```
let tac4d =
    IMP_RES_TAC lemma4a THEN
    IMP_RES_TAC well_formed_DESIGN_sigs_lemma THEN
    IMP_RES_TAC lemma4b THEN
    ASM_REWRITE_TAC []
and tac4e = lemma2_TAC lemma2a'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaB:trace) t)";
                        "SND ((thetaA:trace))";"(sig:name,val:value)"]
and tac4f = lemma2_TAC lemma2b'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaB:trace) t)";
                        "SND ((thetaA:trace))";"(sig:name,val:value)"]
and tac4g = lemma2_TAC lemma2c'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaB:trace) t)";
                        "SND(thetaA:trace))";"(sig:name,val:value)"];;

let QUIET_CONG_THM = save_thm (`QUIET_CONG_THM`,GEN_ALL (PROVE (
 "let A = ARCH ent archA ports declsA csA
  and B = ARCH ent archB ports declsB csB
  and dlysAC = ADD_DELAYS_CS dlysA dlysC
  and dlysBC = ADD_DELAYS_CS dlysB dlysC
  and sigmaAC = sigmaA UNION sigmaC
  and sigmaBC = sigmaB UNION sigmaC
  and gammaAC = gammaA UNION gammaC
  and gammaBC = gammaB UNION gammaC
  and thetaAC t = ((FST (thetaA t)) UNION (FST (thetaC t)),
                   (SND (thetaA t)) UNION (SND (thetaC t)))
  and thetaBC t = ((FST (thetaB t)) UNION (FST (thetaC t)),
                   (SND (thetaB t)) UNION (SND (thetaC t))) in
  let ports' = get_PORTS A
  and ports'' = get_PORTS C in
  let prts = ports' UNION ports'' in
      ((Simulate (dlysA,now,sigmaA,gammaA,thetaA)(DFLAT A) tauA behA /\
        QUIESCE dlysA now tauA gammaA /\
        Simulate (dlysB,now,sigmaB,gammaB,thetaB)(DFLAT B) tauB behB /\
        QUIESCE dlysB now tauB gammaB /\
        Simulate (dlysAC,now,sigmaAC,gammaAC,thetaAC)(DFLAT (DPAR A C))
                 tauAC behAC /\
        QUIESCE dlysAC now tauAC gammaAC /\
        Simulate (dlysBC,now,sigmaBC,gammaBC,thetaBC)(DFLAT (DPAR B C))
                 tauBC behBC /\
        QUIESCE dlysBC now tauBC gammaBC /\
        well_formed_DESIGN C /\
        well_formed_theta thetaC C /\
        EQ_SIGMA sigmaAC sigmaBC ports''/\
        EQ_TAU behA behB ports') ==>
       EQ_TAU behAC behBC prts)",
  CONV_TAC (TOP_DEPTH_CONV let_CONV) THEN tac1 THEN tac2 THENL
    [tac3a;tac3b;tac3c;tac3d;tac3e;tac3f;tac3g;tac3h;
     tac4a;tac4b;tac4c;tac4d;tac4e;tac4f;tac4g;tac4d])));;
```

```
let lemma3_TAC lemma lst =
    let th = SPECL lst lemma in
    ASSUME_TAC th THEN UNDISCH_TAC (concl th) THEN
    ASM_REWRITE_TAC[];;

let tac3a = lemma2_TAC lemma2a
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaA:state";
                        "sigmaB:state";"(sig:name,val:value)"]
and tac3b = lemma2_TAC lemma2b
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaA:state";
                        "sigmaB:state";"(sig:name,val:value)"]
and tac3c = lemma2_TAC lemma2c
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaA:state";
                        "sigmaB:state";"(sig:name,val:value)"]
and tac3d = lemma3_TAC lemma3a
                       ["sigmaA:state";"sigmaC:state";"sigmaB:state";
                        "C:DESIGN";"x:(name#value)"]
and tac3e = lemma2_TAC lemma2a'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaB:state";
                        "sigmaA:state";"(sig:name,val:value)"]
and tac3f = lemma2_TAC lemma2b'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaB:state";
                        "sigmaA:state";"(sig:name,val:value)"]
and tac3g = lemma2_TAC lemma2c'
                       ["behB:transactions";"behA:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "now:time";"sigmaB:state";
                        "sigmaA:state";"(sig:name,val:value)"]
and tac3h = lemma3_TAC lemma3a'
                       ["sigmaB:state";"sigmaC:state";"sigmaA:state";
                        "C:DESIGN";"x:(name#value)"];;

let tac4a = lemma2_TAC lemma2a
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaA:trace) t)";
                        "SND ((thetaB:trace))";"(sig:name,val:value)"]
and tac4b = lemma2_TAC lemma2b
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaA:trace) t)";
                        "SND ((thetaB:trace))";"(sig:name,val:value)"]
and tac4c = lemma2_TAC lemma2c
                       ["behA:transactions";"behB:transactions";
                        "ports:((name)set#(name)set#(name)set)";
                        "t:time";"SND ((thetaA:trace) t)";
                        "SND((thetaB:trace))";"(sig:name,val:value)"];;
```

```
let lemma4a =
    let th1 = ASSUME "well_formed_DESIGN design" in
    let th2 = REWRITE_RULE [well_formed_DESIGN] th1 in
    let th3 = CONJUNCT1 th2 in
        GEN_ALL (DISCH_ALL th3)
and lemma4b =
    let th1 = ASSUME "well_formed_theta theta design" in
    let th2 = REWRITE_RULE [well_formed_theta] th1 in
    let th3 = CONV_RULE let_CONV th2 in
    let th4 = CONJUNCT2 (SPEC_ALL th3) in
    let th5 = DISCH_ALL (GEN_ALL th4) in
        GEN_ALL th5;;

let tac2 =
   let tac = CONV_TAC (ONCE_DEPTH_CONV LEFT_IMP_FORALL_CONV) THEN
             EXISTS_TAC "t:time" THEN ASM_REWRITE_TAC [] THEN DISCH_TAC
   in
   ONCE_ASM_REWRITE_TAC[] THEN REWRITE_TAC [ADD_TO_THETA;EQ_TAU] THEN
   CONV_TAC (DEPTH_CONV let_CONV) THEN BETA_TAC THEN GEN_TAC THEN
   REWRITE_TAC [EXTENSION] THEN
   CONV_TAC (ONCE_DEPTH_CONV SET_SPEC_CONV) THEN
   GEN_TAC THEN REWRITE_TAC [IN_UNION;IN_IMAGE] THEN
   COND_CASES_TAC THEN REWRITE_TAC [IN_UNION] THEN
   UNDISCH_TAC
     "behA = (\t. SND (ADD_TO_THETA sigmaA gammaA thetaA now now t))"
   THEN
   UNDISCH_TAC
     "behB = (\t. SND (ADD_TO_THETA sigmaB gammaB thetaB now now t))"
   THEN
   UNDISCH_TAC
     "behAC = (\t. SND (ADD_TO_THETA (sigmaA UNION sigmaC)
                                     (gammaA UNION gammaC)
                                     (\t.
                                       ((FST(thetaA t)) UNION
                                        (FST(thetaC t)),
                                        (SND(thetaA t)) UNION
                                        (SND(thetaC t))))
                                    now now t))" THEN
   UNDISCH_TAC
     "behBC = (\t. SND (ADD_TO_THETA (sigmaB UNION sigmaC)
                                     (gammaB UNION gammaC)
                                     (\t.
                                       ((FST(thetaB t)) UNION
                                        (FST(thetaC t)),
                                        (SND(thetaB t)) UNION
                                        (SND(thetaC t))))
                                    now now t))" THEN
   REWRITE_TAC [ADD_TO_THETA] THEN
   CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV) THEN
   BETA_TAC THEN REPEAT tac THEN
   REWRITE_TAC [IN_UNION] THEN
   EQ_TAC THEN STRIP_TAC THEN EXISTS_TAC "sig:name" THEN
   EXISTS_TAC "val:value" THEN ASM_REWRITE_TAC[];;

let lemma2_TAC lemma lst =
    let th = SPECL lst lemma in
     ASSUME_TAC th THEN UNDISCH_TAC (concl th) THEN
     ASM_REWRITE_TAC[] THEN STRIP_TAC THEN ASM_REWRITE_TAC[];;
```

```
   (GEN_ALL (PROVE(
      "EQ_TAU tau' tau''((FST ports) UNION
                         ((FST(SND ports)) UNION (SND(SND ports)))) /\
       (tau' now = sigma') /\
       (tau'' now = sigma'') /\
       (FST x) IN (FST ports) /\
       x IN sigma' ==>
       x IN sigma''",tac)),
    GEN_ALL (PROVE(
      "EQ_TAU tau' tau''((FST ports) UNION
                         ((FST(SND ports)) UNION (SND(SND ports)))) /\
       (tau' now = sigma') /\
       (tau'' now = sigma'') /\
       (FST x) IN (FST (SND ports)) /\
       x IN sigma' ==>
       x IN sigma''",tac)),
    GEN_ALL (PROVE(
      "EQ_TAU tau' tau''((FST ports) UNION
                         ((FST(SND ports)) UNION (SND(SND ports)))) /\
       (tau' now = sigma') /\
       (tau'' now = sigma'') /\
       (FST x) IN (SND (SND ports)) /\
       x IN sigma' ==>
       x IN sigma''",tac)));;

let lemma2a' = ONCE_REWRITE_RULE [EQ_TAU_SYM] lemma2a
and lemma2b' = ONCE_REWRITE_RULE [EQ_TAU_SYM] lemma2b
and lemma2c' = ONCE_REWRITE_RULE [EQ_TAU_SYM] lemma2c;;

let lemma3a = GEN_ALL (PROVE(
   "((EQ_SIGMA (sigma' UNION sigma''') (sigma'' UNION sigma''')
              (get_PORTS X))/\
    ((FST x) IN (get_PORTS X)) /\
    (x IN sigma')) ==> ((x IN sigma'') \/ (x IN sigma'''))",
   REWRITE_TAC [EQ_SIGMA] THEN CONV_TAC (ONCE_DEPTH_CONV let_CONV) THEN
   REWRITE_TAC [EXTENSION;IN_UNION;IN_IMAGE] THEN
   CONV_TAC (ONCE_DEPTH_CONV LEFT_AND_FORALL_CONV) THEN
   CONV_TAC LEFT_IMP_FORALL_CONV THEN EXISTS_TAC "x:(name#value)" THEN
   CONV_TAC (ONCE_DEPTH_CONV SET_SPEC_CONV) THEN
   STRIP_TAC THEN FIRST_ASSUM (ASSUME_TAC o fwd) THEN
   UNDISCH_TAC
     "(x IN sigma' \/ x IN sigma''') /\ (FST x) IN (get_PORTS X) ==>
      (?x' y'.
        (x = x',y') /\
        ((x',y':value) IN sigma'' \/ (x',y') IN sigma''') /\
       x' IN (get_PORTS X))" THEN
   ASM_REWRITE_TAC [] THEN STRIP_TAC THEN
   FILTER_ASM_REWRITE_TAC (\tm. (lhs tm = "x:(name#value)")?false)[] THENL
    [FILTER_ASM_REWRITE_TAC
      (\tm. (tm = "(x',y') IN sigma'':state")?false)[]
    ;FILTER_ASM_REWRITE_TAC
      (\tm. (tm = "(x',y') IN sigma''':state")?false)[]]));;

let lemma3a' = ONCE_REWRITE_RULE [EQ_SIGMA_SYM] lemma3a;;

let tac1 =
let tac1a = REWRITE_TAC [DFLAT;get_PORTS] THEN STRIP_TAC
and lemma = GEN_ALL (CONV_RULE (ONCE_DEPTH_CONV SIMENV_CONV)
                     (SPEC "(dlys,now,sigma,gamma,theta):sim_env"
                           lemma1)) in
    tac1a THEN IMP_RES_TAC lemma;;
```

```
let SIMENV_CONV =
    let check1 = assert (\c. let tmp = fst (dest_const c) in
                                ((tmp=`Sim_ENV_DLYS`) or
                                 (tmp=`Sim_ENV_RHO`) or
                                 (tmp=`Sim_ENV_TIME`) or
                                 (tmp=`Sim_ENV_SIGMA`) or
                                 (tmp = `Sim_ENV_GAMMA`) or
                                 (tmp = `Sim_ENV_THETA`)))
    and check2 = assert (\l. not (is_var (hd l)))
    and SIMENV = map (definition `Femto-config`)
                     [`Sim_ENV_DLYS`;`Sim_ENV_RHO`;`Sim_ENV_TIME`;
                      `Sim_ENV_SIGMA`;`Sim_ENV_GAMMA`;`Sim_ENV_THETA`]
    in
    letrec build_conv lst conv =
       if null lst then conv
       else build_conv (tl lst) (conv ORELSEC (REWRITE_CONV (hd lst)))
    in
    let conv = (build_conv (tl SIMENV) (REWRITE_CONV (hd SIMENV)))
               THENC DEPTH_CONV (REWRITE_CONV FST
                                 ORELSEC REWRITE_CONV SND) in
    \tm. (let (_,_) = (check1 # check2) (strip_comb tm) in
          conv tm) ? failwith `SIMENV_CONV`;;

let (lemma2a,lemma2b,lemma2c) =
    let tac =
      REWRITE_TAC [EQ_TAU] THEN CONV_TAC (ONCE_DEPTH_CONV let_CONV) THEN
      CONV_TAC (ONCE_DEPTH_CONV LEFT_AND_FORALL_CONV) THEN
      CONV_TAC LEFT_IMP_FORALL_CONV THEN
      EXISTS_TAC "now:time" THEN REWRITE_TAC [EXTENSION;IN_IMAGE] THEN
      CONV_TAC (ONCE_DEPTH_CONV SET_SPEC_CONV) THEN
      CONV_TAC (ONCE_DEPTH_CONV LEFT_AND_FORALL_CONV) THEN
      CONV_TAC (LEFT_IMP_FORALL_CONV) THEN
      EXISTS_TAC "x:(name#value)" THEN
      CONV_TAC (DEPTH_CONV LEFT_AND_FORALL_CONV) THEN
      CONV_TAC (DEPTH_CONV RIGHT_AND_FORALL_CONV) THEN
      CONV_TAC LEFT_IMP_FORALL_CONV THEN
      EXISTS_TAC "x:(name#value)" THEN
      CONV_TAC (ONCE_DEPTH_CONV RIGHT_AND_FORALL_CONV) THEN
      CONV_TAC LEFT_IMP_FORALL_CONV THEN
      EXISTS_TAC "x:(name#value)" THEN
      STRIP_TAC THEN FIRST_ASSUM (ASSUME_TAC o fwd) THEN
      UNDISCH_TAC
        "(FST x) IN
         ((FST ports) UNION
          ((FST(SND ports)) UNION (SND(SND ports)))) /\
         x IN (tau' now) ==>
         (?sig val'.
           (x = sig,val') /\
           sig IN
           ((FST ports) UNION
            ((FST(SND ports)) UNION (SND(SND ports)))) /\
           (sig,val') IN ((tau'':transactions) now))" THEN
      ASM_REWRITE_TAC[IN_UNION] THEN STRIP_TAC THEN
      UNDISCH_TAC "x=(sig:name,val':value)" THEN
      DISCH_THEN (ASSUME_TAC o SYM) THEN
      UNDISCH_TAC "(sig,val') IN ((tau'':transactions) now)" THEN
      FILTER_ASM_REWRITE_TAC
           (\tm. (rhs tm = "x:(name#value)")?false)[] THEN
      FILTER_ASM_REWRITE_TAC
           (\tm. (rhs tm = "x IN sigma'':state")?false)[]
    in
```

A.11 Theorem 4.3.5.3

```
let ADD_TO_THETA_SIMP = prove_thm(`ADD_TO_THETA_SIMP`,
    "!sigma' gamma' sigma'' gamma'' Tau now later.
      ADD_TO_THETA sigma' gamma'
                   (ADD_TO_THETA sigma'' gamma'' Tau now later)
                   now later =
      ADD_TO_THETA sigma' gamma' Tau now later",
     REPEAT GEN_TAC
       THEN REWRITE_TAC [definition `sim-rules` `ADD_TO_THETA`]
       THEN CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV) THEN GEN_TAC
       THEN BETA_TAC THEN COND_CASES_TAC
       THEN REWRITE_TAC [] THEN COND_CASES_TAC THEN REWRITE_TAC []);;
```

A.12 Theorem 4.4.1

```
let lemma1 = PROVE(
    "!mu Pr tau beh.
      (Simulate mu Pr tau beh /\
       QUIESCE(Sim_ENV_DLYS mu)(Sim_ENV_TIME mu)tau
              (Sim_ENV_GAMMA mu)) ==>
      (beh =
       (\t. (SND ((ADD_TO_THETA
                   (Sim_ENV_SIGMA mu)
                   (Sim_ENV_GAMMA mu)
                   (Sim_ENV_THETA mu)
                   (Sim_ENV_TIME mu)
                   (Sim_ENV_TIME mu)) t))))",
    REPEAT GEN_TAC THEN ONCE_REWRITE_TAC [GEN_Simulate] THEN
    DISCH_THEN (ASSUME_TAC o \th. let [x;y] = CONJUNCTS th in
                                      REWRITE_RULE [y] x) THEN
    ASM_REWRITE_TAC[]);;

let fwd th =
    let (th1,_) = EQ_IMP_RULE th in
    let th2 = CONV_RULE (ONCE_DEPTH_CONV LEFT_AND_EXISTS_CONV) th1 in
    let th3 = CONV_RULE (DEPTH_CONV RIGHT_AND_EXISTS_CONV) th2 in
    let th4 = CONV_RULE (TOP_DEPTH_CONV LEFT_IMP_EXISTS_CONV) th3 in
    let th5 = SPECL ["FST (x:(name#value))";
                     "SND (x:(name#value))"] th4
    in
        REWRITE_RULE [] th5;;
```

A.7 Theorem 4.3.3.3

```
let ZIP_ASSOC = PROVE(
    "!Tau Tau' Tau''.
      (Tau ZIP (Tau' ZIP Tau'')) = ((Tau ZIP Tau') ZIP Tau'')",
    REPEAT GEN_TAC THEN
    REWRITE_TAC [definition `cs-rules` `ZIP`] THEN
    BETA_TAC THEN
    REWRITE_TAC[theorem `sets` `UNION_ASSOC`]);;
```

A.8 Theorem 4.3.3.4

```
let ZIP_EMPTY = PROVE(
    "(!tau. (\t.{}) ZIP tau = tau) /\ (!tau. tau ZIP (\t.{}) = tau)",
    REWRITE_TAC [definition `cs-rules` `ZIP`;UNION_EMPTY;ETA_AX]);;
```

A.9 Theorem 4.3.3.5

```
let CLEAN_ZIP_EMPTY = PROVE(
    "!A B. (CLEAN_ZIP (\t.{}) A B) = (A ZIP B)",
    REPEAT GEN_TAC THEN REWRITE_TAC [CLEAN_ZIP;ZIP] THEN
    CONV_TAC (DEPTH_CONV let_CONV) THEN BETA_TAC THEN
    REWRITE_TAC [DIFF_EMPTY;EMPTY_DIFF;UNION_EMPTY;ETA_AX]);;
```

A.10 Theorem 4.3.3.6

```
let CLEAN_ZIP_SYM = PROVE(
    "!orig A B. (CLEAN_ZIP orig A B) = (CLEAN_ZIP orig B A)",
    REPEAT GEN_TAC THEN REWRITE_TAC [CLEAN_ZIP;ZIP] THEN
    CONV_TAC (DEPTH_CONV let_CONV) THEN
    CONV_TAC FUN_EQ_CONV THEN BETA_TAC THEN
    REWRITE_TAC [IN_IMAGE;DIFF_DEF;UNION_DEF;EXTENSION] THEN
    REPEAT GEN_TAC THEN CONV_TAC  (DEPTH_CONV SET_SPEC_CONV) THEN
    CONV_TAC (TOP_DEPTH_CONV NOT_EXISTS_CONV) THEN
    REWRITE_TAC [DE_MORGAN_THM] THEN
    CONV_TAC (ONCE_DEPTH_CONV NOT_EXISTS_CONV) THEN
    REWRITE_TAC [DE_MORGAN_THM] THEN EQ_TAC THEN
    CONV_TAC (TOP_DEPTH_CONV RIGHT_AND_FORALL_CONV) THEN
    CONV_TAC (TOP_DEPTH_CONV RIGHT_OR_FORALL_CONV) THEN
    CONV_TAC (TOP_DEPTH_CONV RIGHT_IMP_FORALL_CONV) THEN
    REPEAT GEN_TAC THEN
    CONV_TAC (ONCE_DEPTH_CONV LEFT_OR_FORALL_CONV) THEN
    CONV_TAC (ONCE_DEPTH_CONV RIGHT_OR_FORALL_CONV) THEN
    CONV_TAC (ONCE_DEPTH_CONV RIGHT_AND_FORALL_CONV) THEN
    CONV_TAC (ONCE_DEPTH_CONV RIGHT_OR_FORALL_CONV) THEN
    CONV_TAC RIGHT_IMP_FORALL_CONV THEN GEN_TAC THEN
    CONV_TAC (TOP_DEPTH_CONV LEFT_IMP_FORALL_CONV) THEN
    EXISTS_TAC "x':name" THEN EXISTS_TAC "y:value" THEN
    EXISTS_TAC "x'':(name#value)" THEN REPEAT STRIP_TAC THEN
    ASM_REWRITE_TAC[]);;
```

A.2 Theorem 3.3.3.1

```
let lemma4a =
    let th1 = ASSUME "well_formed_DESIGN design" in
    let th2 = REWRITE_RULE [well_formed_DESIGN] th1 in
    let th3 = CONJUNCT1 th2 in
        GEN_ALL (DISCH_ALL th3)
```

A.3 Theorem 4.2.3.1

```
let th1 = PROVE(
    "!A B ports sig val t.
      ((EQ_TAU A B ports) /\ (sig IN ports) /\ (sig,val) IN (A t)) ==>
      (sig,val) IN (B t)",
    REPEAT GEN_TAC THEN
    REWRITE_TAC [definition `Femto-config` `EQ_TAU`] THEN
    CONV_TAC (ONCE_DEPTH_CONV let_CONV) THEN
    REWRITE_TAC [EXTENSION] THEN
    CONV_TAC (ONCE_DEPTH_CONV SET_SPEC_CONV) THEN
    CONV_TAC (TOP_DEPTH_CONV LEFT_AND_FORALL_CONV) THEN
    CONV_TAC (TOP_DEPTH_CONV LEFT_IMP_FORALL_CONV) THEN
    EXISTS_TAC "t:time" THEN EXISTS_TAC "(sig:name,val:value)" THEN
    STRIP_TAC THEN
    FIRST_ASSUM
     \th. let (a,_)=EQ_IMP_RULE th in
          let th1=CONV_RULE (TOP_DEPTH_CONV LEFT_IMP_EXISTS_CONV) a in
          let th2=SPECL ["sig:name";"val:value"] th1 in
              ASSUME_TAC (REWRITE_RULE [] th2) THEN
    RES_TAC THEN ASM_REWRITE_TAC[]);;
```

A.4 Theorem 4.2.3.2

```
let th2 =
    let a=ONCE_REWRITE_RULE[theorem `Femto-config` `EQ_TAU_SYM`]th1 in
    let b=SPECL ["B:transactions";"A:transactions"] a in
        GENL ["A:transactions";"B:transactions"] b;;
```

A.5 Theorem 4.3.3.1

```
let Zip_ZIP_EQ =
    let ZIP = definition `cs-rules` `ZIP`
    and Zip = definition `cs-rules` `Zip` in
    let th = REWRITE_RULE [SYM (SPEC_ALL ZIP)] Zip in
        CONV_RULE (ONCE_DEPTH_CONV let_CONV) th;;
```

A.6 Theorem 4.3.3.2

```
let ZIP_COMM = PROVE(
    "!A B. (A ZIP B) = (B ZIP A)",
    REWRITE_TAC [ZIP] THEN
    CONV_TAC (ONCE_DEPTH_CONV FUN_EQ_CONV) THEN
    REPEAT GEN_TAC THEN BETA_TAC THEN
    REWRITE_TAC [EXTENSION] THEN GEN_TAC THEN
    EQ_TAC THEN STRIP_TAC THEN ONCE_REWRITE_TAC [UNION_COMM] THEN
    ASM_REWRITE_TAC[]);;
```

APPENDIX A

HOL PROOFS

This appendix gives the HOL scripts used to prove the theorems shown in Chapters 3 and 4.

A.1 Theorem 3.3.2.1

```
let well_formed_DESIGN_sigs_lemma =
    prove_thm(`well_formed_DESIGN_sigs_lemma`,
    "!D. (FST(well_formed_DESIGN_sigs D)) ==>
         !sig. (sig IN (get_PORTS D)) ==>
               (sig IN (get_sigs_DESIGN D))",
    let pair = GEN_ALL (SYM (SPEC_ALL PAIR)) in
    let th1 =
        let x = INST_TYPE [(":(name)set",":*");(":(name)set",":**")]
                          pair
        in
            SPEC "get_sigs_CS C" x
    and (th2,th3) =
        let x = INST_TYPE [(":bool",":*");(":(name)set",":**")]
                          pair
        in
           (SPEC "well_formed_DESIGN_sigs D" x,
            SPEC "well_formed_DESIGN_sigs D'" x)
    in
    INDUCT_THEN DESIGN_Induct ASSUME_TAC THEN
    REWRITE_TAC [get_PORTS;get_sigs_DESIGN;IN_UNION;
                 well_formed_DESIGN_sigs] THENL
    [CONV_TAC (DEPTH_CONV let_CONV) THEN REWRITE_TAC [] THEN
     PURE_ONCE_REWRITE_TAC [th1] THEN
     CONV_TAC (DEPTH_CONV let_CONV) THEN REWRITE_TAC [] THEN
     REPEAT GEN_TAC THEN STRIP_TAC THEN ASM_REWRITE_TAC [] THEN
     REWRITE_TAC [IN_UNION] THEN
     REPEAT STRIP_TAC THEN ASM_REWRITE_TAC []
    ;PURE_ONCE_REWRITE_TAC [th2;th3] THEN
     CONV_TAC (ONCE_DEPTH_CONV let_CONV) THEN
     REWRITE_TAC[] THEN STRIP_TAC THEN RES_TAC THEN
     REPEAT STRIP_TAC THEN RES_TAC THEN ASM_REWRITE_TAC[]]);;
```