Zermelo's Theorem: Towers Topple

Johan G. F. Belinfante
2007 October 27

```
In[1]:= SetDirectory["l:"]; << goedel98.25a; << tools.m

        :Package Title: goedel98.25a         2007 October 25 at 12:00 noon

        It is now:  2007 Oct 27 at 18:20

        Loading Simplification Rules

        TOOLS.M                                      Revised 2007 September 19

        weightlimit = 40
```

summary

One way to prove that the axiom of choice implies Zorn's lemma is to make use of a theorem attributed to Zermelo by Halmos.

```
In[2]:= "Paul R. Halmos, Naive Set Theory, Van Nostrand, Princeton, 1960.  See page 63.";
```

Zermelo's theorem says that a covering operation, that is, a unary operation which is contained in the covering relation **K**, cannot have a domain which is closed under unions of chains. The derivation of Zermelo's "towers topple" theorem itself, presented in this notebook, does not make any use of the axiom of choice. Because the derivation contains a few time-consuming steps, the **simplify** and **cond** flags will be cleared.

```
In[3]:= simplify = False; cond = False;
```

To make the derivation more readable, the following temporary definitions will be used. None of these concepts enter into the statement of the theorem itself; they are only used in the proof.

```
In[4]:= towers[x_] := intersection[invar[x], fix[UCHAINS]]

In[5]:= comparable[x_, y_] := or[subclass[x, y], subclass[y, x]]

In[6]:= bicone[x_] := union[P[x], image[S, set[x]]]
```

The bicone of a set **x** is the class of all elements that are comparable to **x**. The **spine** of a class **x** is the subclass of those members of **x** that are comparable to all the members of **x**.

```
In[7]:= class[u, and[member[u, x], forall[v, implies[member[v, x], comparable[u, v]]]]]

Out[7]= spine[S, x]
```

The idea of the proof is to show that if the domain of a covering operation were closed under unions of chains, then one would obtain a contradiction. First it is shown that the bicone of the value **APPLY[x, u]** of **x** at the top spinal element **u** of

the minimal tower of **x** would be invariant under **x**. Since the class of invariant subsets and the class **fix[UCHAINS]** are both closed under intersections, and since any bicone is closed under unions of chains, the intersection of the bicone of **APPLY[x, u]** with the minimal tower would be a tower. It follows that either there are no towers or else the minimal tower is contained in this bicone. The former alternative is ruled out: if the domain of a covering operation were closed under unions of chains, then **domain[x]** would be a tower. On the other hand, the the top spinal member **u** belongs to the domain of **x**, and therefore **APPLY[x, u]** covers **u**. This implies that **APPLY[x, u]** cannot be a spinal element of the minimal tower and therefore the minimal tower cannot be contained in its bicone. The upshot of this apparent contradiction is Zermelo's theorem: there can be no covering operation whose domain is closed under unions of chains.

the comparable and covering lemma

Lemma. Negative form of an equality substitution rule: If **u = v**, then **APPLY[x,u]** and **APPLY[x,v]** are comparable. (The same holds for any functor, not just for **APPLY**.)

```
In[8]:= and[equal[u, v], not[comparable[APPLY[x, u], APPLY[x, v]]]] // NotNotTest

Out[8]= and[equal[u, v], not[subclass[APPLY[x, u], APPLY[x, v]]],
          not[subclass[APPLY[x, v], APPLY[x, u]]]] == False

In[9]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[10]:= or[and[member[u, x], member[v, x]], not[member[u, t]],
           not[member[v, t]], not[subclass[t, x]]] // NotNotTest

Out[10]= or[and[member[u, x], member[v, x]],
           not[member[u, t]], not[member[v, t]], not[subclass[t, x]]] == True

In[11]:= (% /. {x → x_, t → t_, u → u_, v → v_}) /. Equal → SetDelayed
```

Lemma.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p3],
            implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
            {p1 → and[member[u, t], subclass[t, domain[funpart[x]]]],
             p2 -> subclass[funpart[x], w],
             p3 → member[u, domain[funpart[x]]],
             p4 -> member[pair[u, APPLY[funpart[x], u]], w]}]] // Reverse

Out[12]= or[member[pair[u, APPLY[funpart[x], u]], w], not[member[u, t]],
           not[subclass[t, domain[funpart[x]]]], not[subclass[funpart[x], w]]] == True

In[13]:= or[member[pair[u_, APPLY[funpart[x_], u_]], w_], not[member[u_, t_]],
           not[subclass[t_, domain[funpart[x_]]]], not[subclass[funpart[x_], w_]]] := True
```

Lemma.

```
In[14]:= and[member[u, t], not[member[pair[u, APPLY[funpart[x], u]], w]],
           subclass[t, domain[funpart[x]]], subclass[funpart[x], w]] // NotNotTest

Out[14]= and[member[u, t], not[member[pair[u, APPLY[funpart[x], u]], w]],
           subclass[t, domain[funpart[x]]], subclass[funpart[x], w]] == False

In[15]:= (% /. {t → t_, u → u_, x → x_, w → w_}) /. Equal → SetDelayed
```

Comparable and Covering Lemma.

```
In[16]:= SubstTest[or, and[p1, p5, not[p6]], and[p1, not[p7]],
           and[p1, not[p8]], and[p1, p2, p3, p4, p7, p8, not[p5], not[p6]] , p6, not[p1],
           not[p2], not[p3], not[p4], {p1 → and[subclass[t, domain[funpart[x]]],
              member[u, t], member[v, t], subclass[funpart[x], K]],
            p2 → comparable[u, v], p3 → comparable[u, APPLY[funpart[x], v]],
            p4 → comparable[APPLY[funpart[x], u], v], p5 → equal[u, v],
            p6 → comparable[APPLY[funpart[x], u], APPLY[funpart[x], v]],
            p7 → member[pair[u, APPLY[funpart[x], u]], K],
            p8 → member[pair[v, APPLY[funpart[x], v]], K]}] // Reverse

Out[16]= or[and[not[subclass[u, v]], not[subclass[v, u]]],
           and[not[subclass[u, APPLY[funpart[x], v]]], not[subclass[APPLY[funpart[x], v], u]]],
           and[not[subclass[v, APPLY[funpart[x], u]]], not[subclass[APPLY[funpart[x], u], v]]],
           not[member[u, t]], not[member[v, t]], not[subclass[t, domain[funpart[x]]]],
           not[subclass[funpart[x], K]], subclass[APPLY[funpart[x], u], APPLY[funpart[x], v]],
           subclass[APPLY[funpart[x], v], APPLY[funpart[x], u]] == True

In[17]:= (% /. {t → t_, u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

invariance lemma

Lemma. If **u** is a member of the spine of an invariant subclass **t** contained in the domain of a function **x**, then **u** is comparable to **APPLY[x,v]** for every member **v** of **t**.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p5],
            implies[p1, p6], implies[and[p1, p5, p6], p7], not[implies[p1, p7]],
            {p1 → and[member[u, t], member[v, t], subclass[t, domain[funpart[x]]],
               subclass[t, union[image[S, set[u]], P[u]]], subclass[image[funpart[x], t], t]],
             p5 → member[APPLY[funpart[x], v], t], p6 → member[APPLY[funpart[x], v],
               domain[funpart[x]]], p7 → comparable[u, APPLY[funpart[x], v]]}]] // Reverse

Out[18]= or[not[member[u, t]], not[member[v, t]], not[subclass[t, domain[funpart[x]]]],
           not[subclass[t, union[image[S, set[u]], P[u]]]],
           not[subclass[image[funpart[x], t], t]], subclass[u, APPLY[funpart[x], v]],
           subclass[APPLY[funpart[x], v], u]] == True

In[19]:= (% /. {x → x_, t → t_, u → u_, v → v_}) /. Equal → SetDelayed
```

Invariance Lemma. (This takes 65 seconds.)

```
In[20]:= Map[not, SubstTest[and, implies[and[p0, p1, p4, p7], p9],
            implies[p1, p4], implies[p1, p7] , not[implies[and[p0, p1], p9]],
            {p0 -> comparable[v, APPLY[funpart[x], u]],
             p1 -> and[member[u, spine[S, t]], member[v, t], subclass[funpart[x], K],
               subclass[t, domain[funpart[x]]], invariant[funpart[x], t]],
             p4 → comparable[u, v], p7 → comparable[u, APPLY[funpart[x], v]],
             p9 → comparable[APPLY[funpart[x], u], APPLY[funpart[x], v]]}]] // Reverse

Out[20]= or[and[not[subclass[v, APPLY[funpart[x], u]]], not[subclass[APPLY[funpart[x], u], v]]],
           not[member[u, t]], not[member[v, t]], not[subclass[t, domain[funpart[x]]]],
           not[subclass[t, union[image[S, set[u]], P[u]]]],
           not[subclass[funpart[x], K]], not[subclass[image[funpart[x], t], t]],
           subclass[APPLY[funpart[x], u], APPLY[funpart[x], v]],
           subclass[APPLY[funpart[x], v], APPLY[funpart[x], u]]] == True

In[21]:= (% /. {t → t_, u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

The next step is to eliminate the variable **v**.

```
In[22]:= Map[equal[V, #] &,
          SubstTest[class, v, implies[and[subclass[z, p], subclass[t, q], subclass[r, t],
             member[u, s], member[v, t], comparable[v, w]], comparable[APPLY[funpart[x], v], w]],
            {p → K, q → domain[funpart[x]], r → image[funpart[x], t], s → spine[S, t],
             w → APPLY[funpart[x], u], z → funpart[x]}]]

Out[22]= or[and[
             subclass[image[funpart[x], intersection[t, image[S, set[APPLY[funpart[x], u]]]]],
              union[image[S, set[APPLY[funpart[x], u]]], P[APPLY[funpart[x], u]]],
             subclass[image[funpart[x], intersection[t, P[APPLY[funpart[x], u]]],
              union[image[S, set[APPLY[funpart[x], u]]], P[APPLY[funpart[x], u]]]],
            not[member[u, t]], not[subclass[t, domain[funpart[x]]]],
            not[subclass[t, union[image[S, set[u]], P[u]]]],
            not[subclass[funpart[x], K]], not[subclass[image[funpart[x], t], t]]] == True

In[23]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

The **funpart** wrapper, introduced to help cope with the occurrence of the **APPLY** constructor, is no longer needed now that the variable **v** has been eliminated. One can now remove it.

```
In[24]:= SubstTest[implies, equal[x, funpart[y]], implies[
            and[subclass[x, K], subclass[t, domain[x]], invariant[x, t], member[u, spine[S, t]]],
            invariant[x, intersection[t, bicone[APPLY[x, u]]]], y → x] // Reverse

Out[24]= or[and[subclass[image[x, intersection[t, image[S, set[APPLY[x, u]]]]],
              union[image[S, set[APPLY[x, u]]], P[APPLY[x, u]]],
             subclass[image[x, intersection[t, P[APPLY[x, u]]]],
              union[image[S, set[APPLY[x, u]]], P[APPLY[x, u]]]],
            not[FUNCTION[x]], not[member[u, t]], not[subclass[t, domain[x]]],
            not[subclass[t, union[image[S, set[u]], P[u]]]],
            not[subclass[x, K]], not[subclass[image[x, t], t]]] == True

In[25]:= (% /. {t → t_, u → u_, x → x_}) /. Equal → SetDelayed
```

combining the two lemmas

Lemma.

```
In[26]:= SubstTest[implies, conduct[x, t, t],
           conduct[x, intersection[t, y], t], t → A[towers[x]]] // Reverse

Out[26]= subclass[image[x, intersection[y, A[intersection[fix[UCHAINS], invar[x]]]]],
           A[intersection[fix[UCHAINS], invar[x]]]] == True

In[27]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Since the minimal tower and bicones are closed under unions of chains, so is their intersection.

```
In[28]:= SubstTest[implies, and[equal[Uchains[u], u], equal[Uchains[v], v]],
           equal[Uchains[intersection[u, v]], intersection[u, v]],
           {u → A[towers[x]], v → bicone[y]}] // Reverse

Out[28]= equal[Uchains[
            union[intersection[A[intersection[fix[UCHAINS], invar[x]]], image[S, set[y]]],
             intersection[A[intersection[fix[UCHAINS], invar[x]]], P[y]]]],
           union[intersection[A[intersection[fix[UCHAINS], invar[x]]], image[S, set[y]]],
            intersection[A[intersection[fix[UCHAINS], invar[x]]], P[y]]]] == True

In[29]:= Uchains[
           union[intersection[A[intersection[fix[UCHAINS], invar[x_]]], image[S, set[y_]]],
            intersection[A[intersection[fix[UCHAINS], invar[x_]]], P[y_]]]] :=
          union[intersection[A[intersection[fix[UCHAINS], invar[x]]], image[S, set[y]]],
           intersection[A[intersection[fix[UCHAINS], invar[x]]], P[y]]]
```

A simplification rule for sethood statements:

```
In[30]:= equiv[or[equal[0, intersection[fix[UCHAINS], invar[x]]],
            not[member[intersection[A[intersection[fix[UCHAINS], invar[x]]], y], V]]],
          equal[0, intersection[fix[UCHAINS], invar[x]]]]

Out[30]= True

In[31]:= or[equal[0, intersection[fix[UCHAINS], invar[x_]]],
            not[member[intersection[y_, A[intersection[fix[UCHAINS], invar[x]]]], V]]] :=
          equal[0, intersection[fix[UCHAINS], invar[x]]]
```

Lemma.

```
In[32]:= Map[or[#, empty[towers[x]]] &, SubstTest[implies, member[y, towers[x]],
           subclass[A[towers[x]], y], y → intersection[A[towers[x]], bicone[v]]]] // Reverse

Out[32]= or[equal[0, intersection[fix[UCHAINS], invar[x]]],
           not[subclass[image[x, intersection[A[intersection[fix[UCHAINS], invar[x]]],
              image[S, set[v]]]], union[image[S, set[v]], P[v]]]],
           not[subclass[image[x, intersection[A[intersection[fix[UCHAINS], invar[x]]], P[v]]],
             union[image[S, set[v]], P[v]]], subclass[
            A[intersection[fix[UCHAINS], invar[x]]], union[image[S, set[v]], P[v]]] == True

In[33]:= (% /. {x → x_, v → v_}) /. Equal → SetDelayed
```

Corollary.

```
In[34]:= (SubstTest[implies, and[FUNCTION[x], subclass[x, K], subclass[t, domain[x]],
            invariant[x, t], member[u, spine[S, t]], equal[w, bicone[APPLY[x, u]]], invariant[
            x, intersection[t, w]], {u → U[spine[S, t]]}] // Reverse) /. t → A[towers[x]]

Out[34]= or[not[equal[w, union[
            image[S, set[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]],
            P[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]],
           not[FUNCTION[x]], not[subclass[x, K]],
           not[subclass[A[intersection[fix[UCHAINS], invar[x]]], domain[x]]], subclass[
            image[x, intersection[w, A[intersection[fix[UCHAINS], invar[x]]]]], w]] == True

In[35]:= (% /. {x → x_, w → w_}) /. Equal → SetDelayed
```

Corollary. Combining the main two lemmas takes 37 seconds.

```
In[36]:= (Map[not, SubstTest[and, implies[and[p0, p1, p2, p4], p5], implies[p3, p1],
            implies[p3, p4], implies[and[p0, p5], p6], not[implies[and[p0, p2, p3], p6]],
            {p0 -> equal[w, bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]],
             p1 -> FUNCTION[x], p2 -> subclass[x, K],
             p3 → and[member[x, UNOPS], equal[Uchains[domain[x]], domain[x]]], p4 -> subclass[
               A[towers[x]], domain[x]], p5 -> invariant[x, intersection[w, A[towers[x]]]],
             p6 → or[empty[towers[x]], subclass[A[towers[x]], w]]}]] // Reverse) /.
         w -> bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]

Out[36]= or[equal[0, intersection[fix[UCHAINS], invar[x]]],
           not[equal[domain[x], Uchains[domain[x]]]], not[member[x, UNOPS]],
           not[subclass[x, K]], subclass[A[intersection[fix[UCHAINS], invar[x]]], union[
             image[S, set[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]],
             P[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]] == True

In[37]:= (% /. x → x_) /. Equal → SetDelayed
```

Existence of towers.

```
In[38]:= Map[implies[member[x, V], #] &, SubstTest[implies, member[u, v],
           not[empty[v]], {u → domain[x], v → towers[x]}]] // MapNotNot // Reverse

Out[38]= or[not[equal[0, intersection[fix[UCHAINS], invar[x]]]],
          not[equal[domain[x], Uchains[domain[x]]]],
          not[member[x, V]], not[subclass[range[x], domain[x]]] == True

In[39]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[40]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 → and[member[x, UNOPS], subclass[x, K], equal[Uchains[domain[x]], domain[x]]],
            p2 → not[empty[towers[x]]], p3 → subclass[A[towers[x]],
              bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]]}]] // Reverse

Out[40]= or[not[equal[domain[x], Uchains[domain[x]]]], not[member[x, UNOPS]],
          not[subclass[x, K]], subclass[A[intersection[fix[UCHAINS], invar[x]]], union[
            image[S, set[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]],
            P[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]] == True

In[41]:= (% /. x → x_) /. Equal → SetDelayed
```

Restatement.

```
In[42]:= implies[and[member[x, UNOPS], subclass[x, K], equal[Uchains[domain[x]], domain[x]]],
           subclass[A[towers[x]], bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]]]

Out[42]= True
```

the apparent contradiction

Lemma. The top spinal element **u** of the minimal tower is in the domain of **x**.

```
In[43]:= (Map[not, SubstTest[and, implies[p1, p2],
             implies[and[p1, p2], p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
             {p1 → and[equal[u, U[spine[S, A[towers[x]]]]], equal[domain[x], Uchains[
                  domain[x]]], member[x, UNOPS]], p2 → subclass[A[towers[x]], domain[x]],
              p3 → member[u, A[towers[x]]], p4 → member[u, domain[x]]}]] //
            Reverse) /. u -> U[spine[S, A[towers[x]]]]

Out[43]= or[member[U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]], domain[x]],
          not[equal[domain[x], Uchains[domain[x]]]], not[member[x, UNOPS]]] == True

In[44]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. The element **APPLY[x, u]** is strictly bigger than **u**.

```
In[45]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → and[FUNCTION[x], subclass[x, K], member[u, domain[x]]], p2 ->
            member[pair[u, APPLY[x, u]], K], p3 -> not[subclass[APPLY[x, u], u]]}]] // Reverse

Out[45]= or[not[FUNCTION[x]], not[member[u, domain[x]]],
          not[subclass[x, K]], not[subclass[APPLY[x, u], u]]] == True

In[46]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Lemma. The element **APPLY[x, u]** is strictly bigger than **u**.

```
In[47]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → and[FUNCTION[x], subclass[x, K], member[u, domain[x]]],
            p2 -> member[pair[u, APPLY[x, u]], K], p3 -> subclass[u, APPLY[x, u]]}]] // Reverse

Out[47]= or[not[FUNCTION[x]], not[member[u, domain[x]]],
          not[subclass[x, K]], subclass[u, APPLY[x, u]]] == True

In[48]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Simplification rule.

```
In[49]:= equiv[and[member[u, domain[x]], member[APPLY[x, u], v]], member[APPLY[x, u], v]]

Out[49]= True

In[50]:= and[member[u_, domain[x_]], member[APPLY[x_, u_], v_]] := member[APPLY[x, u], v]
```

Lemma. If **u** is the greatest spinal element of the minimal tower, and if **APPLY[x,u]** were also a spinal element of the minimal tower, then **APPLY[x,u]** would be no greater than **u**.

```
In[51]:= SubstTest[implies, member[r, s], subclass[r, U[s]],
           {r → APPLY[x, u], s → spine[S, A[towers[x]]]}] // MapNotNot // Reverse

Out[51]= or[not[member[APPLY[x, u], A[intersection[fix[UCHAINS], invar[x]]]]],
          not[subclass[A[intersection[fix[UCHAINS], invar[x]]],
            union[image[S, set[APPLY[x, u]]], P[APPLY[x, u]]]]],
          subclass[APPLY[x, u], U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]] == True

In[52]:= (% /. {u → u_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[53]:= (SubstTest[implies,
            and[FUNCTION[x], subclass[t, domain[x]], invariant[x, t], member[u, t]],
            member[APPLY[x, u], t], t → A[towers[x]]] // Reverse) /. u → U[spine[S, A[towers[x]]]]

Out[53]= or[member[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]],
           A[intersection[fix[UCHAINS], invar[x]]]], not[FUNCTION[x]],
          not[subclass[A[intersection[fix[UCHAINS], invar[x]]], domain[x]]]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The value **APPLY[x, u]** is in the minimal tower.

```
In[55]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 → and[equal[domain[x], Uchains[domain[x]]], member[x, UNOPS]],
            p2 → FUNCTION[x], p3 → subclass[A[towers[x]], domain[x]],
            p4 → member[APPLY[x, U[spine[S, A[towers[x]]]]], A[towers[x]]]}]] // Reverse

Out[55]= or[member[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]],
           A[intersection[fix[UCHAINS], invar[x]]]],
          not[equal[domain[x], Uchains[domain[x]]]], not[member[x, UNOPS]]] == True

In[56]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[57]:= Map[not, SubstTest[and, implies[p0, p3], implies[p0, p1],
           implies[and[p1, p2, p3], p4], not[implies[and[p0, p2], p4]],
           {p0 → and[equal[domain[x], Uchains[domain[x]]], member[x, UNOPS]],
            p1 → FUNCTION[x], p2 → subclass[x, K],
            p3 → member[U[spine[S, A[towers[x]]]], domain[x]], p4 → not[subclass[
              APPLY[x, U[spine[S, A[towers[x]]]]], U[spine[S, A[towers[x]]]]]]}]] // Reverse

Out[57]= or[not[equal[domain[x], Uchains[domain[x]]]],
          not[member[x, UNOPS]], not[subclass[x, K]],
          not[subclass[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]],
            U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]] == True

In[58]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[59]:= Map[not, SubstTest[and, implies[and[p0, p2], p5], implies[and[p0], p7],
           implies[and[p7, p5], not[p8]], not[implies[and[p0, p2], not[p8]]], {p0 →
             and[equal[domain[x], Uchains[domain[x]]], member[x, UNOPS]], p2 → subclass[x, K],
            p5 → not[subclass[APPLY[x, U[spine[S, A[towers[x]]]]], U[spine[S, A[towers[x]]]]]],
            p7 → member[APPLY[x, U[spine[S, A[towers[x]]]]], A[towers[x]]], p8 →
             subclass[A[towers[x]], bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]]}]] // Reverse

Out[59]= or[not[equal[domain[x], Uchains[domain[x]]]], not[member[x, UNOPS]],
          not[subclass[x, K]], not[subclass[A[intersection[fix[UCHAINS], invar[x]]], union[
            image[S, set[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]],
            P[APPLY[x, U[spine[S, A[intersection[fix[UCHAINS], invar[x]]]]]]]]] == True

In[60]:= (% /. x → x_) /. Equal → SetDelayed
```

Restatement:

```
In[61]:= implies[and[member[x, UNOPS], subclass[x, K], equal[Uchains[domain[x]], domain[x]]],
          not[subclass[A[towers[x]], bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]]]]

Out[61]= True
```

Zermelo's theorem: towers topple

The statements at the end of the two preceding sections appears to contradict each other. Both statements can however be true provided that their common hypothesis is false. This yields Zermelo's theorem: there can be no covering operation whose domain is closed under unions of chains.

Zermelo's theorem. ("Towers topple".)

```
In[62]:= SubstTest[and, implies[p, q], implies[p, not[q]],
          {p -> and[member[x, UNOPS], subclass[x, K], equal[Uchains[domain[x]], domain[x]]],
           q -> subclass[A[towers[x]], bicone[APPLY[x, U[spine[S, A[towers[x]]]]]]]}]

Out[62]= or[not[equal[domain[x], Uchains[domain[x]]]],
           not[member[x, UNOPS]], not[subclass[x, K]]] == True

In[63]:= or[not[equal[domain[x_], Uchains[domain[x_]]]],
           not[member[x_, UNOPS]], not[subclass[x_, K]]] := True
```

Lemma.

```
In[64]:= fix[composite[inverse[IMAGE[id[cart[V, V]]]],
            inverse[IMAGE[FIRST]], S, UCHAINS, IMAGE[FIRST]]] // Normality

Out[64]= fix[composite[inverse[IMAGE[id[cart[V, V]]]], inverse[IMAGE[FIRST]],
            S, UCHAINS, IMAGE[FIRST]]] == image[inverse[IMAGE[FIRST]], fix[UCHAINS]]

In[65]:= fix[composite[inverse[IMAGE[id[cart[V, V]]]], inverse[IMAGE[FIRST]], S,
            UCHAINS, IMAGE[FIRST]]] := image[inverse[IMAGE[FIRST]], fix[UCHAINS]]
```

Corollary of Zermelo's theorem. (Variable-free formulation of Zermelo's tower theorem.)

```
In[66]:= Map[complement, SubstTest[class, x, or[not[equal[domain[x], Uchains[domain[x]]]],
             not[member[x, u]], not[subclass[x, v]]], {u → UNOPS, v → K}]] // InvertFix

Out[66]= intersection[UNOPS, image[inverse[IMAGE[FIRST]], fix[UCHAINS]], P[K]] == 0

In[67]:= intersection[UNOPS, image[inverse[IMAGE[FIRST]], fix[UCHAINS]], P[K]] := 0
```