Mechanized Mathematics for Computer Sciences

EPIT 2015

Assia Mahboubi

Motivations for Today

From a paper proof to a formal proof:

- Searching and using existing libraries;
- Choosing the appropriate statement and proof;
- Reviewing some hints and best practice.

Motivations for Today

From a paper proof to a formal proof:

- Searching and using existing libraries;
- Choosing the appropriate statement and proof;
- Reviewing some hints and best practice.

On the way, a small introduction
to the Mathematical Components libraries.

Running Example

Estimation of the asymptotic behavior of Arthur's cost function:

$$\sum_{\substack{p=1\\ p \text{ prime}}}^{n} \frac{1}{p} = O(\log\ (\log\ \mathrm{n}))$$

where log n is the binary logarithm ($2^{\log\ \mathrm{n}} = n$).

Instructions

Sketch of a Proof

Lemma (0) :

$$\forall n \in \mathbb{N}, \binom{2n}{n} \leq 2^n.$$

Sketch of a Proof

Lemma (0) :

$$\forall n \in \mathbb{N}, \binom{2n}{n} \leq 2^n.$$

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

Sketch of a Proof

Lemma (0) :

$$\forall n \in \mathbb{N}, \binom{2n}{n} \leq 2^n.$$

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

Lemma (2) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \log \mathrm{p} \leq 2n.$$

Sketch of a Proof

Lemma (2) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \log \mathrm{p} \leq 2n.$$

Sketch of a Proof

Lemma (2) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \log \mathrm{p} \leq 2n.$$

Lemma (3) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \frac{1}{p} \leq \frac{2}{\log \mathrm{n}}.$$

Sketch of a Proof

Lemma (3) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \frac{1}{p} \leq \frac{2}{\log \mathrm{n}}.$$

Sketch of a Proof

Lemma (3) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \frac{1}{p} \leq \frac{2}{\log \text{n}}.$$

Lemma (4) :

$$\forall n \in \mathbb{N} \text{ such that, } n > 0, \quad \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \log \text{n}.$$

Sketch of a Proof

Lemma (3) :

$$\forall n \in \mathbb{N}, \sum_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} \frac{1}{p} \leq \frac{2}{\log \mathrm{n}}.$$

Lemma (4) :

$$\forall n \in \mathbb{N} \text{ such that, } n > 0, \quad \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \log \mathrm{n}.$$

Theorem :

$$\forall n \in \mathbb{N} \text{ such that, } n > 4, \quad \sum_{\substack{p=1 \\ p \text{ prime}}}^{n} \frac{1}{p} \leq 1 + 6 \times \log (\log \mathrm{n}).$$

Libraries of Formalized Mathematics

In order to formalize this proof, we are going to combine:

- the Standard Library of Coq;
- the Coquelicot Library on real numbers;
- the Mathematical Components library.

Standard Library

- Shipped with the Coq System;
- Contains basic results of arithmetic;
- Contains an axiomatization of real numbers, and some formalized analysis;
- And more.

Standard Library

- Shipped with the Coq System;
- Contains basic results of arithmetic;
- Contains an axiomatization of real numbers, and some formalized analysis;
- And more.

Good place to start, often not enough.

Coquelicot

- Authors: C. Lelay, G. Melquiond, S. Boldo;
- Extends and improves the standard library of real numbers;
- Is a conservative extension of the latter;
- See C. Lelay's PhD.

Coquelicot

- Authors: C. Lelay, G. Melquiond, S. Boldo;
- Extends and improves the standard library of real numbers;
- Is a conservative extension of the latter;
- See C. Lelay's PhD.

Tested live at the French national exam at university entry level.

Mathematical Components

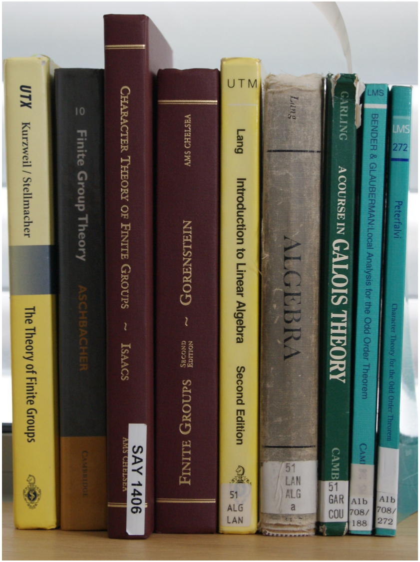
- Authors: the Math. Comp. team (led by G. Gonthier);
- Started with a Coq proof of the Four Colour Theorem;
- Culminates with a proof of the Odd Order Theorem.

Mathematical Components

- Authors: the Math. Comp. team (led by G. Gonthier);
- Started with a Coq proof of the Four Colour Theorem;
- Culminates with a proof of the Odd Order Theorem.

A wide range of formalize libraries of algebra.

Mathematical Components



From book to machine-checked proofs

A reasonable objective:

Keep simple formal proofs of what is trivial on paper.

From book to machine-checked proofs

A reasonable objective:

Keep simple formal proofs of what is trivial on paper.

But this is one of the most difficult task

From book to machine-checked proofs

A reasonable objective:

Keep simple formal proofs of what is trivial on paper.

But this is one of the most difficult task, which requires:

- Understanding the mathematical usages of the field;
- Crafting appropriate libraries accordingly;
- Using notations, inference and automation.

Mathematical Components

- An extension of Coq's tactic language;
- A custom `Search` tool;
- A (large) set of libraries;

Mathematical Components

- An extension of Coq's tactic language;
- A custom `Search` tool;
- A (large) set of libraries;

Material based and support for boolean reflection.

Boolean Reflection

Remember the first lecture and compare:

```
Inductive bool : Set := true : bool | false : bool

Definition orb (b1 b2 : bool) : bool :=
  if b1 then true else b2.
```

with:

```
Definition in_the_margin : Prop := forall (n x y z : nat),
  n > 2 -> x ^ n + y ^ n = z ^ n -> x = 0 /\ y = 0 /\ z = 0.
```

Boolean Reflection

Remember the first lecture and compare:

```
Inductive bool : Set := true : bool | false : bool

Definition orb (b1 b2 : bool) : bool :=
  if b1 then true else b2.
```

with:

```
Definition in_the_margin : Prop := forall (n x y z : nat),
  n > 2 -> x ^ n + y ^ n = z ^ n -> x = 0 /\ y = 0 /\ z = 0.
```

Although we can still write:

```
Lemma orTb : forall b : bool, orb true b = true.
Proof. reflexivity. Qed.
```

Boolean Reflection

Compare:

```
Fixpoint prime_decomp_rec m k a b c e :=
  let p := k.*2.+1 in
  if a is a'.+1 then
    if b - (ifnz e 1 k - c) is b'.+1 then
      [rec m, k, a', b', ifnz c c.-1 (ifnz e p.-2 1), e] else
    if (b == 0) && (c == 0) then
      let b' := k + a' in [rec b'.*2.+3, k, a', b', k.-1, e.+1] else
    let bc' := ifnz e (ifnz b (k, 0) (edivn2 0 c)) (b, c) in
    p ^? e :: ifnz a' [rec m, k.+1, a'.-1, bc'.1 + a', bc'.2, 0] [:: (m, 1)]
  else if (b == 0) && (c == 0) then [:: (p, e.+2)] else p ^? e :: [:: (m, 1)]
where "[ 'rec' m , k , a , b , c , e ]" := (prime_decomp_rec m k a b c e).

Definition prime_decomp n :=
  let: (e2, m2) := elogn2 0 n.-1 n.-1 in
  if m2 < 2 then 2 ^? e2 :: 3 ^? m2 :: [::] else
  let: (a, bc) := edivn m2.-2 3 in
  let: (b, c) := edivn (2 - bc) 2 in
  2 ^? e2 :: [rec m2.*2.+1, 1, a, b, c, 0].

Definition prime p :=
  if prime_decomp p is [:: (_ , 1)] then true else false.
```

Boolean Reflection

With:

```
Definition prime k : Prop :=
   k > 1 /\ forall r d, 1 < d < k -> k <> r * d.
```

Boolean reflection: free theorems

```
(* Order relation on nat *)
Fixpoint le n m := match n, m with
  | 0   , _    => true
  | S _ , 0    => false
  | S n', S m' => le n' m' end.
Notation "a <= b" := (le a b).
```

Boolean reflection: free theorems

```
(* Order relation on nat *)
Fixpoint le n m := match n, m with
  | 0   , _    => true
  | S _ , 0    => false
  | S n', S m' => le n' m' end.
Notation "a <= b" := (le a b).

(*Free theorems, thanks computation *)
Lemma le0n n   : 0 <= n.
Proof. reflexivity. Qed.

Lemma leSS n m : S n <= S m  =  n <= m.
Proof. reflexivity. Qed.
```

Boolean reflection: free theorems

```
(* Order relation on nat *)
Fixpoint le n m := match n, m with
  | 0    , _     => true
  | S _ , 0      => false
  | S n', S m' => le n' m' end.
Notation "a <= b" := (le a b).

(*Free theorems, thanks computation *)
Lemma le0n n    : 0 <= n.
Proof. reflexivity. Qed.

Lemma leSS n m : S n <= S m  =  n <= m.
Proof. reflexivity. Qed.

(* Almost free theorems *)
Lemma lenn n    : n <= n.
Proof. by elim: n. Qed.
```

Boolean reflection and deduction

Free theorems combine well with boolean connectives:

```
n : nat
m : nat
==================
 1 <= S m && (S n <= 0   ==>  b) && P

simpl.
```

Boolean reflection and deduction

Free theorems combine well with boolean connectives:

```
n : nat
m : nat
===================
 1 <= S m && (S n <= 0   ==>  b) && P
```

```
simpl.
```

```
n : nat
m : nat
===================
 P
```

Boolean vs Prop Definitions

Whereas using the relation defined in the standard library:

```
Inductive le (n : nat) : nat -> Prop :=
    le_n : le n n
  | le_S : forall m : nat, le n m -> le n (S m)
```

- The proof of `n <= m` chains `m - n + 1` constructors;
- Local simplifications are less easy.

Boolean Reflection & Classical Logic

Excluded middle is just case analysis:

```
(* Boolean Excluded Middle, never used as such. *)
Lemma EMb (b : bool) : b || ~~b = true.
Proof. by case b. Qed.
```

Boolean Reflection & Classical Logic

Excluded middle is just case analysis:

```
(* Boolean Excluded Middle, never used as such. *)
Lemma EMb (b : bool) : b || ~~b = true.
Proof. by case b. Qed.
```

Contraposition is provable:

```
Lemma contra (c b : bool) :
  (c = true -> b) -> ~~ b = true -> ~~ c = true.

Lemma contraL (c b : bool) :
(c = true -> ~~ b = true ) -> b = true -> ~~ c = true.
```

Boolean Reflection & Classical Logic

Excluded middle is just case analysis:

```
(* Boolean Excluded Middle, never used as such. *)
Lemma EMb (b : bool) : b || ~~b = true.
Proof. by case b. Qed.
```

Contraposition is provable:

```
Lemma contra (c b : bool) :
  (c = true -> b) -> ~~ b = true -> ~~ c = true.

Lemma contraL (c b : bool) :
(c = true -> ~~ b = true ) -> b = true -> ~~ c = true.
```

Allows for local classical assumptions, instead of global axioms.

Practical Issues

The pervasive `_ = true` are hidden by a coercion:

```
Lemma contra (c b : bool) :
  (c = true -> b) -> ~~ b = true -> ~~ c = true.

Lemma contraL (c b : bool) :
(c = true -> ~~ b = true ) -> b = true -> ~~ c = true.
```

is displayed (and input):

```
Lemma contra (c b : bool) :
  (c -> b) -> ~~ b -> ~~ c.

Lemma contraL (c b : bool) :
(c -> ~~ b ) -> b -> ~~ c.
```

Practical Issues

The pervasive `_ = true` are hidden by a coercion:

```
Lemma contra (c b : bool) :
  (c = true -> b) -> ~~ b = true -> ~~ c = true.

Lemma contraL (c b : bool) :
(c = true -> ~~ b = true ) -> b = true -> ~~ c = true.
```

is displayed (and input):

```
Lemma contra (c b : bool) :
  (c -> b) -> ~~ b -> ~~ c.

Lemma contraL (c b : bool) :
(c -> ~~ b ) -> b -> ~~ c.
```

Coq restores `_ = true` when typing requires `Prop` and finds `bool`.

Exercises

Lemma (0) :

$$\forall n \in \mathbb{N}, \binom{2n}{n} \leq 2^n.$$

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

Exercises

Lemma (0) :

$$\forall n \in \mathbb{N}, \binom{2n}{n} \leq 2^n.$$

Lemma (0.5) :

$$\forall p, n \in \mathbb{N}, \text{if p is prime and if } n+1 \leq p \leq 2n, \text{ then } p \mid \binom{2n}{n}.$$

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

Tools

The `rewrite` tactic plays an important role, therefore:

- It can be used for more purposes (`simpl`, `unfold`, `trivial`,...)
- It can perform chained operation;
- It can use a pattern to localize the redex.

Tools

The `rewrite` tactic plays an important role, therefore:

- It can be used for more purposes (`simpl`, `unfold`, `trivial`,...)
- It can perform chained operation;
- It can use a pattern to localize the redex.

When relying on external libraries:

- Read the sources;
- Use the `Search` and `About` commands.

The Rewrite Swiss Knife: Examples

Chaining: `rewrite foo bar` rewrites with `foo`, then `bar`.

Repeating, repeating if possible: `rewrite !foo`, `rewrite ?bar`

Simpl: `rewrite /=` but also `rewrite foo /= bar`

Trivial: `rewrite //` but also `rewrite foo // bar`

Unfold: `rewrite /blah`

Change for convertible: `rewrite -[foo]/blah`

Exact Patterns: `rewrite (X in _ <= X)foo`, `rewrite [LHS]foo`, `rewrite [X in X + _ = _]/=`

Context Patterns: `rewrite (in X in _ <= X)foo`, `rewrite [in LHS]foo`

From Bool to Prop and Back

`move/eqP: h => h` : transforms hypothesis `h : n == m` in the context into `h : n = m`

`apply/eqP` : transforms a goal `n == m` into `n = m`.

`case/orP: h => h` : when `h : p || q`, performs a case analysis: `h : p` in one branch, `h : q` in the other.

`case/andP: h => h1 h2` : when `h : p && q`, introduces both `h1 : p` and `h2 : q`.

`rewrite (negPf h):= when h : ~~ p` : rewrites occurrences of p to `false` in the goal.

Hands On

- Warm up;
- Lemma (0)
- Lemma (0.5)

Do the paper proofs first!

Next steps, if time allows

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

Lemma (4) :

$$\forall n \in \mathbb{N} \text{ such that, } n > 0, \quad \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \log \text{ n}.$$

Iterated Operations

Capital Greek letters:

- Stand for (finitely) iterated operations;
- Allow many idioms describing iteration domains;
- Hide a lot of implicit information:

  name, neutral, properties,...
- But remain remarkably non ambiguous.

We need a modular library that can accommodate this flexibility.

Iterated Operations

The generic data structure is of the form:

```
Definition reducebig R I idx r (op : R -> R -> R) (P : I -> bool) :=
  foldr (fun x y => if (P x) then op x y else y) idx r.
```

- `R : Type`;
- `op : R -> R -> R` is the iterated operation (like addition);
- `idx : R` is the default element, neutral to `op`;
- `r : list I` is the domain of iteration (like the list $[1, .., n]$);
- `P : R -> I` is a filter predicate (like `prime`)

Iterated Operations

The generic data structure is of the form:

```
Definition reducebig R I idx r (op : R -> R -> R) (P : I -> bool) :=
  foldr (fun x y => if (P x) then op x y else y) idx r.
```

- `R : Type`;
- `op : R -> R -> R` is the iterated operation (like addition);
- `idx : R` is the default element, neutral to `op`;
- `r : list I` is the domain of iteration (like the list $[1, .., n]$);
- `P : R -> I` is a filter predicate (like `prime`)

and comes with notations suggesting the properties of `op`.

Iterated Operations

The `bigop` library is generic:

- Register the properties of the operation(s) of interest;
- And you benefit from shared notations and theory;
- Including theory for specific iteration domains

  finite sets, consecutive numbers...

It is based on an extended (type-class like) mechanism of type inference called `Canonical Structure`s.

Hands On

Let us prove:

Lemma (1) :

$$\forall n \in \mathbb{N}, \prod_{\substack{p=n+1 \\ p \text{ prime}}}^{2n} p \leq \binom{2n}{n}.$$

How?

Hands On

Let us prove:

Lemma (4) :

$$\forall n \in \mathbb{N} \text{ such that, } n > 0, \quad \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \log \mathrm{n}.$$

Using a suggestion by Arthur.

Small Bibliography

- The webpage of the libraries (and mailing list etc.);
- An introduction to the Ssreflect language and to the basic libraries, for computer scientists: I. Serguey's lecture notes;
- The User Manual of the Ssreflect language;
- An introduction to Canonical Structures;
- A research paper on "big" operators.