derivation of the Schroeder−Bernsteintheorem

Johan G. F. Belinfante
2003 January 14

```
<< goedel52.q98; << tools.m
```

```
:Package Title: goedel52.q98          2003 January 14 at 8:45 a.m.

It is now:  2003 Jan 14 at 15:29

Loading Simplification Rules

TOOLS.M                       Revised 2002 December 27

weightlimit = 40
```

■ summary

This notebook contains an adaptation of Jean Rubin's proof of the Schroeder−Bernsteintheorem. The reference is:

Jean E. Rubin, "Set Theory for the Mathematician," Holden – Day, San Francisco, 1967. see pages 102 – 104.

The proof is rather long, but the **GOEDEL** program found some steps in her proof to be "obvious" while other steps required considerable expansion. The upshot is that the proof obtained here only resembles her proof in broad outlines. The details are quite different. The final result is a formulation of the Schroeder−Bernsteintheorem as an equation connecting the subclass relation **S** and the equipollence relation **Q**.

```
class[pair[x, y], subclass[x, y]]
```

```
S
```

```
class[pair[x, y], exists[z, and[ONEONE[z], equal[domain[z], x], equal[range[z], y]]]]
```

```
Q
```

An important tool in our proof is the range of the relation **iterate[x,y]**, which yields the smallest **x**−invariant class that contains **y**.

■ union of a bijection and an identity with disjoint domain and range

One of the key ideas is to construct new bijections from old ones by forming unions with identities on sets disjoint from the domain and range of the old one.

```
fix[composite[inverse[x], Di]] // InvertFixTest
```

```
fix[composite[inverse[x], Di]] == fix[composite[Di, x]]
```

```
fix[composite[inverse[x_], Di]] := fix[composite[Di, x]]
```

```
SubstTest[implies, and[subclass[u, v], disjoint[y, v]], disjoint[u, y],
{u -> fix[composite[Di, x]], v -> domain[x]}]

or[equal[0, intersection[y, fix[composite[Di, x]]]],
  not[equal[0, intersection[y, domain[x]]]]] == True

or[equal[0, intersection[y_, fix[composite[Di, x_]]]],
  not[equal[0, intersection[y_, domain[x_]]]]] := True
```

Summary:

```
implies[and[FUNCTION[x], disjoint[domain[x], y]], FUNCTION[union[x, id[y]]]]

True
```

The inverse counterpart:

```
fix[composite[Di, inverse[x]]] // InvertFixTest

fix[composite[Di, inverse[x]]] == fix[composite[x, Di]]

fix[composite[Di, inverse[x_]]] := fix[composite[x, Di]]

SubstTest[implies, and[subclass[u, v], disjoint[y, v]], disjoint[u, y],
{u -> fix[composite[x, Di]], v -> range[x]}]

or[equal[0, intersection[y, fix[composite[x, Di]]]],
  not[equal[0, intersection[y, range[x]]]]] == True

or[equal[0, intersection[y_, fix[composite[x_, Di]]]],
  not[equal[0, intersection[y_, range[x_]]]]] := True
```

Summary:

```
implies[and[FUNCTION[inverse[x]], disjoint[range[x], y]],
 FUNCTION[inverse[union[x, id[y]]]]]

True
```

■ Lemma 1: sethood of restrictions

```
SubstTest[implies, and[subclass[u, v], member[v, V]], member[u, V],
    {u -> composite[x, id[y]], v -> x}]

or[member[composite[x, id[y]], V], not[member[x, V]]] == True

or[member[composite[x_, id[y_]], V], not[member[x_, V]]] := True
```

■ Lemma 2

This lemma is needed for the step in the first paragraph of page 104 in Rubin's proof. It is a property of **range[iterate[x,y]]**. The general rule is:

```
union[y, image[x, range[iterate[x, y]]]]

range[iterate[x, y]]
```

What we want here is a minor variant of this, which is only needed temporarily. The complication is that the **GOEDEL** program rewrites inclusions involving complements to eliminate the complements.

```
SubstTest[subclass, w, range[iterate[z, w]], w -> intersection[complement[y], x]]

subclass[x, union[y, range[iterate[z, intersection[x, complement[y]]]]]] == True

subclass[x_, union[y_, range[iterate[z_, intersection[x_, complement[y_]]]]]] := True

SubstTest[intersection, y, union[u, v],
{u → intersection[complement[y], domain[z]],
  v → image[z, range[iterate[z, intersection[complement[y], domain[z]]]]]}]

intersection[y, range[iterate[z, intersection[complement[y], domain[z]]]]] ==
 intersection[y, image[z, range[iterate[z, intersection[complement[y], domain[z]]]]]]

intersection[y_, range[iterate[z_, intersection[complement[y_], domain[z_]]]]] :=
 intersection[y, image[z, range[iterate[z, intersection[complement[y], domain[z]]]]]]

Map[implies[subclass[y, domain[z]], #] &, SubstTest[subclass, dif[x, w], w,
    {x -> y,
    w -> union[image[z, range[iterate[z, intersection[complement[y], domain[z]]]]],
      intersection[complement[range[
         iterate[z, intersection[complement[y], domain[z]]]], domain[z]]]}]] // Reverse

or[not[subclass[y, domain[z]]],
  subclass[y, union[image[z, range[iterate[z, intersection[complement[y], domain[z]]]]],
    intersection[complement[range[iterate[z, intersection[complement[y], domain[z]]]],
     domain[z]]]]] == True

or[not[subclass[y_, domain[z_]]], subclass[y_,
   union[image[z_, range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
    intersection[complement[range[
       iterate[z_, intersection[complement[y_], domain[z_]]]], domain[z_]]]]] := True
```

■ Lemma 3

The inclusion found above can be strengthened to an equation.

```
SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
    {p -> and[subclass[y, domain[z]], subclass[range[z], y]],
        u -> y,
   v -> union[image[z, range[iterate[z, intersection[complement[y], domain[z]]]]],
     intersection[complement[range[
        iterate[z, intersection[complement[y], domain[z]]]], domain[z]]]}] // Reverse

or[equal[y, union[image[z, range[iterate[z, intersection[complement[y], domain[z]]]]],
    intersection[complement[range[iterate[z, intersection[complement[y], domain[z]]]],
     domain[z]]]], not[subclass[y, domain[z]]], not[subclass[range[z], y]]] == True

or[equal[y_,
   union[image[z_, range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
    intersection[complement[
      range[iterate[z_, intersection[complement[y_], domain[z_]]]], domain[z_]]]],
  not[subclass[y_, domain[z_]]], not[subclass[range[z_], y_]]] := True
```

■ temporary definitions

The following abbreviations will be used to help shorten some statements in the sequel.

```
r = range[iterate[z, intersection[complement[y], domain[z]]]];

h = union[composite[z, id[r]], id[dif[domain[z], r]]];
```

■ h is a set

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
       not[implies[p1, p3]],
  {p1 → member[z, V], p2 → member[domain[z], V], p3 → member[dif[domain[z], r], V]}]]

or[member[
   intersection[complement[range[iterate[z, intersection[complement[y], domain[z]]]]],
    domain[z]], V], not[member[z, V]]] == True

or[member[intersection[
    complement[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
    domain[z_]], V], not[member[z_, V]]] := True

implies[member[z, V], member[h, V]] // NotNotTest

or[and[member[composite[z,
     id[range[iterate[z, intersection[complement[y], domain[z]]]]]], V], member[
    intersection[complement[range[iterate[z, intersection[complement[y], domain[z]]]]],
     domain[z]], V]], not[member[z, V]]] == True

or[and[member[
    composite[z_, id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]]],
    V], member[intersection[
     complement[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
     domain[z_]], V]], not[member[z_, V]]] := True
```

The upshot:

```
implies[member[z, V], member[h, V]]

True
```

■ h is a function

This does not require any new rewrite rules:

```
implies[FUNCTION[z], FUNCTION[h]]

True
```

■ h is one−to−one

```
SubstTest[implies, and[FUNCTION[inverse[u]], disjoint[range[u], v]],
 FUNCTION[inverse[union[u, id[v]]]],
    {u -> composite[z, id[r]],
        v -> dif[domain[z], r]}]
```

```
or[
  not[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]]],
     inverse[z]]]], subclass[intersection[domain[z], fix[
     composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
   range[iterate[z, intersection[complement[y], domain[z]]]]] == True
```

```
or[not[FUNCTION[composite[
     id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]], inverse[z_]]]],
  subclass[intersection[domain[z_], fix[composite[z_,
      id[range[iterate[z_, intersection[complement[y_], domain[z_]]]], Di]]],
   range[iterate[z_, intersection[complement[y_], domain[z_]]]]] := True
```

The following steps took quite some effort.

```
SubstTest[FUNCTION, inverse[union[u, id[v]]], {u -> composite[z, id[r]],
        v -> dif[domain[z], r]}]
```

```
and[equal[0, intersection[fix[composite[inverse[z], id[intersection[complement[
         range[iterate[z, intersection[complement[y], domain[z]]]], domain[z]]], Di]],
    range[iterate[z, intersection[complement[y], domain[z]]]]],
  FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]],
    inverse[z]]], subclass[intersection[domain[z], fix[
     composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
   range[iterate[z, intersection[complement[y], domain[z]]]]] ==
 and[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]],
    inverse[z]]], subclass[intersection[domain[z], fix[
     composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
   range[iterate[z, intersection[complement[y], domain[z]]]]]
```

```
and[equal[0,
   intersection[fix[composite[inverse[z_], id[intersection[complement[range[iterate[
          z_, intersection[complement[y_], domain[z_]]]], domain[z_]]], Di]],
    range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
  FUNCTION[composite[id[range[iterate[z_, intersection[complement[y_], domain[z_]]]],
    inverse[z_]]], subclass[intersection[domain[z_], fix[composite[z_,
      id[range[iterate[z_, intersection[complement[y_], domain[z_]]]], Di]]],
   range[iterate[z_, intersection[complement[y_], domain[z_]]]]] :=
 and[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]],
    inverse[z]]], subclass[intersection[domain[z], fix[
     composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
   range[iterate[z, intersection[complement[y], domain[z]]]]]
```

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
    {p1 -> FUNCTION[inverse[z]], p2 -> FUNCTION[inverse[composite[z,
    id[r]]]],
        p3 -> FUNCTION[inverse[h]]}]]
```

```
or[not[FUNCTION[inverse[z]]], subclass[intersection[domain[z], fix[
     composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
   range[iterate[z, intersection[complement[y], domain[z]]]]] == True
```

```
or[not[FUNCTION[inverse[z_]]], subclass[intersection[domain[z_], fix[composite[z_,
      id[range[iterate[z_, intersection[complement[y_], domain[z_]]]], Di]]],
   range[iterate[z_, intersection[complement[y_], domain[z_]]]]] := True
```

```
or[
  and[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]]],
     inverse[z]]], subclass[intersection[domain[z], fix[
      composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]]], Di]]],
    range[iterate[z, intersection[complement[y], domain[z]]]]],
  not[FUNCTION[inverse[z]]] // NotNotTest

or[
  and[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]]],
     inverse[z]]], subclass[intersection[domain[z], fix[
      composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]]], Di]]],
    range[iterate[z, intersection[complement[y], domain[z]]]]],
  not[FUNCTION[inverse[z]]] == True

or[and[FUNCTION[composite[
     id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]], inverse[z_]]],
   subclass[intersection[domain[z_], fix[composite[z_,
       id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]], Di]]],
    range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
  not[FUNCTION[inverse[z_]]] := True
```

■ synthesis

```
implies[FUNCTION[inverse[z]], FUNCTION[inverse[h]]]

True

implies[member[z, BIJ], member[h, BIJ]] // NotNotTest

or[and[FUNCTION[
    composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]]]],
   FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]]],
     inverse[z]]], member[composite[z,
     id[range[iterate[z, intersection[complement[y], domain[z]]]]], V], member[
    intersection[complement[range[iterate[z, intersection[complement[y], domain[z]]]],
     domain[z]], V], subclass[intersection[domain[z], fix[
      composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
    range[iterate[z, intersection[complement[y], domain[z]]]]],
  not[FUNCTION[z]], not[FUNCTION[inverse[z]]],
  not[member[z, V]]] == True

or[and[FUNCTION[
    composite[z_, id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]]],
   FUNCTION[composite[id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
     inverse[z_]]], member[
    composite[z_, id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
    V], member[intersection[complement[
      range[iterate[z_, intersection[complement[y_], domain[z_]]]], domain[z_]], V],
   subclass[intersection[domain[z_], fix[composite[z_,
       id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]], Di]]],
    range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
  not[FUNCTION[z_]], not[FUNCTION[inverse[z_]]],
  not[member[z_, V]]] := True

implies[member[z, BIJ], member[h, BIJ]]

True
```

■ a key lemma

Bijections yield equipollence statements:

```
SubstTest[implies, member[x, BIJ], member[pair[domain[x], range[x]], Q], x -> h]
```

```
or[member[pair[domain[z], union[
     image[z, range[iterate[z, intersection[complement[y], domain[z]]]]], intersection[
      complement[range[iterate[z, intersection[complement[y], domain[z]]]],
      domain[z]]]], Q], not[FUNCTION[
    composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]]]]],
  not[FUNCTION[composite[id[range[iterate[z, intersection[complement[y], domain[z]]]]],
     inverse[z]]]], not[member[
    composite[z, id[range[iterate[z, intersection[complement[y], domain[z]]]]], V]],
  not[member[intersection[complement[
      range[iterate[z, intersection[complement[y], domain[z]]]], domain[z]], V]],
  not[subclass[intersection[domain[z], fix[composite[z,
       id[range[iterate[z, intersection[complement[y], domain[z]]]], Di]]],
    range[iterate[z, intersection[complement[y], domain[z]]]]]] == True
```

```
or[member[pair[domain[z_],
    union[image[z_, range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
     intersection[complement[range[
        iterate[z_, intersection[complement[y_], domain[z_]]]], domain[z_]]], Q], not[
   FUNCTION[composite[id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
     inverse[z_]]]], not[FUNCTION[composite[z_,
     id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]]]], not[member[
    composite[z_, id[range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
    V]], not[member[intersection[complement[
      range[iterate[z_, intersection[complement[y_], domain[z_]]]], domain[z_]], V]],
  not[subclass[intersection[domain[z_], fix[composite[z_,
       id[range[iterate[z_, intersection[complement[y_], domain[z_]]]], Di]]],
    range[iterate[z_, intersection[complement[y_], domain[z_]]]]]] := True
```

Note that the domain of **h** is the same as that of **z**.

```
domain[h]
```

```
domain[z]
```

Equality substitution requires some manual intervention:

```
SubstTest[implies, and[equal[u, v], member[u, w]], member[v, w],
    {u -> pair[domain[z], range[h]], v -> pair[domain[z], y], w -> Q}]
```

```
or[member[pair[domain[z], y], Q],
  not[equal[singleton[y], singleton[union[image[z, range[
        iterate[z, intersection[complement[y], domain[z]]]]], intersection[complement[
        range[iterate[z, intersection[complement[y], domain[z]]]], domain[z]]]]],
  not[member[pair[domain[z], union[image[z, range[iterate[z,
         intersection[complement[y], domain[z]]]], intersection[complement[range[
         iterate[z, intersection[complement[y], domain[z]]]], domain[z]]]], Q]]] == True
```

```
or[member[pair[domain[z_], y_], Q],
  not[equal[singleton[y_], singleton[union[image[z_, range[
        iterate[z_, intersection[complement[y_], domain[z_]]]]], intersection[
      complement[range[iterate[z_, intersection[complement[y_], domain[z_]]]],
      domain[z_]]]]], not[member[pair[domain[z_],
     union[image[z_, range[iterate[z_, intersection[complement[y_], domain[z_]]]]],
      intersection[complement[range[iterate[z_,
          intersection[complement[y_], domain[z_]]]], domain[z_]]], Q]]] := True
```

The crucial step which combines everything is this:

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
  implies[and[p4, p5], p6], implies[p6, p7], implies[and[p3, p7], p8],
        not[implies[and[p1, p4, p5], p8]],
            {p1 -> member[z, BIJ],
   p2 -> member[h, BIJ], p3 -> member[pair[domain[z], range[h]], Q],
p4 -> subclass[range[z], y], p5 -> subclass[y, domain[z]], p6 -> equal[y, range[h]],
        p7 -> equal[pair[domain[z], y], pair[domain[z], range[h]]],
   p8 -> member[pair[domain[z], y], Q]}]]
```

```
or[member[pair[domain[z], y], Q], not[FUNCTION[z]], not[FUNCTION[inverse[z]]],
  not[member[z, V]], not[subclass[y, domain[z]]], not[subclass[range[z], y]]] == True
```

```
or[member[pair[domain[z_], y_], Q],
  not[FUNCTION[z_]], not[FUNCTION[inverse[z_]]], not[member[z_, V]],
  not[subclass[y_, domain[z_]]], not[subclass[range[z_], y_]]] := True
```

This is what we have found:

```
implies[and[member[z, BIJ], subclass[range[z], y], subclass[y, domain[z]]],
 member[pair[domain[z], y], Q]]
```

```
True
```

■ eliminating the variables y and z

```
Map[equal[V, #] &, SubstTest[class, y,
   implies[and[member[z, u], subclass[range[z], y], subclass[y, domain[z]]],
    member[pair[domain[z], y], v]],
    {u -> BIJ, v -> Q}]] // Reverse
```

```
or[not[FUNCTION[z]], not[FUNCTION[inverse[z]]], not[member[z, V]],
  subclass[intersection[image[S, singleton[range[z]]], P[domain[z]]],
   image[Q, singleton[domain[z]]]]] == True
```

```
or[not[FUNCTION[z_]], not[FUNCTION[inverse[z_]]], not[member[z_, V]],
  subclass[intersection[image[S, singleton[range[z_]]], P[domain[z_]]],
   image[Q, singleton[domain[z_]]]]] := True
```

Restatement:

```
implies[member[z, BIJ],
 subclass[intersection[image[S, singleton[range[z]]], P[domain[z]]],
  image[Q, singleton[domain[z]]]]]
```

```
True
```

Next we eliminate **z**.

```
Map[assert[equal[V, #]] &, SubstTest[class, z, implies[member[z, u],
    subclass[intersection[image[S, singleton[range[z]]], P[domain[z]]],
     image[Q, singleton[domain[z]]]]], u -> BIJ]] // Reverse
```

```
subclass[intersection[composite[Q, S], inverse[S]], Q] == True
```

This is a special case of the Schroeder−Bernsteintheorem:

```
subclass[intersection[composite[Q, S], inverse[S]], Q] := True
```

The following corollary is needed in the next section to derive the general case.

```
SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
    {u -> intersection[composite[Q, S], inverse[S]], v -> Q, w -> Q}]

subclass[composite[intersection[composite[Q, S], inverse[S]], Q], Q] == True

subclass[composite[intersection[composite[Q, S], inverse[S]], Q], Q] := True
```

■ the general case

The derivation of the general case from the special case is done somewhat differently from what Jean Rubin does. The method used here was discovered using abstraction. It avoids having to consider two bijections.

```
SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
    {u -> intersection[composite[cross[inverse[z], Id], FIRST],
    composite[cross[inverse[z], Id], SECOND]],
        v -> composite[cross[inverse[z], Id], SECOND,
    id[cross[composite[z, inverse[z]], Id]]],
        w -> cart[x, y]}]

subclass[intersection[composite[x, z], composite[y, z]],
  composite[intersection[y, composite[x, z, inverse[z]]], z]] == True

% /. {x -> S, y -> inverse[S], z -> Q}

subclass[intersection[composite[Q, S], composite[Q, inverse[S]]],
  composite[intersection[composite[Q, S], inverse[S]], Q]] == True

subclass[intersection[composite[Q, S], composite[Q, inverse[S]]],
  composite[intersection[composite[Q, S], inverse[S]], Q]] := True

SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> intersection[composite[Q, S], composite[Q, inverse[S]]],
        v -> composite[intersection[composite[Q, S], inverse[S]], Q], w -> Q}]

subclass[intersection[composite[Q, S], composite[Q, inverse[S]]], Q] == True

subclass[intersection[composite[Q, S], composite[Q, inverse[S]]], Q] := True
```

The final steps are done to reformulate the Schroeder−Bernsteintheorem as an equation.

```
SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
    {u -> Id, v -> S, w -> Q}]

subclass[Q, composite[Q, S]] == True

subclass[Q, composite[Q, S]] := True

SubstTest[subclass, inverse[u], inverse[v], {u -> Q, v -> composite[Q, S]}]

subclass[Q, composite[Q, inverse[S]]] == True

subclass[Q, composite[Q, inverse[S]]] := True
```

```
SubstTest[and, subclass[u, v], subclass[v, u],
    {u -> intersection[composite[Q, S], composite[Q, inverse[S]]], v -> Q}]
```

```
True == equal[Q, intersection[composite[Q, S], composite[Q, inverse[S]]]]
```

This is the final version of the Schroeder−Bernsteintheorem:

```
intersection[composite[Q, S], composite[Q, inverse[S]]] := Q
```