Printed by Zach Tatlock

```
StructTactics.v
                                                                                                                             StructTactics.v
 Oct 17, 16 7:17
                                                                           Page 1/7
                                                                                          Oct 17, 16 7:17
                                                                                                                                                                    Page 2/7
(** https://github.com/uwplse/StructTact *)
                                                                                                    | [H : exists (name : _), _ |- _ ] =>
                                                                                                      let x := fresh name in
                                                                                                      destruct H as [x]
Ltac subst max :=
 repeat match goal with
                                                                                                  end.
          | [ H : ?X = _ |- _ ] => subst X
| [H : _ = ?X |- _] => subst X
                                                                                        Ltac break_exists_exists :=
         end.
                                                                                          repeat match goal with
                                                                                                    | H:exists _, _ |- _ =>
                                                                                                      let x := fresh "x" in
Ltac inv H := inversion H; subst_max.
Ltac invc H := inv H; clear H.
                                                                                                      destruct H as [x]; exists x
Ltac invcs H := invc H; simpl in *.
                                                                                                  end.
Ltac break if :=
                                                                                        Ltac break and :=
 match goal with
                                                                                          repeat match goal with
   | [ |- context [ if ?X then _ else _ ] ] =>
                                                                                                  | [H : _ /\ _ |- _ ] => destruct H
      match type of X with
                                                                                                  end.
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
                                                                                         Ltac break_and_goal :=
      end
                                                                                            repeat match goal with
                                                                                                    | [ |- _ /\ _ ] => split
    | [ H : context [ if ?X then _ else _ ] |- _] =>
      match type of X with
                                                                                                    end
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
                                                                                        Ltac solve_by_inversion' tac :=
                                                                                          match goal with
      end
 end.
                                                                                           | [H : _ |- _] => solve [inv H; tac]
                                                                                           end.
Ltac break_match_hyp :=
 match goal with
                                                                                         Ltac solve by inversion := solve by inversion' auto.
   | [ H : context [ match ?X with _ => _ end ] |- _] =>
      match type of X with
                                                                                         Ltac apply_fun f H:=
       | sumbool _ _ => destruct X
                                                                                          match type of H with
        | _ => destruct X eqn:?
                                                                                            | ?X = ?Y => assert (f X = f Y)
      end
                                                                                           end.
 end.
                                                                                        Ltac conclude H tac :=
                                                                                           (let H' := fresh in
Ltac break_match_goal :=
 match goal with
                                                                                           match type of H with
   [ [ - context [ match ?X with => end ] ] =>
                                                                                            | ?P -> => assert P as H' by (tac)
      match type of X with
                                                                                            end; specialize (H H'); clear H').
       | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
                                                                                         Ltac concludes :=
      end
                                                                                          match goal with
 end.
                                                                                            | [ H : ?P -> _ |- _ ] => conclude H auto
                                                                                           end.
Ltac break_match := break_match_goal || break_match_hyp.
                                                                                         Ltac forward H :=
Ltac break_inner_match' t :=
                                                                                          let H' := fresh in
match t with
                                                                                           match type of H with
  context[match ?X with _ => _ end] =>
                                                                                            | ?P -> _ => assert P as H'
    break_inner_match' X || destruct X eqn:?
                                                                                            end.
  | _ => destruct t eqn:?
end.
                                                                                         Ltac forwards :=
                                                                                          match goal with
                                                                                           | [ H : ?P -> _ |- _ ] => forward H
Ltac break_inner_match_goal :=
match goal with
                                                                                           end.
 [ [ - context[match ?X with _ => _ end] ] =>
                                                                                         Ltac find_contradiction :=
    break_inner_match' X
end.
                                                                                          match goal with
                                                                                           | [ H : ?X = _, H' : ?X = _ |- _ ] => rewrite H in H'; solve_by_inversion
Ltac break_inner_match_hyp :=
                                                                                           end.
match goal with
 [ H : context[match ?X with _ => _ end] |- _ ] =>
                                                                                         Ltac find_rewrite :=
    break_inner_match' X
                                                                                          match goal with
                                                                                            | [ H : ?X _ _ _ = _, H' : ?X _ _ _ = _ |- _ ] => rewrite H in H'
| [ H : ?X = _, H' : ?X = _ |- _ ] => rewrite H in H'
| [ H : ?X = _, H' : context [ ?X ] |- _ ] => rewrite H in H'
| [ H : ?X = _ |- context [ ?X ] ] => rewrite H
end.
Ltac break_inner_match := break_inner_match_goal || break_inner_match_hyp.
Ltac break_exists :=
                                                                                           end.
 repeat match goal with
```

Printed by Zach Tatlock

Page 4/7

```
StructTactics.v
                                                                                                                                      StructTactics.v
 Oct 17, 16 7:17
                                                                                Page 3/7
                                                                                                 Oct 17, 16 7:17
Ltac find erewrite :=
                                                                                                  end.
 match goal with
   | [ H : ?X _ _ _ = _, H' : ?X _ _ _ = _ |- _ ] => erewrite H in H'
| [ H : ?X = _, H' : ?X = _ |- _ ] => erewrite H in H'
| [ H : ?X = _, H' : context [ ?X ] |- _ ] => erewrite H in H'
                                                                                               Ltac break_let :=
                                                                                                 match goal with
                                                                                                   | [ H : context [ (let (_,_) := ?X in _) ] |- _ ] => destruct X eqn:?
    | [ H : ?X = _ |- context [ ?X ] ] => erewrite H
                                                                                                   | [ |- context [ (let (_,_) := ?X in _) ] ] => destruct X eqn:?
 end.
                                                                                                 end.
Ltac find_rewrite_lem lem :=
                                                                                               Ltac break_or_hyp :=
 match goal with
                                                                                                 match goal with
    | [ H : _ |- _ ] =>
                                                                                                  | [ H : _ \/ _ |- _ ] => invc H
      rewrite lem in H; [idtac]
                                                                                                 end.
 end.
                                                                                               Ltac copy_apply lem H :=
Ltac find_rewrite_lem_by lem t :=
                                                                                                let x := fresh in
                                                                                                pose proof H as x;
 match goal with
   | [ H : _ |- _ ] =>
                                                                                                   apply lem in x.
     rewrite lem in H by t
 end.
                                                                                               Ltac copy_eapply lem H :=
                                                                                                let x := fresh in
Ltac find_erewrite_lem lem :=
                                                                                                 pose proof H as x;
 match goal with
                                                                                                   eapply lem in x.
  | [ H : _ |- _] => erewrite lem in H by eauto
                                                                                               Ltac conclude_using tac :=
 end.
                                                                                                 match goal with
Ltac find_reverse_rewrite :=
                                                                                                  | [ H : ?P -> _ |- _ ] => conclude H tac
 match goal with
                                                                                                 end.
   | [ H : _ = ?X _ _ _, H' : ?X _ _ _ = _ |- _ ] => rewrite <- H in H'
| [ H : _ = ?X, H' : context [ ?X ] |- _ ] => rewrite <- H in H'
                                                                                               Ltac find_higher_order_rewrite :=
    | [ H : _ = ?X |- context [ ?X ] ] => rewrite <- H
                                                                                                 match goal with
 end.
                                                                                                   | [ H : _ = _ |- _ ] => rewrite H in *
                                                                                                   | [ H : forall _, _ = _ |- _ ] => rewrite H in *
Ltac find_inversion :=
                                                                                                   | [ H : forall ____ = _ |- _ ] => rewrite H in *
 match goal with
                                                                                                 end
        H : ?X _ _ _ = ?X _ _ _ |- _ ] => invc H
    | [ H : ?X _ _ _ = ?X _ _ _ |- ] => invc H
| [ H : ?X _ _ = ?X _ _ _ |- ] => invc H
                                                                                               Ltac find_reverse_higher_order_rewrite :=
                                                                                                 match goal with
   | [ H : 2X _ _ = 2X _ _ |-_] => invc H
| [ H : 2X _ _ = 2X _ _ |-_] => invc H
| [ H : 2X _ = 2X _ |-_] => invc H
                                                                                                   | [ H : _ = _ |- _ ] => rewrite <- H in *
                                                                                                   | [ H : forall _, _ = _ |- _ ] => rewrite <- H in *
| [ H : forall _, _ = _ |- _ ] => rewrite <- H in *
 end.
                                                                                                 end.
Ltac prove eg :=
                                                                                               Ltac clean :=
 match goal with
                                                                                                match goal with
   | [ H : ?X ?x1 ?x2 ?x3 = ?X ?y1 ?y2 ?y3 |- _ ] =>
                                                                                                  | [ H : ?X = ?X |- _ ] => clear H
      assert (x1 = y1) by congruence;
                                                                                                 end.
        assert (x^2 = y^2) by congruence;
        assert (x3 = y3) by congruence;
                                                                                               Ltac find_apply_hyp_goal :=
        clear H
                                                                                                match goal with
    | [ H : ?X ?x1 ?x2 = ?X ?y1 ?y2 |- _ ] =>
                                                                                                  | [ H : _ |- _ ] => solve [apply H]
      assert (x1 = y1) by congruence;
                                                                                                 end.
        assert (x^2 = y^2) by congruence;
        clear H
                                                                                               Ltac find_copy_apply_lem_hyp lem :=
    | [ H : ?X ?x1 = ?X ?y1 |- _ ] =>
                                                                                                match goal with
      assert (x1 = y1) by congruence;
                                                                                                  | [ H : _ |- _ ] => copy_apply lem H
        clear H
                                                                                                 end.
 end.
                                                                                               Ltac find_apply_hyp_hyp :=
Ltac tuple_inversion :=
                                                                                                 match goal with
                                                                                                  | [ H : forall _, _ -> _,
H' : _ |- _ ] =>
apply H in H'; [idtac]
 match goal with
   | [ H : (_, _, _, _) = (_, _, _, _) |- _ ] => invc H
| [ H : (_, _, _) = (_, _, _) |- _ ] => invc H
    | [ H : (_, _) = (_, _) |- _ ] => invc H
                                                                                                   | [ H : _ -> _ , H' : _ |- _ ] =>
 end.
                                                                                                     apply H in H'; auto; [idtac]
                                                                                                 end.
Ltac f_apply H f :=
match type of H with
                                                                                               Ltac find_copy_apply_hyp_hyp :=
   | ?X = ?Y =>
                                                                                                 match goal with
      assert (f X = f Y) by (rewrite H; auto)
                                                                                                   | [ H : forall _, _ -> _,
```

Oct 17, 16 7:17 StructTactics.	V Page 5/7	Oct 17, 16 7:17	StructTactics.v	Page 6/7
<pre>H' : _] => copy_apply H H'; [idtac] [H :> _ , H' : _] => copy_apply H H'; auto; [idtac] end.</pre>		remGenIfNotVar f remGenIfNotVar g ?a ?b ?c ?d ?e remGenIfNotVar a remGenIfNotVar b remGenIfNotVar c	?f =>	
<pre>Ltac find_apply_lem_hyp lem := match goal with</pre>		remGenIfNotVar d remGenIfNotVar e remGenIfNotVar e [_ ?a ?b ?c ?d ?e remGenIfNotVar a	=>	
<pre>Ltac find_eapply_lem_hyp lem := match goal with</pre>		remGenIfNotVar b remGenIfNotVar c remGenIfNotVar d remGenIfNotVar e	;	
<pre>Ltac insterU H := match type of H with forall _: ?T, _=> let x := fresh "x" in evar (x : T); let x' := (eval unfold x in x) in clear x; specialize (H x') end.</pre>		<pre> _ ?a ?b ?c ?d => remGenIfNotVar a remGenIfNotVar b remGenIfNotVar c remGenIfNotVar d _ ?a ?b ?c => remGenIfNotVar a remGenIfNotVar c</pre>	;	
Ltac find_insterU := match goal with		<pre> _ ?a ?b => remGenIfNotVar a remGenIfNotVar b _ ?a => remGenIfNotVar a end.</pre>	;	
<pre>Ltac eapply_prop P := match goal with H : P _ => eapply H end. Ltac isVar t := match goal with v : _ => match t with v => idtac end end.</pre>		Ltac generalizeEveryth repeat match goal wi [x : ?T first [match x end match	th] => H with => fail 2 type of H with pontext [x] => fail 2	
<pre>Ltac remGen t := let x := fresh in let H := fresh in remember t as x eqn:H; generalize dependent H.</pre>		<pre>Ltac prep_induction H rememberNonVars H; generalizeEverything! Ltac econcludes := match goal with</pre>		
<pre>Ltac remGenIfNotVar t := first [isVar t remGen t] Ltac rememberNonVars H := match type of H with _ ?a ?b ?c ?d ?e ?f ?g ?h => remGenIfNotVar a; remGenIfNotVar c; remGenIfNotVar c; remGenIfNotVar c; remGenIfNotVar f; remGenIfNotVar f; remGenIfNotVar h _ ?a ?b ?c ?d ?e ?f ?g => remGenIfNotVar a; remGenIfNotVar b;</pre>		<pre> [H : ?P -> _ - end. Ltac find_copy_eapply match goal with</pre>	<pre>> copy_eapply lem H 2 :=], H' : context [Q]] =></pre>	
remGenIfNotVar c; remGenIfNotVar d; remGenIfNotVar e;		match goal with	<pre>2 .], H' : context [Q]] =></pre>	

<pre>end. Ltac copy_eapply_prop_hyp P Q := match goal with</pre>	
<pre>match goal with</pre>	
<pre>match goal with H :> False => exfalso; apply H end. Ltac injc H :=</pre>	
Ltac find_injection := match goal with [H: ?X = ?X] => injc H [H: ?X _ = ?X] => injc H end.	
<pre>Ltac aggressive_rewrite_goal := match goal with H : _ => rewrite H end.</pre>	
<pre>Ltac break_exists_name x := match goal with [H : exists _, _] => destruct H as [x H] end.</pre>	
<pre>Tactic Notation "unify" uconstr(x) "with" uconstr(y) := let Htmp := fresh "Htmp" in refine (let Htmp : False -> x := fun false : False => match false return y with end in _); clear Htmp.</pre>	
<pre>Tactic Notation "on" uconstr(x) "," tactic3(tac) := match goal with [H : ?y] => unify x with y; tac H end.</pre>	
(** generic forward reasoning *)	
<pre>Tactic Notation "fwd" tactic3(tac) "as" ident(H) := simple refine (let H : _ := _ in _); [shelve tac clearbody H].</pre>	
Tactic Notation "fwd" tactic3(tac) := let H := fresh "H" in fwd tac as H.	
Ltac ee := econstructor; eauto.	