CS420

Introduction to the Theory of Computation

Lecture 8: Formal languages

Tiago Cogumbreiro
Today we will learn...

- A summary on module 1, intro do module 2
- Formal languages
- A library of languages
A little taste of dependent types

by David Christiansen. URL: www.youtube.com/watch?v=VxINoKFm-S4

Note: \(\Sigma\) is exists, \(U\) is Prop, \(\Pi\) is forall
What have we learned in Module 1?

1. A programming language to systematically prove logical facts (Coq)
   - Dependently-typed language
   - Inductive types
   - Inductive propositions
   - Recursion and the connection to proofs by induction
What have we learned in Module 1?

1. **A programming language to systematically prove logical facts (Coq)**
   - Dependently-typed language
   - Inductive types
   - Inductive propositions
   - Recursion and the connection to proofs by induction

2. **Learn from the ground up, by assuming nothing**
   - We defined natural numbers, lists
   - We defined operations on natural numbers, lists (eg, +, -, *)
   - We proved facts about natural numbers, lists (eg, addition is commutative, associative, etc)
What have we learned in Module 1?

1. A programming language to systematically prove logical facts (Coq)
   - Dependently-typed language
   - Inductive types
   - Inductive propositions
   - Recursion and the connection to proofs by induction

2. Learn from the ground up, by assuming nothing
   - We defined natural numbers, lists
   - We defined operations on natural numbers, lists (e.g., +, -, *)
   - We proved facts about natural numbers, lists (e.g., addition is commutative, associative, etc)

3. A better understanding of proofs
   - We can look at a theorem and intuit a proof structure (case analysis?, induction?)
   - We can even prove some facts like mindless robots (brute force proofs)
Where are proof assistants used?

- Industry
- Academy
- Education
Where are proof assistants used?

Industry

- CompCert is a C99 compiler *written in Coq* that is proved correct: The *behavior* of the output (machine code) is equivalent to that of the source code (C99).
- CompCert is used in avionics and automotive industries
Where are proof assistants used?

Academy

- Programming Language theory
- Parallel Programming theory
- Networks and distributed systems
- Cryptography
- Math (geometry)
What is programming language theory?

Programming Language theory is the cornerstone of computer science

This fields that studies:

- **abstractions of computation**
  (programming languages, DSLs, APIs, operating systems, distributed systems)
- **PL design & implementation**
  compilers, interpreters
- **quality assurance of code**
  (code analyzers, linters, bug finder)
- **correctness of algorithms**
  (verification)

Related fields

- Logic
- Software Engineering
- DevOps (automation, DSLs)

Who hires PLT scientists?

- Facebook (Automated fault-finding and fixing at Facebook) (ReasonML)
- Microsoft (Thinking above the code) (C#)
- Google (Concurrency is not parallelism) (Go, Dart)
- Amazon (Use of formal methods at AWS)
- NVidia, Intel, ...
We model the behavior of intricate systems

- We identify/prove in which cases such intricate systems **fail** (e.g., data-races being the root causes of deadlocks).
- We build tools that help intricate systems fail less (e.g., detecting deadlocks in distributed programs).

Why?

- **To tame other people's technology** — Marianne Bellotti
- **To find bugs without running or even looking at the code** — Jay Parlar
Where are proof assistants used?

Education

- To teach programming language theory (Benjamin Pierce, UPenn)
- To teach math (Kevin Buzzard, Imperial College)
- To teach logic
- To teach the theory of computing (here!)
What is next in Module 2?

- Formal languages
- Regular expressions
- Finite State Machines
Formal language
Formal language

**Insight:** If we restrict what program can do, then what guarantees can we obtain from the restricted program?

- **Goal:** understanding the boundaries of computation
- **Subject:** decision procedures (a form of program)
- **Method:** introducing levels of restrictions in what programs can do

Decision procedures

- **A yes/no question:** that takes a string as input
- **A program:** that implements said question
Formal language examples

Using the mathematical notation, we simply use the **set-builder** notation to represent formal languages. **Set-membership is acceptance:** $x \in L$ reads as $L$ accepts $x$.

- $L_1 = \{ w \mid w \text{ starts with string } 01 \}$
  - Examples: $01 \in L_1$  $0101 \in L_1$  $\text{foo} \notin L_1$
Formal language examples

Using the mathematical notation, we simply use the **set-builder** notation to represent formal languages. **Set-membership is acceptance:** \( x \in L \) reads as \( L \) accepts \( x \).

- \( L_1 = \{ w \mid w \text{ starts with string } 01 \} \)
  - Examples: \( 01 \in L_1 \quad 0101 \in L_1 \quad \text{foo} \notin L_1 \)
- \( L_2 = \{ w \mid w \text{ contains character } a \} \)
  - Examples: \( 000 \notin L_2 \quad \text{aaaaa} \in L_2 \)
Formal language examples

Using the mathematical notation, we simply use the **set-builder** notation to represent formal languages. **Set-membership is acceptance:** $x \in L$ reads as *L* accepts *x*.

- $L_1 = \{w \mid w \text{ starts with string } 01\}$
  - Examples: $01 \in L_1 \quad 0101 \in L_1 \quad \text{foo} \not\in L_1$
- $L_2 = \{w \mid w \text{ contains character } a\}$
  - Examples: $000 \not\in L_2 \quad \text{aaaaa} \in L_2$
- $L_3 = \{w \mid w \text{ has 3 characters}\}$
  - Examples: $000 \in L_3 \quad \text{aa} \not\in L_3$
Formal language examples

Using the mathematical notation, we simply use the set-builder notation to represent formal languages. **Set-membership is acceptance:** $x \in L$ reads as $L$ accepts $x$.

- $L_1 = \{ w \mid w \text{ starts with string } 01 \}$
  - Examples: $01 \in L_1 \quad 0101 \in L_1 \quad \text{foo} \notin L_1$

- $L_2 = \{ w \mid w \text{ contains character } a \}$
  - Examples: $000 \notin L_2 \quad \text{aaaaa} \in L_2$

- $L_3 = \{ w \mid w \text{ has 3 characters} \}$
  - Examples: $000 \in L_3 \quad \text{aa} \notin L_3$

- $L_4 = \{ w \mid w \text{ is the textual representation of a prime number} \}$
  - Examples: $\text{aa} \notin L_4 \quad 3 \in L_4$
Formal language examples

Using the mathematical notation, we simply use the set-builder notation to represent formal languages. Set-membership is acceptance: $x \in L$ reads as $L$ accepts $x$.

- $L_1 = \{w \mid w \text{ starts with string } 01\}$
  - Examples: $01 \in L_1$  $0101 \in L_1$  $\text{foo} \notin L_1$
- $L_2 = \{w \mid w \text{ contains character a}\}$
  - Examples: $000 \notin L_2$  $\text{aaaaa} \in L_2$
- $L_3 = \{w \mid w \text{ has 3 characters}\}$
  - Examples: $000 \in L_3$  $\text{aa} \notin L_3$
- $L_4 = \{w \mid w \text{ is the textual representation of a prime number}\}$
  - Examples: $\text{aa} \notin L_4$  $3 \in L_4$
- $L_5 = \{w \mid w \text{ is a valid C program}\}$
  - Examples: $\text{void main(){return 0;}} \in L_5$  $\text{aa} \notin L_5$
Formal language examples

Using the mathematical notation, we simply use the set-builder notation to represent formal languages. **Set-membership is acceptance:** $x \in L$ reads as $L$ accepts $x$.

- $L_1 = \{w \mid w \text{ starts with string 01}\}$
  - Examples: $01 \in L_1$  $0101 \in L_1$  $\text{foo} \notin L_1$
- $L_2 = \{w \mid w \text{ contains character a}\}$
  - Examples: $000 \notin L_2$  $\text{aaaaa} \in L_2$
- $L_3 = \{w \mid w \text{ has 3 characters}\}$
  - Examples: $000 \in L_3$  $\text{aa} \notin L_3$
- $L_4 = \{w \mid w \text{ is the textual representation of a prime number}\}$
  - Examples: $\text{aa} \notin L_4$  $3 \in L_4$
- $L_5 = \{w \mid w \text{ is a valid C program}\}$
  - Examples: $\text{void main()\{return 0;\}} \in L_5$  $\text{aa} \notin L_5$
- $L_6 = \{w \mid w \text{ a valid C program and when run returns code 0}\}$
Looking ahead: formal languages

- Formal languages can be grouped and ordered
- Smaller languages represent simpler decision problems
- **Insight 1:** we can develop a restricted set of constructs to write all programs in a group
- **Insight 2:** We can know more about simpler languages

Regular $\subseteq$ Context-Free $\subseteq$ Decidable $\subseteq$ Turing Complete

**Regular**
- $L_1 = \{ w \mid w \text{ starts with string 01} \}$
- $L_2 = \{ w \mid w \text{ contains character a} \}$
- $L_3 = \{ w \mid w \text{ has 3 characters} \}$

**Decidable**
- $L_4 = \{ w \mid w \text{ is a prime number} \}$

**Undecidable**
- $L_6 = \{ w \mid w \text{ a C program and returns code 0} \}$
Formal languages in Coq

How do represent a formal language in Coq?
A **formal language** is a predicate, of type \((\text{list ascii}) \rightarrow \text{Prop}\):

- Takes a **string** \((\text{list ascii})\) and returns a **proof object** (an evidence),
- **Acceptance**: We say that the word is **accepted** by language \(L\) if, and only if \(L \ w\).
A **formal language** is a predicate, of type \((\text{list ascii}) \to \text{Prop}\):

- Takes a **string** \((\text{list ascii})\) and returns a **proof object** (an evidence),
- **Acceptance**: We say that the word is **accepted** by language \(L\) if, and only if \(L \ w\).

### Implementation

(* Boilerplate code *)

```coq
Require Import Coq.Lists.List.
Open Scope char_scope.
Import ListNotations.

(* Definition of a word and a language *)
Definition word := list ascii.  (* Think of it as a typedef *)
Definition language := word \to Prop.
Definition In w L := L w.  (* A word is in the language, if we can show that [L w] holds. *)
```
Strings and their operations

A **string** is a finite sequence of characters. \( \epsilon \) and \([\ ]\) represent an empty string.

Operators

- **Length:** The length of a string, written \( |w| \), is the number of characters that the string contains.
- **Substring:** String \( z \) is a substring of \( w \) if \( z \) appears consecutively within \( w \).
- **Concatenation:** We write \( x \cdot y \) for the string concatenation.
- **Power:** The power operator \( x^n \) where \( x \) is a string and \( n \) is a natural number, defined as \( x \) being concatenated \( n \) times (yields the empty string when \( n = 0 \))

\[
\begin{align*}
car^3 &= \text{carcarcar} \\
car^0 &= \epsilon \\
car^1 &= \text{car}
\end{align*}
\]
Strings in Coq

Require Import Coq.Lists.List.
Open Scope char_scope.
Import ListNotations.
Require Import Turing.Util.

(* Length: *)
Goal \texttt{length ["c";} ; "a";} ; "r"] = 3. Proof. reflexivity. Qed.

(* Concatenation *)
Goal \texttt{"c"} ++ \texttt{"a";} ; "r"] = \texttt{"c";} ; "a";} ; "r"]. Proof. reflexivity. Qed.

(* Power *)
Goal \texttt{pow ["c";} ; "a";} ; "r"] 3 = \texttt{"c";} ; "a";} ; "r";} ; "c";} ; "a";} ; "r"].

\hspace{100pt} \textbf{Proof.} reflexivity. Qed.
Goal \texttt{pow ["c";} ; "a";} ; "r"] 1 = \texttt{"c";} ; "a";} ; "r"]. Proof. reflexivity. Qed.
Goal \texttt{pow ["c";} ; "a";} ; "r"] 0 = []. Proof. reflexivity. Qed.

Coq has its own string data type, but we are not using that in this course.
Example 1

Recall that $\text{language} := \text{word} \rightarrow \text{Prop}$

1. Define a language $L_1$ that only accepts word ["c"; "a"; "r"]
2. Show that $L_1$ accepts ["c"; "a"; "r"]
**Example 1**

Recall that $\text{language} := \text{word} \to \text{Prop}$

1. Define a language $L_1$ that only accepts word $["c"; "a"; "r"]$
2. Show that $L_1$ accepts $["c"; "a"; "r"]$

**Definition** $L_1 w := w = ["c"; "a"; "r"]$. (* Define a language $L_1$ *)

(* Show that "car" is in $L_1" *)

**Lemma** car_in_l1: In $["c"; "a"; "r"] L_1$.

**Proof.**

- unfold $L_1$.
- reflexivity.

Qed.
Example 1 (continued)

3. Show that L1 rejects ["f"; "o"; "o"]
Example 1 (continued)

3. Show that L1 rejects ["f"; "o"; "o"]

(* Show that "foo" is not in L1 *)

Lemma foo_not_in_l1: ~ In ["f"; "o"; "o"] L1.

Proof.
3. Show that $L_1$ rejects ["f"; "o"; "o"]

(* Show that "foo" is not in $L_1$ *)

Lemma foo_not_in_l1: ~ In ["f"; "o"; "o"] $L_1$.

Proof.

\[
\text{unfold not, In. (* a proof by contradiction *)}
\]
\[
\text{(* Goal: } L_1 \text{ ["f"; "o"; "o"] } \rightarrow \text{ False *)}
\]

intros N.

(* N : L1 ["f"; "o"; "o"] *)

(* Goal: False *)

unfold $L_1$ in N.

(* N : ["f"; "o"; "o"] = ["c"; "a"; "r"] *)

inversion N. (* Explosion principle! *)

Qed.
Example 2: Vowel

1. Language L2 accepts strings that consist of a single vowel
Example 2: Vowel

1. Language $L_2$ accepts strings that consist of a single vowel

Definition: $Vowel \ w := w = ["a"]$
\[ \vee w = ["e"] \]
\[ \vee w = ["i"] \]
\[ \vee w = ["o"] \]
\[ \vee w = ["u"]. \]
Example 2 (continued)

2. Show that Vowel accepts ["a"]

Lemma a_in_vowel: In ["a"] Vowel.
  unfold Vowel.
  Print or.
  (* Inductive or (A B : Prop) : Prop := | or_introl : A > A \/ B *)
  (*               | or_intror : B > A \/ B *)
  apply or_introl.
  reflexivity.
Qed.
Example 2 (continuation)

3. Show that Vowel rejects ["a"; "a"]

*Lemma*  
aa_not_in_vowel: \(\sim \text{In} \ ["a"; "a"] \text{Vowel.}\)
3. Show that Vowel rejects ["a"; "a"]

Lemma aa_not_in_vowel: ~ In ["a"; "a"] Vowel.

  unfold Vowel.
  intros N.
  destruct N as [N|N|[N|[N|[N|N]]]] ; inversion N.
Qed.
A library of language operators
A library of language operators

- Recall that our objective is to **group languages**
- We want to have a **compositional** reasoning about languages
- **Idea:** Define an algebra of languages and study how properties behave under this algebra
Language operators

1. Nil
2. Char
3. Union
4. App
Nil

- A language that only accepts the empty word.

Set-builder notation: $\{w \mid w = []\}$ or $\{w \mid w = \epsilon\}$
Nil

A language that only accepts the empty word.

Set-builder notation: \( \{ w \mid w = [] \} \) or \( \{ w \mid w = \epsilon \} \)

Definition \( \text{Nil} \ w := w = [] \).

Correction properties

1. Show that \( \text{Nil} \ [ ] \)
2. Show that if a word is accepted by \( \text{Nil} \), then that word must be \( [ ] \)
Char

A language that accepts a single character (given as parameter).
Char

A language that accepts a single character (given as parameter).

**Definition**  \( \text{Char } c \ (w:\text{word}) := w = [c] \).

**Coercion**  \( \text{Char: ascii} \rightarrow \text{language.} \) (* Allow writing "a" rather than Char "a" *)

**Correction properties**

1. Show that the word \([c]\) is accepted by \(\text{Char } c\): \(\text{Char } c \ [c]\)
2. Show that any word \(w\) accepted by \(\text{Char } c\) must be equal to \([c]\)
Char

A language that accepts a single character (given as parameter).

**Definition** \( \text{Char} \ c \ (w: \text{word}) := w = [c] \).

**Coercion** \( \text{Char}: \text{ascii} \rightarrow \text{language} \). (* Allow writing "a" rather than Char "a" *)

**Correction properties**

1. Show that the word \([c]\) is accepted by \( \text{Char} \ c \): \( \text{Char} \ c \ [c] \)
2. Show that any word \( w \) accepted by \( \text{Char} \ c \) must be equal to \([c]\): Show that any word \([c]\) is in \( \text{Char} \ c \):
Union

- A language that accepts all words of both languages.
Union

A language that accepts all words of both languages.

**Definition** Union (L1 L2:language) w :=
In w L1 \/
In w L2.

**Infix** "U" := Union. (* Define a notation for terseness *)

**Correction properties**

1. If the word is accepted by either L1 or L2, then is accepted by L1 \ U \ L2
2. If the word is accepted by L1 \ U \ L2, then is accepted by either L1 or L2.
App

Language L1  \(\Rightarrow\) L2 accepts a word from L1 concatenated with a word from L2
**App**

Language L1  \(\Rightarrow\) L2 accepts a word from L1 concatenated with a word from L2

**Definition**  
\[
\text{App} \ (L1 \ L2: \text{language}) \ w := \\
\exists \ w_1 \ w_2, \ w = w_1 \ + \ w_2 \land L1 \ w_1 \land L2 \ w_2.
\]

**Correction properties**

1. If \(w_1\) in \(L1\) and \(w_2\) in \(L2\), then \(w_1 \ + \ w_2\) in \(L1 \ L2\).
2. If \(w\) in \(L1 \ L2\), then there exists \(w_1\) in \(L1\) and \(w_2\) in \(L2\) such \(w = w_1 \ + \ w_2\).