

1.

```
1 Definition inj_1 : forall (P Q : Prop), P -> P \/ Q :=
2   fun P Q HP => or_introl HP.
3
4 Theorem inj_1' : forall (P Q : Prop), P -> P \/ Q.
5 Proof.
6   intros P Q HP. left. apply HP.
7 Qed.
8
9 Theorem conj_disj' : forall (P Q R: Prop), (P /\ Q) \/ (P /\ R) -> P /\
  (Q \/ R).
10 Proof.
11   intros.
12   destruct H as [HPQ | HPR].
13   - split. destruct HPQ. apply H. destruct HPQ. left. apply H0.
14   - split. destruct HPR. apply H. destruct HPR. right. apply H0.
15 Qed.
16 Print conj_disj'.
17
18 Definition conj_disj : forall P Q R, (P /\ Q) \/ (P /\ R) -> P /\ (Q \/
  R) :=
19   fun (P Q R : Prop) (H : P /\ Q \/ P /\ R) =>
20     match H with
21     | or_introl x =>
22       (fun HPQ : P /\ Q =>
23         conj
24           match HPQ with
25           | conj x0 x1 =>
26             (fun (H0 : P) (_ : Q) => H0) x0 x1
27           end
28           match HPQ with
29           | conj x0 x1 =>
30             (fun (_ : P) (H1 : Q) => or_introl H1) x0 x1
31           end) x
32     | or_intror x =>
33       (fun HPR : P /\ R =>
34         conj
```

```

35     match HPR with
36     | conj x0 x1 =>
37         (fun (H0 : P) ( _ : R) => H0) x0 x1
38     end
39     match HPR with
40     | conj x0 x1 =>
41         (fun ( _ : P) (H1 : R) => or_intror H1) x0 x1
42     end) x
43 end.

```

File Edit View Navigation Templates Queries Tools Compile Debug Windows Help



scratch ProofObjects Rel Maps IndProp IndPrinciples

```

involving [or], without resorting to tactics. *)
Definition inj_1 : forall (P Q : Prop), P -> P /\ Q :=
  fun P Q HP => or_intror HP.
Theorem inj_1' : forall (P Q : Prop), P -> P /\ Q.
Proof.
  intros P Q HP. left. apply HP.
Qed.
Theorem conj_disj' : forall (P Q R : Prop), (P /\ Q) \/ (P /\ R) -> P /\ (Q \/ R).
Proof.
  intros.
  destruct H as [HPQ | HPR].
  - split. destruct HPQ. apply H. destruct HPQ. left. apply H0.
  - split. destruct HPR. apply H. destruct HPR. right. apply H0.
Qed.
Print conj_disj'.
Definition conj_disj : forall P Q R, (P /\ Q) \/ (P /\ R) -> P /\ (Q \/ R) :=
  fun (P Q R : Prop) (H : P /\ Q \/ P /\ R) =>
  match H with
  | or_intror x =>
    (fun HPQ : P /\ Q =>
      conj
      match HPQ with
      | conj x0 x1 =>
        (fun (H0 : P) ( _ : Q) => H0) x0 x1
      end
      match HPQ with
      | conj x0 x1 =>
        (fun ( _ : P) (H1 : Q) => or_intror H1) x0 x1
      end) x
    | or_intror x =>
      (fun HPR : P /\ R =>
        conj
        match HPR with
        | conj x0 x1 =>
          (fun (H0 : P) ( _ : R) => H0) x0 x1
        end
        match HPR with
        | conj x0 x1 =>
          (fun ( _ : P) (H1 : R) => or_intror H1) x0 x1
        end) x
      end) x
  end.
Definition or_elim : forall (P Q R : Prop), (P \/ Q) -> (P -> R) -> (Q -> R) -> R :=
  fun P Q R HPQ HPR HQR =>
  match HPQ with
  | or_intror HP => HPR HP
  | or_intror HQ => HQR HQ
  end.
Theorem or_elim' : forall (P Q R : Prop), (P \/ Q) -> (P -> R) -> (Q -> R) -> R.
Proof.
  intros P Q R HPQ HPR HQR.
  destruct HPQ as [HP | HQ].

```

☐ Regular expression ☐ Case insensitive

Find: or_elim Next Previous
Replace: Replace Replace All

Ready in ProofObjects.Props.Or

Line: 474 Char: 1 Offset: 16057

0 / 0

Messages Errors Jobs

```

conj_disj' =
fun (P Q R : Prop) (H : P /\ Q \/ P /\ R) =>
match H with
| or_intror x =>
  (fun HPQ : P /\ Q =>
    conj
    match HPQ with
    | conj x0 x1 => (fun (H0 : P) ( _ : Q) => H0) x0 x1
    end
    match HPQ with
    | conj x0 x1 => (fun ( _ : P) (H1 : Q) => or_intror H1) x0 x1
    end) x
  | or_intror x =>
    (fun HPR : P /\ R =>
      conj
      match HPR with
      | conj x0 x1 => (fun (H0 : P) ( _ : R) => H0) x0 x1
      end
      match HPR with
      | conj x0 x1 => (fun ( _ : P) (H1 : R) => or_intror H1) x0 x1
      end) x
    end
  : forall P Q R : Prop, P /\ Q \/ P /\ R -> P /\ (Q \/ R)
Arguments conj_disj' (P Q R) : type_scope _

```

```

Definition conj_disj : forall P Q R, (P /\ Q) \/ (P /\ R) -> P /\ (Q \/ R) :=
fun (P Q R : Prop) (H : P /\ Q \/ P /\ R) =>
match H with
| or_introl x =>
  (fun HPQ : P /\ Q =>
    conj
      match HPQ with
      | conj x0 x1 =>
        (fun (H0 : P) (_ : Q) => H0) x0 x1
      end
    match HPQ with
    | conj x0 x1 =>
      (fun (_ : P) (H1 : Q) => or_introl H1) x0 x1
    end) x
| or_intror x =>
  (fun HPR : P /\ R =>
    conj
      match HPR with
      | conj x0 x1 =>
        (fun (H0 : P) (_ : R) => H0) x0 x1
      end
    match HPR with
    | conj x0 x1 =>
      (fun (_ : P) (H1 : R) => or_intror H1) x0 x1
    end) x
end)

```

2.

```

1 Theorem plus_one_r' : forall n:nat,
2   n + 1 = S n.
3 Proof.
4   apply nat_ind.
5   - reflexivity.
6   - simpl. intros n IH.
7     rewrite IH. reflexivity.
8 Qed.

```

```

Theorem plus_one_r' : forall n:nat,
  n + 1 = S n.
Proof.
  apply nat_ind.
  - reflexivity.
  - simpl. intros n IH.
    rewrite IH. reflexivity.
Qed.

```

3.

```

1 Theorem lt_trans' :
2   transitive lt.
3 Proof.
4   (* Prove this by induction on evidence that [m] is less than [o]. *)
5   unfold lt. unfold transitive.
6   intros n m o Hnm Hmo.
7   induction Hmo as [| m' Hm'o].
8   - apply le_S. apply Hnm.
9   - apply le_S. apply IHm'o.
10  Qed.

```

*(** **** Exercise: 2 stars, standard, optional (le_trans_hard_way)*

*We can also prove [lt_trans] more laboriously by induction,
without using [le_trans]. Do this. *)*

```

Theorem lt_trans' :
  transitive lt.
Proof.
  (* Prove this by induction on evidence that [m] is less than [o]. *)
  unfold lt. unfold transitive.
  intros n m o Hnm Hmo.
  induction Hmo as [| m' Hm'o].
  - apply le_S. apply Hnm.
  - apply le_S. apply IHm'o.
Qed.

```

4.

```

1 Lemma rsc_trans :
2   forall (X:Type) (R: relation X) (x y z : X),
3     clos_refl_trans_1n R x y ->
4     clos_refl_trans_1n R y z ->
5     clos_refl_trans_1n R x z.
6 Proof.
7   intros X R x y z xy.
8   generalize dependent z.
9   induction xy as [| x y' y H H' IH].
10  - trivial.
11  - intros z yz.
12    apply (rt1n_trans R x y' z).
13    + assumption.

```

```
14      + apply IH. assumption.
15 Qed.
```

```
Lemma rsc_trans :
  forall (X:Type) (R: relation X) (x y z : X),
    clos_refl_trans_ltn R x y ->
    clos_refl_trans_ltn R y z ->
    clos_refl_trans_ltn R x z.
```

```
Proof.
  intros X R x y z xy.
  generalize dependent z.
  induction xy as [| x y' y H H' IH].
  - trivial.
  - intros z yz.
    apply (rtln_trans R x y' z).
    + assumption.
    + apply IH. assumption.
```

```
Qed.
```