)

$$\forall g_1,\ g_2,\ s_1,\ s_2, (S_1 \Rightarrow^*_{g_1} s_1) \wedge (S_2 \Rightarrow^*_{g_2} s_2) \rightarrow (S_3 \Rightarrow^*_{g_3} s_1 \cdot s_2)$$

Concatenation (completeness)

$$\forall s_3, (S_3 \Rightarrow^*_{g_3} s_3) \rightarrow \exists s_1, s_2 \,|\, (s_3 = s_1 \cdot s_2) \wedge (S_1 \Rightarrow^*_{g_1} s_1) \wedge (S_2 \Rightarrow^*_{g_2} s_2)$$

Closure Properties

Correctness and Completeness

Kleene star (correctness)

Considering that g_2 is the Kleene star of g_1 and S_2 and S_1 are, respectively, the start symbols of g_2 and g_1):

$$\forall g_1,\ s_1,\ s_2, (S_2 \Rightarrow^*_{g_2} \epsilon) \wedge ((S_2 \Rightarrow^*_{g_2} s_2) \wedge (S_1 \Rightarrow^*_{g_1} s_1) \rightarrow S_2 \Rightarrow^*_{g_2} s_2 \cdot s_1)$$

Kleene star (completeness)

$$\forall s_2, (S_2 \Rightarrow^*_{g_2} s_2) \rightarrow (s_2 = \epsilon) \vee$$

$$(\exists\ s_1,\ s_2' \,|\, (s_2 = s_2' \cdot s_1) \wedge (S_2 \Rightarrow^*_{g_2} s_2') \wedge (S_1 \Rightarrow^*_{g_1} s_1))$$

Grammar Simplification

Objective

For all G, if G is non-empty, then there exists G' such that:

- $L(G) = L(G')$;
- G' has no empty rules (except for one, if G generates the empty string);
- G' has no unit rules;
- G' has no useless symbols;
- G' has no inaccessible symbols;
- The start symbol of G' does not appear on the right-hand side of any other rule of G'.

Grammar Simplification

Empty rule

- An *empty rule* $r \in P$ is a rule whose right-hand side β is empty (e.g. $X \to \epsilon$).

We formalize that for all G, there exists G' such that:

- $L(G) = L(G')$ and
- G' has no empty rules, except for a single rule $S' \to \epsilon$ if $\epsilon \in L(G)$ (S' is the initial symbol of G');
- S' does not appear on the right-hand side of any rule of G'.

Grammar Simplification

Empty rules elimination

From g_1:

1. Construct g_2 such that $L(g_2) = L(g_1) - \epsilon$;
2. Construct g_3 (using g_2) such that:
   - $L(g_3) = L(g_1) \cup \{\epsilon\}$ if $\epsilon \in L(g_1)$ or
   - $L(g_3) = L(g_1)$ if $\epsilon \notin L(g_1)$.

Grammar Simplification

Empty rules elimination

Step 1:

$g_1 \rightsquigarrow g_2$

$$P_1 \underbrace{\rightsquigarrow}_{\leq,\geq,+1} P_2$$

$$\Sigma \underbrace{\rightsquigarrow}_{same} \Sigma$$

$$V_1 \underbrace{\rightsquigarrow}_{+1} V_2$$

$$S_1 \underbrace{\rightsquigarrow}_{new} S_2$$

Grammar Simplification

Empty rules elimination

Step 1:

- For the new set of non-terminals (`non_terminal'`):
  - All the non-terminals of g_1;
  - A fresh new non-terminal symbol (S_2).
- For the new set of rules (`g_emp_rules`):
  - All non-empty rules of g_1;
  - All rules of g_1 with every combination on nullable symbols in the right-hand side removed, except if empty;
  - One new rule: $S_2 \rightarrow S_1$.
- For the new grammar (`g_emp`):
  - The new set of non-terminals;
  - The new set of rules;
  - The new non-terminal (S_2) as the start symbol.

Grammar Simplification

Empty rules elimination

Step 2:

$g_1 \rightsquigarrow g_3$

$$P_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2\ or\ +1} P_3$$

$$\Sigma \underbrace{\rightsquigarrow}_{same\ of\ g_2} \Sigma$$

$$V_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2} V_3$$

$$S_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2} S_3$$

Grammar Simplification

Empty rules elimination

Step 2:

- For the new set of non-terminals (`*`):
  - All the non-terminals of Step 1.
- For the new set of rules (`g_emp'_rules`):
  - All the rules of Step 1;
  - One new rule: $S_2 \rightarrow \epsilon$ if $\epsilon \in L(g_1)$.
- For the new grammar (`g_emp'`):
  - The same set of non-terminals;
  - The new set of rules;
  - The same start symbol (S_2).

Grammar Simplification

Unit rule

- A *unit rule* $r \in P$ is a rule whose right-hand side β contains a single non-terminal symbol (e.g. $X \to Y$).

We formalize that for all G, there exists G' such that:

- $L(G) = L(G')$ and
- G' has no unit rules.

Grammar Simplification

Unit rules elimination

$g_1 \rightsquigarrow g_2$

$$
\begin{array}{lll}
P_1 & \underbrace{\rightsquigarrow}_{\leq,\geq} & P_2 \\
\Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1 & \underbrace{\rightsquigarrow}_{same} & V_2 \\
S_1 & \underbrace{\rightsquigarrow}_{same} & S_2
\end{array}
$$

Grammar Simplification

Unit rules elimination

From g_1:

- For the new set of non-terminals (`*`):
  - All the non-terminals of g_1.
- For the new set of rules (`g_unit_rules`):
  - All non-unit rules of g_1;
  - New rules: one for each `a`, `b`, $right$ such that (i) `unit a b`, (ii) $b \rightarrow right$, (iii) $right$ is not a single non-terminal; the new rule becomes $a \rightarrow right$.
- For the new grammar (`g_unit`):
  - The same set of non-terminals;
  - The new set of rules;
  - The same start symbol (S_1).

Grammar Simplification

Useless symbol

- A symbol $s \in V$ is *useful* if it is possible to derive a string of terminal symbols from it using the rules of the grammar. Otherwise, s is called an *useless symbol*;
- A useful symbol s is one such that $s \Rightarrow^* \omega$, with $\omega \in \Sigma^*$;
- This definition concerns mainly non-terminals, as terminals are trivially useful.

We formalize that, for all G such that $L(G) \neq \emptyset$, there exists G' such that:

- $L(G) = L(G')$ and
- G' has no useless symbols.

Grammar Simplification

Useless symbol elimination

$g_1 \rightsquigarrow g_2$

$$
\begin{array}{ccc}
P_1 & \underbrace{\rightsquigarrow}_{\leq} & P_2 \\
\Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1 & \underbrace{\rightsquigarrow}_{same} & V_2 \\
S_1 & \underbrace{\rightsquigarrow}_{same} & S_2
\end{array}
$$

Grammar Simplification

Useless symbol elimination

From g_1:

- For the new set of non-terminals (*):
  - All the non-terminals of g_1.
- For the new set of rules (`g_use_rules`):
  - All rules of g_1, except those that have useless symbols.
- For the new grammar (`g_use`):
  - The same set of non-terminals;
  - The new set of rules;
  - The same start symbol (S_1, which must be useful).

Grammar Simplification

Inaccessible symbol

- A symbol $s \in V$ is *accessible* if it is part of at least one string generated from the root symbol of the grammar. Otherwise, it is called an *inaccessible symbol*;
- An accessible symbol s is one such that $S \Rightarrow^* \alpha s \beta$, with $\alpha, \beta \in V^*$.

We formalize that for all G, there exists G' such that:

- $L(G) = L(G')$ and
- G' has no inaccessible symbols.

Grammar Simplification

Inaccessible symbol elimination

$g_1 \rightsquigarrow g_2$

$$
\begin{array}{ccc}
P_1 & \underbrace{\rightsquigarrow}_{\leq} & P_2 \\
\Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1 & \underbrace{\rightsquigarrow}_{same} & V_2 \\
S_1 & \underbrace{\rightsquigarrow}_{same} & S_2
\end{array}
$$

Grammar Simplification

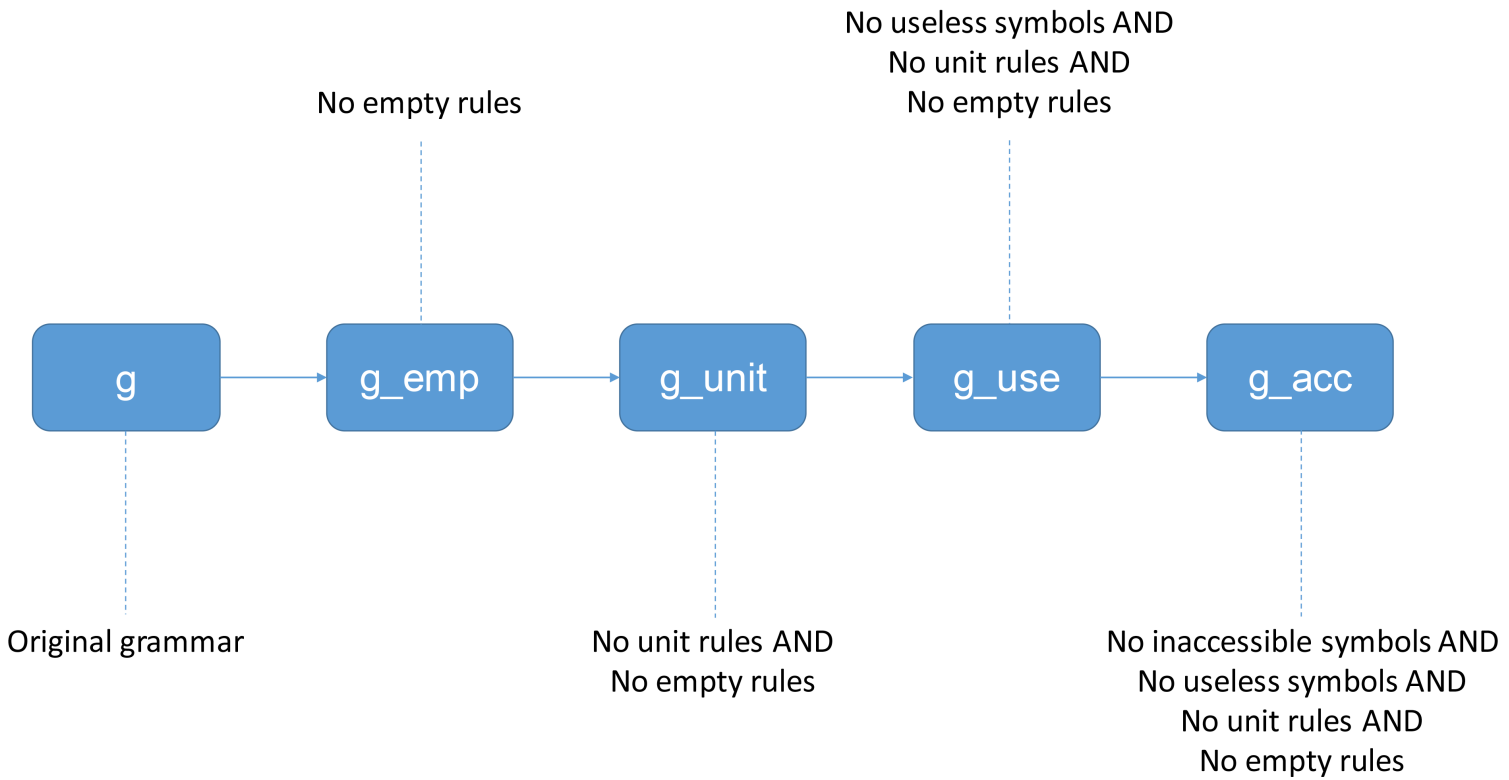
Inaccessible symbol elimination

From g_1:

- For the new set of non-terminals (*):
  - All the non-terminals of g_1.
- For the new set of rules (`g_acc_rules`):
  - All rules of g_1, except those that have inaccessible symbols.
- For the new grammar (`g_acc`):
  - The same set of non-terminals;
  - The new set of rules;
  - The same start symbol (S_1).

Grammar Simplification

Unification



Chomsky Normal Form

Definition

$$\forall\ G = (V, \Sigma, P, S),\ \exists\ G' = (V', \Sigma, P', S')\ |$$

$$L(G) = L(G') \wedge \forall\ (\alpha \rightarrow_{G'} \beta) \in P', (\beta \in \Sigma) \vee (\beta \in N \cdot N)$$

Grammar Simplification

Chomsky Normal Form

From g_1:

1. Construct g_2 such that $L(g_2) = L(g_1) - \epsilon$;
2. Construct g_3 (using g_2) such that $L(g_3) = L(g_2) \cup \{\epsilon\}$.

Grammar Simplification

Chomsky Normal Form

Step 1:

$g_1 \rightsquigarrow g_2$

$$
\begin{array}{lll}
P_1 & \underbrace{\rightsquigarrow}_{*} & P_2 \\
\Sigma & \underbrace{\rightsquigarrow}_{same} & \Sigma \\
V_1 & \underbrace{\rightsquigarrow}_{*} & V_2 \\
S_1 & \underbrace{\rightsquigarrow}_{new} & S_2
\end{array}
$$

Grammar Simplification

Chomsky Normal Form

From g_1 to g_2:

- For the new set of non-terminals:
  - One for every possibile (non-empty) sequence of terminal and non-terminal symbols of g_1: $[...]$
- For the new set of rules:
  - One for every terminal symbol t of g_1: $[t] \rightarrow t$;
  - One for every rule $X \rightarrow t$ of g_1: $[X] \rightarrow t$;
  - One for every rule $left \rightarrow s_1 s_2 \beta$ of g_1: $[left] \rightarrow [s1][s_2 \beta]$;
  - One for every rule $[left] \rightarrow [s_1][s_2 s_3 \beta]$ of g_2: $[s_2 s_3 \beta] \rightarrow [s_2][s_3 \beta]$
- For the new grammar:
  - The new set of non-terminals;
  - The new set of rules;
  - The mapped start symbol ($[S_1]$).

Grammar Simplification

Chomsky Normal Form

Step 2:

$g_1 \rightsquigarrow g_3$

$$P_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2\ +1} P_3$$

$$\Sigma \underbrace{\rightsquigarrow}_{same\ of\ g_2} \Sigma$$

$$V_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2} V_3$$

$$S_1 \underbrace{\rightsquigarrow}_{same\ of\ g_2} S_3$$

Grammar Simplification

Chomsky Normal Form

From g_1 to g_3:

- For the new set of non-terminals:
  - The same of g_2.
- For the new set of rules:
  - The same of g_2;
  - One extra rule: $[S_1] \rightarrow \epsilon$
- For the new grammar:
  - The new set of non-terminals;
  - The new set of rules;
  - The mapped start symbol ($[S_1]$).

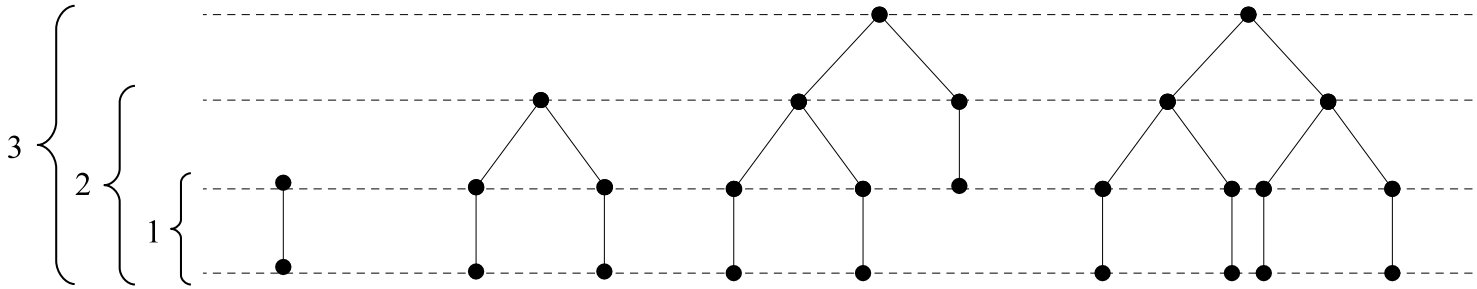
Generic Binary Trees Library

General results on binary trees and their relation to CNF grammars:

- 4,539 lines of Coq script, ~18.9% of the total;
- 84 lemmas;
- Supports the formalization of the Pumping Lemma.
- Based on the definition of `btree`.

Generic Binary Trees Library

```
Inductive btree (non_terminal terminal: Type): Type:=
| bnode_1: non_terminal → terminal → btree
| bnode_2: non_terminal → btree → btree → btree.
```



Pumping Lemma

$$\forall\ \mathcal{L}, (\mathsf{cfl}\ \mathcal{L}) \rightarrow \exists\ n\ |$$

$$\forall\ \alpha, (\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n) \rightarrow$$

$$\exists\ u, v, w, x, y \in \Sigma^*\ |\ (\alpha = uvwxy) \wedge (|vx| \geq 1) \wedge (|vwx| \leq n) \wedge$$

$$\forall\ i, uv^i wx^i y \in \mathcal{L}$$

Summary

- 23,985 lines of Coq script spread in 18 libraries;
- Eight auxiliary libraries contain 11,781 lines of Coq script and correspond to almost half of the formalization (49.1%);
- Two of these auxiliary libraries (`cfg.v` and `trees.v`) sum, alone, 8,932 lines or more than one third (37.2%) of the total;
- 533 lemmas and theorems, 83 definitions and 40 inductive definitions among 1,067 declared names;
- Created and compiled with the Coq Proof Assistant, version 8.4pl4 (June 2014), using CoqIDE for Windows;
- Available for download at `https://github.com/mvmramos/v1`;
- Compiled with the following commands under Cygwin:
  - `coq_makefile *.v > _makefile`
  - `make -f _makefile`
  - `make -f _makefile html`

Discussion

Pumping Lemma

$$\forall\ \mathcal{L}, (\text{cfl}\ \mathcal{L}) \rightarrow \exists\ n\ |$$

$$\forall\ \alpha, (\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n) \rightarrow$$

$$\exists\ u, v, w, x, y \in \Sigma^*\ |\ (\alpha = uvwxy) \wedge (|vx| \geq 1) \wedge (|vwx| \leq n) \wedge$$

$$\forall\ i, uv^i wx^i y \in \mathcal{L}$$

Discussion

Pumping Lemma

$$\forall\ \mathcal{L}, (\text{cfl}\ \mathcal{L}) \rightarrow \exists\ n\ |$$

$$\forall\ \alpha, (\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n) \rightarrow$$

$$\exists\ u, v, w, x, y \in \Sigma^*\ |\ (\alpha = uvwxy) \wedge (|vx| \geq 1) \wedge (|uy| \geq 1) \wedge (|vwx| \leq n) \wedge$$

$$\forall\ i, uv^i wx^i y \in \mathcal{L}$$

Discussion

Pumping Lemma

A variant of the Pumping Lemma, using a smaller value of n, has also been proved. This result uses $n = 2^{k-1} + 1$ instead of $n = 2^k$ (k is the number of non-terminal symbols in the CNF grammar). Since the proof needs a binary tree of height at least $k + 1$ in order to proceed, and since trees of height i have as frontier strings of length maximum 2^{i-1}, it is possible to consider strings of length equal to or greater than $2^{k-1} + 1$ (and not only of length equal to or greater than 2^k) in order to have the corresponding binary tree with height equal to or higher than $k + 1$. This way, two slightly different proofs of the Pumping Lemma have been produced: one with $n = 2^k$ (`pumping_lemma`) and the other with $n = 2^{k-1} + 1$ (`pumping_lemma_v2`).

Discussion

Pumping Lemma

The statement of (`pumping_lemma_v2`) becomes:

$$\forall\ \mathcal{L}, (\text{cfl}\ \mathcal{L}) \rightarrow \exists\ n\ |$$

$$\forall\ \alpha, (\alpha \in \mathcal{L}) \wedge (|\alpha| \geq n) \rightarrow$$

$$\exists\ u, v, w, x, y \in \Sigma^*\ |\ (\alpha = uvwxy) \wedge (|vx| \geq 1) \wedge (|vwx| \leq (n-1) * 2) \wedge$$

$$\forall\ i, uv^i wx^i y \in \mathcal{L}$$

Discussion

Contribution

	Norrish & Barthwal	Firsov & Uustalu	Ramos
Proof assistant	HOL4	Agda	Coq
Closure	✓	×	✓
Simplification	✓	*empty and unit rules*	✓
CNF	✓	✓	✓
GNF	✓	×	×
PDA	✓	×	×
PL	✓	×	✓

Achievements and Future

- A set of libraries that formalizes an important subset of context-free language theory;
- Expertise on interactive theorem proving.
  - Pioneering;
  - Reasoning about context-free language theory;
  - Learning and experimenting in an educational environment;
  - New projects and theories.
- Various possibilities, considered in three different groups:
  - New devices and results;
  - Code extraction;
  - General enhancements.

Contributions

Pioneering

- Bring formalization into an area which has relied so far mostly in informal arguments;
- First formalization of a coherent and complete subset of context-free language theory in the Coq proof assistant;
- Second formalization ever (in any proof assistant) of the Pumping Lemma for context-free languages; first in Coq;
- Second most comprehensive formalization of an important subset of the context-free language theory in any proof assistant.

Contributions

Reasoning about context-free language theory

- Insight into the nature and behaviour of the objects of context-free language theory, as well on the proofs of their properties;
- Ease the development of representations for new and similar devices, and proofs for new results of the theory;
- Ensure that the proofs are correct and that the remaining errors in the informal demonstrations, if any, can finally and definitely be reviewed and corrected.

Contributions

Learning and experimenting in an educational environment

- Teachers, students and professionals;
- Learn and experiment with the objects and concepts of context-free language theory;
- Further practical observations and developments can be done;
- Course on the theoretical foundations of computing, exploring simultaneously or independently:
  - Language theory;
  - Logic;
  - Proof theory;
  - Type theory;
  - Models of computation;
  - Formal mathematics;
  - Interactive theorem provers and Coq.

Contributions

Expertise and knowledge

- The essence of formalization;
- Application of similar principles to the formalization of other theories;
- Multiplication of the knowledge among students and colleagues;
- Technical preparation for dealing with the challenges of theory and computer program developments of the future.

Further Work

New devices and results

- Pushdown automata, including: definition, equivalence of pushdown automata and context-free grammars; equivalence of empty stack and final state acceptance criteria; non-equivalence of the deterministic and the non-deterministic models;
- Elimination of left recursion in context-free grammars and Greibach Normal Form;
- Derivation trees, ambiguity and inherent ambiguity;
- Decidable problems for context-free languages (membership, emptyness and finiteness for example);
- Odgen's Lemma.

Further Work

Code extraction

- Add computational content;
- Extract certified programs for:
  - Closure properties;
  - Grammar simplification;
  - CNF.
- Certified parser generator.

Further Work

General enhancements

- Create a naming policy that can be used rename the various objects and better identify their nature and intended use;
- Eliminate unnecessary definitions and lemmas;
- Make a better grouping of related objetcs and thus a better structuring of the whole formalization;
- Simplify some proof scripts;

Further Work

General enhancements

- Comment the scripts in order to provide a better understanding of their nature.
- Substitute the classical logic proof of the pigeonhole principle for a constructive version;
- Rewrite the contents of the trees.v library, in order to allow that all definitions and results be parametrized on any two types, one for the leafs and the other for the internal nodes of a `btree`;
- Experiment and rewrite in SSReflect.