Proof Trick: Small Inversions[1]

J.-F. Monin

Univ. Joseph Fourier and
LIAMA, Tsinghua Univ. , Beijing

Coq 2, Edinburgh, July 2010

[1]Supported by ANR project SIVES

Simple examples

Consider the following predicate

```
Inductive mul3 : nat -> Prop :=
  | T0 : mul3 0
  | T3: ∀n, mul3 n -> mul3 (3 + n).
```

And the following goals:

```
mul3 2 -> C
mul3 10 -> 0 = 1
∀n, mul3 (3 + n) -> mul3 n
∀n, mul3 n -> {t | div3spec t n}.
```

where

```
Inductive div3spec: nat -> nat -> Prop :=
 | D0 : div3spec 0 0
 | D3 : ∀d n, div3spec d n -> div3spec (S d) (3+n).
```

Solution: inversion

```
Lemma absurd2_inv : mul3 2 -> C.
Proof.
intros H. inversion_clear H.
Qed.
```

```
fun H : mul3 2 =>
let H0 :=
  match H in (mul3 n) return (n = 2 -> C) with
  | T0 =>
      fun H0 : 0 = 2 =>
      (fun H1 : 0 = 2 =>
       let H2 :=
         eq_ind 0
           (fun e : nat => match e with
                           | 0 => True
                           | S _ => False
                           end) I 2 H1 in
       (fun H3 : False => False_ind C H3) H2) H0
```

```
| T3 n H0 =>
      fun H1 : 3 + n = 2 =>
      let H2 :=
        (fun H2 : 3 + n = 2 =>
         let H3 :=
           eq_ind (3 + n)
             (fun e : nat =>
              match e with
              | 0 => False
              | 1 => False
              | 2 => False
              | S (S (S _)) => True
              end) I 2 H2 in
         (fun H4 : False => False_ind (mul3 n -> C) H4) H3) H1 in
      H2 H0
```

Similarly

```
Lemma inv_mul_3plusn_inv :
  ∀n, mul3 (3 + n) -> mul3 n.
Proof.
intros n m. inversion_clear m.
(*
1 subgoal

  n : nat
  H : mul3 n
  ============================
   mul3 n
*)

assumption.
```

Similarly

```
Proof: fun (n : nat) (m : mul3 (3 + n)) =>
       let H :=
         match m in (mul3 n0) return (n0 = 3 + n -> mul3 n) with
         | T0 =>
             fun H : 0 = 3 + n =>
             (fun H0 : 0 = 3 + n =>
              let H1 :=
                eq_ind 0
                  (fun e : nat => match e with
                                  | 0 => True
                                  | S _ => False
                                  end) I (3 + n) H0 in
              (fun H2 : False => False_ind (mul3 n) H2) H1) H
         | T3 n0 H =>
             fun H0 : 3 + n0 = 3 + n =>
             let H1 :=
               (fun H1 : 3 + n0 = 3 + n =>
                let H2 :=
                  f_equal
                    (fun e : nat =>
                     match e with
                     | 0 => n0
                     | 1 => n0
                     | 2 => n0
                     | S (S (S n3)) => n3
                     end) H1 in
                (fun H3 : n0 = n =>
                 eq_ind n (fun n1 : nat => mul3 n1 -> mul3 n)
                   (fun H4 : mul3 n => H4) n0 (sym_eq H3)) H2) H0 in
             H1 H
         end in
       H (refl_equal (3 + n))
```

More realistic example

More realistic example

Not enough room on this slide

Standard inversion

- *Very convenient*, but

Standard inversion

- *Very convenient*, but
- Generates big terms

Standard inversion

- *Very convenient*, but
- Generates big terms
  - corresponding explanation difficult to follow
    an intuitive idea becomes rather mysterious

Standard inversion

- *Very convenient*, but
- Generates big terms
  - corresponding explanation difficult to follow
    an intuitive idea becomes rather mysterious
  - in practice, cannot be inlined in function definitions
    (e.g. above example for division by 3)

Standard inversion

- *Very convenient*, but
- Generates big terms
  - corresponding explanation difficult to follow
    an intuitive idea becomes rather mysterious
  - in practice, cannot be inlined in function definitions
    (e.g. above example for division by 3)
  - uses irrelevant notions (impurity of the method):
    `False_rec, I, refl_equal, eq_ind`

Demystify inversion

Handcrafted inversion

Outline

Illustration

General trick

More advanced examples

Outline

Illustration

```
Lemma absurd2_interact : mul3 2 -> C.
Proof.
intros m.
pose (diag x := match x with 2 => C | _ => True end).
change (diag 2). case m; clear m.
(*
  ============================
   diag 0
*)
  simpl. trivial.
(*
  ============================
   ∀ n : nat, mul3 n -> diag (3 + n)
*)
  simpl. trivial.
Qed.
```

Illustration

```
Definition absurd2_prog : mul3 2 -> C :=
  fun H =>
  let diag x := match x with 2 => C | _ => True end
  in
    match H in (mul3 n) return (diag n) with
      | T0 => I
      | T3 _ _ => I
    end.
```

For fun: without True as well

```
Definition absurd2_prog : mul3 2 -> C :=
  fun H =>
  let diag x := match x with 2 => C | _ => True end
  in
    match H in (mul3 n) return (diag n) with
      | T0 => I
      | T3 _ _ => I
    end.

Definition absurd2_prog_pure : mul3 2 -> C :=
  fun H =>
  let diag x := match x with 2 => C | _ => mul3 2 end
  in
    match H in (mul3 n) return (diag n) with
      | T0 => H
      | T3 _ _ => H
    end.
```

Similarly

Remark: variant with small elimination

```
Lemma inv_mul_3plusn :
    ∀n, mul3 (3 + n) -> mul3 n.
Proof.
intros n m.
exact
(let diag x :=
     match x with S (S (S y)) => y | _ => 3 + n end
 in
     match m in (mul3 n0) return mul3 (diag n0) with
     | T0 => m
     | T3 _ m0 => m0
     end
).
Qed.
```

Outline

General trick

Assume:

• an inductive predicate $P : D \longrightarrow \texttt{Prop}$ having n constructors $C_i : P\ t_i$, with $1 \leq i \leq n$

• a hypothesis $H : P\ t$ such that t matches no term among $\{t_i \mid 1 \leq i \leq n\}$.

Interactive mode

```
pose (diag x := match x with t => C | _ => P t end).
```

where C is the conclusion of the current subgoal, and proceed by case on H.

Direct definition

```
let diag x := match x with t => C | _ => P t in
match H in (P x) return (diag x) with
| C_i  _ ... => H
end.
```

Outline

A non determininistic relation

f g: nat-> nat c: nat

```
Inductive nd : nat -> Prop :=
  | Nc: nd c
  | Nf: forall n, nd n -> nd (f n)
  | Ng: forall n, nd n -> nd (g n).
```

Exponential blow-up : each inversion generates 2 subgoals

Example

```
Definition tr15 := nd (plus 6) (plus 9) 15.
```

A non determininistic relation

```
Lemma tr15 : tr15 40 -> tr15 p.
Proof.
intros t40.
pose (diag n := if nat_beq n 40 then p else 40).
change (tr15 (diag 40)).
assert (gen: forall n, tr15 n -> tr15 (diag n)).
  repeat (exact t40 ||
    intros n t; exact t40 || (case t; clear t n)).
  apply gen; exact t40.
Qed.
```

Much smaller size of .vo files: 36 Ko against 364 K.

Operational Semantics

Taken from *Software Foundations*, B. Pierce

```
Inductive ceval : state -> com -> state -> Prop :=
  | CESkip : forall st,  ceval st CSkip st
  | CEAss  : forall st a1 n l, aeval st a1 = n ->
      (CAss l a1) / st ~~> (override st l n)
  | CESeq : forall c1 c2 st st' st'',
      c1 / st  ~~> st' ->  c2 / st' ~~> st'' ->
      (CSeq c1 c2) / st ~~> st''
```

etc.

```
  where "c1 '/' st '~~>' st'" := (ceval st c1 st').

Definition plus2 : com := X ::= !X +++ A2.
```

Operational Semantics

```
Theorem plus2_spec:
 ∀ st n st', st X = n -> plus2 / st ~~> st' ->
 st' X = n + 2.
Proof.
refine
 ((let e := !X +++ A2 in
   let Heqe := refl_equal e in
     (let diag x y z :=
         match y with
           | v' ::= e' => e' = !v' +++ A2 ->
                          x v' = n -> z v' = n + 2
           | _ => (X ::= e) / st ~~> st'
         end in
         match Heval in (c / s ~~> s')
         return (diag s c s') with
           | CEAss st0 a1 n0 l Heqn0 => _
           |  _ => Heval
         end) Heqe) HX).
```

Can be captured by Ltac.

Conclusion

Small terms for inversion can be obtained

Scripts non trivial but still short

Flexible technique

Strong elimination can be avoided
when the result has sort Prop

More experiments are needed

Conclusion

Small terms for inversion can be obtained

Scripts non trivial but still short

Flexible technique

Strong elimination can be avoided
when the result has sort Prop

More experiments are needed

THANKS