double induction, part 2

Johan G. F. Belinfante and Ming Li
2006 April 13

```
In[1]:= SetDirectory["l:"]; << goedel80.13a; << tools.m

        :Package Title: goedel80.13a          2006 April 13 at 11:08 a.m.

        It is now:  2006 Apr 13 at 14:7

        Loading Simplification Rules

        TOOLS.M                      Revised 2006 March 7

        weightlimit = 40
```

summary

Raymond M. Smullyan and Melvin Fitting present a double induction theorem of a general nature, with applications to functions **x** that satisfy **subclass[x, S]**.

```
In[2]:= "Raymond M. Smullyan and Melvin Fitting, Set Theory and the Continuum Problem,
           Oxford Science Publications, Clarendon Press, Oxford, 1996.  (see page 33)";
```

In the notebook **dbl-ind1.nb**, the following variant of their theorem was derived for the classic case **y = SUCC**, **z = 0**.

```
In[3]:= or[not[subclass[omega, image[inverse[x], set[0]]]],
         not[subclass[composite[SUCC, intersection[inverse[x], x]], x]],
         subclass[cart[omega, omega], x]]

Out[3]= True
```

In this notebook, the general case is derived as a corollary of the classic case. Our target goal will be to prove a similar result, with an arbitrary function **funpart[y]** in place of **SUCC**, and **hull[invar[funpart[y]], set[z]]** in place of **omega**. The latter is the smallest set which is invariant under **funpart[y]**, and contains **set[z]**. It may be obtained by iteratively applying **funpart[y]** to **z**.

```
In[4]:= range[iterate[funpart[y], set[z]]]

Out[4]= hull[invar[funpart[y]], set[z]]
```

some lemmas

Lemma 1.

```
In[5]:= Map[A, SubstTest[image, iterate[u, v], set[0], {u → funpart[x], v → set[setpart[y]]}]]

Out[5]= APPLY[iterate[funpart[x], set[setpart[y]]], 0] == setpart[y]

In[6]:= APPLY[iterate[funpart[x_], set[setpart[y_]]], 0] := setpart[y]
```

Lemma 2.

```
In[7]:= composite[inverse[SUCC], inverse[iterate[x, y]]] // DoubleInverse

Out[7]= composite[inverse[SUCC], inverse[iterate[x, y]]] ==
         composite[inverse[iterate[x, y]], inverse[x]]

In[8]:= composite[inverse[SUCC], inverse[iterate[x_, y_]]] :=
         composite[inverse[iterate[x, y]], inverse[x]]
```

Lemma 3.

```
In[9]:= SubstTest[intersection, range[z], fix[composite[z, y]], z → inverse[x]]

Out[9]= intersection[domain[x], fix[composite[inverse[x], y]]] == fix[composite[inverse[x], y]]

In[10]:= intersection[domain[x_], fix[composite[inverse[x_], y_]]] :=
          fix[composite[inverse[x], y]]
```

Lemma 4.

```
In[11]:= subclass[image[inverse[SUCC], domain[iterate[x, y]]],
           domain[iterate[x, y]]] // AssertTest

Out[11]= subclass[image[inverse[SUCC], domain[iterate[x, y]]], domain[iterate[x, y]]] == True

In[12]:= subclass[image[inverse[SUCC], domain[iterate[x_, y_]]],
           domain[iterate[x_, y_]]] := True
```

Restatement:

```
In[13]:= equal[intersection[domain[iterate[y, z]],
           image[SUCC, complement[domain[iterate[y, z]]]]], 0]

Out[13]= True

In[14]:= intersection[domain[iterate[y_, z_]],
           image[SUCC, complement[domain[iterate[y_, z_]]]]] := 0
```

transposing SUCC and restricting to omega

In the classic version, one can transpose **SUCC**:

```
In[15]:= Map[or[not[subclass[omega, image[inverse[x], set[0]]]],
           not[#], subclass[cart[omega, omega], x]] &,
         subclass[intersection[x, inverse[x]], composite[inverse[SUCC], x]] // AssertTest]

Out[15]= or[not[subclass[omega, image[inverse[x], set[0]]]],
          not[subclass[intersection[x, inverse[x]], composite[inverse[SUCC], x]]],
          subclass[cart[omega, omega], x]] == True

In[16]:= (% /. x → x_) /. Equal → SetDelayed
```

Replacing **x** with its restriction to **omega** yields:

```
In[17]:= SubstTest[or, not[subclass[omega, image[inverse[y], set[0]]]],
          not[subclass[intersection[y, inverse[y]], composite[inverse[SUCC], y]]],
          subclass[cart[omega, omega], y], y → restrict[x, omega, omega]]

Out[17]= or[not[subclass[omega, image[inverse[x], set[0]]]],
          not[subclass[composite[id[omega], intersection[x, inverse[x]], id[omega]],
            composite[inverse[SUCC], id[omega], x]]], subclass[cart[omega, omega], x]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

This version is most suitable for deriving the general case. The key substitution will be to replace **x** with

```
In[19]:= union[composite[inverse[iterate[funpart[y], set[setpart[z]]]],
           x, iterate[funpart[y], set[setpart[z]]]],
          cart[V, complement[domain[iterate[funpart[y], set[setpart[z]]]]]],
          cart[complement[domain[iterate[funpart[y], set[setpart[z]]]]], V]];
```

The result of doing so initially produces a complicated expression which can be simplified. We shall refer to this as the translated classic theorem.

---

a tautology

One expression in the translated classic theorem is miraculously recognized as being a tautology:

```
In[20]:= (subclass[cart[omega, intersection[omega, complement[domain[t]]]],
            union[cart[V, image[inverse[SUCC], intersection[omega, complement[domain[t]]]]],
             cart[complement[domain[t]], omega], composite[inverse[SUCC],
              id[omega], inverse[t], x, t]]] // AssertTest) /. t → iterate[y, z]

Out[20]= subclass[
          cart[omega, intersection[omega, complement[domain[iterate[y, z]]]], union[cart[V,
            image[inverse[SUCC], intersection[omega, complement[domain[iterate[y, z]]]]],
           cart[complement[domain[iterate[y, z]]], omega], composite[inverse[SUCC],
            id[omega], inverse[iterate[y, z]], x, iterate[y, z]]]] == True

In[21]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

simplifying the recursion condition

The recursion condition in the translated classic version also dramatically simplifies:

```
In[24]:= subclass[composite[id[omega], inverse[iterate[funpart[y], set[z]]],
             intersection[x, inverse[x]], iterate[funpart[y], set[z]]],
            union[cart[V, image[inverse[SUCC], intersection[omega,
                complement[domain[iterate[funpart[y], set[z]]]]]]],
             cart[complement[domain[iterate[funpart[y], set[z]]]], omega],
             composite[inverse[SUCC], id[omega], inverse[iterate[funpart[y], set[z]]],
              x, iterate[funpart[y], set[z]]]]] // AssertTest // RotateFix

Out[24]= subclass[composite[id[omega], inverse[iterate[funpart[y], set[z]]],
           intersection[x, inverse[x]], iterate[funpart[y], set[z]]],
          union[cart[V, image[inverse[SUCC],
             intersection[omega, complement[domain[iterate[funpart[y], set[z]]]]]],
           cart[complement[domain[iterate[funpart[y], set[z]]]], omega],
           composite[inverse[SUCC], id[omega],
            inverse[iterate[funpart[y], set[z]]], x, iterate[funpart[y], set[z]]]]] ==
         equal[0, intersection[fix[composite[inverse[funpart[y]], complement[x],
             id[hull[invar[funpart[y]], set[z]]], intersection[x, inverse[x]]]],
           hull[invar[funpart[y]], set[z]]]]

In[25]:= ((First[%] /. {x → x_, y → y_, z → z_}) == Last[%]) /. Equal → SetDelayed
```

the translated classic theorem

At this point the translated classic theorem is:

```
In[26]:= SubstTest[or, not[subclass[omega, image[inverse[w], set[0]]]],
          not[subclass[composite[id[omega], intersection[w, inverse[w]], id[omega]],
            composite[inverse[SUCC], id[omega], w]], subclass[cart[omega, omega], w],
          w -> union[composite[inverse[iterate[funpart[y], set[setpart[z]]]],
             x, iterate[funpart[y], set[setpart[z]]]],
            cart[V, complement[domain[iterate[funpart[y], set[setpart[z]]]]]],
            cart[complement[domain[iterate[funpart[y], set[setpart[z]]]]], V]]]

Out[26]= or[not[equal[0, intersection[fix[composite[inverse[funpart[y]], complement[x],
               id[hull[invar[funpart[y]], set[setpart[z]]]], intersection[x, inverse[x]]]],
             hull[invar[funpart[y]], set[setpart[z]]]]],
           not[subclass[domain[iterate[funpart[y], set[setpart[z]]]],
             image[inverse[iterate[funpart[y], set[setpart[z]]]],
              image[inverse[x], set[setpart[z]]]]], subclass[cart[omega, omega],
            union[cart[V, complement[domain[iterate[funpart[y], set[setpart[z]]]]]],
             cart[complement[domain[iterate[funpart[y], set[setpart[z]]]]], V],
             composite[inverse[iterate[funpart[y], set[setpart[z]]]],
              x, iterate[funpart[y], set[setpart[z]]]]]] == True
```

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

the conclusion

The following step leads us from the conclusion of the translated classic theorem to the desired conclusion in the target theorem.

```
In[28]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u -> cart[omega, omega],
           v -> union[cart[V, complement[domain[iterate[funpart[y], set[z]]]]],
             cart[complement[domain[iterate[funpart[y], set[z]]]], V],
             composite[inverse[iterate[funpart[y], set[z]]], x, iterate[funpart[y], set[z]]]],
           w → cross[iterate[funpart[y], set[z]], iterate[funpart[y], set[z]]]}]
```

```
Out[28]= or[not[subclass[cart[omega, omega],
            union[cart[V, complement[domain[iterate[funpart[y], set[z]]]]],
             cart[complement[domain[iterate[funpart[y], set[z]]]], V], composite[
              inverse[iterate[funpart[y], set[z]]], x, iterate[funpart[y], set[z]]]]]],
           subclass[cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]]],
            x]] == True
```

```
In[29]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

the initial condition

The initial condition in the target theorem implies one of that of the translated classic theorem:

```
In[30]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> hull[invar[funpart[y]], set[z]],
           v -> image[inverse[x], set[z]], w -> inverse[iterate[funpart[y], set[z]]]}]
```

```
Out[30]= or[not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]],
           subclass[domain[iterate[funpart[y], set[z]]],
            image[inverse[iterate[funpart[y], set[z]]], image[inverse[x], set[z]]]]] == True
```

```
In[31]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

the recursion condition

An equivalent version of the recursion condition of the target theorem is this:

```
In[32]:= subclass[intersection[inverse[x], x], union[cart[V, complement[domain[funpart[y]]]],
           composite[inverse[funpart[y]], x]]] // AssertTest

Out[32]= subclass[intersection[x, inverse[x]],
          union[cart[V, complement[domain[funpart[y]]]], composite[inverse[funpart[y]], x]]] ==
         subclass[composite[funpart[y], intersection[x, inverse[x]]], x]

In[33]:= subclass[intersection[x_, inverse[x_]], union[
           cart[V, complement[domain[funpart[y_]]]], composite[inverse[funpart[y_]], x_]]] :=
         subclass[composite[funpart[y], intersection[x, inverse[x]]], x]
```

From the latter one can derive a simple consequence by restricting to the cartesian square of **hull[invar[funpart[y]],set[z]]**. The following lemma is needed to simplify the result:

```
In[36]:= subclass[composite[id[hull[invar[funpart[y]], set[z]]], intersection[x, inverse[x]],
            id[hull[invar[funpart[y]], set[z]]]], union[cart[hull[invar[funpart[y]], set[z]],
             intersection[complement[domain[funpart[y]]], hull[invar[funpart[y]], set[z]]]],
            composite[id[hull[invar[funpart[y]], set[z]]], inverse[funpart[y]],
             x, id[hull[invar[funpart[y]], set[z]]]]]] // AssertTest // RotateFix

Out[36]= subclass[composite[id[hull[invar[funpart[y]], set[z]]], intersection[x, inverse[x]],
           id[hull[invar[funpart[y]], set[z]]]], union[cart[hull[invar[funpart[y]], set[z]],
            intersection[complement[domain[funpart[y]]], hull[invar[funpart[y]], set[z]]]],
           composite[id[hull[invar[funpart[y]], set[z]]], inverse[funpart[y]],
            x, id[hull[invar[funpart[y]], set[z]]]]]] ==
         equal[0, intersection[fix[composite[inverse[funpart[y]], complement[x],
            id[hull[invar[funpart[y]], set[z]]], intersection[x, inverse[x]]]],
           hull[invar[funpart[y]], set[z]]]]

In[37]:= ((First[%] /. {x → x_, y → y_, z → z_}) == Last[%]) /. Equal → SetDelayed
```

Using this result, one finds:

```
In[38]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u -> intersection[x, inverse[x]], v ->
            union[cart[V, complement[domain[funpart[y]]]], composite[inverse[funpart[y]], x]],
           w → id[cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]]]]}]

Out[38]= or[equal[0, intersection[fix[composite[inverse[funpart[y]], complement[x],
              id[hull[invar[funpart[y]], set[z]]], intersection[x, inverse[x]]]],
            hull[invar[funpart[y]], set[z]]]],
           not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]]] == True

In[39]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

finally the actual proof

```
In[41]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4], implies[and[p3, p4], p5],
           implies[p5, p6], not[implies[and[p1, p2], p6]], {p1 -> subclass[
              hull[invar[funpart[y]], set[setpart[z]]], image[inverse[x], set[setpart[z]]]],
            p2 -> subclass[composite[funpart[y], intersection[x, inverse[x]]], x],
            p3 → subclass[domain[iterate[funpart[y], set[setpart[z]]]], image[inverse[
                iterate[funpart[y], set[setpart[z]]]], image[inverse[x], set[setpart[z]]]]],
            p4 → equal[0, intersection[fix[composite[inverse[funpart[y]], complement[x],
                 id[hull[invar[funpart[y]], set[setpart[z]]]], intersection[x, inverse[x]]]],
              hull[invar[funpart[y]], set[setpart[z]]]]], p5 -> subclass[cart[omega, omega],
             union[cart[V, complement[domain[iterate[funpart[y], set[setpart[z]]]]]],
              cart[complement[domain[iterate[funpart[y], set[setpart[z]]]]], V],
              composite[inverse[iterate[funpart[y], set[setpart[z]]]],
               x, iterate[funpart[y], set[setpart[z]]]]]],
            p6 -> subclass[cart[hull[invar[funpart[y]], set[setpart[z]]],
              hull[invar[funpart[y]], set[setpart[z]]], x]}]]

Out[41]= or[not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]], not[subclass[
           hull[invar[funpart[y]], set[setpart[z]]], image[inverse[x], set[setpart[z]]]]],
          subclass[cart[hull[invar[funpart[y]], set[setpart[z]]],
           hull[invar[funpart[y]], set[setpart[z]]], x]] == True

In[42]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

In the next section, this statement will be simplified, removing the **setpart** wrapper.

simplifying the statement

The **setpart** wrapper can be replaced with a sethood literal.

```
In[43]:= SubstTest[implies, equal[z, setpart[w]],
          or[not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]], subclass[
            cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]], x]], w → z]

Out[43]= or[not[member[z, V]],
           not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]], subclass[
            cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]], x]] == True

In[44]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The theorem also holds trivially when **z** is a proper class.

```
In[45]:= SubstTest[implies, equal[0, w],
          or[not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[funpart[y]], w], image[inverse[x], w]]], subclass[
            cart[hull[invar[funpart[y]], w], hull[invar[funpart[y]], w]], x]], w → set[z]]

Out[45]= or[member[z, V], not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]]], subclass[
            cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]]], x]] == True

In[46]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Combining the case of a set and the case of a proper class yields a simplified statement of the general **Double-Induction Theorem**:

```
In[47]:= SubstTest[and, or[p, q], implies[p, q], {p → member[z, V],
            q -> or[not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
              not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]]],
              subclass[cart[hull[invar[funpart[y]], set[z]],
                hull[invar[funpart[y]], set[z]]], x]]}] // Reverse

Out[47]= or[not[subclass[composite[funpart[y], intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[funpart[y]], set[z]], image[inverse[x], set[z]]]], subclass[
            cart[hull[invar[funpart[y]], set[z]], hull[invar[funpart[y]], set[z]]], x]] == True

In[48]:= or[not[subclass[composite[funpart[y_], intersection[x_, inverse[x_]]], x_]],
           not[subclass[hull[invar[funpart[y_]], set[z_]], image[inverse[x_], set[z_]]]],
           subclass[cart[hull[invar[funpart[y_]], set[z_]],
             hull[invar[funpart[y_]], set[z_]]], x_]] := True
```

One could replace the **funpart** wrapper, if desired, with a **FUNCTION** literal:

```
In[49]:= SubstTest[implies, equal[y, funpart[w]],
          or[not[subclass[composite[y, intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[y], set[z]], image[inverse[x], set[z]]]],
           subclass[cart[hull[invar[y], set[z]], hull[invar[y], set[z]]], x]], w → y]

Out[49]= or[not[FUNCTION[y]], not[subclass[composite[y, intersection[x, inverse[x]]], x]],
           not[subclass[hull[invar[y], set[z]], image[inverse[x], set[z]]]],
           subclass[cart[hull[invar[y], set[z]], hull[invar[y], set[z]]], x]] == True

In[50]:= or[not[FUNCTION[y_]], not[subclass[composite[y_, intersection[x_, inverse[x_]]], x_]],
           not[subclass[hull[invar[y_], set[z_]], image[inverse[x_], set[z_]]]],
           subclass[cart[hull[invar[y_], set[z_]], hull[invar[y_], set[z_]]], x_]] := True
```

application to progressing functions

Lemma.

```
In[51]:= subclass[composite[funpart[intersection[S, x]],
            intersection[S, composite[S, funpart[intersection[S, x]]]]],
           union[composite[S, funpart[intersection[S, x]]], inverse[S]]] // AssertTest

Out[51]= subclass[composite[funpart[intersection[S, x]],
            intersection[S, composite[S, funpart[intersection[S, x]]]]],
           union[composite[S, funpart[intersection[S, x]]], inverse[S]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed

In[53]:= SubstTest[or, not[subclass[composite[funpart[w], intersection[z, inverse[z]]], z]],
          not[subclass[hull[invar[funpart[w]], set[0]], image[inverse[z], set[0]]]],
          subclass[cart[hull[invar[funpart[w]], set[0]], hull[invar[funpart[w]], set[0]]], z],
          {w → intersection[S, x],
           z → union[inverse[S], composite[S, funpart[intersection[S, x]]]]}]

Out[53]= subclass[cart[hull[invar[funpart[intersection[S, x]]], set[0]],
            hull[invar[funpart[intersection[S, x]]], set[0]]],
           union[composite[S, funpart[intersection[S, x]]], inverse[S]]] == True

In[54]:= subclass[cart[hull[invar[funpart[intersection[S, x_]]], set[0]],
            hull[invar[funpart[intersection[S, x_]]], set[0]]],
           union[composite[S, funpart[intersection[S, x_]]], inverse[S]]] := True
```

The **funpart** wrapper can be replaced with appropriate literals:

```
In[55]:= SubstTest[implies, equal[x, funpart[intersection[S, y]]],
          subclass[cart[hull[invar[x], set[0]], hull[invar[x], set[0]]],
           union[composite[S, x], inverse[S]]], y → x]

Out[55]= or[not[FUNCTION[x]], not[subclass[x, S]],
           subclass[cart[hull[invar[x], set[0]], hull[invar[x], set[0]]],
            union[composite[S, x], inverse[S]]]] == True

In[56]:= or[not[FUNCTION[x_]], not[subclass[x_, S]],
           subclass[cart[hull[invar[x_], set[0]], hull[invar[x_], set[0]]],
            union[composite[S, x_], inverse[S]]]] := True
```