Fresh development of Zermelo 1908b in Lestrade

Randall Holmes

May 14, 2021

Consider this a proper lab notebook for development of Zermelo's approach to foundations under Lestrade, and also a diary of recovery from COVID-19.

1 Version notes

This subsection will have entries describing the development of the work

Dec 3, 2020: Starting. Setting myself the task.

The idea is to develop Zermelo 1908b, the paper on axiomatics of set theory, by directly reading the text. The first thing I need to decide is how to treat logical notions (do I have a preamble, or can one fold logical primitives into set theory primitives?)

Introduced subset as a primitive rather than implication.

Dec 8, 2020: Continuing. I adopt the working philosophy that I put in just what is in the text, and backfill to add logical principles which might be neeed, when they are actually needed in a proof.

2 Zermelo 1908b in Lestrade, with notes

The text that follows is organized by Zermelo's paragraph numbers.

1. The domain $\mathcal{B}$ of individuals will be represented by the built-in Lestrade type `obj`.

   The relation of equality must be declared. Do we declare inequality or do we declare negation?

   Dec 3 I am experimenting with inequality as a primitive: we will see what reasoning principles we need by following the text.

```
begin Lestrade execution

   >>> declare x obj


   x : obj


   {move 1}

   >>> declare y obj


   y : obj


   {move 1}

   >>> postulate = x y prop


   = : [(x_1 : obj), (y_1 : obj) =>
       (--- : prop)]


   {move 0}

   >>> postulate =/= x y prop
```

```
   =/= : [(x_1 : obj), (y_1 : obj) =>
       (--- : prop)]


   {move 0}
end Lestrade execution
```

2. The primitive relation of membership and the notion of being a set must be declared.

   The primitive function `Isset` witnesses that objects with elements are sets.

```
begin Lestrade execution

   >>> clearcurrent

{move 1}

   >>> declare a obj


   a : obj


   {move 1}

   >>> declare b obj


   b : obj


   {move 1}
```

```
   >>> postulate E a b prop


   E : [(a_1 : obj), (b_1 : obj) =>
       (--- : prop)]


   {move 0}

   >>> postulate set b prop


   set : [(b_1 : obj) => (--- : prop)]


   {move 0}

   >>> declare memberdata that a E b


   memberdata : that a E b


   {move 1}

   >>> postulate Isset memberdata that set \
       b


   Isset : [(.a_1 : obj), (.b_1 : obj), (memberdata_1
       : that .a_1 E .b_1) => (--- : that
       set (.b_1))]


   {move 0}
end Lestrade execution
```

3. The subset relation (implication?)

```
begin Lestrade execution

   >>> declare M obj


   M : obj


   {move 1}

   >>> declare N obj


   N : obj


   {move 1}

   >>> postulate << M N prop


   << : [(M_1 : obj), (N_1 : obj) =>
       (--- : prop)]


   {move 0}

   >>> declare subsetev that M << N


   subsetev : that M << N


   {move 1}
```

```
>>> postulate Subset1 subsetev that set \
    M


Subset1 : [(.M_1 : obj), (.N_1 : obj), (subsetev_1
    : that .M_1 << .N_1) => (--- : that
    set (.M_1))]


{move 0}

>>> postulate Subset2 subsetev that set \
    N


Subset2 : [(.M_1 : obj), (.N_1 : obj), (subsetev_1
    : that .M_1 << .N_1) => (--- : that
    set (.N_1))]


{move 0}

>>> declare x obj


x : obj


{move 1}

>>> declare memberev that x E M


memberev : that x E M


{move 1}
```

```
   >>> postulate Subset3 subsetev memberev \
       that x E N


   Subset3 : [(.M_1 : obj), (.N_1 : obj), (subsetev_1
       : that .M_1 << .N_1), (.x_1 : obj), (memberev_1
       : that .x_1 E .M_1) => (--- : that
       .x_1 E .N_1)]


   {move 0}

   >>> declare subsetev2 [x, memberev => \
          that x E N]


   subsetev2 : [(x_1 : obj), (memberev_1
       : that x_1 E M) => (--- : that x_1
       E N)]


   {move 1}

   >>> postulate Subset4 subsetev2 that M << \
       N


   Subset4 : [(.M_1 : obj), (.N_1 : obj), (subsetev2_1
       : [(x_2 : obj), (memberev_2 : that
          x_2 E .M_1) => (--- : that x_2
          E .N_1)]) => (--- : that .M_1
       << .N_1)]


   {move 0}
end Lestrade execution
```

This is a rather daring approach. `Subset3` is in effect replacing *modus ponens* and `Subset4` is replacing the Deduction Theorem, in the development I am forming in my mind. Of course, the usual forms of these principles (and the usual notion of implication) will prove to be definable, if it all works as I expect.

We need to prove that inclusion is reflexive.

```
begin Lestrade execution

   >>> clearcurrent

{move 1}

   >>> declare M obj


   M : obj


   {move 1}

   >>> open


      {move 2}

      >>> declare x obj


      x : obj


      {move 2}
```

```
   >>> declare memberev that x E M


   memberev : that x E M


   {move 2}

   >>> define reflexinclev x memberev \
       : memberev


   reflexinclev : [(x_1 : obj), (memberev_1
       : that x_1 E M) => (--- : that
       x_1 E M)]


   {move 1}

   >>> close


{move 1}

>>> define Reflexincl M : Subset4 reflexinclev


Reflexincl : [(M_1 : obj) =>
    ({def} Subset4 ([(x_2 : obj), (memberev_2
       : that x_2 E M_1) =>
       ({def} memberev_2 : that x_2 E M_1)]) : that
    M_1 << M_1)]


Reflexincl : [(M_1 : obj) => (---
    : that M_1 << M_1)]
```

```
   {move 0}
end Lestrade execution
```

We need to prove that inclusion is transitive.

```
begin Lestrade execution

   >>> clearcurrent

{move 1}

   >>> declare M obj


   M : obj


   {move 1}

   >>> declare N obj


   N : obj


   {move 1}

   >>> declare R obj


   R : obj


   {move 1}
```

```
>>> declare subsetev1 that M << N

subsetev1 : that M << N

{move 1}

>>> declare subsetev2 that N << R

subsetev2 : that N << R

{move 1}

>>> open

   {move 2}

   >>> declare x obj

   x : obj

   {move 2}

   >>> declare xinmev that x E M

   xinmev : that x E M

   {move 2}
```

```
   >>> define line1 xinmev : Subset3 subsetev1 \
       xinmev


   line1 : [(.x_1 : obj), (xinmev_1
       : that .x_1 E M) => (--- : that
       .x_1 E N)]


   {move 1}

   >>> define line2 xinmev : Subset3 subsetev2 \
       line1 xinmev


   line2 : [(.x_1 : obj), (xinmev_1
       : that .x_1 E M) => (--- : that
       .x_1 E R)]


   {move 1}

   >>> close


{move 1}

>>> Showdec xinmev


xinmev : {function error}


{move ~1 :}

>>> define Transincl subsetev1 subsetev2 \
    : Subset4 line2
```

```
   Transincl : [(.M_1 : obj), (.N_1
       : obj), (.R_1 : obj), (subsetev1_1
       : that .M_1 << .N_1), (subsetev2_1
       : that .N_1 << .R_1) =>
       ({def} Subset4 ([(.x_2 : obj), (xinmev_2
          : that .x_2 E .M_1) =>
          ({def} subsetev2_1 Subset3 subsetev1_1
          Subset3 xinmev_2 : that .x_2 E .R_1)]) : that
       .M_1 << .R_1)]


   Transincl : [(.M_1 : obj), (.N_1
       : obj), (.R_1 : obj), (subsetev1_1
       : that .M_1 << .N_1), (subsetev2_1
       : that .N_1 << .R_1) => (--- : that
       .M_1 << .R_1)]


   {move 0}
end Lestrade execution
```

We will defer consideration of disjointness, unless we make changes in our logical preamble.

4. Definite questions or assertions are simply Lestrade propositions. Definite propositional functions are Lestrade functions from objects to propositions.

   Here is the axiom of extensionality. Note that we have not yet introduced the logic of equality in any obvious way.

```
begin Lestrade execution

   >>> clearcurrent
```

```
{move 1}

   >>> declare M obj


   M : obj


   {move 1}

   >>> declare N obj


   N : obj


   {move 1}

   >>> declare subsetev1 that M << N


   subsetev1 : that M << N


   {move 1}

   >>> declare subsetev2 that N << M


   subsetev2 : that N << M


   {move 1}

   >>> postulate Axiom1 subsetev1 subsetev2 \
       that M = N
```

```
   Axiom1 : [(.M_1 : obj), (.N_1 : obj), (subsetev1_1
       : that .M_1 << .N_1), (subsetev2_1
       : that .N_1 << .M_1) => (--- : that
       .M_1 = .N_1)]


   {move 0}
end Lestrade execution
```

The axiom of elementary sets is made up of parts.

Since we do not yet have any treatment of negation, we say simply that 0 is a subset of every set.

```
begin Lestrade execution

   >>> clearcurrent

{move 1}

   >>> postulate 0 obj


   0 : obj


   {move 0}

   >>> declare M obj


   M : obj


   {move 1}
```

```
>>> postulate Zeroincl M : that 0 << M


Zeroincl : [(M_1 : obj) => (--- : that
    0 << M_1)]


{move 0}

>>> postulate Unit M obj


Unit : [(M_1 : obj) => (--- : obj)]


{move 0}

>>> postulate Unitax1 M that M E Unit \
    M


Unitax1 : [(M_1 : obj) => (--- : that
    M_1 E Unit (M_1))]


{move 0}

>>> declare x obj


x : obj


{move 1}

>>> declare inunitev that x E Unit M
```

```
inunitev : that x E Unit (M)


{move 1}

>>> postulate Unitax2 inunitev that x = M


Unitax2 : [(.M_1 : obj), (.x_1 : obj), (inunitev_1
    : that .x_1 E Unit (.M_1)) => (---
    : that .x_1 = .M_1)]


{move 0}

>>> declare xismev that x = M


xismev : that x = M


{move 1}

>>> postulate Unitax3 xismev that x E Unit \
    M


Unitax3 : [(.M_1 : obj), (.x_1 : obj), (xismev_1
    : that .x_1 = .M_1) => (--- : that
    .x_1 E Unit (.M_1))]


{move 0}

>>> declare xinmev that x E M
```

```
xinmev : that x E M


{move 1}

>>> postulate Unitax4 xinmev that (Unit \
    x) << M


Unitax4 : [(.M_1 : obj), (.x_1 : obj), (xinmev_1
    : that .x_1 E .M_1) => (--- : that
    Unit (.x_1) << .M_1)]


{move 0}

>>> declare y obj


y : obj


{move 1}

>>> postulate Pair x y obj


Pair : [(x_1 : obj), (y_1 : obj) =>
    (--- : obj)]


{move 0}

>>> postulate Pairax1 x y : x E Pair x y


Pairax1 : [(x_1 : obj), (y_1 : obj) =>
    ({def} x_1 E x_1 Pair y_1 : prop)]
```

```
Pairax1 : [(x_1 : obj), (y_1 : obj) =>
    (--- : prop)]


{move 0}

>>> postulate Pairax2 x y : y E Pair x y


Pairax2 : [(x_1 : obj), (y_1 : obj) =>
    ({def} y_1 E x_1 Pair y_1 : prop)]


Pairax2 : [(x_1 : obj), (y_1 : obj) =>
    (--- : prop)]


{move 0}

>>> declare yinmev that y E M


yinmev : that y E M


{move 1}

>>> postulate Pairax3 xinmev yinmev that \
    (Pair x y) << M


Pairax3 : [(.M_1 : obj), (.x_1 : obj), (xinmev_1
    : that .x_1 E .M_1), (.y_1 : obj), (yinmev_1
    : that .y_1 E .M_1) => (--- : that
    (.x_1 Pair .y_1) << .M_1)]
```

```
   {move 0}
end Lestrade execution
```

5. I am not sure how to express the uniqueness results for the elementary sets.