```
(** https://github.com/uwplse/StructTact *)

Ltac subst_max :=
  repeat match goal with
           | [ H : ?X = _ |- _ ]  => subst X
           | [H : _ = ?X |- _] => subst X
         end.

Ltac inv H := inversion H; subst_max.
Ltac invc H := inv H; clear H.
Ltac invcs H := invc H; simpl in *.

Ltac break_if :=
  match goal with
    | [ |- context [ if ?X then _ else _ ] ] =>
      match type of X with
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
      end
    | [ H : context [ if ?X then _ else _ ] |- _] =>
      match type of X with
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
      end
  end.

Ltac break_match_hyp :=
  match goal with
    | [ H : context [ match ?X with _ => _ end ] |- _] =>
      match type of X with
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
      end
  end.

Ltac break_match_goal :=
  match goal with
    | [ |- context [ match ?X with _ => _ end ] ] =>
      match type of X with
        | sumbool _ _ => destruct X
        | _ => destruct X eqn:?
      end
  end.

Ltac break_match := break_match_goal || break_match_hyp.

Ltac break_inner_match’ t :=
 match t with
   | context[match ?X with _ => _ end] =>
     break_inner_match’ X || destruct X eqn:?
   | _ => destruct t eqn:?
 end.

Ltac break_inner_match_goal :=
 match goal with
   | [ |- context[match ?X with _ => _ end] ] =>
     break_inner_match’ X
 end.

Ltac break_inner_match_hyp :=
 match goal with
   | [ H : context[match ?X with _ => _ end] |- _ ] =>
     break_inner_match’ X
 end.

Ltac break_inner_match := break_inner_match_goal || break_inner_match_hyp.

Ltac break_exists :=
  repeat match goal with
           | [H : exists (name : _), _ |- _ ] =>
             let x := fresh name in
             destruct H as [x]
         end.

Ltac break_exists_exists :=
  repeat match goal with
           | H:exists _, _ |- _ =>
             let x := fresh "x" in
             destruct H as [x]; exists x
         end.

Ltac break_and :=
  repeat match goal with
           | [H : _ /\ _ |- _ ] => destruct H
         end.

Ltac break_and_goal :=
    repeat match goal with
             | [ |- _ /\ _ ] => split
           end.

Ltac solve_by_inversion’ tac :=
  match goal with
    | [H : _ |- _] => solve [inv H; tac]
  end.

Ltac solve_by_inversion := solve_by_inversion’ auto.

Ltac apply_fun f H:=
  match type of H with
    | ?X = ?Y => assert (f X = f Y)
  end.

Ltac conclude H tac :=
  (let H’ := fresh in
   match type of H with
     | ?P -> _ => assert P as H’ by (tac)
   end; specialize (H H’); clear H’).

Ltac concludes :=
  match goal with
    | [ H : ?P -> _ |- _ ] => conclude H auto
  end.

Ltac forward H :=
  let H’ := fresh in
   match type of H with
     | ?P -> _ => assert P as H’
   end.

Ltac forwards :=
  match goal with
    | [ H : ?P -> _ |- _ ] => forward H
  end.

Ltac find_contradiction :=
  match goal with
    | [ H : ?X = _, H’ : ?X = _ |- _ ] => rewrite H in H’; solve_by_inversion
  end.

Ltac find_rewrite :=
  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 find_erewrite :=
  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’
    | [ H : ?X = _ |- context [ ?X ] ] => erewrite H
  end.

Ltac find_rewrite_lem lem :=
  match goal with
    | [ H : _ |- _ ] =>
      rewrite lem in H; [idtac]
  end.

Ltac find_rewrite_lem_by lem t :=
  match goal with
    | [ H : _ |- _ ] =>
      rewrite lem in H by t
  end.

Ltac find_erewrite_lem lem :=
  match goal with
    | [ H : _ |- _] => erewrite lem in H by eauto
  end.

Ltac find_reverse_rewrite :=
  match goal with
    | [ 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 find_inversion :=
  match goal with
    | [ H : ?X _ _ _ _ _ _ = ?X _ _ _ _ _ _ |- _ ] => invc H
    | [ H : ?X _ _ _ _ _ = ?X _ _ _ _ _ |- _ ] => invc H
    | [ H : ?X _ _ _ _ = ?X _ _ _ _ |- _ ] => invc H
    | [ H : ?X _ _ _ = ?X _ _ _ |- _ ] => invc H
    | [ H : ?X _ _ = ?X _ _ |- _ ] => invc H
    | [ H : ?X _ = ?X _ |- _ ] => invc H
  end.

Ltac prove_eq :=
  match goal with
    | [ H : ?X ?x1 ?x2 ?x3 = ?X ?y1 ?y2 ?y3 |- _ ] =>
      assert (x1 = y1) by congruence;
        assert (x2 = y2) by congruence;
        assert (x3 = y3) by congruence;
        clear H
    | [ H : ?X ?x1 ?x2 = ?X ?y1 ?y2 |- _ ] =>
      assert (x1 = y1) by congruence;
        assert (x2 = y2) by congruence;
        clear H
    | [ H : ?X ?x1 = ?X ?y1 |- _ ] =>
      assert (x1 = y1) by congruence;
        clear H
  end.

Ltac tuple_inversion :=
  match goal with
    | [ H : (_, _, _, _) = (_, _, _, _) |- _ ] => invc H
    | [ H : (_, _, _) = (_, _, _) |- _ ] => invc H
    | [ H : (_, _) = (_, _) |- _ ] => invc H
  end.

Ltac f_apply H f :=
  match type of H with
    | ?X = ?Y =>
      assert (f X = f Y) by (rewrite H; auto)
```

```
  end.

Ltac break_let :=
  match goal with
    | [ H : context [ (let (_,_) := ?X in _) ] |- _ ] => destruct X eqn:?
    | [ |- context [ (let (_,_) := ?X in _) ] ] => destruct X eqn:?
  end.

Ltac break_or_hyp :=
  match goal with
    | [ H : _ \/ _ |- _ ] => invc H
  end.

Ltac copy_apply lem H :=
  let x := fresh in
  pose proof H as x;
    apply lem in x.

Ltac copy_eapply lem H :=
  let x := fresh in
  pose proof H as x;
    eapply lem in x.

Ltac conclude_using tac :=
  match goal with
    | [ H : ?P -> _ |- _ ] => conclude H tac
  end.

Ltac find_higher_order_rewrite :=
  match goal with
    | [ H : _ = _ |- _ ] => rewrite H in *
    | [ H : forall _, _ = _ |- _ ] => rewrite H in *
    | [ H : forall _ _, _ = _ |- _ ] => rewrite H in *
  end.

Ltac find_reverse_higher_order_rewrite :=
  match goal with
    | [ H : _ = _ |- _ ] => rewrite <- H in *
    | [ H : forall _, _ = _ |- _ ] => rewrite <- H in *
    | [ H : forall _ _, _ = _ |- _ ] => rewrite <- H in *
  end.

Ltac clean :=
  match goal with
    | [ H : ?X = ?X |- _ ] => clear H
  end.

Ltac find_apply_hyp_goal :=
  match goal with
    | [ H : _ |- _ ] => solve [apply H]
  end.

Ltac find_copy_apply_lem_hyp lem :=
  match goal with
    | [ H : _ |- _ ] => copy_apply lem H
  end.

Ltac find_apply_hyp_hyp :=
  match goal with
    | [ H : forall _, _ -> _,
        H’ : _ |- _ ] =>
      apply H in H’; [idtac]
    | [ H : _ -> _ , H’ : _ |- _ ] =>
      apply H in H’; auto; [idtac]
  end.

Ltac find_copy_apply_hyp_hyp :=
  match goal with
    | [ H : forall _, _ -> _,
        H’ : _ |- _ ] =>
      copy_apply H H’; [idtac]
    | [ H : _ -> _ , H’ : _ |- _ ] =>
      copy_apply H H’; auto; [idtac]
  end.

Ltac find_apply_lem_hyp lem :=
  match goal with
    | [ H : _ |- _ ] => apply lem in H
  end.

Ltac find_eapply_lem_hyp lem :=
  match goal with
    | [ H : _ |- _ ] => eapply lem in H
  end.

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.

Ltac find_insterU :=
  match goal with
    | [ H : forall _, _ |- _ ] => insterU H
  end.

Ltac eapply_prop P :=
  match goal with
    | H : P _ |- _ =>
      apply H
  end.

Ltac isVar t :=
    match goal with
      | v : _ |- _ =>
        match t with
          | v => idtac
        end
    end.

Ltac remGen t :=
  let x := fresh in
  let H := fresh in
  remember t as x eqn:H;
    generalize dependent H.

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 b;
      remGenIfNotVar c;
      remGenIfNotVar d;
      remGenIfNotVar e;
      remGenIfNotVar f;
      remGenIfNotVar g;
      remGenIfNotVar h
    | _ ?a ?b ?c ?d ?e ?f ?g =>
      remGenIfNotVar a;
      remGenIfNotVar b;
      remGenIfNotVar c;
      remGenIfNotVar d;
      remGenIfNotVar e;
```

```
      remGenIfNotVar f;
      remGenIfNotVar g
    | _ ?a ?b ?c ?d ?e ?f =>
      remGenIfNotVar a;
      remGenIfNotVar b;
      remGenIfNotVar c;
      remGenIfNotVar d;
      remGenIfNotVar e;
      remGenIfNotVar f
    | _ ?a ?b ?c ?d ?e =>
      remGenIfNotVar a;
      remGenIfNotVar b;
      remGenIfNotVar c;
      remGenIfNotVar d;
      remGenIfNotVar e
    | _ ?a ?b ?c ?d =>
      remGenIfNotVar a;
      remGenIfNotVar b;
      remGenIfNotVar c;
      remGenIfNotVar d
    | _ ?a ?b ?c =>
      remGenIfNotVar a;
      remGenIfNotVar b;
      remGenIfNotVar c
    | _ ?a ?b =>
      remGenIfNotVar a;
      remGenIfNotVar b
    | _ ?a =>
      remGenIfNotVar a
  end.

Ltac generalizeEverythingElse H :=
  repeat match goal with
           | [ x : ?T |- _ ] =>
             first [
                 match H with
                   | x => fail 2
                 end |
                 match type of H with
                   | context [x] => fail 2
                 end |
                 revert x]
         end.

Ltac prep_induction H :=
  rememberNonVars H;
  generalizeEverythingElse H.

Ltac econcludes :=
  match goal with
    | [ H : ?P -> _ |- _ ] => conclude H eauto
  end.

Ltac find_copy_eapply_lem_hyp lem :=
  match goal with
    | [ H : _ |- _ ] => copy_eapply lem H
  end.

Ltac apply_prop_hyp P Q :=
  match goal with
  | [ H : context [ P ], H’ : context [ Q ] |- _ ] =>
    apply H in H’
  end.


Ltac eapply_prop_hyp P Q :=
  match goal with
  | [ H : context [ P ], H’ : context [ Q ] |- _ ] =>
    eapply H in H’
  end.

Ltac copy_eapply_prop_hyp P Q :=
  match goal with
    | [ H : context [ P ], H' : context [ Q ] |- _ ] =>
      copy_eapply H H'
  end.

Ltac find_false :=
  match goal with
    | H : _ -> False |- _ => exfalso; apply H
  end.

Ltac injc H :=
  injection H; clear H; intros; subst_max.

Ltac find_injection :=
  match goal with
    | [ H : ?X _ _ _ _ _ _ = ?X _ _ _ _ _ _ |- _ ] => injc H
    | [ H : ?X _ _ _ _ _ = ?X _ _ _ _ _ |- _ ] => injc H
    | [ H : ?X _ _ _ _ = ?X _ _ _ _ |- _ ] => injc H
    | [ H : ?X _ _ _ = ?X _ _ _ |- _ ] => injc H
    | [ H : ?X _ _ = ?X _ _ |- _ ] => injc H
    | [ H : ?X _ = ?X _ |- _ ] => injc H
  end.

Ltac aggressive_rewrite_goal :=
  match goal with H : _ |- _ => rewrite H end.

Ltac break_exists_name x :=
  match goal with
  | [ H : exists _, _ |- _ ] => destruct H as [x H]
  end.

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.

Tactic Notation "on" uconstr(x) "," tactic3(tac) :=
    match goal with
    | [ H : ?y |- _ ] =>
            unify x with y;
            tac H
    end.

(** generic forward reasoning *)

Tactic Notation "fwd" tactic3(tac) "as" ident(H) :=
    simple refine (let H : _ := _ in _);
    [ shelve
    | tac
    | clearbody H ].

Tactic Notation "fwd" tactic3(tac) :=
    let H := fresh "H" in
    fwd tac as H.


Ltac ee :=
  econstructor; eauto.
```