double induction, part 1

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

```
In[1]:= SetDirectory["l:"]; << goedel80.12a; << tools.m

        :Package Title: goedel80.12a          2006 April 12 at 9:25 p.m.

        It is now:  2006 Apr 13 at 10:54

        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, along 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 this notebook, a variant of their double induction theorem is derived for the classic case of the successor function **SUCC**. The general case, as well as the application, will be obtained as a corollary of this special case in a separate notebook.

the hypotheses

The double induction theorem has two hypotheses, the first of which can be viewed as an initial condition. (Comment: The relation **x** here corresponds what Smullyan and Fitting call **R**.) For the classic case, this hypothesis is:

```
In[3]:= assert[forall[u, implies[member[u, omega], member[pair[u, 0], x]]]]

Out[3]= subclass[omega, image[inverse[x], set[0]]]
```

Their second hypothesis can be viewed as a recursion condition. For the classic case this hypothesis is:

```
In[4]:= assert[
          forall[u, v, implies[and[member[u, omega], member[v, omega], member[pair[u, v], x],
             member[pair[v, u], x]], member[pair[u, succ[v]], x]]]] // RotateFix

Out[4]= subclass[composite[id[omega], SUCC, intersection[x, inverse[x]], id[omega]], x]
```

In this notebook, the recursion hypothesis will be simplified slightly as follows: (Note that both hypotheses are satisfied by **cart[omega, omega]**.)

```
In[5]:= hypotheses[x_] := and[subclass[composite[SUCC, intersection[x, inverse[x]]], x],
          subclass[omega, image[inverse[x], set[0]]]]
```

The function **SUCC** in the recursion hypothesis can be moved from one side of the inclusion to the other. For convenience, this will be made into a temporary rewrite rule. (One could entertain the idea of making this a permanent rewrite rule, but it is not clear in which direction it would best be oriented.)

```
In[6]:= subclass[intersection[x, inverse[x]], composite[inverse[SUCC], x]] // AssertTest

Out[6]= subclass[intersection[x, inverse[x]], composite[inverse[SUCC], x]] ==
         subclass[composite[SUCC, intersection[x, inverse[x]]], x]

In[7]:= subclass[intersection[x_, inverse[x_]], composite[inverse[SUCC], x_]] :=
         subclass[composite[SUCC, intersection[x, inverse[x]]], x]
```

introducing variables

Introducing the variables **u** and **v**, the initial condition yields:

```
In[8]:= Map[or[member[pair[v, u], x], #] &, SubstTest[implies, and[member[r, s], subclass[s, t]],
          member[r, t], {r → v, s → omega, t → image[inverse[x], set[u]]}]]

Out[8]= or[member[pair[v, u], x], not[member[v, omega]],
          not[subclass[omega, image[inverse[x], set[u]]]]] == True

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

In particular, when **v = 0**, this reduces to:

```
In[10]:= SubstTest[or, member[pair[v, u], x], not[member[v, omega]],
          not[subclass[omega, image[inverse[x], set[u]]]], v → 0]

Out[10]= or[member[pair[0, u], x], not[subclass[omega, image[inverse[x], set[u]]]]] == True

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

Similarly, introducing variables into the transposed form of the recursion relation yields:

```
In[12]:= Map[implies[and[member[u, omega], member[v, omega]], #] &,
          SubstTest[implies, and[member[r, s], subclass[s, t]], member[r, t],
           {r → pair[u, v], s -> intersection[x, inverse[x]], t -> composite[inverse[SUCC], x]}]]

Out[12]= or[member[pair[u, succ[v]], x], not[member[u, omega]],
           not[member[v, omega]], not[member[pair[u, v], x]], not[member[pair[v, u], x]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]]] == True

In[13]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

a key lemma

Lemma.

```
In[14]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p1, p2, p3], p4], not[implies[and[p1, p2], p4]], {p1 →
             and[member[u, omega], member[v, omega], subclass[omega, image[inverse[x], set[u]]],
              subclass[composite[SUCC, intersection[x, inverse[x]]], x]],
            p2 -> member[pair[u, v], x], p3 → member[pair[v, u], x],
            p4 -> member[pair[u, succ[v]], x]}]]

Out[14]= or[member[pair[u, succ[v]], x], not[member[u, omega]], not[member[v, omega]],
           not[member[pair[u, v], x]], not[subclass[omega, image[inverse[x], set[u]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]]] == True

In[15]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

The variable **v** will now be eliminated. To achieve this, the key idea is to turn off the **simplify** flag.

```
In[16]:= simplify = False;

In[17]:= Map[equal[V, #] &,
           SubstTest[class, v, or[member[pair[u, v], z], not[member[pair[u, v], y]],
             not[subclass[omega, r]], not[subclass[t, w]]], {y → restrict[x, omega, omega],
             z → composite[inverse[SUCC], x], r -> image[inverse[x], set[u]],
             t -> intersection[x, inverse[x]], w -> composite[inverse[SUCC], x]}]] // Reverse

Out[17]= or[not[member[u, omega]], not[subclass[omega, image[inverse[x], set[u]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]],
           subclass[intersection[omega, image[SUCC, image[x, set[u]]]], image[x, set[u]]]] == True

In[18]:= (% /. {u → u_, x → x_}) /. Equal → SetDelayed
```

a first application of induction

Now use induction.

```
In[19]:= Map[implies[member[u, omega], #] &, SubstTest[implies, INDUCTIVE[z],
           subclass[omega, z], z → intersection[omega, image[x, set[u]]]]]

Out[19]= or[not[member[u, omega]], not[member[pair[u, 0], x]],
           not[subclass[intersection[omega, image[SUCC, image[x, set[u]]]], image[x, set[u]]]],
           subclass[omega, image[x, set[u]]]] == True

In[20]:= (% /. {u → u_, x → x_}) /. Equal → SetDelayed
```

Theorem. (In the terminology of Smullyan and Fitting, this says right-normal implies left-normal.)

```
In[21]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[p1, p4], implies[and[p1, p3, p4], p5], not[implies[p1, p5]], {p1 ->
             and[member[u, omega], hypotheses[x], subclass[omega, image[inverse[x], set[u]]]],
            p2 -> member[pair[0, u], x], p3 → member[pair[u, 0], x],
            p4 -> subclass[intersection[omega, image[SUCC, image[x, set[u]]]], image[x, set[u]]],
            p5 -> subclass[omega, image[x, set[u]]]}]]

Out[21]= or[not[member[u, omega]], not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[omega, image[inverse[x], set[u]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]],
           subclass[omega, image[x, set[u]]] == True

In[22]:= (% /. {u → u_, x → x_}) /. Equal → SetDelayed
```

---

a second application of induction

Theorem.

```
In[23]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p1, p3], p4], implies[and[p1, p2, p4], p5],
           not[implies[p1, p5]], {p1 → and[member[u, omega], member[v, omega],
              hypotheses[x], subclass[omega, image[inverse[x], set[v]]]],
            p2 → member[pair[u, v], x], p3 -> subclass[omega, image[x, set[v]]],
            p4 → member[pair[v, u], x], p5 → member[pair[u, succ[v]], x]}]]

Out[23]= or[member[pair[u, succ[v]], x], not[member[u, omega]],
           not[member[v, omega]], not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[omega, image[inverse[x], set[v]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]]] == True

In[24]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

The variables **u** and **v** will now be eliminated. The result obtained says that the hypotheses imply that **intersection[omega,ub[x,omega]]** is invariant under **SUCC**.

```
In[25]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
         or[member[pair[u, succ[v]], x], not[member[u, omega]],
             not[member[v, omega]], not[subclass[omega, image[inverse[x], set[0]]]],
             not[subclass[omega, image[inverse[x], set[v]]]], not[subclass[s, t]]],
            {s -> intersection[x, inverse[x]], t -> composite[inverse[SUCC], x]}]] // Reverse

Out[25]= or[not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]],
           subclass[cart[omega, intersection[omega, image[SUCC, ub[x, omega]]]], x]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

Induction is used again.

```
In[27]:= SubstTest[implies, INDUCTIVE[z],
          subclass[omega, z], z -> intersection[omega, ub[x, omega]]]

Out[27]= or[not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[cart[omega, intersection[omega, image[SUCC, ub[x, omega]]]], x]],
           subclass[cart[omega, omega], x]] == True

In[28]:= (% /. x → x_) /. Equal → SetDelayed
```

Double Induction Theorem.

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → hypotheses[x], p2 → INDUCTIVE[intersection[omega, ub[x, omega]]],
            p3 → subclass[cart[omega, omega], x]}]]

Out[29]= or[not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[composite[SUCC, intersection[x, inverse[x]]], x]],
           subclass[cart[omega, omega], x]] == True

In[30]:= or[not[subclass[omega, image[inverse[x_], set[0]]]],
           not[subclass[composite[SUCC, intersection[inverse[x_], x_]], x_]],
           subclass[cart[omega, omega], x_]] := True
```

Corollary. The theorem with the original (unsimplified) hypotheses is obtained if one restricts **x** to natural numbers.

```
In[31]:= SubstTest[implies, hypotheses[y],
          subclass[cart[omega, omega], y], y → restrict[x, omega, omega]]

Out[31]= or[not[subclass[omega, image[inverse[x], set[0]]]],
           not[subclass[composite[id[omega], SUCC, intersection[x, inverse[x]], id[omega]], x]],
           subclass[cart[omega, omega], x]] == True
```