Implementation of Zermelo's work of 1908 in Lestrade: Part III, opening of Zermelo well-ordering theorem argument

M. Randall Holmes

March 13, 2020

1 Introduction

This document was originally titled as an essay on the proposition that mathematics is what can be done in Automath (as opposed to what can be done in ZFC, for example). Such an essay is still in in my mind, but this particular document has transformed itself into the large project of implementing Zermelo's two important set theory papers of 1908 in Lestrade, with the further purpose of exploring the actual capabilities of Zermelo's system of 1908 as a mathematical foundation, which we think are perhaps underrated.

This is a new version of this document in modules, designed to make it possible to work more efficiently without repeated execution of slow log files when they do not need to be revisited.

2 Zermelo's 1908 proof of the well-ordering theorem

I am now going to carry out or at least attempt a significant piece of mathematics in Lestrade. I shall attempt to directly translate Zermelo's 1908 proof of the well-ordering theorem (which was published at about the same time as the axiomatization implemented above: they are intimately connected) into a Lestrade proof.

Zermelo starts by stating prerequisites which are found in the axiomatization. Point I: he requires the axiom of Separation, stated above. He points out as an important corollary the existence of relative complements.

As point II he requires the existence of power sets, provided by us in the axiomatization above.

As point III, he notes that Separation implies the existence of intersections of (nonempty) sets.

In earlier versions, declarations of the above points appeared here, but we have moved them back into the second file, which implements the axiomatics paper.

Zermelo states the following Theorem, the central result in his argument (the well ordering theorem follows immediately from this theorem and the axiom of choice, as we will verify below).

Theorem: If with every nonempty subset of a set M an element of the subset is associated by some law as "distinguished element", then $\mathcal{P}(x)$, the set of all subsets of M, possesses one and only one subset $\mathbf{M}$ such that to every arbitrary subset P of M there always corresponds one and only one element P_0 of M that includes P as a subset and contains an element of P as its distinguished element. The set M is well-ordered by $\mathbf{M}$.

The apparent second-order quality of this theorem is dispelled by the proof in Lestrade, in which we see how we can introduce the "law" referred to as a hypothetical without in fact allowing quantification over such laws (and nonetheless successfully carry out the proof of the corollary).

We declare the hypotheses of the theorem, the set M and the unspecified law that, given a subset S of M, allows us to select a distinguished element of S.

```
begin Lestrade execution

   >>> comment load whatismath2


   {move 1}
```

```
   >>> clearcurrent
{move 1}
   >>> declare M obj

   M : obj

   {move 1}
   >>> declare Misset that Isset M

   Misset : that Isset (M)

   {move 1}
   >>> open

      {move 2}
      >>> declare S obj

      S : obj

      {move 2}
      >>> declare x obj

      x : obj
```

```
{move 2}

>>> declare subsetev that S <<= M


subsetev : that S <<= M


{move 2}

>>> declare inev that Exists [x => \
       x E S]


inev : that Exists ([(x_2 : obj) =>
    ({def} x_2 E S : prop)])


{move 2}

>>> postulate thelaw S : obj


thelaw : [(S_1 : obj) => (--- : obj)]


{move 1}

>>> postulate thelawchooses subsetev \
    inev : that (thelaw S) E S


thelawchooses : [(.S_1 : obj), (subsetev_1
    : that .S_1 <<= M), (inev_1 : that
    Exists ([(x_3 : obj) =>
       ({def} x_3 E .S_1 : prop)])) =>
    (--- : that thelaw (.S_1) E .S_1)]
```

```
      {move 1}

      >>> close


   {move 1}
end Lestrade execution
```

It appears for direct implementation of Zermelo's argument that the choice of a distinguished element should return something arbitrary when the argument is empty, and further it is convenient for the choice to work for any set (or indeed atom) at all, formally, though we make no assumptions about the result. Otherwise we need axioms handling proof indifference and there are other embarrassments.

This seems to be a general fact: operations taking sets to sets (or more generally, sets and atoms to sets and atoms) should be universal, or we end up stumbling over the Lestrade type system.

```
begin Lestrade execution

   >>> open


      {move 2}

      >>> declare S obj


      S : obj


      {move 2}

      >>> declare subsetev that S <<= M
```

```
   subsetev : that S <<= M


   {move 2}

   >>> define prime1 S : Complement (S, Usc \
       (thelaw S))


   prime1 : [(S_1 : obj) =>
       ({def} S_1 Complement Usc (thelaw
       (S_1)) : obj)]


   prime1 : [(S_1 : obj) => (--- : obj)]


   {move 1}

   >>> save


   {move 2}

   >>> close


{move 1}

>>> declare S1 obj


S1 : obj


{move 1}
```

```
>>> define prime2 thelaw, S1 : prime1 \
    S1


prime2 : [(thelaw_1 : [(S_2 : obj) =>
       (--- : obj)]), (S1_3 : obj) =>
    ({def} S1_3 Complement Usc (thelaw_1
    (S1_3)) : obj)]


prime2 : [(thelaw_1 : [(S_2 : obj) =>
       (--- : obj)]), (S1_3 : obj) =>
    (--- : obj)]


{move 0}

>>> open


   {move 2}

   >>> define prime S : prime2 thelaw, S


   prime : [(S_1 : obj) =>
       ({def} prime2 (thelaw, S_1) : obj)]


   prime : [(S_1 : obj) => (--- : obj)]


   {move 1}
end Lestrade execution
```

An important operation in Zermelo's argument takes each subset A of M to the subset $A' = A \setminus \{a\}$, where a is the distinguished element of A if A is nonempty. This is defined above. Below, we prove its natural comprehension

axiom.

```
begin Lestrade execution

      >>> open


         {move 3}

         >>> declare u obj


         u : obj


         {move 3}

         >>> open


            {move 4}

            >>> declare hyp1 that u E prime \
                S


            hyp1 : that u E prime (S)


            {move 4}

            >>> declare hyp2 that (u E S) & ~ (u = thelaw \
                S)


            hyp2 : that (u E S) & ~ (u = thelaw
             (S))
```

```
{move 4}

>>> define line1 hyp1 : Iff1 \
    (hyp1, Ui (u, Compax (S, Usc \
    (thelaw S))))


line1 : [(hyp1_1 : that u E prime
    (S)) =>
    ({def} hyp1_1 Iff1 u Ui S Compax
    Usc (thelaw (S)) : that
    (u E S) & ~ (u E Usc (thelaw
    (S))))]


line1 : [(hyp1_1 : that u E prime
    (S)) => (--- : that (u E S) & ~ (u E Usc
    (thelaw (S))))]


{move 3}

>>> define line2 hyp1 : Simp2 \
    (line1 hyp1)


line2 : [(hyp1_1 : that u E prime
    (S)) =>
    ({def} Simp2 (line1 (hyp1_1)) : that
    ~ (u E Usc (thelaw (S))))]


line2 : [(hyp1_1 : that u E prime
    (S)) => (--- : that ~ (u E Usc
    (thelaw (S))))]
```

```
{move 3}

>>> open


   {move 5}

   >>> declare hyp3 that u = thelaw \
       S


   hyp3 : that u = thelaw (S)


   {move 5}

   >>> define line3 : Inusc2 \
       (thelaw S)


   line3 : [
       ({def} Inusc2 (thelaw
       (S)) : that thelaw (S) E thelaw
       (S) ; thelaw (S))]


   line3 : that thelaw (S) E thelaw
    (S) ; thelaw (S)


   {move 4}

   >>> declare v obj


   v : obj
```

```
{move 5}

>>> define line4 hyp3 : Subs \
    (Eqsymm hyp3, [v => v E (Usc \
       (thelaw S))], line3)


line4 : [(hyp3_1 : that
    u = thelaw (S)) =>
    ({def} Subs (Eqsymm (hyp3_1), [(v_2
       : obj) =>
       ({def} v_2 E Usc (thelaw
       (S)) : prop)], line3) : that
    u E Usc (thelaw (S)))]


line4 : [(hyp3_1 : that
    u = thelaw (S)) => (---
    : that u E Usc (thelaw
    (S)))]


{move 4}

>>> define line5 hyp3 : line4 \
    hyp3 Mp line2 hyp1


line5 : [(hyp3_1 : that
    u = thelaw (S)) =>
    ({def} line4 (hyp3_1) Mp
    line2 (hyp1) : that ??)]


line5 : [(hyp3_1 : that
    u = thelaw (S)) => (---
    : that ??)]
```

```
   {move 4}

   >>> close


{move 4}

>>> define line6 hyp1 : Conj \
    (Simp1 line1 hyp1, Negintro \
    line5)


line6 : [(hyp1_1 : that u E prime
    (S)) =>
    ({def} Simp1 (line1 (hyp1_1)) Conj
    Negintro ([(hyp3_5 : that
       u = thelaw (S)) =>
       ({def} Subs (Eqsymm (hyp3_5), [(v_7
          : obj) =>
          ({def} v_7 E Usc (thelaw
          (S)) : prop)], Inusc2
       (thelaw (S))) Mp line2
       (hyp1_1) : that ??)]) : that
    (u E S) & ~ (u = thelaw
    (S)))]


line6 : [(hyp1_1 : that u E prime
    (S)) => (--- : that (u E S) & ~ (u = thelaw
    (S)))]


{move 3}

>>> open
```

```
{move 5}

>>> declare hyp4 that u E Usc \
    (thelaw S)


hyp4 : that u E Usc (thelaw
 (S))


{move 5}

>>> define line8 hyp4 : Mp \
    (Inusc1 hyp4, Simp2 hyp2)


line8 : [(hyp4_1 : that
    u E Usc (thelaw (S))) =>
    ({def} Inusc1 (hyp4_1) Mp
    Simp2 (hyp2) : that ??)]


line8 : [(hyp4_1 : that
    u E Usc (thelaw (S))) =>
    (--- : that ??)]


{move 4}

>>> close


{move 4}

>>> define line9 hyp2 : Conj \
    (Simp1 hyp2, Negintro line8)
```

```
line9 : [(hyp2_1 : that (u E S) & ~ (u = thelaw
    (S))) =>
    ({def} Simp1 (hyp2_1) Conj
    Negintro ([(hyp4_3 : that
       u E Usc (thelaw (S))) =>
       ({def} Inusc1 (hyp4_3) Mp
       Simp2 (hyp2_1) : that
       ??)]) : that (u E S) & ~ (u E Usc
    (thelaw (S))))]


line9 : [(hyp2_1 : that (u E S) & ~ (u = thelaw
    (S))) => (--- : that
    (u E S) & ~ (u E Usc (thelaw
    (S))))]


{move 3}

>>> define line10 hyp2 : Iff2 \
    (line9 hyp2, Ui (u, Compax \
    (S, Usc (thelaw S))))


line10 : [(hyp2_1 : that (u E S) & ~ (u = thelaw
    (S))) =>
    ({def} line9 (hyp2_1) Iff2
    u Ui S Compax Usc (thelaw
    (S)) : that u E S Complement
    Usc (thelaw (S)))]


line10 : [(hyp2_1 : that (u E S) & ~ (u = thelaw
    (S))) => (--- : that
    u E S Complement Usc (thelaw
    (S)))]
```

```
   {move 3}

   >>> close


{move 3}

>>> define bothways u : Dediff line6, line10


bothways : [(u_1 : obj) =>
    ({def} Dediff ([(hyp1_7 : that
       u_1 E prime (S)) =>
       ({def} Simp1 (hyp1_7 Iff1
       u_1 Ui S Compax Usc (thelaw
       (S))) Conj Negintro ([(hyp3_9
          : that u_1 = thelaw (S)) =>
          ({def} Subs (Eqsymm (hyp3_9), [(v_11
             : obj) =>
             ({def} v_11 E Usc (thelaw
             (S)) : prop)], Inusc2
          (thelaw (S))) Mp Simp2
          (hyp1_7 Iff1 u_1 Ui S Compax
          Usc (thelaw (S))) : that
          ??)]) : that (u_1 E S) & ~ (u_1
       = thelaw (S)))], [(hyp2_7
       : that (u_1 E S) & ~ (u_1
       = thelaw (S))) =>
       ({def} Simp1 (hyp2_7) Conj
       Negintro ([(hyp4_10 : that
          u_1 E Usc (thelaw (S))) =>
          ({def} Inusc1 (hyp4_10) Mp
          Simp2 (hyp2_7) : that
          ??)]) Iff2 u_1 Ui S Compax
       Usc (thelaw (S)) : that
       u_1 E S Complement Usc (thelaw
       (S)))]) : that (u_1
```

```
        E prime (S)) == (u_1 E S) & ~ (u_1
        = thelaw (S)))]


   bothways : [(u_1 : obj) => (---
        : that (u_1 E prime (S)) ==
        (u_1 E S) & ~ (u_1 = thelaw
        (S)))]


   {move 2}

   >>> close


{move 2}

>>> define primeax subsetev : Ug bothways


primeax : [(.S_1 : obj), (subsetev_1
    : that .S_1 <<= M) =>
    ({def} Ug ([(u_3 : obj) =>
       ({def} Dediff ([(hyp1_4 : that
          u_3 E prime (.S_1)) =>
          ({def} Simp1 (hyp1_4 Iff1
          u_3 Ui .S_1 Compax Usc (thelaw
          (.S_1))) Conj Negintro
          ([(hyp3_6 : that u_3 = thelaw
             (.S_1)) =>
             ({def} Subs (Eqsymm (hyp3_6), [(v_8
                : obj) =>
                ({def} v_8 E Usc (thelaw
                (.S_1)) : prop)], Inusc2
             (thelaw (.S_1))) Mp
             Simp2 (hyp1_4 Iff1 u_3
             Ui .S_1 Compax Usc (thelaw
             (.S_1))) : that ??)]) : that
```

```
                (u_3 E .S_1) & ~ (u_3 = thelaw
                (.S_1)))], [(hyp2_4
                : that (u_3 E .S_1) & ~ (u_3
                = thelaw (.S_1))) =>
                ({def} Simp1 (hyp2_4) Conj
                Negintro ([(hyp4_7 : that
                   u_3 E Usc (thelaw (.S_1))) =>
                   ({def} Inusc1 (hyp4_7) Mp
                   Simp2 (hyp2_4) : that
                   ??)]) Iff2 u_3 Ui .S_1
                Compax Usc (thelaw (.S_1)) : that
                u_3 E .S_1 Complement Usc
                (thelaw (.S_1)))]) : that
             (u_3 E prime (.S_1)) == (u_3
             E .S_1) & ~ (u_3 = thelaw (.S_1)))]) : that
          Forall ([(x'_12 : obj) =>
             ({def} (x'_12 E prime (.S_1)) ==
             (x'_12 E .S_1) & ~ (x'_12
             = thelaw (.S_1)) : prop)]))]


       primeax : [(.S_1 : obj), (subsetev_1
           : that .S_1 <<= M) => (--- : that
           Forall ([(x'_12 : obj) =>
              ({def} (x'_12 E prime (.S_1)) ==
              (x'_12 E .S_1) & ~ (x'_12
              = thelaw (.S_1)) : prop)]))]


       {move 1}

       >>> close


   {move 1}
end Lestrade execution
```

Now we are ready to define the central concept of this central argument,

the idea of a Θ-chain. The definition of a Θ-chain has four clauses, and there are a tiresome number of occurrences in this document of proofs of the four statements required to show that a particular set is a Θ-chain.

```
begin Lestrade execution

   >>> declare C11 obj


   C11 : obj


   {move 1}

   >>> declare D11 obj


   D11 : obj


   {move 1}

   >>> declare F11 obj


   F11 : obj


   {move 1}

   >>> define thetachain1 M thelaw, C11 \
       : (M E C11) & (C11 <<= Sc (M)) & Forall \
       [D11 => (D11 E C11) -> (prime D11) E C11] & Forall \
       [D11 => Forall [F11 => ((D11 <<= C11) & (F11 \
             E D11)) -> (Intersection D11 \
             F11) E C11]]
```

```
thetachain1 : [(M_1 : obj), (thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (C11_3
    : obj) =>
    ({def} (M_1 E C11_3) & (C11_3 <<=
    Sc (M_1)) & Forall ([(D11_8 : obj) =>
       ({def} (D11_8 E C11_3) -> prime2
       (thelaw_1, D11_8) E C11_3 : prop)]) & Forall
    ([(D11_8 : obj) =>
       ({def} Forall ([(F11_9 : obj) =>
          ({def} ((D11_8 <<= C11_3) & F11_9
          E D11_8) -> (D11_8 Intersection
          F11_9) E C11_3 : prop)]) : prop)]) : prop)]


thetachain1 : [(M_1 : obj), (thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (C11_3
    : obj) => (--- : prop)]


{move 0}

>>> open


   {move 2}

   >>> declare C obj


   C : obj


   {move 2}

   >>> declare D obj
```

```
D : obj


{move 2}

>>> declare F obj


F : obj


{move 2}

>>> define thetachain C : thetachain1 \
    M thelaw, C


thetachain : [(C_1 : obj) =>
    ({def} thetachain1 (M, thelaw, C_1) : prop)]


thetachain : [(C_1 : obj) => (---
    : prop)]


{move 1}

>>> declare thetaev that thetachain \
    C


thetaev : that thetachain (C)


{move 2}

>>> declare G obj
```

```
G : obj


{move 2}

>>> declare ginev that G E C


ginev : that G E C


{move 2}

>>> define Setsinchains1 thetaev ginev \
    : Simp1 (Simp2 (Iff1 (Mp (ginev, Ui \
    (G, Simp1 (Simp1 (Simp2 thetaev)))), Ui \
    G Scthm M)))


Setsinchains1 : [(.C_1 : obj), (thetaev_1
    : that thetachain (.C_1)), (.G_1
    : obj), (ginev_1 : that .G_1
    E .C_1) =>
    ({def} Simp1 (Simp2 (ginev_1
    Mp .G_1 Ui Simp1 (Simp1 (Simp2
    (thetaev_1))) Iff1 .G_1 Ui Scthm
    (M))) : that Isset (.G_1))]


Setsinchains1 : [(.C_1 : obj), (thetaev_1
    : that thetachain (.C_1)), (.G_1
    : obj), (ginev_1 : that .G_1
    E .C_1) => (--- : that Isset (.G_1))]


{move 1}
```

```
   >>> save


   {move 2}

   >>> close


{move 1}

>>> declare C10 obj


C10 : obj


{move 1}

>>> declare thetaev10 that thetachain \
    C10


thetaev10 : that thetachain (C10)


{move 1}

>>> declare G10 obj


G10 : obj


{move 1}

>>> declare ginev10 that G10 E C10
```

```
ginev10 : that G10 E C10


{move 1}

>>> define Setsinchains2 Misset thelawchooses, thetaev10 \
    ginev10 : Setsinchains1 thetaev10 ginev10


Setsinchains2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
          ({def} x_4 E .S_2 : prop)])) =>
       (--- : that .thelaw_1 (.S_2) E .S_2)]), (.C10_3
    : obj), (thetaev10_3 : that thetachain1
    (.M_1, .thelaw_1, .C10_3)), (.G10_3
    : obj), (ginev10_3 : that .G10_3
    E .C10_3) =>
    ({def} Simp1 (Simp2 (ginev10_3 Mp
    .G10_3 Ui Simp1 (Simp1 (Simp2 (thetaev10_3))) Iff1
    .G10_3 Ui Scthm (.M_1))) : that
    Isset (.G10_3))]


Setsinchains2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
          ({def} x_4 E .S_2 : prop)])) =>
       (--- : that .thelaw_1 (.S_2) E .S_2)]), (.C10_3
    : obj), (thetaev10_3 : that thetachain1
    (.M_1, .thelaw_1, .C10_3)), (.G10_3
    : obj), (ginev10_3 : that .G10_3
```

```
    E .C10_3) => (--- : that Isset (.G10_3))]


  {move 0}

  >>> open


    {move 2}

    >>> define Setsinchains thetaev ginev \
        : Setsinchains2 Misset, thelawchooses, thetaev \
        ginev


    Setsinchains : [(.C_1 : obj), (thetaev_1
        : that thetachain (.C_1)), (.G_1
        : obj), (ginev_1 : that .G_1
        E .C_1) =>
        ({def} Setsinchains2 (Misset, thelawchooses, thetaev_1, ginev_1) : th
        Isset (.G_1))]


    Setsinchains : [(.C_1 : obj), (thetaev_1
        : that thetachain (.C_1)), (.G_1
        : obj), (ginev_1 : that .G_1
        E .C_1) => (--- : that Isset (.G_1))]


    {move 1}

    >>> close


  {move 1}
end Lestrade execution
```

We did some extra work to ensure that `thetachain` is defined in terms

of a notion at move 0 which will not be expanded everywhere it occurs.

We then proved that each element of a Θ-chain is a set and again did work to control definitional expansion.

We then need to prove that $\mathcal{P}(M)$ is a Θ-chain.

```
begin Lestrade execution

   >>> open


      {move 2}

      >>> open


         {move 3}

         >>> define line1 : Misset Mp Ui \
             M Inownpowerset


         line1 : Misset Mp M Ui Inownpowerset


         line1 : that M E Sc (M)


         {move 2}

         >>> define line2 : (Misset Mp Ui \
             M Scofsetisset) Mp Ui Sc M Subsetrefl


         line2 : Misset Mp M Ui Scofsetisset
          Mp Sc (M) Ui Subsetrefl
```

```
         line2 : that Sc (M) <<= Sc (M)


         {move 2}
end Lestrade execution
```

The first two lines establish the first two of the four points needed to show that $\mathcal{P}(M)$ is a Θ-chain.

```
begin Lestrade execution

         >>> open


            {move 4}

            >>> declare u obj


            u : obj


            {move 4}

            >>> open


               {move 5}

               >>> declare usinev that u E Sc \
                   M


               usinev : that u E Sc (M)


               {move 5}
```

```
>>> define line3 : Fixform \
    (Isset (prime u), Separation3 \
    (Refleq prime u))


line3 : [
    ({def} Isset (prime (u)) Fixform
    Separation3 (Refleq (prime
    (u))) : that Isset
    (prime (u)))]


line3 : that Isset (prime
 (u))


{move 4}
end Lestrade execution
```

Note the devious use of `Separation3` to avoid having to write out the defining predicate of the prime operation. The use of the axiom `Separation2` is essential: the result of applying the prime operation might be empty, and we do want it to be a set.

```
begin Lestrade execution

>>> define line4 usinev : Simp1 \
    (Simp2 (Iff1 (usinev, Ui \
    u (Scthm M))))


line4 : [(usinev_1 : that
    u E Sc (M)) =>
    ({def} Simp1 (Simp2 (usinev_1
    Iff1 u Ui Scthm (M))) : that
    Isset (u))]
```

```
line4 : [(usinev_1 : that
    u E Sc (M)) => (---
    : that Isset (u))]


{move 4}

>>> open


   {move 6}

   >>> declare v obj


   v : obj


   {move 6}

   >>> open


      {move 7}

      >>> declare vinev that \
          v E prime u


      vinev : that v E prime
       (u)


      {move 7}

      >>> define line5 : Ui \
```

```
    v primeax (Iff1 (usinev, Ui \
    u Scthm M))


line5 : v Ui primeax
 (usinev Iff1 u Ui Scthm
 (M))


line5 : that (v E prime
 (u)) == (v E u) & ~ (v = thelaw
 (u))


{move 6}

>>> define line6 vinev \
    : Simp1 (Iff1 (vinev, line5))


line6 : [(vinev_1
    : that v E prime
    (u)) =>
    ({def} Simp1 (vinev_1
    Iff1 line5) : that
    v E u)]


line6 : [(vinev_1
    : that v E prime
    (u)) => (---
    : that v E u)]


{move 6}

>>> define line7 vinev \
    : Mp line6 vinev (Ui \
```

```
        v Simp1 (Iff1 (usinev, Ui \
        u Scthm M)))


    line7 : [(vinev_1
        : that v E prime
        (u)) =>
        ({def} line6 (vinev_1) Mp
        v Ui Simp1 (usinev
        Iff1 u Ui Scthm (M)) : that
        v E M)]


    line7 : [(vinev_1
        : that v E prime
        (u)) => (---
        : that v E M)]


    {move 6}

    >>> close


{move 6}

>>> define line8 v : Ded \
    line7


line8 : [(v_1 : obj) =>
    ({def} Ded ([(vinev_9
        : that v_1 E prime
        (u)) =>
        ({def} Simp1 (vinev_9
        Iff1 v_1 Ui primeax
        (usinev Iff1 u Ui
        Scthm (M))) Mp
```

```
          v_1 Ui Simp1 (usinev
          Iff1 u Ui Scthm (M)) : that
          v_1 E M)]) : that
       (v_1 E prime (u)) ->
       v_1 E M)]


   line8 : [(v_1 : obj) =>
       (--- : that (v_1 E prime
       (u)) -> v_1 E M)]


   {move 5}

   >>> close


{move 5}

>>> define line9 usinev : Ug \
    line8


line9 : [(usinev_1 : that
    u E Sc (M)) =>
    ({def} Ug ([(v_3 : obj) =>
       ({def} Ded ([(vinev_4
          : that v_3 E prime
          (u)) =>
          ({def} Simp1 (vinev_4
          Iff1 v_3 Ui primeax
          (usinev_1 Iff1 u Ui
          Scthm (M))) Mp
          v_3 Ui Simp1 (usinev_1
          Iff1 u Ui Scthm (M)) : that
          v_3 E M)]) : that
       (v_3 E prime (u)) ->
       v_3 E M)]) : that
```

```
    Forall ([(x'_13 : obj) =>
       ({def} (x'_13 E prime
       (u)) -> x'_13 E M : prop)]))]


line9 : [(usinev_1 : that
    u E Sc (M)) => (---
    : that Forall ([(x'_13
       : obj) =>
       ({def} (x'_13 E prime
       (u)) -> x'_13 E M : prop)]))]


{move 4}

>>> define line10 usinev : Fixform \
    ((prime u) <<= M, Conj \
    (line9 usinev, Conj (line3, Misset)))


line10 : [(usinev_1 : that
    u E Sc (M)) =>
    ({def} (prime (u) <<=
    M) Fixform line9 (usinev_1) Conj
    line3 Conj Misset : that
    prime (u) <<= M)]


line10 : [(usinev_1 : that
    u E Sc (M)) => (---
    : that prime (u) <<=
    M)]


{move 4}

>>> define line11 usinev : Iff2 \
    (line10 usinev, Ui (prime \
```

```
        u, Scthm M))


   line11 : [(usinev_1 : that
        u E Sc (M)) =>
        ({def} line10 (usinev_1) Iff2
        prime (u) Ui Scthm (M) : that
        prime (u) E Sc (M))]


   line11 : [(usinev_1 : that
        u E Sc (M)) => (---
        : that prime (u) E Sc
        (M))]


   {move 4}

   >>> close


{move 4}

>>> define line12 u : Ded line11


line12 : [(u_1 : obj) =>
    ({def} Ded ([(usinev_7
        : that u_1 E Sc (M)) =>
        ({def} ((prime (u_1) <<=
        M) Fixform Ug ([(v_11
            : obj) =>
            ({def} Ded ([(vinev_12
                : that v_11 E prime
                (u_1)) =>
                ({def} Simp1 (vinev_12
                Iff1 v_11 Ui primeax
                (usinev_7 Iff1 u_1
```

```
                Ui Scthm (M))) Mp
                v_11 Ui Simp1 (usinev_7
                Iff1 u_1 Ui Scthm
                (M)) : that v_11
                E M)]) : that
             (v_11 E prime (u_1)) ->
             v_11 E M)]) Conj
          (Isset (prime (u_1)) Fixform
          Separation3 (Refleq (prime
          (u_1)))) Conj Misset) Iff2
          prime (u_1) Ui Scthm
          (M) : that prime (u_1) E Sc
          (M))]) : that (u_1
       E Sc (M)) -> prime (u_1) E Sc
       (M))]


   line12 : [(u_1 : obj) => (---
       : that (u_1 E Sc (M)) ->
       prime (u_1) E Sc (M))]


   {move 3}

   >>> close


{move 3}

>>> define line13 : Ug line12


line13 : Ug ([(u_3 : obj) =>
    ({def} Ded ([(usinev_4 : that
       u_3 E Sc (M)) =>
       ({def} ((prime (u_3) <<=
       M) Fixform Ug ([(v_8 : obj) =>
          ({def} Ded ([(vinev_9
```

```
                    : that v_8 E prime (u_3)) =>
                    ({def} Simp1 (vinev_9
                    Iff1 v_8 Ui primeax
                    (usinev_4 Iff1 u_3
                    Ui Scthm (M))) Mp
                    v_8 Ui Simp1 (usinev_4
                    Iff1 u_3 Ui Scthm (M)) : that
                    v_8 E M)]) : that
                 (v_8 E prime (u_3)) ->
                 v_8 E M)]) Conj (Isset
              (prime (u_3)) Fixform
              Separation3 (Refleq (prime
              (u_3)))) Conj Misset) Iff2
              prime (u_3) Ui Scthm (M) : that
              prime (u_3) E Sc (M))]) : that
           (u_3 E Sc (M)) -> prime (u_3) E Sc
           (M))])


      line13 : that Forall ([(x'_16
           : obj) =>
           ({def} (x'_16 E Sc (M)) ->
           prime (x'_16) E Sc (M) : prop)])


      {move 2}
end Lestrade execution
```

Here is the third statement needed to verify that $\mathcal{P}(M)$ is a Θ-chain.

```
begin Lestrade execution

      >>> open


         {move 4}
```

```
>>> declare u obj


u : obj


{move 4}

>>> open


   {move 5}

   >>> declare v obj


   v : obj


   {move 5}

   >>> open


      {move 6}

      >>> declare hyp that (u <<= \
          Sc M) & v E u


      hyp : that (u <<= Sc (M)) & v E u


      {move 6}

      >>> define line14 hyp : Simp2 \
          hyp Mp Intax u v
```

```
line14 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    ({def} Simp2 (hyp_1) Mp
    u Intax v : that Forall
    ([(x''_2 : obj) =>
       ({def} (x''_2 E u Intersection
       v) == Forall ([(B1_4
          : obj) =>
          ({def} (B1_4
          E u) -> x''_2
          E B1_4 : prop)]) : prop)]))]


line14 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    (--- : that Forall
    ([(x''_2 : obj) =>
       ({def} (x''_2 E u Intersection
       v) == Forall ([(B1_4
          : obj) =>
          ({def} (B1_4
          E u) -> x''_2
          E B1_4 : prop)]) : prop)]))]


{move 5}

>>> open


   {move 7}

   >>> declare w obj


   w : obj
```

```
{move 7}

>>> open


   {move 8}

   >>> declare hyp2 \
       that w E Intersection \
       u v


   hyp2 : that w E u Intersection
    v


   {move 8}

   >>> define line15 \
       hyp2 : Mp (Simp2 \
       hyp, Ui v (Iff1 \
       (hyp2, Ui w line14 \
       hyp)))


   line15 : [(hyp2_1
       : that w E u Intersection
       v) =>
       ({def} Simp2
       (hyp) Mp v Ui
       hyp2_1 Iff1 w Ui
       line14 (hyp) : that
       w E v)]


   line15 : [(hyp2_1
       : that w E u Intersection
```

```
    v) => (--- : that
    w E v)]


{move 7}

>>> define line16 \
    : Mp (Simp2 hyp, Ui \
    v Simp1 (Simp1 hyp))


line16 : Simp2 (hyp) Mp
 v Ui Simp1 (Simp1
 (hyp))


line16 : that v E Sc
 (M)


{move 7}

>>> define line17 \
    : Iff1 (line16, Ui \
    v Scthm M)


line17 : [
    ({def} line16
    Iff1 v Ui Scthm
    (M) : that v <<=
    M)]


line17 : that v <<=
 M
```

```
{move 7}

>>> define line18 \
    hyp2 : Mp (line15 \
    hyp2, Ui w Simp1 \
    line17)


line18 : [(hyp2_1
    : that w E u Intersection
    v) =>
    ({def} line15
    (hyp2_1) Mp
    w Ui Simp1 (line17) : that
    w E M)]


line18 : [(hyp2_1
    : that w E u Intersection
    v) => (--- : that
    w E M)]


{move 7}

>>> close


{move 7}

>>> define line19 w : Ded \
    line18


line19 : [(w_1 : obj) =>
    ({def} Ded ([(hyp2_11
       : that w_1 E u Intersection
       v) =>
```

```
            ({def} Simp2
            (hyp) Mp v Ui
            hyp2_11 Iff1 w_1
            Ui line14 (hyp) Mp
            w_1 Ui Simp1 (Simp2
            (hyp) Mp v Ui
            Simp1 (Simp1
            (hyp)) Iff1
            v Ui Scthm (M)) : that
            w_1 E M)]) : that
        (w_1 E u Intersection
        v) -> w_1 E M)]


   line19 : [(w_1 : obj) =>
        (--- : that (w_1
        E u Intersection
        v) -> w_1 E M)]


   {move 6}

   >>> close


{move 6}

>>> define line20 hyp : Ug \
    line19


line20 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    ({def} Ug ([(w_3
       : obj) =>
       ({def} Ded ([(hyp2_4
          : that w_3 E u Intersection
          v) =>
```

```
            ({def} Simp2
            (hyp_1) Mp v Ui
            hyp2_4 Iff1 w_3
            Ui line14 (hyp_1) Mp
            w_3 Ui Simp1 (Simp2
            (hyp_1) Mp v Ui
            Simp1 (Simp1
            (hyp_1)) Iff1
            v Ui Scthm (M)) : that
            w_3 E M)]) : that
         (w_3 E u Intersection
         v) -> w_3 E M)]) : that
      Forall ([(x'_14 : obj) =>
         ({def} (x'_14 E u Intersection
         v) -> x'_14 E M : prop)]))]


line20 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    (--- : that Forall
    ([(x'_14 : obj) =>
       ({def} (x'_14 E u Intersection
       v) -> x'_14 E M : prop)]))]


{move 5}

>>> define line21 : Fixform \
    (Isset (Intersection \
    u v), Separation3 (Refleq \
    (Intersection u v)))


line21 : [
    ({def} Isset (u Intersection
    v) Fixform Separation3
    (Refleq (u Intersection
    v)) : that Isset (u Intersection
```

```
    v))]


line21 : that Isset (u Intersection
 v)


{move 5}

>>> define line22 hyp : Fixform \
    ((Intersection u v) <<= \
    M, Conj (line20 hyp, Conj \
    (line21, Misset)))


line22 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    ({def} ((u Intersection
    v) <<= M) Fixform
    line20 (hyp_1) Conj
    line21 Conj Misset : that
    (u Intersection v) <<=
    M)]


line22 : [(hyp_1 : that
    (u <<= Sc (M)) & v E u) =>
    (--- : that (u Intersection
    v) <<= M)]


{move 5}

>>> define line23 hyp : Iff2 \
    (line22 hyp, Ui (Intersection \
    u v, Scthm M))
```

```
   line23 : [(hyp_1 : that
       (u <<= Sc (M)) & v E u) =>
       ({def} line22 (hyp_1) Iff2
       (u Intersection v) Ui
       Scthm (M) : that (u Intersection
       v) E Sc (M))]


   line23 : [(hyp_1 : that
       (u <<= Sc (M)) & v E u) =>
       (--- : that (u Intersection
       v) E Sc (M))]


   {move 5}

   >>> close


{move 5}

>>> define line24 v : Ded \
    line23


line24 : [(v_1 : obj) =>
    ({def} Ded ([(hyp_9
       : that (u <<= Sc (M)) & v_1
       E u) =>
       ({def} (((u Intersection
       v_1) <<= M) Fixform
       Ug ([(w_13 : obj) =>
          ({def} Ded ([(hyp2_14
             : that w_13 E u Intersection
             v_1) =>
             ({def} Simp2
             (hyp_9) Mp v_1
             Ui hyp2_14 Iff1
```

```
                w_13 Ui Simp2
                (hyp_9) Mp u Intax
                v_1 Mp w_13 Ui
                Simp1 (Simp2
                (hyp_9) Mp v_1
                Ui Simp1 (Simp1
                (hyp_9)) Iff1
                v_1 Ui Scthm (M)) : that
                w_13 E M)]) : that
             (w_13 E u Intersection
             v_1) -> w_13 E M)]) Conj
          (Isset (u Intersection
          v_1) Fixform Separation3
          (Refleq (u Intersection
          v_1))) Conj Misset) Iff2
          (u Intersection v_1) Ui
          Scthm (M) : that (u Intersection
          v_1) E Sc (M))]) : that
       ((u <<= Sc (M)) & v_1
       E u) -> (u Intersection
       v_1) E Sc (M))]


   line24 : [(v_1 : obj) =>
       (--- : that ((u <<=
       Sc (M)) & v_1 E u) ->
       (u Intersection v_1) E Sc
       (M))]


   {move 4}

   >>> close


{move 4}

>>> define line25 u : Ug line24
```

```
line25 : [(u_1 : obj) =>
    ({def} Ug ([(v_3 : obj) =>
       ({def} Ded ([(hyp_4
          : that (u_1 <<= Sc
          (M)) & v_3 E u_1) =>
          ({def} (((u_1 Intersection
          v_3) <<= M) Fixform
          Ug ([(w_8 : obj) =>
             ({def} Ded ([(hyp2_9
                : that w_8 E u_1
                Intersection v_3) =>
                ({def} Simp2
                (hyp_4) Mp v_3
                Ui hyp2_9 Iff1
                w_8 Ui Simp2 (hyp_4) Mp
                u_1 Intax v_3
                Mp w_8 Ui Simp1
                (Simp2 (hyp_4) Mp
                v_3 Ui Simp1 (Simp1
                (hyp_4)) Iff1
                v_3 Ui Scthm (M)) : that
                w_8 E M)]) : that
             (w_8 E u_1 Intersection
             v_3) -> w_8 E M)]) Conj
          (Isset (u_1 Intersection
          v_3) Fixform Separation3
          (Refleq (u_1 Intersection
          v_3))) Conj Misset) Iff2
          (u_1 Intersection v_3) Ui
          Scthm (M) : that (u_1
          Intersection v_3) E Sc
          (M))]) : that ((u_1
       <<= Sc (M)) & v_3 E u_1) ->
       (u_1 Intersection v_3) E Sc
       (M))]) : that Forall
    ([(x'_20 : obj) =>
```

```
          ({def} ((u_1 <<= Sc
          (M)) & x'_20 E u_1) ->
          (u_1 Intersection x'_20) E Sc
          (M) : prop)]))]


   line25 : [(u_1 : obj) => (---
       : that Forall ([(x'_20
          : obj) =>
          ({def} ((u_1 <<= Sc
          (M)) & x'_20 E u_1) ->
          (u_1 Intersection x'_20) E Sc
          (M) : prop)]))]


   {move 3}

   >>> close


{move 3}

>>> define line26 : Ug line25


line26 : Ug ([(u_3 : obj) =>
    ({def} Ug ([(v_4 : obj) =>
       ({def} Ded ([(hyp_5 : that
          (u_3 <<= Sc (M)) & v_4
          E u_3) =>
          ({def} (((u_3 Intersection
          v_4) <<= M) Fixform Ug
          ([(w_9 : obj) =>
             ({def} Ded ([(hyp2_10
                : that w_9 E u_3
                Intersection v_4) =>
                ({def} Simp2 (hyp_5) Mp
                v_4 Ui hyp2_10 Iff1
```

```
                        w_9 Ui Simp2 (hyp_5) Mp
                        u_3 Intax v_4 Mp
                        w_9 Ui Simp1 (Simp2
                        (hyp_5) Mp v_4
                        Ui Simp1 (Simp1
                        (hyp_5)) Iff1
                        v_4 Ui Scthm (M)) : that
                        w_9 E M)]) : that
                     (w_9 E u_3 Intersection
                     v_4) -> w_9 E M)]) Conj
                  (Isset (u_3 Intersection
                  v_4) Fixform Separation3
                  (Refleq (u_3 Intersection
                  v_4))) Conj Misset) Iff2
                  (u_3 Intersection v_4) Ui
                  Scthm (M) : that (u_3
                  Intersection v_4) E Sc
                  (M))]) : that ((u_3
               <<= Sc (M)) & v_4 E u_3) ->
               (u_3 Intersection v_4) E Sc
               (M))]) : that Forall
            ([(x'_4 : obj) =>
               ({def} ((u_3 <<= Sc (M)) & x'_4
               E u_3) -> (u_3 Intersection
               x'_4) E Sc (M) : prop)]))])


         line26 : that Forall ([(x'_19
             : obj) =>
             ({def} Forall ([(x'_20 : obj) =>
                ({def} ((x'_19 <<= Sc (M)) & x'_20
                E x'_19) -> (x'_19 Intersection
                x'_20) E Sc (M) : prop)]) : prop)])


         {move 2}
end Lestrade execution
```

Here is the fourth and last statement needed to verify that the power set of M is a Θ-chain.

```
begin Lestrade execution

         >>> close


      {move 2}

      >>> define thetascm1 : Fixform (thetachain \
          (Sc M), Conj (line1, Conj (line2, Conj \
          (line13, line26))))


      thetascm1 : [
          ({def} thetachain (Sc (M)) Fixform
          Misset Mp M Ui Inownpowerset Conj
          Misset Mp M Ui Scofsetisset Mp Sc
          (M) Ui Subsetrefl Conj Ug ([(u_11
             : obj) =>
             ({def} Ded ([(usinev_12 : that
                u_11 E Sc (M)) =>
                ({def} ((prime (u_11) <<=
                M) Fixform Ug ([(v_16
                   : obj) =>
                   ({def} Ded ([(vinev_17
                      : that v_16 E prime
                      (u_11)) =>
                      ({def} Simp1 (vinev_17
                      Iff1 v_16 Ui primeax
                      (usinev_12 Iff1 u_11
                      Ui Scthm (M))) Mp
                      v_16 Ui Simp1 (usinev_12
                      Iff1 u_11 Ui Scthm (M)) : that
                      v_16 E M)]) : that
                   (v_16 E prime (u_11)) ->
```

```
         v_16 E M)]) Conj (Isset
      (prime (u_11)) Fixform
      Separation3 (Refleq (prime
      (u_11)))) Conj Misset) Iff2
      prime (u_11) Ui Scthm (M) : that
      prime (u_11) E Sc (M))]) : that
   (u_11 E Sc (M)) -> prime
   (u_11) E Sc (M))]) Conj
Ug ([(u_11 : obj) =>
   ({def} Ug ([(v_12 : obj) =>
      ({def} Ded ([(hyp_13 : that
         (u_11 <<= Sc (M)) & v_12
         E u_11) =>
         ({def} (((u_11 Intersection
         v_12) <<= M) Fixform
         Ug ([(w_17 : obj) =>
            ({def} Ded ([(hyp2_18
               : that w_17 E u_11
               Intersection v_12) =>
               ({def} Simp2 (hyp_13) Mp
               v_12 Ui hyp2_18 Iff1
               w_17 Ui Simp2 (hyp_13) Mp
               u_11 Intax v_12 Mp
               w_17 Ui Simp1 (Simp2
               (hyp_13) Mp v_12
               Ui Simp1 (Simp1
               (hyp_13)) Iff1
               v_12 Ui Scthm (M)) : that
               w_17 E M)]) : that
            (w_17 E u_11 Intersection
            v_12) -> w_17 E M)]) Conj
         (Isset (u_11 Intersection
         v_12) Fixform Separation3
         (Refleq (u_11 Intersection
         v_12))) Conj Misset) Iff2
         (u_11 Intersection v_12) Ui
         Scthm (M) : that (u_11
         Intersection v_12) E Sc
```

```
               (M))]) : that ((u_11
            <<= Sc (M)) & v_12 E u_11) ->
            (u_11 Intersection v_12) E Sc
            (M))]) : that Forall
         ([(x'_12 : obj) =>
            ({def} ((u_11 <<= Sc (M)) & x'_12
            E u_11) -> (u_11 Intersection
            x'_12) E Sc (M) : prop)]))]) : that
      thetachain (Sc (M)))]


   thetascm1 : that thetachain (Sc (M))


   {move 1}

   >>> close


{move 1}

>>> define thetascm2 Misset thelawchooses \
    : thetascm1


thetascm2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
          ({def} x_4 E .S_2 : prop)])) =>
       (--- : that .thelaw_1 (.S_2) E .S_2)]),
    ({let} .prime_1 : [(S_2 : obj) =>
       ({def} prime2 (.thelaw_1, S_2) : obj)]),
    ({let} .thetachain_3 : [(C_13 : obj) =>
       ({def} thetachain1 (.M_1, .thelaw_1, C_13) : prop)]) =>
    ({def} .thetachain_3 (Sc (.M_1)) Fixform
```

```
Misset_1 Mp .M_1 Ui Inownpowerset Conj
Misset_1 Mp .M_1 Ui Scofsetisset Mp
Sc (.M_1) Ui Subsetrefl Conj Ug ([(u_17
   : obj) =>
   ({def} Ded ([(usinev_18 : that
      u_17 E Sc (.M_1)) =>
      ({def} ((.prime_1 (u_17) <<=
      .M_1) Fixform Ug ([(v_22
         : obj) =>
         ({def} Ded ([(vinev_23
            : that v_22 E .prime_1
            (u_17)) =>
            ({def} Simp1 (vinev_23
            Iff1 v_22 Ui Ug ([(u_32
               : obj) =>
               ({def} Dediff ([(hyp1_33
                  : that u_32 E .prime_1
                  (u_17)) =>
                  ({def} Simp1 (hyp1_33
                  Iff1 u_32 Ui u_17
                  Compax Usc (.thelaw_1
                  (u_17))) Conj
                  Negintro ([(hyp3_35
                     : that u_32 = .thelaw_1
                     (u_17)) =>
                     ({def} Subs (Eqsymm
                     (hyp3_35), [(v_37
                        : obj) =>
                        ({def} v_37
                        E Usc (.thelaw_1
                        (u_17)) : prop)], Inusc2
                     (.thelaw_1 (u_17))) Mp
                     Simp2 (hyp1_33
                     Iff1 u_32 Ui u_17
                     Compax Usc (.thelaw_1
                     (u_17))) : that
                     ??)]) : that
                  (u_32 E u_17) & ~ (u_32
```

```
                  = .thelaw_1 (u_17)))], [(hyp2_33
                  : that (u_32 E u_17) & ~ (u_32
                  = .thelaw_1 (u_17))) =>
                  ({def} Simp1 (hyp2_33) Conj
                  Negintro ([(hyp4_36
                     : that u_32 E Usc
                     (.thelaw_1 (u_17))) =>
                     ({def} Inusc1
                     (hyp4_36) Mp
                     Simp2 (hyp2_33) : that
                     ??)]) Iff2
                  u_32 Ui u_17 Compax
                  Usc (.thelaw_1 (u_17)) : that
                  u_32 E u_17 Complement
                  Usc (.thelaw_1 (u_17)))]) : that
               (u_32 E .prime_1 (u_17)) ==
               (u_32 E u_17) & ~ (u_32
               = .thelaw_1 (u_17)))])) Mp
            v_22 Ui Simp1 (usinev_18
            Iff1 u_17 Ui Scthm (.M_1)) : that
            v_22 E .M_1)]) : that
         (v_22 E .prime_1 (u_17)) ->
         v_22 E .M_1)]) Conj (Isset
      (.prime_1 (u_17)) Fixform
      Separation3 (Refleq (.prime_1
      (u_17)))) Conj Misset_1) Iff2
      .prime_1 (u_17) Ui Scthm (.M_1) : that
      .prime_1 (u_17) E Sc (.M_1))]) : that
   (u_17 E Sc (.M_1)) -> .prime_1
   (u_17) E Sc (.M_1))]) Conj
Ug ([(u_17 : obj) =>
   ({def} Ug ([(v_18 : obj) =>
      ({def} Ded ([(hyp_19 : that
         (u_17 <<= Sc (.M_1)) & v_18
         E u_17) =>
         ({def} (((u_17 Intersection
         v_18) <<= .M_1) Fixform
         Ug ([(w_23 : obj) =>
```

```
                ({def} Ded ([(hyp2_24
                   : that w_23 E u_17 Intersection
                   v_18) =>
                   ({def} Simp2 (hyp_19) Mp
                   v_18 Ui hyp2_24 Iff1
                   w_23 Ui Simp2 (hyp_19) Mp
                   u_17 Intax v_18 Mp w_23
                   Ui Simp1 (Simp2 (hyp_19) Mp
                   v_18 Ui Simp1 (Simp1
                   (hyp_19)) Iff1 v_18
                   Ui Scthm (.M_1)) : that
                   w_23 E .M_1)]) : that
                (w_23 E u_17 Intersection
                v_18) -> w_23 E .M_1)]) Conj
             (Isset (u_17 Intersection
             v_18) Fixform Separation3
             (Refleq (u_17 Intersection
             v_18))) Conj Misset_1) Iff2
             (u_17 Intersection v_18) Ui
             Scthm (.M_1) : that (u_17
             Intersection v_18) E Sc (.M_1))]) : that
          ((u_17 <<= Sc (.M_1)) & v_18
          E u_17) -> (u_17 Intersection
          v_18) E Sc (.M_1))]) : that
       Forall ([(x'_18 : obj) =>
          ({def} ((u_17 <<= Sc (.M_1)) & x'_18
          E u_17) -> (u_17 Intersection
          x'_18) E Sc (.M_1) : prop)]))]) : that
    .thetachain_3 (Sc (.M_1)))]


thetascm2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
          ({def} x_4 E .S_2 : prop)])) =>
```

```
      (--- : that .thelaw_1 (.S_2) E .S_2)]),
    ({let} .prime_1 : [(S_2 : obj) =>
      ({def} prime2 (.thelaw_1, S_2) : obj)]),
    ({let} .thetachain_3 : [(C_13 : obj) =>
      ({def} thetachain1 (.M_1, .thelaw_1, C_13) : prop)]) =>
    (--- : that .thetachain_3 (Sc (.M_1)))]


{move 0}

>>> open


   {move 2}

   >>> define thetascm : thetascm2 Misset, thelawchooses


   thetascm : [
       ({def} Misset thetascm2 thelawchooses
       : that .thetachain_1 (Sc (M)))]


   thetascm : that .thetachain_1 (Sc
    (M))


(* quit *)

   {move 1}
end Lestrade execution
```

Here we have proved that $\mathcal{P}(M)$ is a Θ-chain.

Notice that we take this theorem down to move 0 then bring it back up and define a new move 1 theorem with the same content in terms of the move 0 concept. This prevents future references to this theorem from expanding to the very large term appearing above.

```
begin Lestrade execution

      >>> clearcurrent

{move 2}

      >>> define Thetachain : Set ((Sc \
          (Sc M)), thetachain)


      Thetachain : Sc (Sc (M)) Set thetachain


      Thetachain : obj


      {move 1}

      >>> open


         {move 3}

         >>> declare C obj


         C : obj


         {move 3}

         >>> open


            {move 4}

            >>> declare hyp1 that thetachain \
                C
```

```
hyp1 : that thetachain (C)


{move 4}

>>> declare hyp2 that C E Thetachain


hyp2 : that C E Thetachain


{move 4}

>>> define line1 hyp1 : Iff2 \
    (Simp1 (Simp2 hyp1), Ui C Scthm \
    Sc M)


line1 : [(hyp1_1 : that thetachain
    (C)) =>
    ({def} Simp1 (Simp2 (hyp1_1)) Iff2
    C Ui Scthm (Sc (M)) : that
    C E Sc (Sc (M)))]


line1 : [(hyp1_1 : that thetachain
    (C)) => (--- : that C E Sc
    (Sc (M)))]


{move 3}

>>> define line2 hyp1 : Fixform \
    (C E Thetachain, Iff2 (Conj \
    (line1 hyp1, hyp1), Ui (C, Separation \
    ((Sc (Sc M)), thetachain))))
```

```
   line2 : [(hyp1_1 : that thetachain
       (C)) =>
       ({def} (C E Thetachain) Fixform
       line1 (hyp1_1) Conj hyp1_1
       Iff2 C Ui Sc (Sc (M)) Separation
       thetachain : that C E Thetachain)]


   line2 : [(hyp1_1 : that thetachain
       (C)) => (--- : that C E Thetachain)]


   {move 3}

   >>> define line3 hyp2 : Simp2 \
       (Iff1 (hyp2, Ui (C, Separation \
       ((Sc (Sc M)), thetachain))))


   line3 : [(hyp2_1 : that C E Thetachain) =>
       ({def} Simp2 (hyp2_1 Iff1
       C Ui Sc (Sc (M)) Separation
       thetachain) : that thetachain
       (C))]


   line3 : [(hyp2_1 : that C E Thetachain) =>
       (--- : that thetachain (C))]


   {move 3}

   >>> close


{move 3}
```

```
>>> define line4 C : Dediff line3, line2


line4 : [(C_1 : obj) =>
    ({def} Dediff ([(hyp2_12
       : that C_1 E Thetachain) =>
       ({def} Simp2 (hyp2_12 Iff1
       C_1 Ui Sc (Sc (M)) Separation
       thetachain) : that thetachain
       (C_1))], [(hyp1_12
       : that thetachain (C_1)) =>
       ({def} (C_1 E Thetachain) Fixform
       Simp1 (Simp2 (hyp1_12)) Iff2
       C_1 Ui Scthm (Sc (M)) Conj
       hyp1_12 Iff2 C_1 Ui Sc (Sc
       (M)) Separation thetachain
       : that C_1 E Thetachain)]) : that
    (C_1 E Thetachain) == thetachain
    (C_1))]


line4 : [(C_1 : obj) => (---
    : that (C_1 E Thetachain) ==
    thetachain (C_1))]


{move 2}

>>> close


{move 2}

>>> define Thetachainax : Ug line4


Thetachainax : Ug ([(C_3 : obj) =>
```

```
        ({def} Dediff ([(hyp2_4 : that
           C_3 E Thetachain) =>
           ({def} Simp2 (hyp2_4 Iff1 C_3
           Ui Sc (Sc (M)) Separation
           thetachain) : that thetachain
           (C_3))], [(hyp1_4 : that
           thetachain (C_3)) =>
           ({def} (C_3 E Thetachain) Fixform
           Simp1 (Simp2 (hyp1_4)) Iff2
           C_3 Ui Scthm (Sc (M)) Conj
           hyp1_4 Iff2 C_3 Ui Sc (Sc (M)) Separation
           thetachain : that C_3 E Thetachain)]) : that
        (C_3 E Thetachain) == thetachain
        (C_3))])


    Thetachainax : that Forall ([(x'_14
        : obj) =>
        ({def} (x'_14 E Thetachain) ==
        thetachain (x'_14) : prop)])


    {move 1}
end Lestrade execution
```

We prove that the collection of all Θ-chains is a set, and in particular a subset of $\mathcal{P}^2(M)$.

We now define the Θ-chain which implements the desired well-ordering (though we have to verify subsequently that that is what it is).

```
begin Lestrade execution

    >>> define Mbold1 : Intersection Thetachain \
        Sc M


    Mbold1 : [
```

```
       ({def} Thetachain Intersection
       Sc (M) : obj)]


   Mbold1 : obj


   {move 1}

   >>> close


{move 1}

>>> define Mbold2 Misset thelawchooses \
    : Mbold1


Mbold2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
          ({def} x_4 E .S_2 : prop)])) =>
       (--- : that .thelaw_1 (.S_2) E .S_2)]) =>
    ({def} (Sc (Sc (.M_1)) Set [(C_3
       : obj) =>
       ({def} thetachain1 (.M_1, .thelaw_1, C_3) : prop)]) Intersection
    Sc (.M_1) : obj)]


Mbold2 : [(.M_1 : obj), (Misset_1
    : that Isset (.M_1)), (.thelaw_1
    : [(S_2 : obj) => (--- : obj)]), (thelawchooses_1
    : [(.S_2 : obj), (subsetev_2 : that
       .S_2 <<= .M_1), (inev_2 : that
       Exists ([(x_4 : obj) =>
```

```
            ({def} x_4 E .S_2 : prop)])) =>
          (--- : that .thelaw_1 (.S_2) E .S_2)]) =>
       (--- : obj)]


   {move 0}

   >>> open


      {move 2}

      >>> define Mbold : Mbold2 Misset, thelawchooses


      Mbold : [
          ({def} Misset Mbold2 thelawchooses
          : obj)]


      Mbold : obj


      {move 1}
end Lestrade execution
```

We now have the tedious task of directly verifying that **M** is a Θ-chain, which Zermelo dismisses as a side remark!

We note that Zermelo's text suggests that we should prove that any intersection of Θ-chains is a Θ-chain, but it appears that the only case of this we need is that **M** itself is a Θ-chain.

```
begin Lestrade execution

      >>> clearcurrent

{move 2}
```

```
>>> declare C obj


C : obj


{move 2}

>>> declare D obj


D : obj


{move 2}

>>> open


   {move 3}

   >>> define Mboldax1 : Intax Thetachain \
       Sc M


   Mboldax1 : [
       ({def} Thetachain Intax Sc (M) : that
       (Sc (M) E Thetachain) ->
       Forall ([(x''_3 : obj) =>
          ({def} (x''_3 E Thetachain
          Intersection Sc (M)) ==
          Forall ([(B1_5 : obj) =>
             ({def} (B1_5 E Thetachain) ->
             x''_3 E B1_5 : prop)]) : prop)]))]


   Mboldax1 : that (Sc (M) E Thetachain) ->
```

```
 Forall ([(x''_3 : obj) =>
    ({def} (x''_3 E Thetachain
    Intersection Sc (M)) == Forall
    ([(B1_5 : obj) =>
       ({def} (B1_5 E Thetachain) ->
       x''_3 E B1_5 : prop)]) : prop)])


{move 2}

>>> define line1 : Ui Sc M Thetachainax


line1 : Sc (M) Ui Thetachainax


line1 : that (Sc (M) E Thetachain) ==
 thetachain (Sc (M))


{move 2}

>>> define line2 : Iff2 thetascm \
    line1

general failure of objectsort line 2989

(paused, type something to continue) >
objectof a non object line 2749: {function error}

(paused, type something to continue) >
objectof a non object line 2749: {function error}

(paused, type something to continue) >
general failure of functionsort line 3030

(paused, type something to continue) >
general failure of functionsort line 3030
```

```
(paused, type something to continue) >
general failure of functionsort line 3030

(paused, type something to continue) >
Failure in comparing {function error} to prop line 3073

(paused, type something to continue) >
Object type error in thetascm Iff2 line1

(paused, type something to continue) >
Typefix failure with application term

implicitarglist failure line 1905

(paused, type something to continue) >
Parse or typefix error inIff2()

(paused, type something to continue) >

         >>> close


      {move 2}

      >>> define Mboldax : Fixform (Forall \
          [C => (C E Mbold) == Forall [D => \
                (D E Thetachain) -> C E D]], Mp \
          line2 Mboldax1)

Fixform (Forall [C => (C E Mbold) == Forall [D => (D E Thetachain) -> C E D]],

(paused, type something to continue) >
end Lestrade execution
```

Above, we develop the most convenient definition of the extension of **M**. I am not sure it is actually used much (though it is used at least once). I believe that development of `Separation4` caused separation axioms for

particular constructions to be used much less.

```
begin Lestrade execution

      >>> clearcurrent

{move 2}

      >>> open


         {move 3}

         >>> declare F obj


         F : obj


         {move 3}

         >>> open


            {move 4}

            >>> declare ftheta that F E Thetachain


            ftheta : that F E Thetachain


            {move 4}

            >>> define line1 ftheta : Iff1 \
                (ftheta, Ui F Thetachainax)
```

```
   line1 : [(ftheta_1 : that F E Thetachain) =>
       ({def} ftheta_1 Iff1 F Ui
       Thetachainax : that thetachain
       (F))]


   line1 : [(ftheta_1 : that F E Thetachain) =>
       (--- : that thetachain (F))]


   {move 3}

   >>> define line2 ftheta : Simp1 \
       line1 ftheta


   line2 : [(ftheta_1 : that F E Thetachain) =>
       ({def} Simp1 (line1 (ftheta_1)) : that
       M E F)]


   line2 : [(ftheta_1 : that F E Thetachain) =>
       (--- : that M E F)]


   {move 3}

   >>> close


{move 3}

>>> define Linea1 F : Ded line2


Linea1 : [(F_1 : obj) =>
    ({def} Ded ([(ftheta_7 : that
```

```
          F_1 E Thetachain) =>
          ({def} Simp1 (ftheta_7 Iff1
          F_1 Ui Thetachainax) : that
          M E F_1)]) : that (F_1
       E Thetachain) -> M E F_1)]


   Linea1 : [(F_1 : obj) => (---
       : that (F_1 E Thetachain) ->
       M E F_1)]


   {move 2}

   >>> close


{move 2}

>>> define Lineb1 : Ug Linea1


Lineb1 : Ug ([(F_3 : obj) =>
    ({def} Ded ([(ftheta_4 : that
       F_3 E Thetachain) =>
       ({def} Simp1 (ftheta_4 Iff1
       F_3 Ui Thetachainax) : that
       M E F_3)]) : that (F_3 E Thetachain) ->
    M E F_3)])


Lineb1 : that Forall ([(x'_9 : obj) =>
    ({def} (x'_9 E Thetachain) ->
    M E x'_9 : prop)])


{move 1}
```

```
      >>> define Line1 : Iff2 (Lineb1, Ui \
          M Mboldax)

Iff2 (Lineb1, Ui M Mboldax) is not well-formed

(paused, type something to continue) >

      >>> clearcurrent

{move 2}
end Lestrade execution
```

Here is the first component of the proof that $\mathbf{M}$ is a Θ-chain.

```
begin Lestrade execution

      >>> open


         {move 3}

         >>> open


            {move 4}

            >>> declare A obj


            A : obj


            {move 4}

            >>> open
```

```
                {move 5}

                >>> declare ainev that A E Mbold


                ainev : that A E Mbold


                {move 5}

                >>> define line1 ainev : Mp \
                    (Iff2 (thetascm, Ui Sc \
                    M Thetachainax), Ui Sc M, Iff1 \
                    (ainev, Ui A Mboldax))

[ainev => Mp (Iff2 (thetascm, Ui Sc M Thetachainax), Ui Sc M, Iff1 (ainev, Ui A

(paused, type something to continue) >

                >>> close


            {move 4}

            >>> define line2 A : Ded line1

[A => Ded line1] is not well-formed

(paused, type something to continue) >

            >>> close


         {move 3}

         >>> define Line3 : Ug line2

Ug line2 is not well-formed
```

```
(paused, type something to continue) >

         >>> close


      {move 2}

      >>> define Line4 : Fixform ((Mbold) <<= \
          Sc M, Conj (Line3, Conj (Inhabited \
          Line1, Sc2 M)))

Fixform ((Mbold) <<= Sc M, Conj (Line3, Conj (Inhabited Line1, Sc2 M))) is not

(paused, type something to continue) >

      >>> clearcurrent

{move 2}
end Lestrade execution
```

Here is the second component of the proof that **M** is a Θ-chain.

```
begin Lestrade execution

      >>> open


         {move 3}

         >>> declare F obj


         F : obj


         {move 3}
```

```
>>> open

   {move 4}

   >>> declare finmbold that F E (Mbold)


   finmbold : that F E Mbold


   {move 4}

   >>> open


      {move 5}

      >>> declare G obj


      G : obj


      {move 5}

      >>> open


         {move 6}

         >>> declare gtheta that \
             G E Thetachain


         gtheta : that G E Thetachain
```

```
                    {move 6}

                    >>> define line1 gtheta \
                        : Ui (F, Simp1 Simp2 \
                        Simp2 Iff1 (gtheta, Ui \
                        G Thetachainax))


                    line1 : [(gtheta_1 : that
                        G E Thetachain) =>
                        ({def} F Ui Simp1 (Simp2
                        (Simp2 (gtheta_1 Iff1
                        G Ui Thetachainax))) : that
                        (F E G) -> prime2
                        (thelaw, F) E G)]


                    line1 : [(gtheta_1 : that
                        G E Thetachain) =>
                        (--- : that (F E G) ->
                        prime2 (thelaw, F) E G)]


                    {move 5}

                    >>> define line2 gtheta \
                        : Mp (gtheta, Ui (G, Iff1 \
                        (finmbold, Ui F Mboldax)))

[gtheta => Mp (gtheta, Ui (G, Iff1 (finmbold, Ui F Mboldax)))] is not well-form

(paused, type something to continue) >

                    >>> define line3 gtheta \
                        : Mp line2 gtheta line1 \
                        gtheta
```

```
[gtheta => Mp line2 gtheta line1 gtheta] is not well-formed

(paused, type something to continue) >

                  >>> close


               {move 5}

               >>> define line4 G : Ded line3

[G => Ded line3] is not well-formed

(paused, type something to continue) >

               >>> close


            {move 4}

            >>> define line5 finmbold : Ug \
                line4

[finmbold => Ug line4] is not well-formed

(paused, type something to continue) >

            >>> define line6 finmbold : Iff2 \
                (line5 finmbold, Ui (prime \
                F, Mboldax))

[finmbold => Iff2 (line5 finmbold, Ui (prime F, Mboldax))] is not well-formed

(paused, type something to continue) >

            >>> close
```

```
         {move 3}

         >>> define line7 F : Ded line6

[F => Ded line6] is not well-formed

(paused, type something to continue) >

         >>> close


      {move 2}

      >>> define Linea8 : Ug line7

Ug line7 is not well-formed

(paused, type something to continue) >

      >>> save


      {move 2}

      >>> close


   {move 1}

   >>> define Lineb8 Misset thelawchooses \
       : Linea8

[Misset thelawchooses => Linea8] is not well-formed

(paused, type something to continue) >

   >>> open
```

```
      {move 2}

      >>> define Line8 : Lineb8 Misset, thelawchooses

Lineb8 Misset, thelawchooses is not well-formed

(paused, type something to continue) >
end Lestrade execution
```

Here is the third component of the proof that $\mathbf{M}$ is a Θ-chain. Note the importance of preventing definitional expansion here!

```
begin Lestrade execution

      >>> open


         {move 3}

         >>> declare H obj


         H : obj


         {move 3}

         >>> open


            {move 4}

            >>> declare J obj


            J : obj
```

```
{move 4}

>>> open


   {move 5}

   >>> declare thehyp that (H <<= \
       Mbold) & J E H


   thehyp : that (H <<= Mbold) & J E H


   {move 5}

   >>> open


      {move 6}

      >>> declare K obj


      K : obj


      {move 6}

      >>> open


         {move 7}

         >>> declare ktheta that \
             K E Thetachain
```

```
ktheta : that K E Thetachain


{move 7}

>>> define line1 ktheta \
    : Iff1 (ktheta, Ui \
    K Thetachainax)


line1 : [(ktheta_1
    : that K E Thetachain) =>
    ({def} ktheta_1
    Iff1 K Ui Thetachainax
    : that thetachain
    (K))]


line1 : [(ktheta_1
    : that K E Thetachain) =>
    (--- : that thetachain
    (K))]


{move 6}

>>> define line2 ktheta \
    : Ui J, Ui H, Simp2 \
    Simp2 Simp2 line1 ktheta


line2 : [(ktheta_1
    : that K E Thetachain) =>
    ({def} J Ui H Ui
    Simp2 (Simp2 (Simp2
    (line1 (ktheta_1)))) : that
```

```
    ((H <<= K) & J E H) ->
    (H Intersection
    J) E K)]


line2 : [(ktheta_1
    : that K E Thetachain) =>
    (--- : that ((H <<=
    K) & J E H) ->
    (H Intersection
    J) E K)]


{move 6}

>>> open


   {move 8}

   >>> declare P obj


   P : obj


   {move 8}

   >>> open


      {move 9}

      >>> declare phyp \
          that P E H


      phyp : that P E H
```

```
                        {move 9}

                        >>> define line3 \
                            phyp : Mp (phyp, Ui \
                            P Simp1 Simp1 \
                            thehyp)


                        line3 : [(phyp_1
                            : that P E H) =>
                            ({def} phyp_1
                            Mp P Ui Simp1
                            (Simp1 (thehyp)) : that
                            P E Mbold)]


                        line3 : [(phyp_1
                            : that P E H) =>
                            (--- : that
                            P E Mbold)]


                        {move 8}

                        >>> define line4 \
                            phyp : Mp (ktheta, K Ui \
                            Iff1 line3 phyp, Ui \
                            P Mboldax)

[phyp => Mp (ktheta, K Ui Iff1 line3 phyp, Ui P Mboldax)] is not well-formed

(paused, type something to continue) >

                        >>> close
```

```
                        {move 8}

                        >>> define line5 \
                            P : Ded line4

[P => Ded line4] is not well-formed

(paused, type something to continue) >

                        >>> close


                    {move 7}

                    >>> define test1 ktheta \
                        : Ug line5

[ktheta => Ug line5] is not well-formed

(paused, type something to continue) >

                    >>> define test2 ktheta \
                        : Inhabited Simp2 thehyp


                    test2 : [(ktheta_1
                        : that K E Thetachain) =>
                        ({def} Inhabited
                        (Simp2 (thehyp)) : that
                        Isset (H))]


                    test2 : [(ktheta_1
                        : that K E Thetachain) =>
                        (--- : that Isset
                        (H))]
```

```
                    {move 6}

                    >>> define test3 ktheta \
                        : Inhabited (Mp (Simp2 \
                        thehyp, line5 J))

[ktheta => Inhabited (Mp (Simp2 thehyp, line5 J))] is not well-formed

(paused, type something to continue) >

                    >>> define line6 ktheta \
                        : Fixform (H <<= K, Conj \
                        (test1 ktheta, Conj \
                        (test2 ktheta, test3 \
                        ktheta)))

[ktheta => Fixform (H <<= K, Conj (test1 ktheta, Conj (test2 ktheta, test3 kthe

(paused, type something to continue) >

                    >>> define linea7 ktheta \
                        : Mp (Conj (line6 \
                        ktheta, Simp2 thehyp), line2 \
                        ktheta)

[ktheta => Mp (Conj (line6 ktheta, Simp2 thehyp), line2 ktheta)] is not well-fo

(paused, type something to continue) >

                    >>> close


                  {move 6}

                  >>> define line8 K : Ded \
                      linea7

[K => Ded linea7] is not well-formed
```

```
(paused, type something to continue) >

                >>> close


              {move 5}

              >>> define line9 thehyp : Ug \
                  line8

[thehyp => Ug line8] is not well-formed

(paused, type something to continue) >

              >>> define line10 : Ui (H Intersection \
                  J, Mboldax)

Ui (H Intersection J, Mboldax) is not well-formed

(paused, type something to continue) >

              >>> define line11 thehyp : Iff2 \
                  (line9 thehyp, line10)

[thehyp => Iff2 (line9 thehyp, line10)] is not well-formed

(paused, type something to continue) >

              >>> close


            {move 4}

            >>> define line12 J : Ded line11

[J => Ded line11] is not well-formed
```

```
(paused, type something to continue) >

            >>> close


         {move 3}

         >>> define line13 H : Ug line12

[H => Ug line12] is not well-formed

(paused, type something to continue) >

         >>> close


      {move 2}

      >>> define Linea14 : Ug line13

Ug line13 is not well-formed

(paused, type something to continue) >

      >>> save


      {move 2}

      >>> close


   {move 1}

   >>> define Lineb14 Misset thelawchooses \
       : Linea14

[Misset thelawchooses => Linea14] is not well-formed
```

```
(paused, type something to continue) >

   >>> open


      {move 2}

      >>> define Line14 : Lineb14 Misset, thelawchooses

Lineb14 Misset, thelawchooses is not well-formed

(paused, type something to continue) >
end Lestrade execution
```

Here is the fourth component of the proof that $\mathbf{M}$ is a Θ-chain.

```
begin Lestrade execution

      >>> define Mboldtheta1 : Fixform (thetachain \
          (Mbold), Conj (Line1, Conj (Line4, Conj \
          (Line8, Line14))))

Fixform (thetachain (Mbold), Conj (Line1, Conj (Line4, Conj (Line8, Line14))))

(paused, type something to continue) >

      >>> close


   {move 1}

   >>> define Mboldtheta2 Misset thelawchooses \
       : Mboldtheta1

[Misset thelawchooses => Mboldtheta1] is not well-formed
```

```
(paused, type something to continue) >

   >>> open


      {move 2}

      >>> define Mboldtheta : Mboldtheta2 \
          Misset thelawchooses

Mboldtheta2 Misset thelawchooses is not well-formed

(paused, type something to continue) >
end Lestrade execution
```

`Mboldtheta` asserts that $\mathbf{M}$ is a Θ-chain.