Strengthening the *inversion* Tactic in Coq

Anne Mulhern

Department of Computer Sciences
University of Wisconsin-Madison

July 9, 2010

The *inversion* Tactic in Coq

The *destruct* tactic on steroids.

Example : `False`

Inductive False : Prop :=

destruct and *inversion*

	H : False / the goal
destruct H	✓
inversion H	✓

Example: `and`

```
Inductive and (A B : Prop) : Prop :=
    conj : A -> B -> A /\ B
```

destruct and *inversion*

	$H : A \wedge B$ the goal
destruct H	$H : A$ $H0 : B$ the goal
inversion H	$H : A \wedge B$ $H0 : A$ $H1 : B$ the goal

Example: ex

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
    ex_intro : forall x : A, P x -> ex P
```

destruct and *inversion*

	H : ex P goal in *Prop*	H : ex P goal in *{Set, Type}*
destruct H	H : A H0 : P x goal in *Prop*	failure
inversion H	*H : ex P* x : A H0 : P x goal in *Prop*	failure

Keeping the Universes Separate

Allowed

```
proj1_sig =
fun (A : Type) (P : A -> Prop) (e : sig P) =>
let (a, _) := e in a
     : forall (A : Type) (P : A -> Prop),
       sig P -> A
```

Forbidden

```
proj1_ex =
fun (A : Type) (P : A -> Prop) (e : ex P) =>
let (a, _) := e in a
     : forall (A : Type) (P : A -> Prop),
       ex P -> A
```

Keeping the Universes Separate

It is always possible to invert an hypothesis under any of the following conditions:

1. The type of the hypothesis is in *Set* or *Type* or the type of the goal is in *Prop*.
2. The type of the hypothesis has at most one constructor and the types of the arguments to that constructor are all in *Prop*, e.g., `and`.

   `False` zero constructors

   `and` just one constructor with two arguments in *Prop*

It should also be possible to invert an hypothesis under the following condition:

Rule 3

It is possible to construct a function that, when applied to the hypothesis, yields a result that satisfies condition 2.

Rule 3

It is possible to construct a function that, when applied to the hypothesis, yields a result that satisfies condition 2 and the result type of the function is strongly related to the hypotheses with which the *inversion* tactic would have supplied the proof context if it had been able to proceed.

Example: Locally Nameless Representation

```
Inductive exp : Set :=
    var_b : nat -> exp
  | var_f : expvar -> exp
  | app : exp -> exp -> exp
  | abs : exp -> exp

Inductive lc_exp : exp -> Prop :=
    lc_var_f : forall x : expvar, lc_exp (var_f x)
  | lc_app : forall e1 e2 : exp,
             lc_exp e1 -> lc_exp e2 -> lc_exp (app e1 e2)
  | lc_abs : forall e : exp,
             (forall x : expvar,
              lc_exp (open_exp_wrt_exp e (var_f x))) ->
             lc_exp (abs e)
```

http://www.cis.upenn.edu/~baydemir/papers/lngen/index.html

Example: Locally Nameless Representation

destruct and *inversion*

		e : exp H : lc_exp e goal in *Prop*	
destruct H	x : expvar goal in *Prop*	e1 : exp e2 : exp H : lc_exp e1 H0 : lc_exp e2 goal in *Prop*	e : exp H : forall x : expvar, . . . goal in *Prop*
inversion H	*e : exp* *H : lc_exp e* x : expvar H0 : var_f x = e goal in *Prop*	*e : exp* *H : lc_exp e* e1 : exp e2 : exp H0 : lc_exp e1 H1 : lc_exp e2 H2 : app e1 e2 = e goal in *Prop*	*e : exp* *H : lc_exp e* e0 : exp H0 : forall x : expvar, . . . H1 : abs e0 = e goal in *Prop*

Powerful statement about equality not on, `H`, the inverted hypothesis, but on `e`, the value on which it is dependent.

Example: Locally Nameless Representation

inversion

	simple inversion H	*inversion* H
	H : lc_exp (abs (var_b 0)) x : expvar H0 : var_f x = abs (var_b 0) goal in *Prop*	✓
H : lc_exp (abs (var_b 0)) goal in *Prop*	*H : lc_exp (abs (var_b 0))* e1 : exp e2 : exp H0 : lc_exp e1 H1 : lc_exp e2 H2 : app e1 e2 = abs (var_b 0) goal in *Prop*	✓
	H : lc_exp (abs (var_b 0)) e0 : exp H0 : forall x : expvar, . . . H1 : abs e0 = abs (var_b 0) goal in *Prop*	*H : lc_exp (abs (var_b 0))* e0 : exp H0 : forall x : expvar, . . . H1 : e0 = var_b 0 goal in *Prop*

Equality statements allow the *inversion* tactic to eliminate all but one of the remaining subgoals, as long as the goal is in *Prop*.

Example: Locally Nameless Representation

	simple inversion H	*inversion* H
	H : lc_exp (abs (var_b 0)) x : expvar H0 : var_f x = abs (var_b 0) goal in *Prop*	✓
H : lc_exp (abs (var_b 0)) goal in *Prop*	*H : lc_exp (abs (var_b 0))* e1 : exp e2 : exp H0 : lc_exp e1 H1 : lc_exp e2 H2 : app e1 e2 = abs (var_b 0) goal in *Prop*	✓
	H : lc_exp (abs (var_b 0)) e0 : exp H0 : forall x : expvar, . . . H1 : abs e0 = abs (var_b 0) goal in *Prop*	*H : lc_exp (abs (var_b 0))* e0 : exp **H0 : forall x : expvar, . . .** H1 : e0 = var_b 0 goal in *Prop*

It is possible to extract the statement of a lemma from this table, i.e.,

```
lc_exp (abs (var_b 0))
→
forall x : expvar, lc_exp (open_exp_wrt_exp (var_b 0) (var_f x))
```

Example: Locally Nameless Representation

Hypothesis	Lemma
lc_exp (abs *e*)	lc_exp (abs e) $\rightarrow$ forall x : expvar, lc_exp (open_exp_wrt_exp *e* (var_f x))
lc_exp (app *e1* *e2*)	lc_exp (app e1 e2) $\rightarrow$ lc_exp e1 $\wedge$ lc_exp e2
lc_exp (var_f *x*)	lc_exp (var_f *x*) $\rightarrow$ True
lc_exp (var_b *x*)	lc_exp (var_b *x*) $\rightarrow$ False

Rule 3

It is possible to construct a function that, when applied to the hypothesis, yields a result that satisfies condition 2.

General Approach

Within my *stronger_inversion* tactic, automatically construct the necessary inversion lemma and apply it to the hypothesis to be inverted, generalizing the result and thereby inserting it within the context.

Arguments in *Set* or *Type*

- *stronger_inversion* derives contradictions to eliminate generated subgoals, in the same way as the *inversion* tactic.
- However, it cannot insert hypotheses with types in *Set* or *Type* into the context.
- If hypotheses with types in *Set* or *Type* occur among the constructor's arguments they must be eliminated somehow.

Non-example: hypothesis is equal to a constant

	simple inversion H	*stronger_inversion* H
	H : lc_exp (abs (var_b 0)) x : expvar H0 : var_f x = abs (var_b 0) --- the goal in *Prop*	✓
H : lc_exp (abs (var_b 0)) --- the goal in *Prop*	*H : lc_exp (abs (var_b 0))* e1 : exp e2 : exp H0 : lc_exp e1 H1 : lc_exp e2 H2 : app e1 e2 = abs (var_b 0) --- the goal in *Prop*	✓
	H : lc_exp (abs (var_b 0)) e0 : exp H0 : forall x : expvar, . . . H1 : abs e0 = abs (var_b 0) --- the goal in *Prop*	*H : lc_exp (abs (var_b 0))* ~~e0 : exp~~ **H0 : forall x : expvar, . . .** ~~H1 : e0 = var_b 0~~ --- the goal in { *Set, Type* }

Substitution eliminates `e0`, i.e., it can be replaced everywhere with the right hand side.

Non-example: hypotheses are equal to existing variables

	simple inversion H	*stronger_inversion* H
	e0 : exp e1 : exp *H : lc_exp (app e0 e1)* x : expvar H0 : var_f x = app e0 e1 --- the goal in *Prop*	✓
e0 : exp **e1 : exp** **H : lc_exp (app e0 e1)** --- the goal in *Prop*	*e0 : exp* *e1 : exp* *H : lc_exp (app e0 e1)* e2 : exp e3 : exp H0 : lc_exp e2 H1 : lc_exp e3 H2 : app e2 e3 = app e0 e1 --- the goal in *Prop*	*e0 : exp* *e1 : exp* *H : lc_exp (app e0 e1)* ~~e2 : exp~~ ~~e3 : exp~~ H0 : lc_exp e2 → lc_exp e0 H1 : lc_exp e3 → lc_exp e1 ~~H2 : app e2 e3 = app e0 e1~~ --- the goal in { *Set, Type* }
	e0 : exp *e1 : exp* *H : lc_exp (app e0 e1)* e0 : exp H0 : forall x : expvar, . . . H1 : abs e0 = abs (var_b 0) --- the goal in *Prop*	✓

Another substitution example.

Example: Bad Version of Locally Nameless Representation

```
Inductive exp : Set :=
    var_b : nat -> exp
  | var_f : expvar -> exp
  | app : exp -> exp -> exp
  | abs : exp -> exp

Inductive lc_exp : exp -> Prop :=
    lc_var_f : forall x : expvar, lc_exp (var_f x)
  | lc_app : forall e1 e2 : exp,
             lc_exp e1 -> lc_exp e2 -> lc_exp (app e1 e2)
  | lc_abs : forall (e : exp) (x : expvar),
              lc_exp (open_exp_wrt_exp e (var_f x)) ->
             lc_exp (abs e)

Inductive lc_exp : exp -> Prop :=
    lc_var_f : forall x : expvar, lc_exp (var_f x)
  | lc_app : forall e1 e2 : exp,
             lc_exp e1 -> lc_exp e2 -> lc_exp (app e1 e2)
  | lc_abs : forall e : exp,
             (forall x : expvar,
              lc_exp (open_exp_wrt_exp e (var_f x))) ->
             lc_exp (abs e)
```

Implementation

- Entirely within the *Ltac* language.
- Makes use of *external* tactic to memoize inversion lemmas.

Conclusion

- A generalization of the *inversion* tactic.
- Memoizes inversion lemmas for reuse via the *external* tactic.
- The *Derive Inversion* command is consistent with the *inversion* tactic and should be changed consistently.

A Generalization of the *inversion* Tactic

- An untypable *match* expression is transformed to an equivalent and typable application. The problematical *match* expression is encapsulated in the abstraction.
- Instead of all this complexity and building an abstraction would it be better just to change the type system to allow *match* expressions in those cases in which *stronger_inversion* can proceed?[1]

[1] Perhaps extraction could make use of closed union types?

Memoization

- A tactic that automates memoization might be useful in many contexts.
- There are a number of possible approaches.
- An automatic replacement for the *Derive Inversion* vernacular command.

Strengthening the *inversion* Tactic in Coq

Anne Mulhern

Department of Computer Sciences
University of Wisconsin-Madison

July 9, 2010