Implementation of Zermelo 1908b in Lestrade

Randall Holmes

January 10, 2022

This document contains a paraphrase of Zermelo's second 1908 paper, introducing a now standard axiomatization of set theory (though not in the exact form it has taken later), along with an implementation of the theory of this paper (or something very close to it) under Lestrade. The intent of the paraphrase is to capture everything important that Zermelo says. We preserve the numbered heading structure of the original paper.

1. Set theory is concerned with a domain **B** of individuals, which we shall call objects, and among which are the sets. If symbols a and b denote the same object, we write $a = b$, and otherwise we write $a \neq b$. We say of an object that it exists if it belongs to **B** and of a class of objects **K** that there exists objects of the class **K** if **B** contains at least one element of this class.

```
begin Lestrade execution

   >>> declare a obj

   a : obj

   {move 1}

   >>> declare b obj

   b : obj
```

```
{move 1}

>>> postulate set a prop

set : [(a_1 : obj) => (--- : prop)]

{move 0}

>>> postulate = a b prop

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

{move 0}

>>> declare K [a => prop]

K : [(a_1 : obj) => (--- : prop)]

{move 1}

>>> postulate Exists K prop

Exists : [(K_1 : [(a_2 : obj) =>
       (--- : prop)]) => (--- : prop)]

{move 0}

>>> declare c obj

c : obj

{move 1}

>>> declare witnessev that K c

witnessev : that K (c)
```

```
{move 1}

>>> postulate Proveexists witnessev that \
    Exists K

Proveexists : [(.K_1 : [(a_2 : obj) =>
       (--- : prop)]), (.c_1 : obj), (witnessev_1
    : that .K_1 (.c_1)) => (--- : that
    Exists (.K_1))]

{move 0}

>>> declare A prop

A : prop

{move 1}

>>> declare existsev that Exists K

existsev : that Exists (K)

{move 1}

>>> declare fromwitness [c, witnessev \
       => that A]

fromwitness : [(c_1 : obj), (witnessev_1
    : that K (c_1)) => (--- : that
    A)]

{move 1}

>>> postulate Fromexixts existsev fromwitness \
    that A

Fromexixts : [(.K_1 : [(a_2 : obj) =>
       (--- : prop)]), (.A_1 : prop), (existsev_1
```

```
    : that Exists (.K_1)), (fromwitness_1
    : [(c_2 : obj), (witnessev_2 : that
       .K_1 (c_2)) => (--- : that .A_1)]) =>
    (--- : that .A_1)]

{move 0}

>>> postulate Refleq a that a = a

Refleq : [(a_1 : obj) => (--- : that
    a_1 = a_1)]

{move 0}

>>> declare aisb that a = b

aisb : that a = b

{move 1}

>>> declare P [a => prop]

P : [(a_1 : obj) => (--- : prop)]

{move 1}

>>> declare pev that P a

pev : that P (a)

{move 1}

>>> postulate Subs aisb P, pev that P b

Subs : [(.a_1 : obj), (.b_1 : obj), (aisb_1
    : that .a_1 = .b_1), (P_1 : [(a_2
       : obj) => (--- : prop)]), (pev_1
    : that P_1 (.a_1)) => (--- : that
```

```
    P_1 (.b_1))]

{move 0}

>>> define Subs2 aisb pev : Subs aisb \
    P, pev

Subs2 : [(.a_1 : obj), (.b_1 : obj), (aisb_1
    : that .a_1 = .b_1), (.P_1 : [(a_2
       : obj) => (--- : prop)]), (pev_1
    : that .P_1 (.a_1)) =>
    ({def} Subs (aisb_1, .P_1, pev_1) : that
    .P_1 (.b_1))]

Subs2 : [(.a_1 : obj), (.b_1 : obj), (aisb_1
    : that .a_1 = .b_1), (.P_1 : [(a_2
       : obj) => (--- : prop)]), (pev_1
    : that .P_1 (.a_1)) => (--- : that
    .P_1 (.b_1))]

{move 0}
end Lestrade execution
```

We implement **B** as the Lestrade primitive sort `obj`. We implement classes (whatever exactly they are here) as propositional functions on `obj`. We implement equality and existence of elements of a class as primitives. We defer the declaration of inequality until we can define it.

We will add rules for reasoning as it becomes clear from the text what we need along these lines. Rules for the existential quantifier and equality are now present.

2. There is a primitive relation $a \in b$ between objects: read "a is an element of the set b". If an object b has an element a, we can deduce that b is a set. There is exactly one set which does not have elements, as we will see later.

```
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}

   >>> declare inev that a E b

   inev : that a E b

   {move 1}

   >>> postulate Proveset inev that set b

   Proveset : [(.a_1 : obj), (.b_1 : obj), (inev_1
       : that .a_1 E .b_1) => (--- : that
       set (.b_1))]

   {move 0}
```

```
end Lestrade execution
```

Here we have declared membership and the rule of deduction from an object having an element to its being a set.

3. If every element x of a set M is also an element of the set N, so that from $x \in M$ it always follows that $x \in N$, we say that M is a subset of N and we write $M \subset N$. We always have $M \subseteq M$ and from $M \subseteq N$ and $N \subseteq R$ it always follows that $M \subseteq R$. Two sets M and N are said to be disjoint if they possess no common element, or if no element of M is an element of N.

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> declare x obj

   x : obj

   {move 1}

   >>> declare M obj

   M : obj

   {move 1}

   >>> declare inev that x E M

   inev : that x E M

   {move 1}

   >>> declare N obj
```

```
N : obj

{move 1}

>>> declare Mset that set M

Mset : that set (M)

{move 1}

>>> declare Nset that set N

Nset : that set (N)

{move 1}

>>> postulate C M N prop

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

{move 0}

>>> declare subsetev that M C N

subsetev : that M C N

{move 1}

>>> postulate C1 inev subsetev that x E N

C1 : [(.x_1 : obj), (.M_1 : obj), (inev_1
    : that .x_1 E .M_1), (.N_1 : obj), (subsetev_1
    : that .M_1 C .N_1) => (--- : that
    .x_1 E .N_1)]

{move 0}
```

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

subsetev2 : [(x_1 : obj), (inev_1
    : that x_1 E M) => (--- : that x_1
    E N)]

{move 1}

>>> postulate C2 Mset Nset subsetev2 that \
    M C N

C2 : [(.M_1 : obj), (.N_1 : obj), (Mset_1
    : that set (.M_1)), (Nset_1 : that
    set (.N_1)), (subsetev2_1 : [(x_2
       : obj), (inev_2 : that x_2 E .M_1) =>
       (--- : that x_2 E .N_1)]) =>
    (--- : that .M_1 C .N_1)]

{move 0}

>>> postulate C3 subsetev that set M

C3 : [(.M_1 : obj), (.N_1 : obj), (subsetev_1
    : that .M_1 C .N_1) => (--- : that
    set (.M_1))]

{move 0}

>>> postulate C4 subsetev that set N

C4 : [(.M_1 : obj), (.N_1 : obj), (subsetev_1
    : that .M_1 C .N_1) => (--- : that
    set (.N_1))]

{move 0}
```

```
>>> define Creflexive Mset : C2 Mset Mset \
    [x, inev => inev]

Creflexive : [(.M_1 : obj), (Mset_1
    : that set (.M_1)) =>
    ({def} C2 (Mset_1, Mset_1, [(x_2
       : obj), (inev_2 : that x_2 E .M_1) =>
       ({def} inev_2 : that x_2 E .M_1)]) : that
    .M_1 C .M_1)]

Creflexive : [(.M_1 : obj), (Mset_1
    : that set (.M_1)) => (--- : that
    .M_1 C .M_1)]

{move 0}

>>> declare R obj

R : obj

{move 1}

>>> declare Rset that set R

Rset : that set (R)

{move 1}

>>> declare Mnsubset that M C N

Mnsubset : that M C N

{move 1}

>>> declare Nrsubset that N C R

Nrsubset : that N C R
```

```
{move 1}

>>> open

   {move 2}

   >>> declare z obj

   z : obj

   {move 2}

   >>> declare zinev that z E M

   zinev : that z E M

   {move 2}

   >>> define line1 zinev : C1 zinev Mnsubset

   line1 : [(.z_1 : obj), (zinev_1
       : that .z_1 E M) => (--- : that
       .z_1 E N)]

   {move 1}

   >>> define line2 zinev : C1 (line1 \
       zinev, Nrsubset)

   line2 : [(.z_1 : obj), (zinev_1
       : that .z_1 E M) => (--- : that
       .z_1 E R)]

   {move 1}

   >>> close

{move 1}
```

```
   >>> define Ctransitive Mnsubset Nrsubset \
       : C2 C3 Mnsubset C4 Nrsubset line2

   Ctransitive : [(.M_1 : obj), (.N_1
       : obj), (.R_1 : obj), (Mnsubset_1
       : that .M_1 C .N_1), (Nrsubset_1
       : that .N_1 C .R_1) =>
       ({def} C2 (C3 (Mnsubset_1), C4
       (Nrsubset_1), [(.z_2 : obj), (zinev_2
          : that .z_2 E .M_1) =>
          ({def} (zinev_2 C1 Mnsubset_1) C1
          Nrsubset_1 : that .z_2 E .R_1)]) : that
       .M_1 C .R_1)]

   Ctransitive : [(.M_1 : obj), (.N_1
       : obj), (.R_1 : obj), (Mnsubset_1
       : that .M_1 C .N_1), (Nrsubset_1
       : that .N_1 C .R_1) => (--- : that
       .M_1 C .R_1)]

   {move 0}
end Lestrade execution
```

We are using `C` for the subset relation. Note that we have actually introduced `C` as a primitive and given it logical rules analogous to those we have used for implication elsewhere: $M \subseteq N$ is taken to correlate with existence of a proof that M is a set, a proof that N is a set, and a method for getting from evidence for $x \in M$ to evidence for $x \in N$ for any x. The text is not unfriendly to such an interpretation.

We do not for the moment have the tools to express disjointness, because we do not yet have a natural way to discuss negation. We are expecting to get this shortly.

Notice that we have proved the stated theorems of reflexivity and transitivity of the subset relation: the functions `Creflexive` and `Ctransitive` embody these theorems.

4. A question or assertion is said to be definite if the fundamental relations of the domain in combination with axioms and logical laws determine whether it is true or not. A propositional function $P(x)$ is definite if each of its values for particular objects in the class R over which it ranges is thus definite. It is interesting to note that Zermelo seems to consider $\in$ as definite a priori, but wants to justify definiteness of $=$ by an appeal to the equivalence of $a = b$ and $a \in \{b\}$ below.

   Our first axioms:

   axiom I (extensionality): For any sets M, N, if $M \subseteq N$ and $N \subseteq M$ then $M = N$. A set is determined by its elements.

   list notation for sets: By $\{a_1, a_2, \ldots, a_n\}$ we mean the set with just the listed a_i's as elements.

   axiom II (elementary sets): There is a (fictitious, sic) set 0 with no elements. For each object, there is a set $\{a\}$ which has only a as an element. For any objects a, b there is a set $\{a, b\}$ which has a and b as elements, and no element x distinct from a and b.

   About definiteness, we do not have much to say. Any proposition we write in Lestrade notation we suppose to be definite in a suitable sense, and so our propositional functions (classes) are also definite.

   We do not at the moment have an implementation of full list notation.

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> declare a obj

   a : obj

   {move 1}

   >>> declare b obj

   b : obj
```

```
{move 1}

>>> declare aincb that a C b

aincb : that a C b

{move 1}

>>> declare binca that b C a

binca : that b C a

{move 1}

>>> postulate Ext aincb binca that a = b

Ext : [(.a_1 : obj), (.b_1 : obj), (aincb_1
    : that .a_1 C .b_1), (binca_1 : that
    .b_1 C .a_1) => (--- : that .a_1
    = .b_1)]

{move 0}

>>> postulate 0 obj

0 : obj

{move 0}

>>> postulate Empty1 that set 0

Empty1 : that set (0)

{move 0}

>>> postulate Empty2 a that 0 C a
```

```
Empty2 : [(a_1 : obj) => (--- : that
    0 C a_1)]

{move 0}

>>> postulate Pair a b obj

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

{move 0}

>>> postulate Pair2 a b that a E Pair \
    a b

Pair2 : [(a_1 : obj), (b_1 : obj) =>
    (--- : that a_1 E a_1 Pair b_1)]

{move 0}

>>> postulate Pair3 a b that b E Pair \
    a b

Pair3 : [(a_1 : obj), (b_1 : obj) =>
    (--- : that b_1 E a_1 Pair b_1)]

{move 0}

>>> declare c obj

c : obj

{move 1}

>>> declare ainc that a E c

ainc : that a E c
```

```
{move 1}

>>> declare binc that b E c

binc : that b E c

{move 1}

>>> postulate Pair4 ainc binc that (Pair \
    a b) C c

Pair4 : [(.a_1 : obj), (.b_1 : obj), (.c_1
    : obj), (ainc_1 : that .a_1 E .c_1), (binc_1
    : that .b_1 E .c_1) => (--- : that
    (.a_1 Pair .b_1) C .c_1)]

{move 0}

>>> define Unit a : Pair a a

Unit : [(a_1 : obj) =>
    ({def} a_1 Pair a_1 : obj)]

Unit : [(a_1 : obj) => (--- : obj)]

{move 0}
end Lestrade execution
```

We provide code for the indicated axioms. So far it is unclear how we are going to talk about negation. We follow modern treatments in regarding the singleton set as a special case of the pair.

Some things that Zermelo says here will be proved later. It is worth noting that we take $0 \subseteq M$ as axiomatic, where he regards it as provable.

5. According to Axiom I the elementary sets $\{a\}$ and $\{a, b\}$ are always uniquely determined and there is only a single empty set. The question as to whether $a = b$ is definite because it is equivalent to $a \in \{b\}$ (this

is proved later, because we require Axiom III for our formal proof of this).

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> open

      {move 2}

      >>> postulate empt obj

      empt : obj

      {move 1}

      >>> postulate empt1 that set empt

      empt1 : that set (empt)

      {move 1}

      >>> declare d obj

      d : obj

      {move 2}

      >>> postulate empt2 d that empt C d

      empt2 : [(d_1 : obj) => (--- : that
          empt C d_1)]

      {move 1}
```

```
   >>> define ev1 : Empty2 empt

   ev1 : that 0 C empt

   {move 1}

   >>> define ev2 : empt2 0

   ev2 : that empt C 0

   {move 1}

   >>> define ev3 : Ext ev2 ev1

   ev3 : that empt = 0

   {move 1}

   >>> close

{move 1}

>>> define Emptyunique empt empt1 empt2 \
    : ev3

Emptyunique : [(empt_1 : obj), (empt1_1
    : that set (empt_1)), (empt2_1
    : [(d_2 : obj) => (--- : that empt_1
       C d_2)]) =>
    ({def} empt2_1 (0) Ext Empty2 (empt_1) : that
    empt_1 = 0)]

Emptyunique : [(empt_1 : obj), (empt1_1
    : that set (empt_1)), (empt2_1
    : [(d_2 : obj) => (--- : that empt_1
       C d_2)]) => (--- : that empt_1
    = 0)]
```

```
{move 0}

>>> open

   {move 2}

   >>> declare d obj

   d : obj

   {move 2}

   >>> declare e obj

   e : obj

   {move 2}

   >>> postulate pair d e obj

   pair : [(d_1 : obj), (e_1 : obj) =>
       (--- : obj)]

   {move 1}

   >>> postulate pair2 d e that d E pair \
       d e

   pair2 : [(d_1 : obj), (e_1 : obj) =>
       (--- : that d_1 E d_1 pair e_1)]

   {move 1}

   >>> postulate pair3 d e that e E pair \
       d e

   pair3 : [(d_1 : obj), (e_1 : obj) =>
       (--- : that e_1 E d_1 pair e_1)]
```

```
{move 1}

>>> declare f obj

f : obj

{move 2}

>>> declare dinf that d E f

dinf : that d E f

{move 2}

>>> declare einf that e E f

einf : that e E f

{move 2}

>>> postulate pair4 dinf einf that \
    (pair d e) C f

pair4 : [(.d_1 : obj), (.e_1 : obj), (.f_1
    : obj), (dinf_1 : that .d_1 E .f_1), (einf_1
    : that .e_1 E .f_1) => (--- : that
    (.d_1 pair .e_1) C .f_1)]

{move 1}

>>> define pair5 d e : Pair4 (pair2 \
    d e, pair3 d e)

pair5 : [(d_1 : obj), (e_1 : obj) =>
    (--- : that (d_1 Pair e_1) C d_1
    pair e_1)]
```

```
   {move 1}

   >>> define pair6 d e : pair4 (Pair2 \
       d e, Pair3 d e)

   pair6 : [(d_1 : obj), (e_1 : obj) =>
       (--- : that (d_1 pair e_1) C d_1
       Pair e_1)]

   {move 1}

   >>> close

{move 1}

>>> declare a obj

a : obj

{move 1}

>>> declare b obj

b : obj

{move 1}

>>> define Pairunique pair, pair2, pair3, pair4, a b : Ext \
    (pair6 a b, pair5 a b)

Pairunique : [(pair_1 : [(d_2 : obj), (e_2
       : obj) => (--- : obj)]), (pair2_1
    : [(d_2 : obj), (e_2 : obj) =>
       (--- : that d_2 E d_2 pair e_2)]), (pair3_1
    : [(d_2 : obj), (e_2 : obj) =>
       (--- : that e_2 E d_2 pair e_2)]), (pair4_1
    : [(.d_2 : obj), (.e_2 : obj), (.f_2
       : obj), (dinf_2 : that .d_2 E .f_2), (einf_2
```

```
            : that .e_2 E .f_2) => (--- : that
            (.d_2 pair .e_2) C .f_2)]), (a_1
         : obj), (b_1 : obj) =>
         ({def} (a_1 pair4 b_1, a_1 Pair
         b_1, a_1 Pair2 b_1, a_1 Pair3 b_1) Ext
         (a_1 pair2 b_1) Pair4 a_1 pair3 b_1
         : that (a_1 pair b_1) = a_1 Pair
         b_1)]

      Pairunique : [(pair_1 : [(d_2 : obj), (e_2
            : obj) => (--- : obj)]), (pair2_1
         : [(d_2 : obj), (e_2 : obj) =>
            (--- : that d_2 E d_2 pair e_2)]), (pair3_1
         : [(d_2 : obj), (e_2 : obj) =>
            (--- : that e_2 E d_2 pair e_2)]), (pair4_1
         : [(.d_2 : obj), (.e_2 : obj), (.f_2
            : obj), (dinf_2 : that .d_2 E .f_2), (einf_2
            : that .e_2 E .f_2) => (--- : that
            (.d_2 pair .e_2) C .f_2)]), (a_1
         : obj), (b_1 : obj) => (--- : that
         (a_1 pair b_1) = a_1 Pair b_1)]

      {move 0}
   end Lestrade execution
```

Above I demonstrate the uniqueness of the empty set and the pair by showing that if we postulate another operation with the same properties, it produces the same outputs. The indicated proof of definiteness of equality we will prove with the assistance of axiom 3 below.

6. The null set is a subset of every set: $0 \subseteq M$ (this was an axiom for us above). A subset of M that differs from both 0 and M is called a *part* of M. The sets 0 and $\{a\}$ do not have parts. (I won't define this yet, but I am stating it for the record in case I need it.)

7. The axiom of separation.

Axiom III: For any propositional function $P(x)$ and set M, there is a set $\{x \in M : P(x)\}$ which contains exactly the elements of M for which $P(x)$ is true.

By giving us a large amount of freedom in defining sets, Axiom III gives a substitute for the natural but inconsistent unrestricted axiom of comprehension.

It differs from the inconsistent axiom in the following ways:

(a) Sets cannot be independently defined by this axiom, but are always separated as subsets from sets already given.

(b) (a consideration which does not operate for us) The predicate used to separate must be definite. [We do not see a way to formalize this: all predicates that we introduce are definite.]

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> declare M obj

   M : obj

   {move 1}

   >>> declare x obj

   x : obj

   {move 1}

   >>> declare P [x => prop]

   P : [(x_1 : obj) => (--- : prop)]
```

```
{move 1}

>>> postulate Sep M P : obj

Sep : [(M_1 : obj), (P_1 : [(x_2
       : obj) => (--- : prop)]) =>
    (--- : obj)]

{move 0}

>>> postulate Sep1 M P that set (Sep \
    M P)

Sep1 : [(M_1 : obj), (P_1 : [(x_2
       : obj) => (--- : prop)]) =>
    (--- : that set (M_1 Sep P_1))]

{move 0}

>>> declare inev that x E M

inev : that x E M

{move 1}

>>> declare pev that P x

pev : that P (x)

{move 1}

>>> postulate Sep2 P, inev, pev that \
    x E Sep M P

Sep2 : [(.M_1 : obj), (.x_1 : obj), (P_1
    : [(x_2 : obj) => (--- : prop)]), (inev_1
    : that .x_1 E .M_1), (pev_1 : that
    P_1 (.x_1)) => (--- : that .x_1
```

```
    E .M_1 Sep P_1)]

{move 0}

>>> define Sepa2 inev pev : Sep2 (P, inev, pev)

Sepa2 : [(.M_1 : obj), (.x_1 : obj), (.P_1
    : [(x_2 : obj) => (--- : prop)]), (inev_1
    : that .x_1 E .M_1), (pev_1 : that
    .P_1 (.x_1)) =>
    ({def} Sep2 (.P_1, inev_1, pev_1) : that
    .x_1 E .M_1 Sep .P_1)]

Sepa2 : [(.M_1 : obj), (.x_1 : obj), (.P_1
    : [(x_2 : obj) => (--- : prop)]), (inev_1
    : that .x_1 E .M_1), (pev_1 : that
    .P_1 (.x_1)) => (--- : that .x_1
    E .M_1 Sep .P_1)]

{move 0}

>>> declare inev2 that x E Sep M P

inev2 : that x E M Sep P

{move 1}

>>> postulate Sep3 inev2 that x E M

Sep3 : [(.M_1 : obj), (.x_1 : obj), (.P_1
    : [(x_2 : obj) => (--- : prop)]), (inev2_1
    : that .x_1 E .M_1 Sep .P_1) => (---
    : that .x_1 E .M_1)]

{move 0}

>>> declare Mset that set M
```

```
Mset : that set (M)

{move 1}

>>> declare z obj

z : obj

{move 1}

>>> declare zinev that z E Sep M P

zinev : that z E M Sep P

{move 1}

>>> define Sepsubset3 P, Mset : C2 (Sep1 \
    M P, Mset, [z zinev => Sep3 zinev])

Sepsubset3 : [(.M_1 : obj), (P_1
    : [(x_2 : obj) => (--- : prop)]), (Mset_1
    : that set (.M_1)) =>
    ({def} C2 (.M_1 Sep1 P_1, Mset_1, [(z_2
       : obj), (zinev_2 : that z_2 E .M_1
       Sep P_1) =>
       ({def} Sep3 (zinev_2) : that
       z_2 E .M_1)]) : that (.M_1 Sep
    P_1) C .M_1)]

Sepsubset3 : [(.M_1 : obj), (P_1
    : [(x_2 : obj) => (--- : prop)]), (Mset_1
    : that set (.M_1)) => (--- : that
    (.M_1 Sep P_1) C .M_1)]

{move 0}

>>> postulate Sep4 inev2 that P x
```

```
   Sep4 : [(.M_1 : obj), (.x_1 : obj), (.P_1
       : [(x_2 : obj) => (--- : prop)]), (inev2_1
       : that .x_1 E .M_1 Sep .P_1) => (---
       : that .P_1 (.x_1))]

   {move 0}
end Lestrade execution
```

This block of Lestrade code implements Axiom III. We note that our formalization allows us to define $\{x \in y : \phi\}$ when y is not a set, obtaining the empty set. This is why `Sepsubset3` (the theorem that $\{x \in M : \phi\} \subseteq M$) requires the hypothesis that M is a set.

Next, we are going to use Axiom III to prove a theorem deferred from a section above, that $a = b$ is equivalent to $a \in \{b\}$.

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> declare A prop

   A : prop

   {move 1}

   >>> declare aa that A

   aa : that A

   {move 1}

   >>> define Fixform A aa : aa

   Fixform : [(A_1 : prop), (aa_1 : that
       A_1) =>
```

```
    ({def} aa_1 : that A_1)]

Fixform : [(A_1 : prop), (aa_1 : that
    A_1) => (--- : that A_1)]

{move 0}

>>> declare a obj

a : obj

{move 1}

>>> declare b obj

b : obj

{move 1}

>>> open

   {move 2}

   >>> declare ev1 that a E Unit b

   ev1 : that a E Unit (b)

   {move 2}

   >>> declare ev2 that a = b

   ev2 : that a = b

   {move 2}

   >>> declare z obj

   z : obj
```

```
{move 2}

>>> define line1 z : Fixform (z E Unit \
    z, Pair2 z z)

line1 : [(z_1 : obj) => (--- : that
    z_1 E Unit (z_1))]

{move 1}

>>> declare x obj

x : obj

{move 2}

>>> define line2 ev2 : Subs (ev2, [x => \
       a E Unit x], line1 a)

line2 : [(ev2_1 : that a = b) =>
    (--- : that a E Unit (b))]

{move 1}

>>> define line3 : Sep2 ([x => x = b], line1 \
    b, Refleq b)

line3 : that b E Unit (b) Sep [(x_3
    : obj) =>
    ({def} x_3 = b : prop)]

{move 1}

>>> define line4 : Pair4 (line3, line3)

line4 : that (b Pair b) C Unit (b) Sep
 [(x_3 : obj) =>
```

```
        ({def} x_3 = b : prop)]

   {move 1}

   >>> define line5 ev1 : C1 (ev1, line4)

   line5 : [(ev1_1 : that a E Unit (b)) =>
        (--- : that a E Unit (b) Sep
        [(x_3 : obj) =>
           ({def} x_3 = b : prop)])]

   {move 1}

   >>> define line6 ev1 : Sep4 (line5 \
        ev1)

   line6 : [(ev1_1 : that a E Unit (b)) =>
        (--- : that a = b)]

   {move 1}

   >>> close

{move 1}

>>> declare Ev1 that a E Unit b

Ev1 : that a E Unit (b)

{move 1}

>>> declare Ev2 that a = b

Ev2 : that a = b

{move 1}

>>> define Unittoeq Ev1 : line6 Ev1
```

```
   Unittoeq : [(.a_1 : obj), (.b_1 : obj), (Ev1_1
       : that .a_1 E Unit (.b_1)) =>
       ({def} Sep4 (Ev1_1 C1 Sep2 ([(x_5
          : obj) =>
          ({def} x_5 = .b_1 : prop)], (.b_1
       E Unit (.b_1)) Fixform .b_1 Pair2
       .b_1, Refleq (.b_1)) Pair4 Sep2
       ([(x_5 : obj) =>
          ({def} x_5 = .b_1 : prop)], (.b_1
       E Unit (.b_1)) Fixform .b_1 Pair2
       .b_1, Refleq (.b_1))) : that .a_1
       = .b_1)]

   Unittoeq : [(.a_1 : obj), (.b_1 : obj), (Ev1_1
       : that .a_1 E Unit (.b_1)) => (---
       : that .a_1 = .b_1)]

   {move 0}

   >>> define Eqtounit Ev2 : line2 Ev2

   Eqtounit : [(.a_1 : obj), (.b_1 : obj), (Ev2_1
       : that .a_1 = .b_1) =>
       ({def} Subs (Ev2_1, [(x_2 : obj) =>
          ({def} .a_1 E Unit (x_2) : prop)], (.a_1
       E Unit (.a_1)) Fixform .a_1 Pair2
       .a_1) : that .a_1 E Unit (.b_1))]

   Eqtounit : [(.a_1 : obj), (.b_1 : obj), (Ev2_1
       : that .a_1 = .b_1) => (--- : that
       .a_1 E Unit (.b_1))]

   {move 0}
end Lestrade execution
```

The fact that we use Axiom III to prove this indicates that our defini-

tion of the pair (and the singleton set) is slightly weaker than Zermelo's (largely implicit) definition. The apparent weakness has to do with the fact that we have not assumed much in the way of specifically logical notions, and we actually use set theoretical notions introduced by Zermelo to handle logic.

The function `Fixform` is a frequently used utility for forcing Lestrade to display the type of a defined expression in a desired form. It helps to make output readable.

8. If $M_1 \subseteq M$, then M always possesses another subset, $M - M_1$, the complement of M_1 (relative to M) which contains all those elements of M that are *not* elements of M_1. The complement of $M - M_1$ is M_1 again. If $M_1 = M$, its complement is the null set 0; the complement of any part of M is again a part of M.

   Here we have to come to terms with the logical notion of negation. For any sentence ϕ, we can actually define $\neg\phi$ as $\{x \in \{0\} : \phi\} \subseteq 0$. We can then define $M - M_1$ as $\{x \in M : \neg x \in M_1\}$. But perhaps it is simpler to define $M - M_1$ directly as $\{x \in M : \{y \in \{x\} : y \in M_1\} \subseteq 0\}$. The assertion that $M - (M - M_1) = M_1$ could simply be taken as an axiom (effectively our version of double negation).

   At some point we need to work up some version of standard logical rules.

```
begin Lestrade execution

   >>> clearcurrent
{move 1}

   >>> declare x obj

   x : obj

   {move 1}

   >>> declare y obj
```

```
y : obj

{move 1}

>>> declare z obj

z : obj

{move 1}

>>> define Notin x y : Sep (Unit x, [z => \
       z E y]) C 0

Notin : [(x_1 : obj), (y_1 : obj) =>
    ({def} (Unit (x_1) Sep [(z_3
       : obj) =>
       ({def} z_3 E y_1 : prop)]) C 0 : prop)]

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

{move 0}

>>> declare M obj

M : obj

{move 1}

>>> declare M1 obj

M1 : obj

{move 1}

>>> declare subsetev that M1 C M

subsetev : that M1 C M
```

```
{move 1}

>>> define Complement subsetev : M Sep \
    [z => z Notin M1]

Complement : [(.M_1 : obj), (.M1_1
    : obj), (subsetev_1 : that .M1_1
    C .M_1) =>
    ({def} .M_1 Sep [(z_2 : obj) =>
       ({def} z_2 Notin .M1_1 : prop)] : obj)]

Complement : [(.M_1 : obj), (.M1_1
    : obj), (subsetev_1 : that .M1_1
    C .M_1) => (--- : obj)]

{move 0}

>>> define - M M1 : M Sep [z => z Notin \
       M1]

- : [(M_1 : obj), (M1_1 : obj) =>
    ({def} M_1 Sep [(z_2 : obj) =>
       ({def} z_2 Notin M1_1 : prop)] : obj)]

- : [(M_1 : obj), (M1_1 : obj) =>
    (--- : obj)]

{move 0}

>>> define Compincl subsetev : Fixform \
    (Complement subsetev C M, Sepsubset3 \
    ([z => z Notin M1], C4 subsetev))

Compincl : [(.M_1 : obj), (.M1_1
    : obj), (subsetev_1 : that .M1_1
    C .M_1) =>
    ({def} (Complement (subsetev_1) C .M_1) Fixform
```

```
        Sepsubset3 ([(z_3 : obj) =>
           ({def} z_3 Notin .M1_1 : prop)], C4
        (subsetev_1)) : that Complement
        (subsetev_1) C .M_1)]

    Compincl : [(.M_1 : obj), (.M1_1
        : obj), (subsetev_1 : that .M1_1
        C .M_1) => (--- : that Complement
        (subsetev_1) C .M_1)]

    {move 0}

    >>> postulate Complement2 subsetev that \
        M1 = Complement (Compincl subsetev)

    Complement2 : [(.M_1 : obj), (.M1_1
        : obj), (subsetev_1 : that .M1_1
        C .M_1) => (--- : that .M1_1 = Complement
        (Compincl (subsetev_1)))]

    {move 0}
end Lestrade execution
```

I provide the usual relative complement operation with universal applicability. Zermelo's operation (`Complement`) takes on an unfamiliar appearance in Lestrade, since it is natural to present the fact that $M_1 \subseteq M$ as its only explicit argument.

The axiom `Complement2` is a singularly unreadable form of the double complement axiom. By adding the declaration of `Compincl` with its type forced to a sensible form, I make it easier to see what `Complement2` says.

We can define the notions of inequality and disjointness deferred from earlier.

```
begin Lestrade execution
```

```
   >>> clearcurrent
{move 1}

   >>> declare M obj

   M : obj

   {move 1}

   >>> declare N obj

   N : obj

   {move 1}

   >>> define =/= M N : M Notin Unit N

   =/= : [(M_1 : obj), (N_1 : obj) =>
       ({def} M_1 Notin Unit (N_1) : prop)]

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

   {move 0}

   >>> declare Mset that set M

   Mset : that set (M)

   {move 1}

   >>> declare Nset that set N

   Nset : that set (N)

   {move 1}
```

```
   >>> declare x obj

   x : obj

   {move 1}

   >>> define disj Mset Nset : (M Sep [x => \
          x E N]) = 0

   disj : [(.M_1 : obj), (.N_1 : obj), (Mset_1
       : that set (.M_1)), (Nset_1 : that
       set (.N_1)) =>
       ({def} (.M_1 Sep [(x_3 : obj) =>
          ({def} x_3 E .N_1 : prop)]) = 0 : prop)]

   disj : [(.M_1 : obj), (.N_1 : obj), (Mset_1
       : that set (.M_1)), (Nset_1 : that
       set (.N_1)) => (--- : prop)]

   {move 0}
end Lestrade execution
```

Proving symmetry for inequality would be a useful exercise.

The definition of disjointness might better be deferred until after the imminent definition of the intersection operation on sets, but it can be stated here.