Procrastination

A proof engineering technique

Armaël Guéneau

Proving existential quantifications

```
Goal ∃ x, ... complicated expression ....
Proof.
  (* ??? *)
```

Guessing the witness

```
Goal ∃ x, ... complicated expression ....
Proof.
  exists 0.
```

Guessing the witness

```
Goal ∃ x, ... complicated expression ....
Proof.
  exists 0.
  ...
Qed.
```

Guessing the witness

```
Goal ∃ x, ... complicated expression ....
Proof.
  exists 53367.
```

Guessing the witness

```
Goal ∃ x, … complicated expression ….
Proof.
  exists 53367.
```

- Hard to guess

Guessing the witness

```
Goal ∃ x, ... complicated expression ....
Proof.
  exists 53367.
```

- Hard to guess
- Found using a trial-and-error process?

Guessing the witness

```
Goal ∃ x, … complicated expression ….
Proof.
  exists 53367.
```

- Hard to guess
- Found using a trial-and-error process?
- “Magic witness” may be large or duplicate information

Guessing the witness

```
Goal ∃ x, … complicated expression ….
Proof.
  exists 387.
```

- Hard to guess
- Found using a trial-and-error process?
- “Magic witness” may be large or duplicate information

Guessing the witness

```
Goal ∃ x, … complicated expression ….
Proof.
  exists 387.
```

- Hard to guess
- Found using a trial-and-error process?
- “Magic witness” may be large or duplicate information
- Not maintainable wrt. refactoring or changes in the definitions

Motivation: asymptotic complexity proofs

In the context of *A Fistful of Dollars: Formalizing Asymptotic Complexity Claims via Deductive Program Verification*, Guéneau, Charguéraud, Pottier, ESOP'18.

```
Goal
  ∃ (f: nat → nat),
    (∀ n,
     1 + (if zerop n then 0 else 1 + max (f (n / 2)) (f (n − (n / 2) − 1)))
           ⩽ f n)
     ∧
    f ∈ O(log2).
```

Motivation: asymptotic complexity proofs

In the context of *A Fistful of Dollars: Formalizing Asymptotic Complexity Claims via Deductive Program Verification*, Guéneau, Charguéraud, Pottier, ESOP'18.

```
Goal
  ∃ (f: nat → nat),
    (∀ n,
     1 + (if zerop n then 0 else 1 + max (f (n / 2)) (f (n − (n / 2) − 1)))
          ⩽ f n)
     ∧
     f ∈ O(log2).
Proof. exists (fun n ⇒ if zerop n then 1 else 2 * log2 n + 3).
```

The problem: how to delay providing a witness?

The solution: “use evars!”

The solution: “use evars!”

but maybe not naively…

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
```

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
  ...
  (* ⊢ ?x = 5 *)
```

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
  ...
  (* ⊢ ?x = 5 *)
  reflexivity.
Qed.
```

Useful if *equalities* are discovered about the evar.

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
```

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
  …
  (* ⊢ ?x ⩾ 5 *)
```

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
  …
  (* ⊢ ?x ⩾ 5 *)
  (* x := 5 ? *)
```

Basic evar workflow

```
Goal ∃ x, P x.
Proof.
  eexists.
  (* ⊢ P ?x *)
  …
  (* ⊢ ?x ⩾ 5 *)
  (* x := 5 ? *)

  …
  (* ⊢ ?x ⩾ 8 *)
```

“Big-enough”

Cyril Cohen’s *bigenough* library enables new tricks with evars:

```
Goal ∃ x, P x.
Proof.
  exists_big x nat. (* x := ... ⊢ P x *)
```

“Big-enough”

Cyril Cohen’s *bigenough* library enables new tricks with evars:

```
Goal ∃ x, P x.
Proof.
  exists_big x nat. (* x := ... ⊢ P x *)
  …
  (* x := ... ⊢ x ⩾ 5 *)
```

“Big-enough”

Cyril Cohen’s *bigenough* library enables new tricks with evars:

```
Goal ∃ x, P x.
Proof.
  exists_big x nat. (* x := ... ⊢ P x *)
  …
  (* x := ... ⊢ x ⩾ 5 *)
  big.
```

“Big-enough”

Cyril Cohen’s *bigenough* library enables new tricks with evars:

```
Goal ∃ x, P x.
Proof.
  exists_big x nat. (* x := ... ⊢ P x *)
  …
  (* x := ... ⊢ x ⩾ 5 *)
  big.
  …
  (* x := ... ⊢ x ⩾ 8 *)
  big.
```

“Big-enough”

Cyril Cohen’s *bigenough* library enables new tricks with evars:

```
Goal ∃ x, P x.
Proof.
  exists_big x nat. (* x := ... ⊢ P x *)
  …
  (* x := ... ⊢ x ⩾ 5 *)
  big.
  …
  (* x := ... ⊢ x ⩾ 8 *)
  big.
  …
  close.
Qed.
```

Useful if *lower bounds* are discovered about x.

Builds an iterated maximum.

“Big-enough”

- What if arbitrary side-conditions can be encountered?
  E.g. “$x \geqslant 5$” and “$x \leqslant 8$”

- We might also be inferring not an integer but a function f satisfying “$\forall n,\ f(n) \geqslant 1 + f(n-1)$”

Hard to guess the shape of the solution beforehand (e.g. as an iterated maximum).

Procrastination, as a small Coq library

- Allows deferring arbitrary side-conditions about zero or several variables;
- Does not try to solve these, simply gathers them;
- **Enforces** a separation of the proof in two phases: (1) side conditions are collected; (2) side conditions are solved and variables are instantiated.

Demo

Procrastination: key properties

- When collecting side-conditions about variables, these variables are “rigid”
- Side-conditions are collected and gathered under a common context, in a single subgoal
- `begin defer` .. `end defer` blocks can be arbitrarily nested

Implementing Procrastination in a nutshell

```
Lemma defer_1 : ∀A (P: Prop) (Q: A → Prop),
  (∀ a, Q a → P) →
  (∃ a, Q a) →
  P.
```

```
begin defer assuming a :=
  eapply defer_1; [intros a ? |].
```

Start procrastinating now!

opam install coq-procrastination

```
Require Import Procrastination.Procrastination.
```

https://github.com/Armael/coq-procrastination