```
begin Lestrade execution

   >>> comment comment This draft demonstrates \
       that having definitions which look


   {move 1}

   >>> comment comment transitory in world \
       0 can be very very useful .Look at the


   {move 1}

   >>> comment comment proofs of Then and \
       Else in this version : they are awful \
       .


   {move 1}

   >>> comment comment They result from


   {move 1}

   >>> comment comment what seems an ideologically \
       sound notion of moving some


   {move 1}

   >>> comment comment declarations local \
       to their proof into deeper worlds so


   {move 1}

   >>> comment that they do not clutter world \
       0 .Bad move, as it turns out .


   {move 1}

   >>> comment comment At the same time, it \
```

```
    is certainly a demonstration of how


{move 1}

>>> comment comment to encapsulate stuff \
    when appropriate : compare with the main


{move 1}

>>> comment comment version to see how \
    many fewer declarations appear in world \
    0


{move 1}

>>> comment related to Then and Else .


{move 1}

>>> comment comment Automath file 37 translation \
    .This must be run with the Lestrade version \
    of


{move 1}

>>> comment comment July 8 or later, with \
    changes in saved world management .The \
    new saved world


{move 1}

>>> comment comment management allows \
    simulation of the Automath context device, and \
    prevents


{move 1}

>>> comment name cluttering of world 1 .
```

```
{move 1}

>>> comment this version makes full use \
    of implicit arguments .Proof lines are \
    generally much shorter .


{move 1}

>>> comment * A := E B ; P R O P


{move 1}

>>> declare A prop


A : prop


{move 1}

>>> comment A * B := E B ; P R O P


{move 1}

>>> declare B prop


B : prop


{move 1}

>>> comment B * I M P := [T, A] B ; P R O P


{move 1}

>>> save B


{move 1 : B}
```

```
>>> postulate Imp A B : prop


Imp : [(A_1 : prop), (B_1 : prop) =>
    (--- : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare T that A


   T : that A


   {move 2}

   >>> postulate Ded T : that B


   Ded : [(T_1 : that A) => (--- : that
       B)]


   {move 1 : B}

   >>> close


{move 1 : B}

>>> postulate Imppf Ded : that Imp A B


Imppf : [(.A_1 : prop), (.B_1 : prop), (Ded_1
    : [(T_2 : that .A_1) => (--- : that
       .B_1)]) => (--- : that .A_1
    Imp .B_1)]


{move 0}
```

```
>>> postulate Imppffull A B Ded : that \
    Imp A B


Imppffull : [(A_1 : prop), (B_1 : prop), (Ded_1
    : [(T_2 : that A_1) => (--- : that
       B_1)]) => (--- : that A_1 Imp
    B_1)]


{move 0}

>>> declare X that A


X : that A


{move 1 : B}

>>> declare Y that A Imp B


Y : that A Imp B


{move 1 : B}

>>> postulate Mp X Y : that B


Mp : [(.A_1 : prop), (.B_1 : prop), (X_1
    : that .A_1), (Y_1 : that .A_1 Imp
    .B_1) => (--- : that .B_1)]


{move 0}

>>> postulate Mpfull A B X Y : that B


Mpfull : [(A_1 : prop), (B_1 : prop), (X_1
    : that A_1), (Y_1 : that A_1 Imp
    B_1) => (--- : that B_1)]
```

```
{move 0}

>>> comment * C O N := P N ; P R O P


{move 1 : B}

>>> postulate Con prop


Con : prop


{move 0}

>>> comment A * N O T := I M P (A, C O N) ; P R O P


{move 1 : B}

>>> define Not A : A Imp Con


Not : [(A_1 : prop) =>
    ({def} A_1 Imp Con : prop)]


Not : [(A_1 : prop) => (--- : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare Xx that A Imp Con


   Xx : that A Imp Con


   {move 2}
```

```
   >>> define negfix Xx : Xx


   negfix : [(Xx_1 : that A Imp Con) =>
       (--- : that A Imp Con)]


   {move 1 : B}

   >>> close


{move 1 : B}

>>> define Negfix A : Imppffull (A Imp \
    Con, Not A, negfix)


Negfix : [(A_1 : prop) =>
    ({def} Imppffull (A_1 Imp Con, Not
    (A_1), [(Xx_2 : that A_1 Imp Con) =>
       ({def} Xx_2 : that A_1 Imp Con)]) : that
    (A_1 Imp Con) Imp Not (A_1))]


Negfix : [(A_1 : prop) => (--- : that
    (A_1 Imp Con) Imp Not (A_1))]


{move 0}

>>> open


   {move 2}

   >>> declare aa that A


   aa : that A


   {move 2}

   >>> postulate neg aa : that Con
```

```
   neg : [(aa_1 : that A) => (---
       : that Con)]


   {move 1 : B}

   >>> close


{move 1 : B}

>>> define Negproof neg : Mp (Imppf neg, Negfix \
    A)


Negproof : [(.A_1 : prop), (neg_1
    : [(aa_2 : that .A_1) => (--- : that
       Con)]) =>
    ({def} Imppf (neg_1) Mp Negfix (.A_1) : that
    Not (.A_1))]


Negproof : [(.A_1 : prop), (neg_1
    : [(aa_2 : that .A_1) => (--- : that
       Con)]) => (--- : that Not (.A_1))]


{move 0}

>>> comment B * I := E B ; I M P (A, B)


{move 1 : B}

>>> clearcurrent B


{move 1 : B}

>>> declare I that A Imp B


I : that A Imp B
```

```
{move 1 : B}

>>> save I


{move 1 : I}

>>> comment I * N := E3 ; N O T (B)


{move 1 : I}

>>> declare N that Not B


N : that Not (B)


{move 1 : I}

>>> comment N * C O N T R A P O S := [T, A] << \
    T > I > N ; N O T (A)


{move 1 : I}

>>> open


   {move 2}

   >>> declare T that A


   T : that A


   {move 2}

   >>> define step1 T : Mp T I


   step1 : [(T_1 : that A) => (---
       : that B)]
```

```
   {move 1 : I}

   >>> define step2 T : Mp (step1 T, N)


   step2 : [(T_1 : that A) => (---
       : that Con)]


   {move 1 : I}

   >>> close


{move 1 : I}

>>> define Contrapos I N : Negproof step2


Contrapos : [(.A_1 : prop), (.B_1
    : prop), (I_1 : that .A_1 Imp .B_1), (N_1
    : that Not (.B_1)) =>
    ({def} Negproof ([(T_2 : that .A_1) =>
       ({def} T_2 Mp I_1 Mp N_1 : that
       Con)]) : that Not (.A_1))]


Contrapos : [(.A_1 : prop), (.B_1
    : prop), (I_1 : that .A_1 Imp .B_1), (N_1
    : that Not (.B_1)) => (--- : that
    Not (.A_1))]


{move 0}

>>> comment A * A0 := E B ; A


{move 1 : I}

>>> clearcurrent I


{move 1 : I}

>>> declare A0 that A
```

```
A0 : that A


{move 1 : I}

>>> save A0


{move 1 : A0}

>>> comment A0 * T H1 := [T, N O T (A)] < A0 \
    > [T] ; N O T (N O T (A))


{move 1 : A0}

>>> open


   {move 2}

   >>> declare T that Not A


   T : that Not (A)


   {move 2}

   >>> define step1 T : Mp A0 T


   step1 : [(T_1 : that Not (A)) =>
       (--- : that Con)]


   {move 1 : A0}

   >>> close


{move 1 : A0}

>>> define Th1 A0 : Negproof step1
```

```
Th1 : [(.A_1 : prop), (A0_1 : that
    .A_1) =>
    ({def} Negproof ([(T_2 : that Not
       (.A_1)) =>
       ({def} A0_1 Mp T_2 : that Con)]) : that
    Not (Not (.A_1)))]


Th1 : [(.A_1 : prop), (A0_1 : that
    .A_1) => (--- : that Not (Not (.A_1)))]


{move 0}

>>> clearcurrent A0


{move 1 : A0}

>>> save A0


{move 1 : A0}

>>> comment A * N := E B ; N O T (N O T (A))


{move 1 : A0}

>>> declare N that Not Not A


N : that Not (Not (A))


{move 1 : A0}

>>> comment N * D B L N E G L A W := P N ; A


{move 1 : A0}

>>> postulate Dblneglaw N : that A
```

```
Dblneglaw : [(.A_1 : prop), (N_1
    : that Not (Not (.A_1))) => (---
    : that .A_1)]


{move 0}

>>> comment B * I := E B ; I M P (A, B)


{move 1 : A0}

>>> comment already declared


{move 1 : A0}

>>> comment I * J := E B ; I M P (N O T (A), B)


{move 1 : A0}

>>> declare J that (Not A) Imp B


J : that Not (A) Imp B


{move 1 : A0}

>>> comment J * A N Y C A S E := D B L N E G L A W (B, [T, N O T (B)] << \
    C O N T R A P O S (A, B, I, T) > J > T) ; B


{move 1 : A0}

>>> open


   {move 2}

   >>> declare bb that Not B


   bb : that Not (B)
```

```
   {move 2}

   >>> define step1 bb : Contrapos I bb


   step1 : [(bb_1 : that Not (B)) =>
       (--- : that Not (A))]


   {move 1 : A0}

   >>> define step2 bb : Contrapos (J, bb)


   step2 : [(bb_1 : that Not (B)) =>
       (--- : that Not (Not (A)))]


   {move 1 : A0}

   >>> define step3 bb : Mp (step1 bb, step2 \
       bb)


   step3 : [(bb_1 : that Not (B)) =>
       (--- : that Con)]


   {move 1 : A0}

   >>> close


{move 1 : A0}

>>> define Anycase I J : Dblneglaw (Negproof \
    (step3))


Anycase : [(.A_1 : prop), (.B_1 : prop), (I_1
    : that .A_1 Imp .B_1), (J_1 : that
    Not (.A_1) Imp .B_1) =>
    ({def} Dblneglaw (Negproof ([(bb_3
       : that Not (.B_1)) =>
```

```
      ({def} I_1 Contrapos bb_3 Mp J_1
      Contrapos bb_3 : that Con)])) : that
    .B_1)]


Anycase : [(.A_1 : prop), (.B_1 : prop), (I_1
    : that .A_1 Imp .B_1), (J_1 : that
    Not (.A_1) Imp .B_1) => (--- : that
    .B_1)]


{move 0}

>>> clearcurrent I


{move 1 : I}

>>> save I


{move 1 : I}

>>> comment B * N := E B ; N O T (A)


{move 1 : I}

>>> declare N that Not A


N : that Not (A)


{move 1 : I}

>>> comment N comment T H2 := [T, A] D B L N E G L A W (B, [U, N O T (B)] < T > N ; I


{move 1 : I}

>>> open


   {move 2}
```

```
>>> declare T that A


T : that A


{move 2}

>>> open


   {move 3}

   >>> declare U that Not B


   U : that Not (B)


   {move 3}

   >>> define step1 U : Mp T N


   step1 : [(U_1 : that Not (B)) =>
       (--- : that Con)]


   {move 2}

   >>> close


{move 2}

>>> define step2 T : Dblneglaw (Negproof \
    (step1))


step2 : [(T_1 : that A) => (---
    : that B)]


{move 1 : I}

>>> close
```

```
{move 1 : I}

>>> comment comment Notice that Th2 has \
    a proposition parameter,


{move 1 : I}

>>> comment comment because B cannot be \
    extracted from the argument


{move 1 : I}

>>> comment supplied (a proof of not \
    A) .


{move 1 : I}

>>> define Th2 B N : Imppf step2


Th2 : [(.A_1 : prop), (B_1 : prop), (N_1
    : that Not (.A_1)) =>
    ({def} Imppf ([(T_2 : that .A_1) =>
       ({def} Dblneglaw (Negproof ([(U_4
          : that Not (B_1)) =>
          ({def} T_2 Mp N_1 : that Con)])) : that
       B_1)]) : that .A_1 Imp B_1)]


Th2 : [(.A_1 : prop), (B_1 : prop), (N_1
    : that Not (.A_1)) => (--- : that
    .A_1 Imp B_1)]


{move 0}

>>> comment B * A0 := E B ; A


{move 1 : I}
```

```
>>> comment already declared


{move 1 : I}

>>> comment A0 * N := E B ; N O T (B)


{move 1 : I}

>>> clearcurrent A0


{move 1 : A0}

>>> save A0


{move 1 : A0}

>>> declare N that Not B


N : that Not (B)


{move 1 : A0}

>>> comment N * T H3 := [T, I M P (A, B)] << \
    A0 > T > N ; N O T (I M P (A, B))


{move 1 : A0}

>>> open


   {move 2}

   >>> declare T that Imp A B


   T : that A Imp B


   {move 2}
```

```
   >>> define step1 T : Mp A0 T


   step1 : [(T_1 : that A Imp B) =>
       (--- : that B)]


   {move 1 : A0}

   >>> define step2 T : Mp (step1 T, N)


   step2 : [(T_1 : that A Imp B) =>
       (--- : that Con)]


   {move 1 : A0}

   >>> close


{move 1 : A0}

>>> define Th3 A0 N : Negproof (step2)


Th3 : [(.A_1 : prop), (.B_1 : prop), (A0_1
    : that .A_1), (N_1 : that Not (.B_1)) =>
    ({def} Negproof ([(T_2 : that .A_1
       Imp .B_1) =>
       ({def} A0_1 Mp T_2 Mp N_1 : that
       Con)]) : that Not (.A_1 Imp
    .B_1))]


Th3 : [(.A_1 : prop), (.B_1 : prop), (A0_1
    : that .A_1), (N_1 : that Not (.B_1)) =>
    (--- : that Not (.A_1 Imp .B_1))]


{move 0}

>>> comment B * N := E B ; N O T (I M P (A, B))
```

```
{move 1 : A0}

>>> clearcurrent I


{move 1 : I}

>>> save I


{move 1 : I}

>>> declare N that Not (A Imp B)


N : that Not (A Imp B)


{move 1 : I}

>>> save N


{move 1 : N}

>>> comment N * T H4 := D B L N E G L A W (A, [T, N O T (A)] < T H2 \
    (A, B, T) > N


{move 1 : N}

>>> open


   {move 2}

   >>> declare T that Not A


   T : that Not (A)


   {move 2}

   >>> define step1 T : Th2 B T
```

```
   step1 : [(T_1 : that Not (A)) =>
       (--- : that A Imp B)]


   {move 1 : N}

   >>> define step2 T : Mp (step1 T, N)


   step2 : [(T_1 : that Not (A)) =>
       (--- : that Con)]


   {move 1 : N}

   >>> close


{move 1 : N}

>>> define Th4 N : Dblneglaw (Negproof \
    (step2))


Th4 : [(.A_1 : prop), (.B_1 : prop), (N_1
    : that Not (.A_1 Imp .B_1)) =>
    ({def} Dblneglaw (Negproof ([(T_3
       : that Not (.A_1)) =>
       ({def} .B_1 Th2 T_3 Mp N_1 : that
       Con)])) : that .A_1)]


Th4 : [(.A_1 : prop), (.B_1 : prop), (N_1
    : that Not (.A_1 Imp .B_1)) => (---
    : that .A_1)]


{move 0}

>>> clearcurrent N


{move 1 : N}

>>> comment N * T H5 := [T, B] < [U, A] T > N
```

```
{move 1 : N}

>>> open


   {move 2}

   >>> declare T that B


   T : that B


   {move 2}

   >>> open


      {move 3}

      >>> declare U that A


      U : that A


      {move 3}

      >>> define step1 U : T


      step1 : [(U_1 : that A) => (---
          : that B)]


      {move 2}

      >>> close


   {move 2}

   >>> define step2 T : Mp ((Imppf step1), N)
```

```
   step2 : [(T_1 : that B) => (---
       : that Con)]


   {move 1 : N}

   >>> close


{move 1 : N}

>>> define Th5 N : Negproof step2


Th5 : [(.A_1 : prop), (.B_1 : prop), (N_1
    : that Not (.A_1 Imp .B_1)) =>
    ({def} Negproof ([(T_2 : that .B_1) =>
       ({def} Imppf ([(U_4 : that .A_1) =>
          ({def} T_2 : that .B_1)]) Mp
       N_1 : that Con)]) : that Not
    (.B_1))]


Th5 : [(.A_1 : prop), (.B_1 : prop), (N_1
    : that Not (.A_1 Imp .B_1)) => (---
    : that Not (.B_1))]


{move 0}

>>> comment B * O R := I M P (N O T (A), B) ; P R O P


{move 1 : N}

>>> clearcurrent I


{move 1 : I}

>>> save I


{move 1 : I}
```

```
>>> define Or A B : (Not A) Imp B


Or : [(A_1 : prop), (B_1 : prop) =>
    ({def} Not (A_1) Imp B_1 : prop)]


Or : [(A_1 : prop), (B_1 : prop) =>
    (--- : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare X2 that (Not A) Imp B


   X2 : that Not (A) Imp B


   {move 2}

   >>> define orfix X2 : X2


   orfix : [(X2_1 : that Not (A) Imp
       B) => (--- : that Not (A) Imp
       B)]


   {move 1 : I}

   >>> close


{move 1 : I}

>>> define Orfix A B : Imppffull ((Not \
    A) Imp B, Or A B, orfix)


Orfix : [(A_1 : prop), (B_1 : prop) =>
```

```
    ({def} Imppffull (Not (A_1) Imp
    B_1, A_1 Or B_1, [(X2_2 : that
       Not (A_1) Imp B_1) =>
       ({def} X2_2 : that Not (A_1) Imp
       B_1)]) : that (Not (A_1) Imp
    B_1) Imp A_1 Or B_1)]


Orfix : [(A_1 : prop), (B_1 : prop) =>
    (--- : that (Not (A_1) Imp B_1) Imp
    A_1 Or B_1)]


{move 0}

>>> comment B * A0 := E B ; A


{move 1 : I}

>>> declare A0 that A


A0 : that A


{move 1 : I}

>>> comment A0 * O R I1 := T H2 (N O T (A), B, T H1 \
    (A, A0)) ; O R (A, B)


{move 1 : I}

>>> define Ori1 B A0 : Mp (Th2 (B, Th1 \
    A0), Orfix A, B)


Ori1 : [(.A_1 : prop), (B_1 : prop), (A0_1
    : that .A_1) =>
    ({def} B_1 Th2 Th1 (A0_1) Mp .A_1
    Orfix B_1 : that .A_1 Or B_1)]


Ori1 : [(.A_1 : prop), (B_1 : prop), (A0_1
    : that .A_1) => (--- : that .A_1
```

```
    Or B_1)]


{move 0}

>>> comment B * B0 := E B ; B


{move 1 : I}

>>> clearcurrent I


{move 1 : I}

>>> declare A0 that A


A0 : that A


{move 1 : I}

>>> declare B0 that B


B0 : that B


{move 1 : I}

>>> save B0


{move 1 : B0}

>>> comment B0 * O R I2 := [T, N O T (A)] B0 \
    ; O R (A, B)


{move 1 : B0}

>>> open


   {move 2}
```

```
   >>> declare Nn that Not A


   Nn : that Not (A)


   {move 2}

   >>> define oristep Nn : B0


   oristep : [(Nn_1 : that Not (A)) =>
       (--- : that B)]


   {move 1 : B0}

   >>> close


{move 1 : B0}

>>> define Ori2 A B0 : Mp (Imppf (oristep), Orfix \
    A B)


Ori2 : [(A_1 : prop), (.B_1 : prop), (B0_1
    : that .B_1) =>
    ({def} Imppf ([(Nn_3 : that Not
       (A_1)) =>
       ({def} B0_1 : that .B_1)]) Mp
    A_1 Orfix .B_1 : that A_1 Or .B_1)]


Ori2 : [(A_1 : prop), (.B_1 : prop), (B0_1
    : that .B_1) => (--- : that A_1 Or
    .B_1)]


{move 0}

>>> comment B * O := E B ; O R (A, B)


{move 1 : B0}
```

```
>>> clearcurrent B0


{move 1 : B0}

>>> declare O that Or A B


O : that A Or B


{move 1 : B0}

>>> save O


{move 1 : O}

>>> comment O * N := E B ; N O T (A)


{move 1 : O}

>>> declare nota that Not A


nota : that Not (A)


{move 1 : O}

>>> save nota


{move 1 : nota}

>>> comment N * N O T C A S E1 := < N > O ; B


{move 1 : nota}

>>> define Notcase1 O nota : Mp (nota, O)


Notcase1 : [(.A_1 : prop), (.B_1
```

```
    : prop), (O_1 : that .A_1 Or .B_1), (nota_1
    : that Not (.A_1)) =>
    ({def} nota_1 Mp O_1 : that .B_1)]


Notcase1 : [(.A_1 : prop), (.B_1
    : prop), (O_1 : that .A_1 Or .B_1), (nota_1
    : that Not (.A_1)) => (--- : that
    .B_1)]


{move 0}

>>> comment O * N := E B ; N O T (B)


{move 1 : nota}

>>> clearcurrent nota


{move 1 : nota}

>>> declare notb that Not B


notb : that Not (B)


{move 1 : nota}

>>> save notb


{move 1 : notb}

>>> comment N * N O T C A S E2 := D B L N E G L A W (A, C O N T R A P O S (N O T A, B


{move 1 : notb}

>>> define Notcase2 O notb : Dblneglaw \
    (Contrapos (O, notb))


Notcase2 : [(.A_1 : prop), (.B_1
```

```
    : prop), (O_1 : that .A_1 Or .B_1), (notb_1
    : that Not (.B_1)) =>
    ({def} Dblneglaw (O_1 Contrapos notb_1) : that
    .A_1)]


Notcase2 : [(.A_1 : prop), (.B_1
    : prop), (O_1 : that .A_1 Or .B_1), (notb_1
    : that Not (.B_1)) => (--- : that
    .A_1)]


{move 0}

>>> comment B * C := E B ; P R O P


{move 1 : notb}

>>> clearcurrent B


{move 1 : B}

>>> declare C prop


C : prop


{move 1 : B}

>>> comment C * O := E B ; O R (A, B)


{move 1 : B}

>>> declare O that A Or B


O : that A Or B


{move 1 : B}

>>> comment O * I := E B ; I M P (A, C)
```

```
{move 1 : B}

>>> declare I that A Imp C


I : that A Imp C


{move 1 : B}

>>> comment I * J := E B ; I M P (B, C)


{move 1 : B}

>>> declare J that B Imp C


J : that B Imp C


{move 1 : B}

>>> comment J * O R E := A N Y C A S E (A, C, I, [T, Not \
       A] << T >, O > J >) ; C


{move 1 : B}

>>> open


   {move 2}

   >>> declare T that Not A


   T : that Not (A)


   {move 2}

   >>> define step1 T : Mp (T, O)
```

```
   step1 : [(T_1 : that Not (A)) =>
       (--- : that B)]


   {move 1 : B}

   >>> define step2 T : Mp (step1 T, J)


   step2 : [(T_1 : that Not (A)) =>
       (--- : that C)]


   {move 1 : B}

   >>> close


{move 1 : B}

>>> define Ore O I J : Anycase (I, Imppf \
    (step2))


Ore : [(.A_1 : prop), (.B_1 : prop), (.C_1
    : prop), (O_1 : that .A_1 Or .B_1), (I_1
    : that .A_1 Imp .C_1), (J_1 : that
    .B_1 Imp .C_1) =>
    ({def} I_1 Anycase Imppf ([(T_3
       : that Not (.A_1)) =>
       ({def} T_3 Mp O_1 Mp J_1 : that
       .C_1)]) : that .C_1)]


Ore : [(.A_1 : prop), (.B_1 : prop), (.C_1
    : prop), (O_1 : that .A_1 Or .B_1), (I_1
    : that .A_1 Imp .C_1), (J_1 : that
    .B_1 Imp .C_1) => (--- : that .C_1)]


{move 0}

>>> comment B * A N D := N O T (I M P (A, N O T (B))) ; P R O P
```

```
{move 1 : B}

>>> clearcurrent B0


{move 1 : B0}

>>> define And A B : Not (A Imp Not B)


And : [(A_1 : prop), (B_1 : prop) =>
    ({def} Not (A_1 Imp Not (B_1)) : prop)]


And : [(A_1 : prop), (B_1 : prop) =>
    (--- : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare fixand that And A B


   fixand : that A And B


   {move 2}

   >>> define andfix fixand : fixand


   andfix : [(fixand_1 : that A And
       B) => (--- : that A And B)]


   {move 1 : B0}

   >>> close


{move 1 : B0}
```

```
>>> define Andfix A B : Imppffull (Not \
    (A Imp Not B), A And B, andfix)


Andfix : [(A_1 : prop), (B_1 : prop) =>
    ({def} Imppffull (Not (A_1 Imp Not
    (B_1)), A_1 And B_1, [(fixand_2
       : that A_1 And B_1) =>
       ({def} fixand_2 : that A_1 And
       B_1)]) : that Not (A_1 Imp Not
    (B_1)) Imp A_1 And B_1)]


Andfix : [(A_1 : prop), (B_1 : prop) =>
    (--- : that Not (A_1 Imp Not (B_1)) Imp
    A_1 And B_1)]


{move 0}

>>> comment B * A0 := E B ; A


{move 1 : B0}

>>> comment already declared


{move 1 : B0}

>>> comment A0 * B0 := E B ; B


{move 1 : B0}

>>> comment use B0 already declared


{move 1 : B0}

>>> comment B0 * A N D I := T H3 (A, N O T (B), A0, T H1 \
    (B, B0)) ; A N D (A, B)


{move 1 : B0}
```

```
>>> define Andi A0 B0 : Mp (Th3 (A0, Th1 \
    B0), Andfix A B)


Andi : [(.A_1 : prop), (.B_1 : prop), (A0_1
    : that .A_1), (B0_1 : that .B_1) =>
    ({def} A0_1 Th3 Th1 (B0_1) Mp .A_1
    Andfix .B_1 : that .A_1 And .B_1)]


Andi : [(.A_1 : prop), (.B_1 : prop), (A0_1
    : that .A_1), (B0_1 : that .B_1) =>
    (--- : that .A_1 And .B_1)]


{move 0}

>>> comment B * A1 := E B ; A N D (A, B)


{move 1 : B0}

>>> declare A1 that A And B


A1 : that A And B


{move 1 : B0}

>>> comment A1 * A N D E1 := T H4 (A, N O T B, A1) ; A


{move 1 : B0}

>>> define Ande1 A1 : Th4 (A1)


Ande1 : [(.A_1 : prop), (.B_1 : prop), (A1_1
    : that .A_1 And .B_1) =>
    ({def} Th4 (A1_1) : that .A_1)]


Ande1 : [(.A_1 : prop), (.B_1 : prop), (A1_1
    : that .A_1 And .B_1) => (--- : that
```

```
    .A_1)]


{move 0}

>>> comment A1 * A N D E2 := D B L N E G L A W (B, T H5 \
    (A, N O T (B), A1))


{move 1 : B0}

>>> define Ande2 A1 : Dblneglaw (Th5 \
    (A1))


Ande2 : [(.A_1 : prop), (.B_1 : prop), (A1_1
    : that .A_1 And .B_1) =>
    ({def} Dblneglaw (Th5 (A1_1)) : that
    .B_1)]


Ande2 : [(.A_1 : prop), (.B_1 : prop), (A1_1
    : that .A_1 And .B_1) => (--- : that
    .B_1)]


{move 0}

>>> comment * N A T := P N ; T Y P E


{move 1 : B0}

>>> clearcurrent

{move 1}

>>> postulate Nat type


Nat : type


{move 0}

>>> comment * P := E B ; [x : N A T] P R O P
```

```
{move 1}

>>> comment comment Notice the characteristic \
    Lestrade maneuver


{move 1}

>>> comment to declare an abstraction \
    variable


{move 1}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> postulate P x prop


   P : [(x_1 : in Nat) => (--- : prop)]


   {move 1}

   >>> close


{move 1}

>>> save P


{move 1 : P}
```

```
>>> comment P * A L L := P ; P R O P


{move 1 : P}

>>> comment comment Here we have to do \
    some work ;


{move 1 : P}

>>> comment comment we are up against \
    the quite


{move 1 : P}

>>> comment comment different treatment \
    of proof


{move 1 : P}

>>> comment types in Lestrade .


{move 1 : P}

>>> comment comment It is quite hard to \
    make sense


{move 1 : P}

>>> comment comment of without carefully \
    thinking


{move 1 : P}

>>> comment comment about the weird subtyping \
    in


{move 1 : P}
```

```
>>> comment metatypes in Automath .


{move 1 : P}

>>> postulate All P : prop


All : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) => (--- : prop)]


{move 0}

>>> declare xx in Nat


xx : in Nat


{move 1 : P}

>>> declare ev that All P


ev : that All (P)


{move 1 : P}

>>> postulate Alle xx ev : that P xx


Alle : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (xx_1 : in
    Nat), (ev_1 : that All (.P_1)) =>
    (--- : that .P_1 (xx_1))]


{move 0}

>>> clearcurrent P


{move 1 : P}
```

```
>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> postulate univev x : that P x


   univev : [(x_1 : in Nat) => (---
       : that P (x_1))]


   {move 1 : P}

   >>> close


{move 1 : P}

>>> postulate Alli univev : that All P


Alli : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (univev_1
    : [(x_2 : in Nat) => (--- : that
       .P_1 (x_2))]) => (--- : that
    All (.P_1))]


{move 0}

>>> clearcurrent P


{move 1 : P}

>>> comment P * S O M E := N O T (A L L ([X, N A T] N O T (< X > P))) ; P R O P
```

```
{move 1 : P}

>>> open


   {move 2}

   >>> declare xxx in Nat


   xxx : in Nat


   {move 2}

   >>> define Notp xxx : Not (P xxx)


   Notp : [(xxx_1 : in Nat) => (---
       : prop)]


   {move 1 : P}

   >>> close


{move 1 : P}

>>> define Some P : Not (All Notp)


Some : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) =>
    ({def} Not (All ([(xxx_3 : in
       Nat) =>
       ({def} Not (P_1 (xxx_3)) : prop)])) : prop)]


Some : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) => (--- : prop)]


{move 0}
```

```
>>> comment P * K := E B ; N A T


{move 1 : P}

>>> save Notp


{move 1 : Notp}

>>> open


   {move 2}

   >>> declare fixsome that Some P


   fixsome : that Some (P)


   {move 2}

   >>> define somefix fixsome : fixsome


   somefix : [(fixsome_1 : that Some
       (P)) => (--- : that Some (P))]


   {move 1 : Notp}

   >>> close


{move 1 : Notp}

>>> define Somefix P : Imppffull (Not \
    (All Notp), Some P, somefix)


Somefix : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) =>
    ({def} Imppffull (Not (All ([(xxx_4
       : in Nat) =>
```

```
       ({def} Not (P_1 (xxx_4)) : prop)])), Some
    (P_1), [(fixsome_2 : that Some
       (P_1)) =>
       ({def} fixsome_2 : that Some (P_1))]) : that
    Not (All ([(xxx_4 : in Nat) =>
       ({def} Not (P_1 (xxx_4)) : prop)])) Imp
    Some (P_1))]


Somefix : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) => (--- : that
    Not (All ([(xxx_4 : in Nat) =>
       ({def} Not (P_1 (xxx_4)) : prop)])) Imp
    Some (P_1))]


{move 0}

>>> clearcurrent Notp


{move 1 : Notp}

>>> declare K in Nat


K : in Nat


{move 1 : Notp}

>>> comment K * K P := E B ; < K > P


{move 1 : Notp}

>>> declare Kp that P K


Kp : that P (K)


{move 1 : Notp}

>>> comment Kp * S O M E I := [T, [X, N A T] N O T (< X > P)] < K P >< \
    K > T
```

```
{move 1 : Notp}

>>> open


   {move 2}

   >>> declare counterev that All Notp


   counterev : that All (Notp)


   {move 2}

   >>> define step1 counterev : Alle K counterev


   step1 : [(counterev_1 : that All
       (Notp)) => (--- : that Notp
       (K))]


   {move 1 : Notp}

   >>> define step2 counterev : Mp (Kp, step1 \
       counterev)


   step2 : [(counterev_1 : that All
       (Notp)) => (--- : that Con)]


   {move 1 : Notp}

   >>> close


{move 1 : Notp}

>>> define Somei P, K, Kp : Mp (Negproof \
    (step2), Somefix P)
```

```
Somei : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (K_1 : in
    Nat), (Kp_1 : that P_1 (K_1)) =>
    ({def} Negproof ([(counterev_3
       : that All ([(xxx_5 : in Nat) =>
          ({def} Not (P_1 (xxx_5)) : prop)])) =>
       ({def} Kp_1 Mp K_1 Alle counterev_3
       : that Con)]) Mp Somefix (P_1) : that
    Some (P_1))]


Somei : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (K_1 : in
    Nat), (Kp_1 : that P_1 (K_1)) =>
    (--- : that Some (P_1))]


{move 0}

>>> clearcurrent Notp


{move 1 : Notp}

>>> comment P * A := E B ; P R O P


{move 1 : Notp}

>>> declare A prop


A : prop


{move 1 : Notp}

>>> comment A * S := E B ; S O M E (P)


{move 1 : Notp}

>>> declare S that Some P


S : that Some (P)
```

```
{move 1 : Notp}

>>> comment S * A0 := E B ; [X : N A T] [T, < X > P)] A


{move 1 : Notp}

>>> open


   {move 2}

   >>> declare xxx in Nat


   xxx : in Nat


   {move 2}

   >>> declare T that P xxx


   T : that P (xxx)


   {move 2}

   >>> postulate A0 xxx T that A


   A0 : [(xxx_1 : in Nat), (T_1 : that
       P (xxx_1)) => (--- : that A)]


   {move 1 : Notp}

   >>> close


{move 1 : Notp}

>>> comment comment +1
```

```
{move 1 : Notp}

>>> comment A0 * N := E B ; N O T (A)

{move 1 : Notp}

>>> open

   {move 2}

   >>> declare nota1 that Not A

   nota1 : that Not (A)

   {move 2}

   >>> comment N * K := E B ; N A T

   {move 2}

   >>> open

      {move 3}

      >>> declare kk in Nat

      kk : in Nat

      {move 3}

      >>> comment K * T1 := C O N T R A P O S (< K > P, A, < K > A0, N) ; N O T (< K >

      {move 3}

      >>> open
```

```
   {move 4}

   >>> declare zorch that P kk


   zorch : that P (kk)


   {move 4}

   >>> define counterzorch zorch \
       : A0 kk zorch


   counterzorch : [(zorch_1 : that
       P (kk)) => (--- : that
       A)]


   {move 3}

   >>> close


{move 3}

>>> define A1 kk : Imppf (counterzorch)


A1 : [(kk_1 : in Nat) => (---
    : that P (kk_1) Imp A)]


{move 2}

>>> define step1 kk : Contrapos \
    (A1 kk, nota1)


step1 : [(kk_1 : in Nat) => (---
    : that Not (P (kk_1)))]


{move 2}
```

```
      >>> comment N * T2 := < [X : N A T] T1 \
          (X) > S ; C O N


      {move 3}

      >>> close


   {move 2}

   >>> define step2 nota1 : Alli step1


   step2 : [(nota1_1 : that Not (A)) =>
       (--- : that All ([(x'_2 : in
          Nat) =>
          ({def} Not (P (x'_2)) : prop)]))]


   {move 1 : Notp}

   >>> define step3 nota1 : Mp (step2 \
       nota1, S)


   step3 : [(nota1_1 : that Not (A)) =>
       (--- : that Con)]


   {move 1 : Notp}

   >>> close


{move 1 : Notp}

>>> comment AO S O M E E := D B L N E G L A W (A, [T, N O T (A)] T2 \
    -1 (T)) ; A


{move 1 : Notp}

>>> comment comment Note that in the proof \
    of Somee, though
```

```
{move 1 : Notp}

>>> comment comment in general terms it \
    is clear that the logical


{move 1 : Notp}

>>> comment comment structure is similar, the \
    details of the


{move 1 : Notp}

>>> comment comment type system are different \
    enough that it


{move 1 : Notp}

>>> comment is hard to compare the terms \
    .


{move 1 : Notp}

>>> define Somee S, A0 : Dblneglaw (Negproof \
    (step3))


Somee : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (.A_1 : prop), (S_1
    : that Some (.P_1)), (A0_1 : [(xxx_2
       : in Nat), (T_2 : that .P_1 (xxx_2)) =>
       (--- : that .A_1)]) =>
    ({def} Dblneglaw (Negproof ([(nota1_3
       : that Not (.A_1)) =>
       ({def} Alli ([(kk_5 : in Nat) =>
          ({def} Imppf ([(zorch_7 : that
             .P_1 (kk_5)) =>
             ({def} kk_5 A0 zorch_7 : that
             .A_1)]) Contrapos nota1_3
          : that Not (.P_1 (kk_5)))]) Mp
       S_1 : that Con)])) : that .A_1)]
```

```
   Somee : [(.P_1 : [(x_2 : in Nat) =>
          (--- : prop)]), (.A_1 : prop), (S_1
       : that Some (.P_1)), (A0_1 : [(xxx_2
          : in Nat), (T_2 : that .P_1 (xxx_2)) =>
          (--- : that .A_1)]) => (---
       : that .A_1)]


   {move 0}

   >>> clearcurrent

{move 1}

   >>> comment * K := E B ; N A T


   {move 1}

   >>> declare K in Nat


   K : in Nat


   {move 1}

   >>> comment K * L := E B ; N A T


   {move 1}

   >>> declare L in Nat


   L : in Nat


   {move 1}

   >>> comment L * I S := P N ; P R O P


   {move 1}
```

```
>>> save L


{move 1 : L}

>>> postulate Is K L : prop


Is : [(K_1 : in Nat), (L_1 : in Nat) =>
    (--- : prop)]


{move 0}

>>> comment K * R E F L E Q := P N ; I S (K, K)


{move 1 : L}

>>> postulate Refleq K that Is (K, K)


Refleq : [(K_1 : in Nat) => (--- : that
    K_1 Is K_1)]


{move 0}

>>> comment L * I := E B ; I S {K, L)


{move 1 : L}

>>> declare I that Is K L


I : that K Is L


{move 1 : L}

>>> comment I * P := E B ; [X, N A T] P R O P


{move 1 : L}
```

```
>>> open

   {move 2}

   >>> declare x in Nat

   x : in Nat

   {move 2}

   >>> postulate P x : prop

   P : [(x_1 : in Nat) => (--- : prop)]

   {move 1 : L}

   >>> close

{move 1 : L}

>>> save P

{move 1 : P}

>>> comment P * K P := E B ; < K > P

{move 1 : P}

>>> declare Kp that P K

Kp : that P (K)

{move 1 : P}

>>> comment K P * E Q P R E D1 := P N ; < L > P
```

```
{move 1 : P}

>>> comment comment That we actually need \
    the predicate argument


{move 1 : P}

>>> comment comment (though it could \
    be inferred) comes from the


{move 1 : P}

>>> comment comment fact that we do not \
    want to make all substitutions


{move 1 : P}

>>> comment of L for K when we use K = L .


{move 1 : P}

>>> comment an implicit argument version \
    might have uses .


{move 1 : P}

>>> postulate Eqpred1 I P, Kp : that \
    P L


Eqpred1 : [(.K_1 : in Nat), (.L_1
    : in Nat), (I_1 : that .K_1 Is .L_1), (P_1
    : [(x_2 : in Nat) => (--- : prop)]), (Kp_1
    : that P_1 (.K_1)) => (--- : that
    P_1 (.L_1))]


{move 0}

>>> comment I * S Y M E Q := E Q P R E D1 \
```

```
    ([X : N A T] I S (X, K), R E F L E Q (K)) ; I S (L, K)


{move 1 : P}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> define thepred x : Is (x, K)


   thepred : [(x_1 : in Nat) => (---
       : prop)]


   {move 1 : P}

   >>> close


{move 1 : P}

>>> comment right here we use a non - inferrable \
    predicate with Eqpred1 .


{move 1 : P}

>>> define Symeq I : Eqpred1 I thepred, Refleq \
    K


Symeq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (I_1 : that .K_1 Is .L_1) =>
    ({def} Eqpred1 (I_1, [(x_2 : in
       Nat) =>
```

```
      ({def} x_2 Is .K_1 : prop)], Refleq
    (.K_1)) : that .L_1 Is .K_1)]


Symeq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (I_1 : that .K_1 Is .L_1) =>
    (--- : that .L_1 Is .K_1)]


{move 0}

>>> clearcurrent P


{move 1 : P}

>>> comment P * L P := E B ; < L > P


{move 1 : P}

>>> declare Lp that P L


Lp : that P (L)


{move 1 : P}

>>> comment L P * E Q P R E D2 := E Q P R E D1 \
    (L, K, S Y M E Q (K, L, I), P, L P) ; < K > P


{move 1 : P}

>>> define Eqpred2 I P, Lp : Eqpred1 \
    (Symeq (I), P, Lp)


Eqpred2 : [(.K_1 : in Nat), (.L_1
    : in Nat), (I_1 : that .K_1 Is .L_1), (P_1
    : [(x_2 : in Nat) => (--- : prop)]), (Lp_1
    : that P_1 (.L_1)) =>
    ({def} Eqpred1 (Symeq (I_1), P_1, Lp_1) : that
    P_1 (.K_1))]
```

```
Eqpred2 : [(.K_1 : in Nat), (.L_1
    : in Nat), (I_1 : that .K_1 Is .L_1), (P_1
    : [(x_2 : in Nat) => (--- : prop)]), (Lp_1
    : that P_1 (.L_1)) => (--- : that
    P_1 (.K_1))]


{move 0}

>>> comment L * M := E B ; Nat


{move 1 : P}

>>> clearcurrent L


{move 1 : L}

>>> declare M in Nat


M : in Nat


{move 1 : L}

>>> comment M * I := E B ; I S (K, L)


{move 1 : L}

>>> save M


{move 1 : M}

>>> declare I that K Is L


I : that K Is L


{move 1 : M}
```

```
>>> comment I * J := E B ; I S (L, M)


{move 1 : M}

>>> declare J that L Is M


J : that L Is M


{move 1 : M}

>>> comment J * T R E Q := E Q P R E D1 \
    (L, M, J, [X : N A T] I S (K, X), I)


{move 1 : M}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> define thepred x : Is (K, x)


   thepred : [(x_1 : in Nat) => (---
       : prop)]


   {move 1 : M}

   >>> close


{move 1 : M}
```

```
>>> define Treq I J : Eqpred1 (J, thepred, I)


Treq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (I_1 : that
    .K_1 Is .L_1), (J_1 : that .L_1
    Is .M_1) =>
    ({def} Eqpred1 (J_1, [(x_2 : in
       Nat) =>
       ({def} .K_1 Is x_2 : prop)], I_1) : that
    .K_1 Is .M_1)]


Treq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (I_1 : that
    .K_1 Is .L_1), (J_1 : that .L_1
    Is .M_1) => (--- : that .K_1 Is .M_1)]


{move 0}

>>> clearcurrent M


{move 1 : M}

>>> comment M * I := E B ; I S (K, M)


{move 1 : M}

>>> declare I that K Is M


I : that K Is M


{move 1 : M}

>>> comment I * J := E B ; I S (L, M)


{move 1 : M}

>>> declare J that L Is M
```

```
J : that L Is M


{move 1 : M}

>>> comment J * C O N V E Q := T R E Q (K, M, L, I, S Y M E Q (L, M, J)) ; I S (K, L)


{move 1 : M}

>>> define Conveq I J : Treq (I, Symeq \
    (J))


Conveq : [(.K_1 : in Nat), (.L_1
    : in Nat), (.M_1 : in Nat), (I_1
    : that .K_1 Is .M_1), (J_1 : that
    .L_1 Is .M_1) =>
    ({def} I_1 Treq Symeq (J_1) : that
    .K_1 Is .L_1)]


Conveq : [(.K_1 : in Nat), (.L_1
    : in Nat), (.M_1 : in Nat), (I_1
    : that .K_1 Is .M_1), (J_1 : that
    .L_1 Is .M_1) => (--- : that .K_1
    Is .L_1)]


{move 0}

>>> clearcurrent M


{move 1 : M}

>>> comment M * I := E B ; I S (M, K)


{move 1 : M}

>>> declare I that M Is K


I : that M Is K
```

```
{move 1 : M}

>>> comment I * J := E B ; I S (M, L)


{move 1 : M}

>>> declare J that M Is L


J : that M Is L


{move 1 : M}

>>> comment J * D I V E Q := T R E Q (K, M, L, S Y M E Q (M, K, I), J) ; I S (K, L)


{move 1 : M}

>>> define Diveq I J : Treq (Symeq (I), J)


Diveq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (I_1 : that
    .M_1 Is .K_1), (J_1 : that .M_1
    Is .L_1) =>
    ({def} Symeq (I_1) Treq J_1 : that
    .K_1 Is .L_1)]


Diveq : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (I_1 : that
    .M_1 Is .K_1), (J_1 : that .M_1
    Is .L_1) => (--- : that .K_1 Is .L_1)]


{move 0}

>>> clearcurrent M


{move 1 : M}
```

```
>>> comment M * N := E B ; N A T

{move 1 : M}

>>> declare N in Nat

N : in Nat

{move 1 : M}

>>> comment N * I := E B ; I S (K, L)

{move 1 : M}

>>> declare I that K Is L

I : that K Is L

{move 1 : M}

>>> comment I * J := E B ; I S (L, M)

{move 1 : M}

>>> declare J that L Is M

J : that L Is M

{move 1 : M}

>>> comment J * I0 := E B ; I S (M, N)

{move 1 : M}

>>> declare I0 that M Is N
```

```
IO : that M Is N


{move 1 : M}

>>> comment IO * T R3 E Q := T R E Q (K, M, N, T R E Q (K, L, M, I, J), IO)


{move 1 : M}

>>> define Treq3 I J IO : Treq (Treq \
    (I, J), IO)


Treq3 : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (.N_1
    : in Nat), (I_1 : that .K_1 Is .L_1), (J_1
    : that .L_1 Is .M_1), (IO_1 : that
    .M_1 Is .N_1) =>
    ({def} I_1 Treq J_1 Treq IO_1 : that
    .K_1 Is .N_1)]


Treq3 : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (.N_1
    : in Nat), (I_1 : that .K_1 Is .L_1), (J_1
    : that .L_1 Is .M_1), (IO_1 : that
    .M_1 Is .N_1) => (--- : that .K_1
    Is .N_1)]


{move 0}

>>> clearcurrent

{move 1}

>>> comment * P := E B ; [X : N A T] P R O P


{move 1}

>>> open
```

```
   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> postulate P x prop


   P : [(x_1 : in Nat) => (--- : prop)]


   {move 1}

   >>> close


{move 1}

>>> save P


{move 1 : P}

>>> comment P * N O T T W O := [X, N A T] [Y, N A T] [T, < X > P] [U, < Y > P] I S (X


{move 1 : P}

>>> comment comment I am forced to take \
    a different tack


{move 1 : P}

>>> comment due to not having weird Automath \
    subtyping


{move 1 : P}

>>> open
```

```
{move 2}

>>> declare x in Nat


x : in Nat


{move 2}

>>> open


   {move 3}

   >>> declare y in Nat


   y : in Nat


   {move 3}

   >>> define bothptheneq y : ((P x) And \
       (P y)) Imp (x Is y)


   bothptheneq : [(y_1 : in Nat) =>
       (--- : prop)]


   {move 2}

   >>> close


{move 2}

>>> define bothptheneq2 x : All bothptheneq


bothptheneq2 : [(x_1 : in Nat) =>
    (--- : prop)]
```

```
   {move 1 : P}

   >>> close


{move 1 : P}

>>> define Nottwo P : All bothptheneq2


Nottwo : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) =>
    ({def} All ([(x_2 : in Nat) =>
       ({def} All ([(y_3 : in Nat) =>
          ({def} (P_1 (x_2) And P_1
          (y_3)) Imp x_2 Is y_3 : prop)]) : prop)]) : prop)]


Nottwo : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) => (--- : prop)]


{move 0}

>>> clearcurrent P


{move 1 : P}

>>> comment P * O N E := A N D (S O M E (P), N O T T W O (P)) ; P R O P


{move 1 : P}

>>> define One P : (Some P) And (Nottwo \
    P)


One : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) =>
    ({def} Some (P_1) And Nottwo (P_1) : prop)]


One : [(P_1 : [(x_2 : in Nat) =>
       (--- : prop)]) => (--- : prop)]
```

```
{move 0}

>>> comment P * O := E B ; O N E


{move 1 : P}

>>> declare O that One P


O : that One (P)


{move 1 : P}

>>> comment O * I N D I V I D U A L := \
    P N ; N A T


{move 1 : P}

>>> postulate Individual O : in Nat


Individual : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (O_1 : that
    One (.P_1)) => (--- : in Nat)]


{move 0}

>>> comment O * A X I N D I V I D U A L := \
    P N ; < I N D I V I D U A L > P


{move 1 : P}

>>> postulate Axindividual O : that P (Individual \
    O)


Axindividual : [(.P_1 : [(x_2 : in
       Nat) => (--- : prop)]), (O_1
    : that One (.P_1)) => (--- : that
```

```
    .P_1 (Individual (O_1)))]

{move 0}

>>> clearcurrent B

{move 1 : B}

>>> comment * A K := E B ; N A T

{move 1 : B}

>>> comment already declared

{move 1 : B}

>>> comment A * K := E B ; N A T

{move 1 : B}

>>> open

   {move 2}

   >>> declare K in Nat

   K : in Nat

   {move 2}

   >>> comment K * L := E B ; N A T

   {move 2}

   >>> declare L in Nat
```

```
L : in Nat


{move 2}

>>> save L


{move 2 : L}

>>> comment comment +3


{move 2 : L}

>>> comment L * N := E B ; N A T


{move 2 : L}

>>> declare N in Nat


N : in Nat


{move 2 : L}

>>> comment N * P R O P1 := I M P (A, I S (N, K)) ; P R O P


{move 2 : L}

>>> define Prop1 K L N : A Imp (N Is \
    K)


Prop1 : [(K_1 : in Nat), (L_1
    : in Nat), (N_1 : in Nat) =>
    (--- : prop)]


{move 1 : B}

>>> comment N * P R O P2 := I M P (N O T (A), I S (N, L)) ; P R O P
```

```
{move 2 : L}

>>> define Prop2 K L N : (Not A) Imp \
    (N Is L)


Prop2 : [(K_1 : in Nat), (L_1
    : in Nat), (N_1 : in Nat) =>
    (--- : prop)]


{move 1 : B}

>>> comment N * P R O P3 := A N D (P R O P1, P R O P2) ; P R O P


{move 2 : L}

>>> define Prop3 K L N : (Prop1 K L N) And \
    Prop2 K L N


Prop3 : [(K_1 : in Nat), (L_1
    : in Nat), (N_1 : in Nat) =>
    (--- : prop)]


{move 1 : B}

>>> open


   {move 3}

   >>> declare xxx that Prop3 K L N


   xxx : that Prop3 (K, L, N)


   {move 3}

   >>> define xxxid xxx : xxx
```

```
   xxxid : [(xxx_1 : that Prop3 (K, L, N)) =>
       (--- : that Prop3 (K, L, N))]


   {move 2 : L}

   >>> close


{move 2 : L}

>>> define Propfix3 K L N : Imppffull \
    ((Prop1 K L N) And Prop2 K L N, Prop3 \
    K L N, xxxid)


Propfix3 : [(K_1 : in Nat), (L_1
    : in Nat), (N_1 : in Nat) =>
    (--- : that (Prop1 (K_1, L_1, N_1) And
    Prop2 (K_1, L_1, N_1)) Imp
    Prop3 (K_1, L_1, N_1))]


{move 1 : B}

>>> comment L * A0 := E B ; A


{move 2 : L}

>>> open


   {move 3}

   >>> declare A0 that A


   A0 : that A


   {move 3}

   >>> comment A0 * T1 := A N D I (P R O P1 \
       (K), P R O P2 (K), [T, A] R E F L E Q (K), T H2 \
       (N O T (A), I S (K, L), T H1 \
```

```
    (A, AO))) ; P R O P3 (K)


{move 3}

>>> declare yyy in Nat


yyy : in Nat


{move 3}

>>> define Propa1 yyy : Prop1 K L yyy


Propa1 : [(yyy_1 : in Nat) =>
    (--- : prop)]


{move 2 : L}

>>> define Propa2 yyy : Prop2 K L yyy


Propa2 : [(yyy_1 : in Nat) =>
    (--- : prop)]


{move 2 : L}

>>> define Propa3 yyy : Prop3 K L yyy


Propa3 : [(yyy_1 : in Nat) =>
    (--- : prop)]


{move 2 : L}

>>> save yyy


{move 3 : yyy}

>>> open
```

```
   {move 4}

   >>> declare T that A


   T : that A


   {move 4}

   >>> define step1 T : Refleq K


   step1 : [(T_1 : that A) =>
       (--- : that K Is K)]


   {move 3 : yyy}

   >>> close


{move 3 : yyy}

>>> define step2 : Imppf step1


step2 : that A Imp K Is K


{move 2 : L}

>>> define T1 A0 : Mp ((Andi (step2, Th2 \
    (K Is L, Th1 A0))), Propfix3 \
    K L K)


T1 : [(A0_1 : that A) => (---
    : that Prop3 (K, L, K))]


{move 2 : L}

>>> comment A0 * T2 := S O M E I ([X, N A T] P R O P3 \
```

```
    (X), K, T1) ; S O M E ([X, N A T] P R O P3 \
    (X))


{move 3 : yyy}

>>> define T2 A0 : Somei (Propa3, K, T1 \
    A0)


T2 : [(A0_1 : that A) => (---
    : that Some (Propa3))]


{move 2 : L}

>>> comment L * A1 := E B ; N O T (A)


{move 3 : yyy}

>>> declare A1 that Not A


A1 : that Not (A)


{move 3 : yyy}

>>> comment A1 * T3 := A N D I (P R O P1 \
    (L), P R O P2 (L), T H2 (A, I S (L, K), A1), [T, N O T (A)] R E F L E Q (L)
    (L)


{move 3 : yyy}

>>> open


   {move 4}

   >>> declare T that Not A


   T : that Not (A)
```

```
   {move 4}

   >>> define lprop T : Refleq L


   lprop : [(T_1 : that Not (A)) =>
       (--- : that L Is L)]


   {move 3 : yyy}

   >>> close


{move 3 : yyy}

>>> define lprop2 : Imppf (lprop)


lprop2 : that Not (A) Imp L Is
 L


{move 2 : L}

>>> define T3 A1 : Mp (Andi (Th2 \
    (L Is K, A1), lprop2), Propfix3 \
    K L L)


T3 : [(A1_1 : that Not (A)) =>
    (--- : that Prop3 (K, L, L))]


{move 2 : L}

>>> comment A1 * T4 := S O M E I ([X, N A T] P R O P3 \
    (X), L, T3) ; S O M E ([X : N A T] P R O P3 \
    (X))


{move 3 : yyy}

>>> define T4 A1 : Somei (Propa3, L, T3 \
    A1)
```

```
   T4 : [(A1_1 : that Not (A)) =>
       (--- : that Some (Propa3))]


   {move 2 : L}

   >>> comment L * E X I S T E N C E := \
       A N Y C A S E (A, S O M E ([X, N A T] P R O P3 \
       (X), [T, A] T2 (T), [T, N O T (A)] T4 \
       (T)) ; S O M E ([X, N A T] P R O P3 \
       (X))


   {move 3 : yyy}

   >>> close


{move 2 : L}

>>> define Existence K L : Anycase \
    (Imppf T2, Imppf (T4))


Existence : [(K_1 : in Nat), (L_1
    : in Nat) => (--- : that Some
    ([(yyy_2 : in Nat) =>
       ({def} Prop3 (K_1, L_1, yyy_2) : prop)]))]


{move 1 : B}

>>> clearcurrent L


{move 2 : L}

>>> open yyy


   {move 3 : yyy}

   >>> comment L * M := E B ; N A T
```

```
{move 3 : yyy}

>>> declare M in Nat


M : in Nat


{move 3 : yyy}

>>> comment M * P := E B ; P R O P3 \
    (M)


{move 3 : yyy}

>>> declare M2 in Nat


M2 : in Nat


{move 3 : yyy}

>>> declare P that Propa3 M


P : that Propa3 (M)


{move 3 : yyy}

>>> comment P * A0 := E B ; A


{move 3 : yyy}

>>> declare a0 that A


a0 : that A


{move 3 : yyy}
```

```
>>> comment A0 * T5 := < A0 > A N D E1 \
    (P R O P1 (M), P R O P2 (M), P) ; I S (M, K)


{move 3 : yyy}

>>> define T5 P a0 : Mp a0 (Ande1 \
    (P))


T5 : [(.M_1 : in Nat), (P_1
    : that Propa3 (.M_1)), (a0_1
    : that A) => (--- : that .M_1
    Is K)]


{move 2 : L}

>>> comment P * A1 := E B ; N O T (A)


{move 3 : yyy}

>>> declare a1 that Not A


a1 : that Not (A)


{move 3 : yyy}

>>> comment A1 * T6 := < A1 > A N D E2 \
    (P R O P1 (M), P R O P2 (M), P) ; I S (M, L)


{move 3 : yyy}

>>> define T6 P a1 : Mp (a1, Ande2 \
    (P))


T6 : [(.M_1 : in Nat), (P_1
    : that Propa3 (.M_1)), (a1_1
    : that Not (A)) => (--- : that
    .M_1 Is L)]
```

```
{move 2 : L}

>>> comment M * N := E B ; N A T


{move 3 : yyy}

>>> comment already declared as \
    M2 above


{move 3 : yyy}

>>> comment N * P := E B ; P R O P3 \
    (M)


{move 3 : yyy}

>>> comment already declared


{move 3 : yyy}

>>> comment P * Q := E B ; P R O P3 \
    (M2)


{move 3 : yyy}

>>> declare Q that Propa3 M2


Q : that Propa3 (M2)


{move 3 : yyy}

>>> comment Q * A0 := E B ; A


{move 3 : yyy}

>>> comment already declared
```

```
{move 3 : yyy}

>>> open

   {move 4}

   >>> declare aa0 that A

   aa0 : that A

   {move 4}

   >>> declare aa1 that Not A

   aa1 : that Not (A)

   {move 4}

   >>> comment A0 * T7 := C O N V E Q (M, N, K, T5 \
       (M, P, A0), T5 (N, Q, A0)) ; I S (M, N)

   {move 4}

   >>> define T7 aa0 : Conveq (T5 \
       (P, aa0), T5 (Q, aa0))

   T7 : [(aa0_1 : that A) =>
       (--- : that M Is M2)]

   {move 3 : yyy}

   >>> comment Q * A1 := E B ; N O T (A)

   {move 4}

   >>> comment already declared
```

```
   {move 4}

   >>> comment A1 * T8 := C O N V E Q (M, N, L, T6 \
       (M, P, A1), T6 (N, Q, A1)) ; I S (M, N)


   {move 4}

   >>> define T8 aa1 : Conveq (T6 \
       (P, aa1), T6 (Q, aa1))


   T8 : [(aa1_1 : that Not (A)) =>
       (--- : that M Is M2)]


   {move 3 : yyy}

   >>> comment Q * U N I C I T Y := \
       A N Y C A S E (A, I S (M, N), [T, A] T7 \
       (T), [T, N O T (A)] T8 \
       (T)) ; I S (M, N)


   {move 4}

   >>> close


{move 3 : yyy}

>>> define Unicity1 P Q : Anycase \
    (Imppf T7, Imppf (T8))


Unicity1 : [(.M_1 : in Nat), (.M2_1
    : in Nat), (P_1 : that Propa3
    (.M_1)), (Q_1 : that Propa3
    (.M2_1)) => (--- : that .M_1
    Is .M2_1)]


{move 2 : L}
```

```
   >>> close

{move 2 : L}

>>> declare m in Nat

m : in Nat

{move 2 : L}

>>> declare m2 in Nat

m2 : in Nat

{move 2 : L}

>>> declare p that Propa3 m

p : that Propa3 (m)

{move 2 : L}

>>> declare q that Propa3 m2

q : that Propa3 (m2)

{move 2 : L}

>>> define Unicity K L p q : Unicity1 \
    p q

Unicity : [(K_1 : in Nat), (L_1
    : in Nat), (.m_1 : in Nat), (.m2_1
    : in Nat), (p_1 : that Prop3
    (K_1, L_1, .m_1)), (q_1 : that
    Prop3 (K_1, L_1, .m2_1)) =>
```

```
    (--- : that .m_1 Is .m2_1)]


{move 1 : B}

>>> open


   {move 3}

   >>> declare x1 in Nat


   x1 : in Nat


   {move 3}

   >>> open


      {move 4}

      >>> declare x2 in Nat


      x2 : in Nat


      {move 4}

      >>> open


         {move 5}

         >>> declare pp that (Propa3 \
             x1) And Propa3 x2


         pp : that Propa3 (x1) And
          Propa3 (x2)


         {move 5}
```

```
   >>> define qq pp : Ande1 (pp)


   qq : [(pp_1 : that Propa3
       (x1) And Propa3 (x2)) =>
       (--- : that Propa3 (x1))]


   {move 4}

   >>> define rr pp : Ande2 (pp)


   rr : [(pp_1 : that Propa3
       (x1) And Propa3 (x2)) =>
       (--- : that Propa3 (x2))]


   {move 4}

   >>> define ss pp : Unicity1 \
       (qq pp, (rr pp))


   ss : [(pp_1 : that Propa3
       (x1) And Propa3 (x2)) =>
       (--- : that x1 Is x2)]


   {move 4}

   >>> close


{move 4}

>>> define tt x2 : Imppf (ss)


tt : [(x2_1 : in Nat) => (---
    : that (Propa3 (x1) And
    Propa3 (x2_1)) Imp x1 Is
    x2_1)]


{move 3}
```

```
      >>> comment define theprop1 x2 \
          : ((Propa3 x1) And Propa3 \
          x2) Imp x1 Is x2


      {move 4}

      >>> close


   {move 3}

   >>> define uu x1 : Alli tt


   uu : [(x1_1 : in Nat) => (---
       : that All ([(x'_2 : in Nat) =>
          ({def} (Propa3 (x1_1) And
          Propa3 (x'_2)) Imp x1_1
          Is x'_2 : prop)]))]


   {move 2 : L}

   >>> comment define theprop2 x1 : All \
       theprop1


   {move 3}

   >>> close


{move 2 : L}

>>> define Uniqueness K L : Alli uu


Uniqueness : [(K_1 : in Nat), (L_1
    : in Nat) => (--- : that All ([(x'_2
       : in Nat) =>
       ({def} All ([(x'_3 : in Nat) =>
          ({def} (Prop3 (K_1, L_1, x'_2) And
          Prop3 (K_1, L_1, x'_3)) Imp
          x'_2 Is x'_3 : prop)]) : prop)]))]
```

```
{move 1 : B}

>>> comment comment L * T9 := A N D I (S O M E ([X, N A T] P R O P3 \
    (X)), N O T T W O ([X, N A T] P R O P3 \
    (X)), E X I S T E N C E,


{move 2 : L}

>>> comment [X, N A T] [Y, N A T] [T, P R O P3 \
       (X)] [U, P R O P3 (Y)] U N I C I T Y (X, Y, T, U)) ; O N E ([X, N A T] P R O
    (X))


{move 2 : L}

>>> define T9 K L : Andi (Existence \
    K L, Uniqueness K L)


T9 : [(K_1 : in Nat), (L_1 : in
    Nat) => (--- : that Some ([(yyy_3
       : in Nat) =>
       ({def} Prop3 (K_1, L_1, yyy_3) : prop)]) And
    All ([(x'_3 : in Nat) =>
       ({def} All ([(x'_4 : in Nat) =>
          ({def} (Prop3 (K_1, L_1, x'_3) And
          Prop3 (K_1, L_1, x'_4)) Imp
          x'_3 Is x'_4 : prop)]) : prop)]))]


{move 1 : B}

>>> comment L * NO := I N D I V I D U A L ([X, N A T] P R O P3 \
    (X), T9) ; N A T


{move 2 : L}

>>> comment deferred


{move 2 : L}
```

```
>>> comment define Ifthenelse A K L : Individual \
    (T9 A K L)


{move 2 : L}

>>> define T10 K L : Axindividual (T9 \
    K L)


T10 : [(K_1 : in Nat), (L_1 : in
    Nat) => (--- : that Prop3 (K_1, L_1, Individual
    (K_1 T9 L_1)))]


{move 1 : B}

>>> declare M in Nat


M : in Nat


{move 2 : L}

>>> declare P that Propa3 M


P : that Propa3 (M)


{move 2 : L}

>>> comment P * A0 := E B ; A


{move 2 : L}

>>> declare a0 that A


a0 : that A


{move 2 : L}
```

```
   >>> declare a1 that Not A


   a1 : that Not (A)


   {move 2 : L}

   >>> define Ta5 P a0 : T5 P a0


   Ta5 : [(.K_1 : in Nat), (.L_1
       : in Nat), (.M_1 : in Nat), (P_1
       : that Prop3 (.K_1, .L_1, .M_1)), (a0_1
       : that A) => (--- : that .M_1
       Is .K_1)]


   {move 1 : B}

   >>> define Ta6 P a1 : T6 P a1


   Ta6 : [(.K_1 : in Nat), (.L_1
       : in Nat), (.M_1 : in Nat), (P_1
       : that Prop3 (.K_1, .L_1, .M_1)), (a1_1
       : that Not (A)) => (--- : that
       .M_1 Is .L_1)]


   {move 1 : B}

   >>> close


{move 1 : B}

>>> declare K in Nat


K : in Nat


{move 1 : B}

>>> declare L in Nat
```

```
L : in Nat


{move 1 : B}

>>> define Ifthenelse A K L : Individual \
    (T9 K L)


Ifthenelse : [(A_1 : prop), (K_1
    : in Nat), (L_1 : in Nat) =>
    ({def} Individual (Imppf ([(A0_5
       : that A_1) =>
       ({def} Somei ([(yyy_6 : in Nat) =>
          ({def} (A_1 Imp yyy_6 Is K_1) And
          Not (A_1) Imp yyy_6 Is L_1
          : prop)], K_1, Imppf ([(T_9
          : that A_1) =>
          ({def} Refleq (K_1) : that
          K_1 Is K_1)]) Andi (K_1 Is
       L_1) Th2 Th1 (A0_5) Mp Imppffull
       ((A_1 Imp K_1 Is K_1) And Not
       (A_1) Imp K_1 Is L_1, (A_1 Imp
       K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1, [(xxx_8 : that (A_1
          Imp K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1) =>
          ({def} xxx_8 : that (A_1 Imp
          K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1)])) : that Some
       ([(yyy_6 : in Nat) =>
          ({def} (A_1 Imp yyy_6 Is K_1) And
          Not (A_1) Imp yyy_6 Is L_1
          : prop)]))]) Anycase Imppf
    ([(A1_5 : that Not (A_1)) =>
       ({def} Somei ([(yyy_6 : in Nat) =>
          ({def} (A_1 Imp yyy_6 Is K_1) And
          Not (A_1) Imp yyy_6 Is L_1
          : prop)], L_1, (L_1 Is K_1) Th2
       A1_5 Andi Imppf ([(T_9 : that
          Not (A_1)) =>
          ({def} Refleq (L_1) : that
          L_1 Is L_1)]) Mp Imppffull
       ((A_1 Imp L_1 Is K_1) And Not
```

```
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_8 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_8 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_6 : in Nat) =>
      ({def} (A_1 Imp yyy_6 Is K_1) And
      Not (A_1) Imp yyy_6 Is L_1
      : prop)]))]) Andi Alli
([(x1_4 : in Nat) =>
   ({def} Alli ([(x2_5 : in Nat) =>
      ({def} Imppf ([(pp_6 : that
         ((A_1 Imp x1_4 Is K_1) And
         Not (A_1) Imp x1_4 Is L_1) And
         (A_1 Imp x2_5 Is K_1) And
         Not (A_1) Imp x2_5 Is L_1) =>
         ({def} Imppf ([(aa0_8
            : that A_1) =>
            ({def} aa0_8 Mp Ande1
            (Ande1 (pp_6)) Conveq
            aa0_8 Mp Ande1 (Ande2
            (pp_6)) : that x1_4
            Is x2_5)]) Anycase Imppf
         ([(aa1_8 : that Not (A_1)) =>
            ({def} aa1_8 Mp Ande2
            (Ande1 (pp_6)) Conveq
            aa1_8 Mp Ande2 (Ande2
            (pp_6)) : that x1_4
            Is x2_5)]) : that x1_4
         Is x2_5)]) : that (((A_1
      Imp x1_4 Is K_1) And Not (A_1) Imp
      x1_4 Is L_1) And (A_1 Imp x2_5
      Is K_1) And Not (A_1) Imp
      x2_5 Is L_1) Imp x1_4 Is x2_5)]) : that
   All ([(x'_5 : in Nat) =>
      ({def} (((A_1 Imp x1_4 Is
      K_1) And Not (A_1) Imp x1_4
      Is L_1) And (A_1 Imp x'_5 Is
      K_1) And Not (A_1) Imp x'_5
      Is L_1) Imp x1_4 Is x'_5 : prop)]))])) : in
Nat)]
```

```
Ifthenelse : [(A_1 : prop), (K_1
    : in Nat), (L_1 : in Nat) => (---
    : in Nat)]


{move 0}

>>> comment L * I F T H E N E L S E * N0 \
    -3 ; N A T


{move 1 : B}

>>> comment already declared


{move 1 : B}

>>> comment L * A0 := E B ; A


{move 1 : B}

>>> declare A0 that A


A0 : that A


{move 1 : B}

>>> comment A0 * T H E N := T5 -3 (N0 \
    -3 ,T10"-3",A0) ; IS(IFTHENELSE,K)



{move 1 : B}

>>> define Then A K L A0 : Ta5 (T10 K L, A0)


Then : [(A_1 : prop), (K_1 : in Nat), (L_1
    : in Nat), (A0_1 : that A_1) =>
    ({def} A0_1 Mp Ande1 (Axindividual
    (Imppf ([(A0_7 : that A_1) =>
       ({def} Somei ([(yyy_8 : in Nat) =>
```

```
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)], K_1, Imppf ([(T_11
      : that A_1) =>
      ({def} Refleq (K_1) : that
      K_1 Is K_1)]) Andi (K_1 Is
   L_1) Th2 Th1 (A0_7) Mp Imppffull
   ((A_1 Imp K_1 Is K_1) And Not
   (A_1) Imp K_1 Is L_1, (A_1 Imp
   K_1 Is K_1) And Not (A_1) Imp
   K_1 Is L_1, [(xxx_10 : that (A_1
      Imp K_1 Is K_1) And Not (A_1) Imp
      K_1 Is L_1) =>
      ({def} xxx_10 : that (A_1 Imp
      K_1 Is K_1) And Not (A_1) Imp
      K_1 Is L_1)])) : that Some
   ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)]))]) Anycase Imppf
([(A1_7 : that Not (A_1)) =>
   ({def} Somei ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)], L_1, (L_1 Is K_1) Th2
   A1_7 Andi Imppf ([(T_11 : that
      Not (A_1)) =>
      ({def} Refleq (L_1) : that
      L_1 Is L_1)]) Mp Imppffull
   ((A_1 Imp L_1 Is K_1) And Not
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_10 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_10 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)]))]) Andi Alli
([(x1_6 : in Nat) =>
   ({def} Alli ([(x2_7 : in Nat) =>
      ({def} Imppf ([(pp_8 : that
         ((A_1 Imp x1_6 Is K_1) And
```

```
          Not (A_1) Imp x1_6 Is L_1) And
          (A_1 Imp x2_7 Is K_1) And
          Not (A_1) Imp x2_7 Is L_1) =>
          ({def} Imppf ([(aa0_10
             : that A_1) =>
             ({def} aa0_10 Mp Ande1
             (Ande1 (pp_8)) Conveq
             aa0_10 Mp Ande1 (Ande2
             (pp_8)) : that x1_6
             Is x2_7)]) Anycase Imppf
          ([(aa1_10 : that Not (A_1)) =>
             ({def} aa1_10 Mp Ande2
             (Ande1 (pp_8)) Conveq
             aa1_10 Mp Ande2 (Ande2
             (pp_8)) : that x1_6
             Is x2_7)]) : that x1_6
          Is x2_7)]) : that (((A_1
       Imp x1_6 Is K_1) And Not (A_1) Imp
       x1_6 Is L_1) And (A_1 Imp x2_7
       Is K_1) And Not (A_1) Imp
       x2_7 Is L_1) Imp x1_6 Is x2_7)]) : that
    All ([(x'_7 : in Nat) =>
       ({def} (((A_1 Imp x1_6 Is
       K_1) And Not (A_1) Imp x1_6
       Is L_1) And (A_1 Imp x'_7 Is
       K_1) And Not (A_1) Imp x'_7
       Is L_1) Imp x1_6 Is x'_7 : prop)])])]))) : that
 Individual (Imppf ([(A0_6 : that
    A_1) =>
    ({def} Somei ([(yyy_7 : in Nat) =>
       ({def} (A_1 Imp yyy_7 Is K_1) And
       Not (A_1) Imp yyy_7 Is L_1
       : prop)], K_1, Imppf ([(T_10
       : that A_1) =>
       ({def} Refleq (K_1) : that
       K_1 Is K_1)]) Andi (K_1 Is
    L_1) Th2 Th1 (A0_6) Mp Imppffull
    ((A_1 Imp K_1 Is K_1) And Not
    (A_1) Imp K_1 Is L_1, (A_1 Imp
    K_1 Is K_1) And Not (A_1) Imp
    K_1 Is L_1, [(xxx_9 : that (A_1
       Imp K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1) =>
       ({def} xxx_9 : that (A_1 Imp
       K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1)])) : that Some
```

```
   ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)]))]) Anycase Imppf
([(A1_6 : that Not (A_1)) =>
   ({def} Somei ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)], L_1, (L_1 Is K_1) Th2
   A1_6 Andi Imppf ([(T_10 : that
      Not (A_1)) =>
      ({def} Refleq (L_1) : that
      L_1 Is L_1)]) Mp Imppffull
   ((A_1 Imp L_1 Is K_1) And Not
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_9 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_9 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)]))]) Andi Alli
([(x1_5 : in Nat) =>
   ({def} Alli ([(x2_6 : in Nat) =>
      ({def} Imppf ([(pp_7 : that
         ((A_1 Imp x1_5 Is K_1) And
         Not (A_1) Imp x1_5 Is L_1) And
         (A_1 Imp x2_6 Is K_1) And
         Not (A_1) Imp x2_6 Is L_1) =>
         ({def} Imppf ([(aa0_9
            : that A_1) =>
            ({def} aa0_9 Mp Ande1
            (Ande1 (pp_7)) Conveq
            aa0_9 Mp Ande1 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) Anycase Imppf
         ([(aa1_9 : that Not (A_1)) =>
            ({def} aa1_9 Mp Ande2
            (Ande1 (pp_7)) Conveq
            aa1_9 Mp Ande2 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) : that x1_5
```

```
            Is x2_6)]) : that (((A_1
          Imp x1_5 Is K_1) And Not (A_1) Imp
          x1_5 Is L_1) And (A_1 Imp x2_6
          Is K_1) And Not (A_1) Imp
          x2_6 Is L_1) Imp x1_5 Is x2_6)]) : that
       All ([(x'_6 : in Nat) =>
          ({def} (((A_1 Imp x1_5 Is
          K_1) And Not (A_1) Imp x1_5
          Is L_1) And (A_1 Imp x'_6 Is
          K_1) And Not (A_1) Imp x'_6
          Is L_1) Imp x1_5 Is x'_6 : prop)]))])) Is
    K_1)]


Then : [(A_1 : prop), (K_1 : in Nat), (L_1
    : in Nat), (A0_1 : that A_1) =>
    (--- : that Individual (Imppf ([(A0_6
       : that A_1) =>
       ({def} Somei ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)], K_1, Imppf ([(T_10
          : that A_1) =>
          ({def} Refleq (K_1) : that
          K_1 Is K_1)]) Andi (K_1 Is
       L_1) Th2 Th1 (A0_6) Mp Imppffull
       ((A_1 Imp K_1 Is K_1) And Not
       (A_1) Imp K_1 Is L_1, (A_1 Imp
       K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1, [(xxx_9 : that (A_1
          Imp K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1) =>
          ({def} xxx_9 : that (A_1 Imp
          K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1)])) : that Some
       ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)]))]) Anycase Imppf
    ([(A1_6 : that Not (A_1)) =>
       ({def} Somei ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)], L_1, (L_1 Is K_1) Th2
       A1_6 Andi Imppf ([(T_10 : that
          Not (A_1)) =>
```

```
      ({def} Refleq (L_1) : that
      L_1 Is L_1)]) Mp Imppffull
   ((A_1 Imp L_1 Is K_1) And Not
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_9 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_9 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)]))]) Andi Alli
([(x1_5 : in Nat) =>
   ({def} Alli ([(x2_6 : in Nat) =>
      ({def} Imppf ([(pp_7 : that
         ((A_1 Imp x1_5 Is K_1) And
         Not (A_1) Imp x1_5 Is L_1) And
         (A_1 Imp x2_6 Is K_1) And
         Not (A_1) Imp x2_6 Is L_1) =>
         ({def} Imppf ([(aa0_9
            : that A_1) =>
            ({def} aa0_9 Mp Ande1
            (Ande1 (pp_7)) Conveq
            aa0_9 Mp Ande1 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) Anycase Imppf
         ([(aa1_9 : that Not (A_1)) =>
            ({def} aa1_9 Mp Ande2
            (Ande1 (pp_7)) Conveq
            aa1_9 Mp Ande2 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) : that x1_5
         Is x2_6)]) : that (((A_1
      Imp x1_5 Is K_1) And Not (A_1) Imp
      x1_5 Is L_1) And (A_1 Imp x2_6
      Is K_1) And Not (A_1) Imp
      x2_6 Is L_1) Imp x1_5 Is x2_6)]) : that
   All ([(x'_6 : in Nat) =>
      ({def} (((A_1 Imp x1_5 Is
      K_1) And Not (A_1) Imp x1_5
      Is L_1) And (A_1 Imp x'_6 Is
      K_1) And Not (A_1) Imp x'_6
      Is L_1) Imp x1_5 Is x'_6 : prop)]))])) Is
```

```
    K_1)]


{move 0}

>>> comment L * A1 := E B ; N O T (A)


{move 1 : B}

>>> declare A1 that Not A


A1 : that Not (A)


{move 1 : B}

>>> comment A1 * E L S E := T6 -3 (No \
    -3 ,T10"-3",A1) ; IS(IFTHENELSE,L)



{move 1 : B}

>>> define Else A K L A1 : Ta6 (T10 K L, A1)


Else : [(A_1 : prop), (K_1 : in Nat), (L_1
    : in Nat), (A1_1 : that Not (A_1)) =>
    ({def} A1_1 Mp Ande2 (Axindividual
    (Imppf ([(A0_7 : that A_1) =>
       ({def} Somei ([(yyy_8 : in Nat) =>
          ({def} (A_1 Imp yyy_8 Is K_1) And
          Not (A_1) Imp yyy_8 Is L_1
          : prop)], K_1, Imppf ([(T_11
          : that A_1) =>
          ({def} Refleq (K_1) : that
          K_1 Is K_1)]) Andi (K_1 Is
       L_1) Th2 Th1 (A0_7) Mp Imppffull
       ((A_1 Imp K_1 Is K_1) And Not
       (A_1) Imp K_1 Is L_1, (A_1 Imp
       K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1, [(xxx_10 : that (A_1
          Imp K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1) =>
```

```
      ({def} xxx_10 : that (A_1 Imp
      K_1 Is K_1) And Not (A_1) Imp
      K_1 Is L_1)])) : that Some
   ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)]))]) Anycase Imppf
([(A1_7 : that Not (A_1)) =>
   ({def} Somei ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)], L_1, (L_1 Is K_1) Th2
   A1_7 Andi Imppf ([(T_11 : that
      Not (A_1)) =>
      ({def} Refleq (L_1) : that
      L_1 Is L_1)]) Mp Imppffull
   ((A_1 Imp L_1 Is K_1) And Not
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_10 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_10 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_8 : in Nat) =>
      ({def} (A_1 Imp yyy_8 Is K_1) And
      Not (A_1) Imp yyy_8 Is L_1
      : prop)]))]) Andi Alli
([(x1_6 : in Nat) =>
   ({def} Alli ([(x2_7 : in Nat) =>
      ({def} Imppf ([(pp_8 : that
         ((A_1 Imp x1_6 Is K_1) And
         Not (A_1) Imp x1_6 Is L_1) And
         (A_1 Imp x2_7 Is K_1) And
         Not (A_1) Imp x2_7 Is L_1) =>
         ({def} Imppf ([(aa0_10
            : that A_1) =>
            ({def} aa0_10 Mp Ande1
            (Ande1 (pp_8)) Conveq
            aa0_10 Mp Ande1 (Ande2
            (pp_8)) : that x1_6
            Is x2_7)]) Anycase Imppf
         ([(aa1_10 : that Not (A_1)) =>
            ({def} aa1_10 Mp Ande2
            (Ande1 (pp_8)) Conveq
```

```
            aa1_10 Mp Ande2 (Ande2
            (pp_8)) : that x1_6
            Is x2_7)]) : that x1_6
         Is x2_7)]) : that (((A_1
      Imp x1_6 Is K_1) And Not (A_1) Imp
      x1_6 Is L_1) And (A_1 Imp x2_7
      Is K_1) And Not (A_1) Imp
      x2_7 Is L_1) Imp x1_6 Is x2_7)]) : that
   All ([(x'_7 : in Nat) =>
      ({def} (((A_1 Imp x1_6 Is
      K_1) And Not (A_1) Imp x1_6
      Is L_1) And (A_1 Imp x'_7 Is
      K_1) And Not (A_1) Imp x'_7
      Is L_1) Imp x1_6 Is x'_7 : prop)]))]))) : that
Individual (Imppf ([(A0_6 : that
   A_1) =>
   ({def} Somei ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)], K_1, Imppf ([(T_10
      : that A_1) =>
      ({def} Refleq (K_1) : that
      K_1 Is K_1)]) Andi (K_1 Is
   L_1) Th2 Th1 (A0_6) Mp Imppffull
   ((A_1 Imp K_1 Is K_1) And Not
   (A_1) Imp K_1 Is L_1, (A_1 Imp
   K_1 Is K_1) And Not (A_1) Imp
   K_1 Is L_1, [(xxx_9 : that (A_1
      Imp K_1 Is K_1) And Not (A_1) Imp
      K_1 Is L_1) =>
      ({def} xxx_9 : that (A_1 Imp
      K_1 Is K_1) And Not (A_1) Imp
      K_1 Is L_1)])) : that Some
   ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)]))]) Anycase Imppf
([(A1_6 : that Not (A_1)) =>
   ({def} Somei ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)], L_1, (L_1 Is K_1) Th2
   A1_6 Andi Imppf ([(T_10 : that
      Not (A_1)) =>
      ({def} Refleq (L_1) : that
      L_1 Is L_1)]) Mp Imppffull
```

```
   ((A_1 Imp L_1 Is K_1) And Not
   (A_1) Imp L_1 Is L_1, (A_1 Imp
   L_1 Is K_1) And Not (A_1) Imp
   L_1 Is L_1, [(xxx_9 : that (A_1
      Imp L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1) =>
      ({def} xxx_9 : that (A_1 Imp
      L_1 Is K_1) And Not (A_1) Imp
      L_1 Is L_1)])) : that Some
   ([(yyy_7 : in Nat) =>
      ({def} (A_1 Imp yyy_7 Is K_1) And
      Not (A_1) Imp yyy_7 Is L_1
      : prop)]))]) Andi Alli
([(x1_5 : in Nat) =>
   ({def} Alli ([(x2_6 : in Nat) =>
      ({def} Imppf ([(pp_7 : that
         ((A_1 Imp x1_5 Is K_1) And
         Not (A_1) Imp x1_5 Is L_1) And
         (A_1 Imp x2_6 Is K_1) And
         Not (A_1) Imp x2_6 Is L_1) =>
         ({def} Imppf ([(aa0_9
            : that A_1) =>
            ({def} aa0_9 Mp Ande1
            (Ande1 (pp_7)) Conveq
            aa0_9 Mp Ande1 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) Anycase Imppf
         ([(aa1_9 : that Not (A_1)) =>
            ({def} aa1_9 Mp Ande2
            (Ande1 (pp_7)) Conveq
            aa1_9 Mp Ande2 (Ande2
            (pp_7)) : that x1_5
            Is x2_6)]) : that x1_5
         Is x2_6)]) : that (((A_1
      Imp x1_5 Is K_1) And Not (A_1) Imp
      x1_5 Is L_1) And (A_1 Imp x2_6
      Is K_1) And Not (A_1) Imp
      x2_6 Is L_1) Imp x1_5 Is x2_6)]) : that
   All ([(x'_6 : in Nat) =>
      ({def} (((A_1 Imp x1_5 Is
      K_1) And Not (A_1) Imp x1_5
      Is L_1) And (A_1 Imp x'_6 Is
      K_1) And Not (A_1) Imp x'_6
      Is L_1) Imp x1_5 Is x'_6 : prop)]))])) Is
L_1)]
```

```
Else : [(A_1 : prop), (K_1 : in Nat), (L_1
    : in Nat), (A1_1 : that Not (A_1)) =>
    (--- : that Individual (Imppf ([(A0_6
       : that A_1) =>
       ({def} Somei ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)], K_1, Imppf ([(T_10
          : that A_1) =>
          ({def} Refleq (K_1) : that
          K_1 Is K_1)]) Andi (K_1 Is
       L_1) Th2 Th1 (A0_6) Mp Imppffull
       ((A_1 Imp K_1 Is K_1) And Not
       (A_1) Imp K_1 Is L_1, (A_1 Imp
       K_1 Is K_1) And Not (A_1) Imp
       K_1 Is L_1, [(xxx_9 : that (A_1
          Imp K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1) =>
          ({def} xxx_9 : that (A_1 Imp
          K_1 Is K_1) And Not (A_1) Imp
          K_1 Is L_1)])) : that Some
       ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)]))]) Anycase Imppf
    ([(A1_6 : that Not (A_1)) =>
       ({def} Somei ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
          Not (A_1) Imp yyy_7 Is L_1
          : prop)], L_1, (L_1 Is K_1) Th2
       A1_6 Andi Imppf ([(T_10 : that
          Not (A_1)) =>
          ({def} Refleq (L_1) : that
          L_1 Is L_1)]) Mp Imppffull
       ((A_1 Imp L_1 Is K_1) And Not
       (A_1) Imp L_1 Is L_1, (A_1 Imp
       L_1 Is K_1) And Not (A_1) Imp
       L_1 Is L_1, [(xxx_9 : that (A_1
          Imp L_1 Is K_1) And Not (A_1) Imp
          L_1 Is L_1) =>
          ({def} xxx_9 : that (A_1 Imp
          L_1 Is K_1) And Not (A_1) Imp
          L_1 Is L_1)])) : that Some
       ([(yyy_7 : in Nat) =>
          ({def} (A_1 Imp yyy_7 Is K_1) And
```

```
            Not (A_1) Imp yyy_7 Is L_1
            : prop)]))]) Andi Alli
      ([(x1_5 : in Nat) =>
         ({def} Alli ([(x2_6 : in Nat) =>
            ({def} Imppf ([(pp_7 : that
               ((A_1 Imp x1_5 Is K_1) And
               Not (A_1) Imp x1_5 Is L_1) And
               (A_1 Imp x2_6 Is K_1) And
               Not (A_1) Imp x2_6 Is L_1) =>
               ({def} Imppf ([(aa0_9
                  : that A_1) =>
                  ({def} aa0_9 Mp Ande1
                  (Ande1 (pp_7)) Conveq
                  aa0_9 Mp Ande1 (Ande2
                  (pp_7)) : that x1_5
                  Is x2_6)]) Anycase Imppf
               ([(aa1_9 : that Not (A_1)) =>
                  ({def} aa1_9 Mp Ande2
                  (Ande1 (pp_7)) Conveq
                  aa1_9 Mp Ande2 (Ande2
                  (pp_7)) : that x1_5
                  Is x2_6)]) : that x1_5
               Is x2_6)]) : that (((A_1
            Imp x1_5 Is K_1) And Not (A_1) Imp
            x1_5 Is L_1) And (A_1 Imp x2_6
            Is K_1) And Not (A_1) Imp
            x2_6 Is L_1) Imp x1_5 Is x2_6)]) : that
         All ([(x'_6 : in Nat) =>
            ({def} (((A_1 Imp x1_5 Is
            K_1) And Not (A_1) Imp x1_5
            Is L_1) And (A_1 Imp x'_6 Is
            K_1) And Not (A_1) Imp x'_6
            Is L_1) Imp x1_5 Is x'_6 : prop)]))])) Is
      L_1)]


   {move 0}

   >>> clearcurrent

{move 1}

   >>> comment * S E T := P N ; T Y P E


   {move 1}
```

```
>>> postulate Set type


Set : type


{move 0}

>>> comment * K := E B ; N A T


{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment K * S := E B ; S E T


{move 1}

>>> declare S in Set


S : in Set


{move 1}

>>> comment S * I N := P N ; P R O P


{move 1}

>>> postulate In K S : prop


In : [(K_1 : in Nat), (S_1 : in Set) =>
    (--- : prop)]
```

```
   {move 0}

   >>> comment * P := E B ; [X, N A T] P R O P


   {move 1}

   >>> clearcurrent

{move 1}

   >>> open


      {move 2}

      >>> declare x1 in Nat


      x1 : in Nat


      {move 2}

      >>> postulate P x1 : prop


      P : [(x1_1 : in Nat) => (--- : prop)]


      {move 1}

      >>> close


   {move 1}

   >>> comment P * S E T O F := P N ; S E T


   {move 1}

   >>> postulate Setof P : in Set
```

```
Setof : [(P_1 : [(x1_2 : in Nat) =>
       (--- : prop)]) => (--- : in
    Set)]


{move 0}

>>> comment P * K := E B ; N A T


{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment K * K P := E B ; < K > P


{move 1}

>>> declare Kp that P K


Kp : that P (K)


{move 1}

>>> comment K P * I N I := P N ; I N (K, S E T O F (P))


{move 1}

>>> postulate Ini P, K Kp that K In Setof \
    P


Ini : [(P_1 : [(x1_2 : in Nat) =>
       (--- : prop)]), (K_1 : in
    Nat), (Kp_1 : that P_1 (K_1)) =>
    (--- : that K_1 In Setof (P_1))]
```

```
{move 0}

>>> comment K * I := E B ; I N (K, S E T O F (P))


{move 1}

>>> declare I that K In Setof P


I : that K In Setof (P)


{move 1}

>>> comment I * I N E := P N ; < K > P


{move 1}

>>> postulate Ine K I that P K


Ine : [(.P_1 : [(x1_2 : in Nat) =>
       (--- : prop)]), (K_1 : in
    Nat), (I_1 : that K_1 In Setof (.P_1)) =>
    (--- : that .P_1 (K_1))]


{move 0}

>>> clearcurrent

{move 1}

>>> comment + N A T U R A L S


{move 1}

>>> comment * 1 := P N ; N A T


{move 1}
```

```
>>> postulate 1 in Nat


1 : in Nat


{move 0}

>>> comment * K := E B ; N A T


{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment K * S U C := P N ; N A T


{move 1}

>>> postulate Suc K in Nat


Suc : [(K_1 : in Nat) => (--- : in
    Nat)]


{move 0}

>>> comment K * L := E B ; N A T


{move 1}

>>> declare L in Nat


L : in Nat
```

```
{move 1}

>>> comment L * I := E B ; I S (K, L)


{move 1}

>>> save L


{move 1 : L}

>>> declare I that K Is L


I : that K Is L


{move 1 : L}

>>> comment I * A X2 := E Q P R E D1 (K, L, I, [X, N A T] I S (S U C (K), S U C (X)),


{move 1 : L}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> define keqx x : (Suc K) Is (Suc \
       x)


   keqx : [(x_1 : in Nat) => (---
       : prop)]
```

```
   {move 1 : L}

   >>> close


{move 1 : L}

>>> define Ax2 K L I : Eqpred1 (I, keqx, Refleq \
    Suc K)


Ax2 : [(K_1 : in Nat), (L_1 : in
    Nat), (I_1 : that K_1 Is L_1) =>
    ({def} Eqpred1 (I_1, [(x_2 : in
       Nat) =>
       ({def} Suc (K_1) Is Suc (x_2) : prop)], Refleq
    (Suc (K_1))) : that Suc (K_1) Is
    Suc (L_1))]


Ax2 : [(K_1 : in Nat), (L_1 : in
    Nat), (I_1 : that K_1 Is L_1) =>
    (--- : that Suc (K_1) Is Suc (L_1))]


{move 0}

>>> comment K * A X3 := P N ; N O T (I S (S U C (K), 1))


{move 1 : L}

>>> postulate Ax3 K : that Not (Suc K Is \
    1)


Ax3 : [(K_1 : in Nat) => (--- : that
    Not (Suc (K_1) Is 1))]


{move 0}

>>> clearcurrent L
```

```
{move 1 : L}

>>> comment L * I := E B ; I S (S U C (K), S U C (L))


{move 1 : L}

>>> declare I that (Suc K) Is (Suc \
    L)


I : that Suc (K) Is Suc (L)


{move 1 : L}

>>> comment I * A X4 := P N ; I S (K, L)


{move 1 : L}

>>> postulate Ax4 I : that K Is L


Ax4 : [(.K_1 : in Nat), (.L_1 : in
    Nat), (I_1 : that Suc (.K_1) Is
    Suc (.L_1)) => (--- : that .K_1
    Is .L_1)]


{move 0}

>>> clearcurrent

{move 1}

>>> comment * S := E B ; S E T


{move 1}

>>> declare S in Set


S : in Set
```

```
{move 1}

>>> comment S * P R O G R E S S I V E := \
    A L L ([X, N A T] I M P (I N (X, S), I N (S U C (X), S))) ; P R O P


{move 1}

>>> open


   {move 2}

   >>> declare s in Set


   s : in Set


   {move 2}

   >>> open


      {move 3}

      >>> declare x in Nat


      x : in Nat


      {move 3}

      >>> define progress x : (x In s) Imp \
          Suc x In s


      progress : [(x_1 : in Nat) =>
          (--- : prop)]


      {move 2}

      >>> close
```

```
   {move 2}

   >>> define Progressive s : All progress


   Progressive : [(s_1 : in Set) =>
       (--- : prop)]


   {move 1}

   >>> close


{move 1}

>>> comment S * P := E B ; P R O G R E S S I V E (S)


{move 1}

>>> declare P that Progressive S


P : that Progressive (S)


{move 1}

>>> comment P * I := E B ; I N (1, S)


{move 1}

>>> declare I that 1 In S


I : that 1 In S


{move 1}

>>> comment I * K := E B ; N A T
```

```
{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment K * A X5 := P N ; I N (K, S)


{move 1}

>>> comment comment Again, the issue \
    is definition expansion !


{move 1}

>>> comment why won' t it accept S as \
    implicit ?


{move 1}

>>> comment comment it does now .The implicit \
    argument inference feature


{move 1}

>>> comment does not always play nicely \
    with definitions .


{move 1}

>>> postulate Ax5 P I K : that K In S


Ax5 : [(.S_1 : in Set), (P_1 : that
    All ([(x_3 : in Nat) =>
       ({def} (x_3 In .S_1) Imp Suc
```

```
        (x_3) In .S_1 : prop)])), (I_1
      : that 1 In .S_1), (K_1 : in Nat) =>
      (--- : that K_1 In .S_1)]


   {move 0}

   >>> clearcurrent

{move 1}

   >>> comment * P := E B ; [X, N A T] P R O P


   {move 1}

   >>> open


      {move 2}

      >>> declare x in Nat


      x : in Nat


      {move 2}

      >>> postulate P x prop


      P : [(x_1 : in Nat) => (--- : prop)]


      {move 1}

      >>> close


   {move 1}

   >>> comment P * 1 P := E B ; <1 > P


   {move 1}
```

```
>>> declare Onep that P 1


Onep : that P (1)


{move 1}

>>> comment 1 P * A := E B ; A L L) [X, N A T] I M P (< X > P, < S U C (X) > P))


{move 1}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> define progress x : P x Imp P Suc \
       x


   progress : [(x_1 : in Nat) => (---
       : prop)]


   {move 1}

   >>> close


{move 1}

>>> declare A that All progress


A : that All (progress)
```

```
{move 1}

>>> comment A * K := E B ; N A T


{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment +0


{move 1}

>>> comment A * S0 := S E T O F (P) ; S E T


{move 1}

>>> define S0 P : Setof P


S0 : [(P_1 : [(x_2 : in Nat) => (---
       : prop)]) =>
    ({def} Setof (P_1) : in Set)]


S0 : [(P_1 : [(x_2 : in Nat) => (---
       : prop)]) => (--- : in Set)]


{move 0}

>>> comment A * T1 := I N I (P, 1, 1 P) ; I N (1, S0)


{move 1}
```

```
>>> define T1 P, Onep : Ini P, 1 Onep


T1 : [(P_1 : [(x_2 : in Nat) => (---
       : prop)]), (Onep_1 : that P_1
    (1)) =>
    ({def} Ini (P_1, 1, Onep_1) : that
    1 In Setof (P_1))]


T1 : [(P_1 : [(x_2 : in Nat) => (---
       : prop)]), (Onep_1 : that P_1
    (1)) => (--- : that 1 In Setof
    (P_1))]


{move 0}

>>> comment K * I := E B ; I N (K, S0)


{move 1}

>>> declare I that K In S0 P


I : that K In S0 (P)


{move 1}

>>> comment I * T2 := I N I (P, S U C (K), < I N E (P, K, I) >< \
    K > A) ; I N (S U C (K), S0)


{move 1}

>>> comment -0


{move 1}

>>> comment K * I N D U C T I O N := Ine \
    (P, K, A X5 (S0 -0, [X, N A T] [T .I \
       N (X, S0 -0 )]T2"-0"(X,T),T1"-0",K)) ; <K>P
```

```
{move 1}

>>> open


   {move 2}

   >>> declare x in Nat


   x : in Nat


   {move 2}

   >>> open


      {move 3}

      >>> declare ev that x In S0 P


      ev : that x In S0 (P)


      {move 3}

      >>> define step1 ev : Ine x ev


      step1 : [(ev_1 : that x In S0
          (P)) => (--- : that P (x))]


      {move 2}

      >>> define step2 ev : Alle x A


      step2 : [(ev_1 : that x In S0
          (P)) => (--- : that progress
          (x))]
```

```
   {move 2}

   >>> define step3 ev : Mp (step1 \
       ev, step2 ev)


   step3 : [(ev_1 : that x In S0
       (P)) => (--- : that P (Suc
       (x)))]


   {move 2}

   >>> define step4 ev : Ini P, Suc \
       x step3 ev


   step4 : [(ev_1 : that x In S0
       (P)) => (--- : that Suc (x) In
       Setof (P))]


   {move 2}

   >>> close


{move 2}

>>> define progress2 x : Imppf (step4)


progress2 : [(x_1 : in Nat) => (---
    : that (x_1 In S0 (P)) Imp Suc
    (x_1) In Setof (P))]


{move 1}

>>> comment define progressive2 x : Imp \
    (x In S0 P, (Suc x) In S0 P)


{move 2}
```

```
   >>> close


{move 1}

>>> comment comment why could I not make \
    P implicit ?


{move 1}

>>> comment solved : it is hidden in a defined \
    concept progress that isn' t expanded \
    .


{move 1}

>>> comment fixing it also required eta \
    reduction to be added !


{move 1}

>>> define step5 A : Alli progress2


step5 : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (A_1 : that
    All ([(x_3 : in Nat) =>
       ({def} .P_1 (x_3) Imp .P_1 (Suc
       (x_3)) : prop)])) =>
    ({def} Alli ([(x_2 : in Nat) =>
       ({def} Imppf ([(ev_3 : that
          x_2 In S0 (.P_1)) =>
          ({def} Ini (.P_1, Suc (x_2), x_2
          Ine ev_3 Mp x_2 Alle A_1) : that
          Suc (x_2) In Setof (.P_1))]) : that
       (x_2 In S0 (.P_1)) Imp Suc (x_2) In
       Setof (.P_1))]) : that All
    ([(x'_2 : in Nat) =>
       ({def} (x'_2 In S0 (.P_1)) Imp
       Suc (x'_2) In Setof (.P_1) : prop)]))]


step5 : [(.P_1 : [(x_2 : in Nat) =>
```

```
        (--- : prop)]), (A_1 : that
     All ([(x_3 : in Nat) =>
        ({def} .P_1 (x_3) Imp .P_1 (Suc
        (x_3)) : prop)])) => (---
     : that All ([(x'_2 : in Nat) =>
        ({def} (x'_2 In S0 (.P_1)) Imp
        Suc (x'_2) In Setof (.P_1) : prop)]))]


{move 0}

>>> define Induction Onep A, K : Ax5 \
    (step5 A, T1 P, Onep, K)


Induction : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (Onep_1 : that
    .P_1 (1)), (A_1 : that All ([(x_3
       : in Nat) =>
       ({def} .P_1 (x_3) Imp .P_1 (Suc
       (x_3)) : prop)])), (K_1
    : in Nat) =>
    ({def} Ax5 (step5 (A_1), T1 (.P_1, Onep_1), K_1) : that
    K_1 In S0 ([(x'_3 : in Nat) =>
       ({def} .P_1 (x'_3) : prop)]))]


Induction : [(.P_1 : [(x_2 : in Nat) =>
       (--- : prop)]), (Onep_1 : that
    .P_1 (1)), (A_1 : that All ([(x_3
       : in Nat) =>
       ({def} .P_1 (x_3) Imp .P_1 (Suc
       (x_3)) : prop)])), (K_1
    : in Nat) => (--- : that K_1 In S0
    ([(x'_3 : in Nat) =>
       ({def} .P_1 (x'_3) : prop)]))]


{move 0}

>>> clearcurrent

{move 1}

>>> comment * K := E B ; N A T
```

```
{move 1}

>>> declare K in Nat


K : in Nat


{move 1}

>>> comment K * L := E B ; N A T


{move 1}

>>> declare L in Nat


L : in Nat


{move 1}

>>> comment L * L E := [S, S E T] [T, P R O G R E S S I V E (S)] I M P (I N (K, S), I


{move 1}

>>> comment comment This definition has \
    significant preliminaries :


{move 1}

>>> comment comment we introduce Progressive \
    defined in world 0,


{move 1}

>>> comment comment which we avoided in \
    formulating the axioms .


{move 1}
```

```
>>> comment I suspect we will need it \
    .


{move 1}

>>> declare S in Set


S : in Set


{move 1}

>>> open


   {move 2}

   >>> declare K1 in Nat


   K1 : in Nat


   {move 2}

   >>> define progressive1 K1 : (K1 In \
       S) Imp Suc K1 In S


   progressive1 : [(K1_1 : in Nat) =>
       (--- : prop)]


   {move 1}

   >>> close


{move 1}

>>> define Progressive S : All progressive1
```

```
Progressive : [(S_1 : in Set) =>
    ({def} All ([(K1_2 : in Nat) =>
       ({def} (K1_2 In S_1) Imp Suc
       (K1_2) In S_1 : prop)]) : prop)]


Progressive : [(S_1 : in Set) => (---
    : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare S1 in Set


   S1 : in Set


   {move 2}

   >>> define leprop S1 : (Progressive \
       S1) Imp (K In S1) Imp L In S1


   leprop : [(S1_1 : in Set) => (---
       : prop)]


   {move 1}

   >>> close


{move 1}

>>> comment comment I have to define the \
    universal quantifier for sets .


{move 1}
```

```
>>> comment Automath gets it for free \
    from the evil subtyping .


{move 1}

>>> open


   {move 2}

   >>> declare S1 in Set


   S1 : in Set


   {move 2}

   >>> postulate P S1 prop


   P : [(S1_1 : in Set) => (--- : prop)]


   {move 1}

   >>> close


{move 1}

>>> save P


{move 1 : P}

>>> postulate Alls P : prop


Alls : [(P_1 : [(S1_2 : in Set) =>
       (--- : prop)]) => (--- : prop)]


{move 0}
```

```
>>> declare xx in Set


xx : in Set


{move 1 : P}

>>> declare ev that Alls P


ev : that Alls (P)


{move 1 : P}

>>> postulate Allse xx ev : that P xx


Allse : [(.P_1 : [(S1_2 : in Set) =>
       (--- : prop)]), (xx_1 : in
    Set), (ev_1 : that Alls (.P_1)) =>
    (--- : that .P_1 (xx_1))]


{move 0}

>>> clearcurrent P


{move 1 : P}

>>> open


   {move 2}

   >>> declare x in Set


   x : in Set


   {move 2}

   >>> postulate univev x : that P x
```

```
   univev : [(x_1 : in Set) => (---
       : that P (x_1))]


   {move 1 : P}

   >>> close


{move 1 : P}

>>> postulate Allsi univev : that Alls \
    P


Allsi : [(.P_1 : [(S1_2 : in Set) =>
       (--- : prop)]), (univev_1
    : [(x_2 : in Set) => (--- : that
       .P_1 (x_2))]) => (--- : that
    Alls (.P_1))]


{move 0}

>>> define Le K L : Alls leprop


Le : [(K_1 : in Nat), (L_1 : in Nat) =>
    ({def} Alls ([(S1_2 : in Set) =>
       ({def} Progressive (S1_2) Imp
       (K_1 In S1_2) Imp L_1 In S1_2
       : prop)]) : prop)]


Le : [(K_1 : in Nat), (L_1 : in Nat) =>
    (--- : prop)]


{move 0}

>>> open


   {move 2}
```

```
   >>> declare T that Le K L


   T : that K Le L


   {move 2}

   >>> define Tid T : T


   Tid : [(T_1 : that K Le L) => (---
       : that K Le L)]


   {move 1 : P}

   >>> close


{move 1 : P}

>>> define Lefix K L : Imppffull (Alls \
    leprop, Le K L, Tid)


Lefix : [(K_1 : in Nat), (L_1 : in
    Nat) =>
    ({def} Imppffull (Alls ([(S1_3
       : in Set) =>
       ({def} Progressive (S1_3) Imp
       (K_1 In S1_3) Imp L_1 In S1_3
       : prop)]), K_1 Le L_1, [(T_2
       : that K_1 Le L_1) =>
       ({def} T_2 : that K_1 Le L_1)]) : that
    Alls ([(S1_3 : in Set) =>
       ({def} Progressive (S1_3) Imp
       (K_1 In S1_3) Imp L_1 In S1_3
       : prop)]) Imp K_1 Le L_1)]


Lefix : [(K_1 : in Nat), (L_1 : in
    Nat) => (--- : that Alls ([(S1_3
       : in Set) =>
       ({def} Progressive (S1_3) Imp
```

```
       (K_1 In S1_3) Imp L_1 In S1_3
       : prop)]) Imp K_1 Le L_1)]


{move 0}

>>> comment K * R E F L L E := [S, S E T] {T, P R O G R E S S I V E (S)] [U, I N (K, S


{move 1 : P}

>>> open


   {move 2}

   >>> declare S1 in Set


   S1 : in Set


   {move 2}

   >>> open


      {move 3}

      >>> declare T that Progressive S1


      T : that Progressive (S1)


      {move 3}

      >>> open


         {move 4}

         >>> declare U that K In S1


         U : that K In S1
```

```
      {move 4}

      >>> define uid U : U


      uid : [(U_1 : that K In S1) =>
          (--- : that K In S1)]


      {move 3}

      >>> close


   {move 3}

   >>> define step1 T : Imppf (uid)


   step1 : [(T_1 : that Progressive
       (S1)) => (--- : that (K In
       S1) Imp K In S1)]


   {move 2}

   >>> close


{move 2}

>>> define step2 S1 : Imppf (step1)


step2 : [(S1_1 : in Set) => (---
    : that Progressive (S1_1) Imp
    (K In S1_1) Imp K In S1_1)]


{move 1 : P}

>>> comment define prop1 S1 : (Progressive \
    S1) Imp (K In S1) Imp K In S1
```

```
   {move 2}

   >>> close


{move 1 : P}

>>> define step3 K : Allsi step2


step3 : [(K_1 : in Nat) =>
    ({def} Allsi ([(S1_2 : in Set) =>
       ({def} Imppf ([(T_3 : that Progressive
          (S1_2)) =>
          ({def} Imppf ([(U_4 : that
             K_1 In S1_2) =>
             ({def} U_4 : that K_1 In
             S1_2)]) : that (K_1 In
          S1_2) Imp K_1 In S1_2)]) : that
       Progressive (S1_2) Imp (K_1 In
       S1_2) Imp K_1 In S1_2)]) : that
    Alls ([(S1'_2 : in Set) =>
       ({def} Progressive (S1'_2) Imp
       (K_1 In S1'_2) Imp K_1 In S1'_2
       : prop)]))]


step3 : [(K_1 : in Nat) => (--- : that
    Alls ([(S1'_2 : in Set) =>
       ({def} Progressive (S1'_2) Imp
       (K_1 In S1'_2) Imp K_1 In S1'_2
       : prop)]))]


{move 0}

>>> define Reflle K : Mp (step3 K, Lefix \
    K K)


Reflle : [(K_1 : in Nat) =>
    ({def} step3 (K_1) Mp K_1 Lefix
    K_1 : that K_1 Le K_1)]
```

```
Reflle : [(K_1 : in Nat) => (--- : that
    K_1 Le K_1)]


{move 0}

>>> clearcurrent L


{move 1 : L}

>>> comment L * M := E B ; N A T


{move 1 : L}

>>> declare M in Nat


M : in Nat


{move 1 : L}

>>> comment M * L1 := E B ; L E (K, L)


{move 1 : L}

>>> declare L1 that K Le L


L1 : that K Le L


{move 1 : L}

>>> comment L1 * L2 := E B ; L E (L, M)


{move 1 : L}

>>> declare L2 that L Le M


L2 : that L Le M
```

```
{move 1 : L}

>>> comment +*0


{move 1 : L}

>>> comment L2 * S := E B ; S E T


{move 1 : L}

>>> open


   {move 2}

   >>> declare S in Set


   S : in Set


   {move 2}

   >>> comment S * P := E B ; P R O G R E S S I V E (S)


   {move 2}

   >>> open


      {move 3}

      >>> declare P that Progressive S


      P : that Progressive (S)


      {move 3}

      >>> comment P * I := E B ; I N (K, S)
```

```
{move 3}

>>> open


   {move 4}

   >>> declare I that K In S


   I : that K In S


   {move 4}

   >>> comment I * T3 := < I >< \
       P >< S > L1 ; I N (L, S)


   {move 4}

   >>> open


      {move 5}

      >>> declare S1 in Set


      S1 : in Set


      {move 5}

      >>> comment define steptarget1 \
          S1 : Progressive S1 Imp (K In \
          S1) Imp L In S1


      {move 5}

      >>> close
```

```
{move 4}

>>> define step1 : Allse S L1


step1 : that Progressive (S) Imp
 (K In S) Imp L In S


{move 3}

>>> define step2 : Mp (P, step1)


step2 : that (K In S) Imp L In
 S


{move 3}

>>> comment it is a bad thing \
    that there is something called \
    step3 ; cleanup needed


{move 4}

>>> define stepa3 I : Mp (I, step2)


stepa3 : [(I_1 : that K In
    S) => (--- : that L In S)]


{move 3}

>>> comment I * T4 := < T3 >< \
    P >< S > L2 ; I N (M, S)


{move 4}

>>> open


   {move 5}
```

```
   >>> declare S1 in Set


   S1 : in Set


   {move 5}

   >>> comment define steptarget2 \
       S1 : Progressive S1 Imp (L In \
       S1) Imp M In S1


   {move 5}

   >>> close


{move 4}

>>> define stepa4 : Allse S L2


stepa4 : that Progressive (S) Imp
 (L In S) Imp M In S


{move 3}

>>> define stepa5 : Mp (P, stepa4)


stepa5 : that (L In S) Imp
 M In S


{move 3}

>>> define stepa6 I : Mp (stepa3 \
    I, stepa5)


stepa6 : [(I_1 : that K In
    S) => (--- : that M In S)]
```

```
      {move 3}

      >>> close


    {move 3}

    >>> define stepa7 P : Imppf (stepa6)


    stepa7 : [(P_1 : that Progressive
        (S)) => (--- : that (K In
        S) Imp M In S)]


    {move 2}

    >>> close


  {move 2}

  >>> define stepa8 S : Imppf (stepa7)


  stepa8 : [(S_1 : in Set) => (---
      : that Progressive (S_1) Imp (K In
      S_1) Imp M In S_1)]


  {move 1 : L}

  >>> comment define stepatarget9 S : (Progressive \
      S) Imp (K In S) Imp M In S


  {move 2}

  >>> close


{move 1 : L}

>>> define stepa9 L1 L2 : Allsi stepa8
```

```
stepa9 : [(.K_1 : in Nat), (.L_1
    : in Nat), (.M_1 : in Nat), (L1_1
    : that .K_1 Le .L_1), (L2_1 : that
    .L_1 Le .M_1) =>
    ({def} Allsi ([(S_2 : in Set) =>
       ({def} Imppf ([(P_3 : that Progressive
          (S_2)) =>
          ({def} Imppf ([(I_4 : that
             .K_1 In S_2) =>
             ({def} I_4 Mp P_3 Mp S_2
             Allse L1_1 Mp P_3 Mp S_2 Allse
             L2_1 : that .M_1 In S_2)]) : that
          (.K_1 In S_2) Imp .M_1 In S_2)]) : that
       Progressive (S_2) Imp (.K_1 In
       S_2) Imp .M_1 In S_2)]) : that
    Alls ([(S1_2 : in Set) =>
       ({def} Progressive (S1_2) Imp
       (.K_1 In S1_2) Imp .M_1 In S1_2
       : prop)]))]


stepa9 : [(.K_1 : in Nat), (.L_1
    : in Nat), (.M_1 : in Nat), (L1_1
    : that .K_1 Le .L_1), (L2_1 : that
    .L_1 Le .M_1) => (--- : that Alls
    ([(S1_2 : in Set) =>
       ({def} Progressive (S1_2) Imp
       (.K_1 In S1_2) Imp .M_1 In S1_2
       : prop)]))]


{move 0}

>>> comment -0


{move 1 : L}

>>> comment L2 * T R L E := [S, S E T] [T, P R O G R E S S I V E (S)] [U, I N (K, S)]
    -0 (S, T, U) ; L E (K, M)


{move 1 : L}

>>> define Trle L1 L2 : Mp (stepa9 L1 \
```

```
    L2, Lefix K M)


Trle : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (L1_1
    : that .K_1 Le .L_1), (L2_1 : that
    .L_1 Le .M_1) =>
    ({def} L1_1 stepa9 L2_1 Mp .K_1 Lefix
    .M_1 : that .K_1 Le .M_1)]


Trle : [(.K_1 : in Nat), (.L_1 : in
    Nat), (.M_1 : in Nat), (L1_1
    : that .K_1 Le .L_1), (L2_1 : that
    .L_1 Le .M_1) => (--- : that .K_1
    Le .M_1)]


{move 0}

>>> comment starting p .19


{move 1 : L}

>>> clearcurrent L


{move 1 : L}

>>> comment L * L1 := E B ; L E (S U C (K), S U C (L))


{move 1 : L}

>>> declare L1 that (Suc L) Le Suc L


L1 : that Suc (L) Le Suc (L)


{move 1 : L}

>>> comment +2
```

```
{move 1 : L}

>>> comment L1 * S := E B ; S E T

{move 1 : L}

>>> declare S in Set

S : in Set

{move 1 : L}

>>> comment S * P := E B ; P R O G R E S S I V E (S)

{move 1 : L}

>>> declare P that Progressive S

P : that Progressive (S)

{move 1 : L}

>>> comment P * M := E B ; N A T

{move 1 : L}

>>> declare M in Nat

M : in Nat

{move 1 : L}

>>> comment M * N := E B ; N A T

{move 1 : L}
```

```
>>> declare N in Nat


N : in Nat


{move 1 : L}

>>> comment comment There are clear signs \
    here that I need to encapsulate


{move 1 : L}

>>> comment some earlier material for \
    namespace control .


{move 1 : L}

>>> comment N * P R O P1 := A N D (I N (N, S), I S (S U C (N), M)) ; P R O P


{move 1 : L}

>>> define Propa1 S M N : (N In S) And \
    ((Suc N) Is M)


Propa1 : [(S_1 : in Set), (M_1 : in
    Nat), (N_1 : in Nat) =>
    ({def} (N_1 In S_1) And Suc (N_1) Is
    M_1 : prop)]


Propa1 : [(S_1 : in Set), (M_1 : in
    Nat), (N_1 : in Nat) => (--- : prop)]


{move 0}

>>> comment P * S0 := S E T O F ([X, N A T] S O M E ([Y, N A T] P R O P1 \
    (X, Y))) ; S E T


{move 1 : L}
```

```
>>> open

   {move 2}

   >>> declare x0 in Nat


   x0 : in Nat


   {move 2}

   >>> open


      {move 3}

      >>> declare y0 in Nat


      y0 : in Nat


      {move 3}

      >>> define propa1 y0 : Propa1 S x0 \
          y0


      propa1 : [(y0_1 : in Nat) =>
          (--- : prop)]


      {move 2}

      >>> close


   {move 2}

   >>> define sa0 x0 : Some propa1


   sa0 : [(x0_1 : in Nat) => (---
```

```
         : prop)]


      {move 1 : L}

      >>> close


   {move 1 : L}

   >>> define Sa0 S M N : Setof sa0


   Sa0 : [(S_1 : in Set), (M_1 : in
       Nat), (N_1 : in Nat) =>
       ({def} Setof ([(x0_2 : in Nat) =>
          ({def} Some ([(y0_3 : in Nat) =>
             ({def} Propa1 (S_1, x0_2, y0_3) : prop)]) : prop)]) : in
       Set)]


   Sa0 : [(S_1 : in Set), (M_1 : in
       Nat), (N_1 : in Nat) => (--- : in
       Set)]


   {move 0}
end Lestrade execution
```