iteration stops or loops forever

Johan G. F. Belinfante
2006 May 25

```
In[1]:= SetDirectory["l:"]; << goedel81.21b; << tools.m

        :Package Title: goedel81.21b        2006 May 21 at 8:20 p.m.

        It is now:  2006 May 25 at 14:20

        Loading Simplification Rules

        TOOLS.M                      Revised 2006 May 8

        weightlimit = 40
```

summary

If one repeatedly applies a function, starting at some point, the same value could be encountered twice. If that happens, the iteration process goes into a loop and never ends. So, contrapositively, if an iteration process terminates after a finite number of steps, then all values obtained up to that point must be distinct. In other words, if **x** is a function for which the domain of **iterate[x, set[y]]** is finite, then the function **iterate[x, set[y]]** is one-to-one. This basic fact about iteration is derived in this notebook, although most of the time a more general situation is addressed. The requirement that **x** be a function will be dropped, and any starting class **y** will be permitted, not necessarily a singleton. The idea is that if one nonempty vertical section of **iterate[x,y]** is ever equal to a later one, then the same holds for the next pair of vertical sections, and so on, forever. Therefore, in this case, the domain of **iterate[x,y]** is **omega**.

the key observation

Lemma.

```
In[2]:= Assoc[iterate[x, y], id[omega], image[power[SUCC], set[z]]] // Reverse

Out[2]= composite[iterate[x, y], image[power[SUCC], set[z]]] == composite[iterate[x, y], plus[z]]

In[3]:= composite[iterate[x_, y_], image[power[SUCC], set[z_]]] :=
          composite[iterate[x, y], plus[z]]
```

This is the key observation: when two vertical sections of **iterate[x,y]** are equal, then all subsequent pairs of vertical sections are also equal.

```
In[4]:= SubstTest[implies,
         and[equal[composite[u, w], composite[w, SUCC]], equal[image[w, set[0]], v]],
         equal[composite[w, id[omega]], iterate[u, v]], {u → x, v → image[iterate[x, y], set[u]],
          w → composite[iterate[x, y], plus[v]]}]

Out[4]= or[equal[composite[iterate[x, y], plus[u]], composite[iterate[x, y], plus[v]]],
          not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]]] == True

In[5]:= or[equal[composite[iterate[x_, y_], plus[u_]], composite[iterate[x_, y_], plus[v_]]],
          not[equal[image[iterate[x_, y_], set[u_]], image[iterate[x_, y_], set[v_]]]]] := True
```

consequences for the domain

Lemma.

```
In[6]:= SubstTest[implies, equal[w, z], equal[domain[w], domain[z]],
         {w → composite[iterate[x, y], plus[u]], z → composite[iterate[x, y], plus[v]]}]

Out[6]= or[equal[image[inverse[plus[u]], domain[iterate[x, y]]],
           image[inverse[plus[v]], domain[iterate[x, y]]]], not[
           equal[composite[iterate[x, y], plus[u]], composite[iterate[x, y], plus[v]]]]] == True

In[7]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Combining this result with that of the preceding section, one finds:

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
          {p1 -> equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]],
           p2 -> equal[composite[iterate[x, y], plus[u]], composite[iterate[x, y], plus[v]]],
           p3 -> equal[image[inverse[plus[u]], domain[iterate[x, y]]],
             image[inverse[plus[v]], domain[iterate[x, y]]]]}]]

Out[8]= or[equal[image[inverse[plus[u]], domain[iterate[x, y]]],
           image[inverse[plus[v]], domain[iterate[x, y]]]],
          not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]]] == True

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

an inclusion

Lemma.

```
In[10]:= subclass[image[inverse[plus[x]], nat[y]], natsub[nat[y], x]] // AssertTest

Out[10]= subclass[image[inverse[plus[x]], nat[y]], natsub[nat[y], x]] == True

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

Removing the **nat** wrapper, one finds:

```
In[12]:= SubstTest[implies, equal[y, nat[z]],
          subclass[image[inverse[plus[x]], y], natsub[y, x]], z → y]

Out[12]= or[not[member[y, omega]], subclass[image[inverse[plus[x]], y], natsub[y, x]]] == True

In[13]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The same holds when **y** is not a natural number.

```
In[14]:= Map[or[#, member[y, omega]] &, SubstTest[implies, equal[v, V],
           subclass[u, v], {u -> image[inverse[plus[x]], y], v -> natsub[y, x]}]]

Out[14]= or[member[y, omega], subclass[image[inverse[plus[x]], y], natsub[y, x]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Combining these two cases, one finds this general inclusion:

```
In[16]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 -> member[y, omega],
            p2 -> subclass[image[inverse[plus[x]], y], natsub[y, x]]}] // Reverse

Out[16]= subclass[image[inverse[plus[x]], y], natsub[y, x]] == True

In[17]:= subclass[image[inverse[plus[x_]], y_], natsub[y_, x_]] := True
```

an equation

In this section it is shown that the inclusion derived in the preceding section can, under certain circumstances, be replaced with an equation. In the following lemma, the temporary variable **w** is deliberately not wrapped to facilitate its immediate removal.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[and[p1, p2], p4]],
           {p1 -> member[nat[x], nat[y]], p2 -> member[w, natsub[nat[y], nat[x]]],
            p3 → not[member[nat[y], nat[x]]], p4 -> member[natadd[w, nat[x]], nat[y]]}]]

Out[18]= or[member[natadd[w, nat[x]], nat[y]],
           not[member[w, natsub[nat[y], nat[x]]]], not[member[nat[x], nat[y]]]] == True

In[19]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Removing the variable **w** yields an inclusion in the opposite direction of that considered in the preceding section.

```
In[20]:= Map[equal[V, #] &, SubstTest[class, w,
            implies[and[member[nat[x], t], member[w, u]], member[w, v]], {t → nat[y],
             u -> natsub[nat[y], nat[x]], v -> image[inverse[plus[nat[x]]], nat[y]]}]] // Reverse

Out[20]= or[not[member[nat[x], nat[y]]],
           subclass[natsub[nat[y], nat[x]], image[inverse[plus[nat[x]]], nat[y]]]] == True

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

Combing this inclusion with that obtained in the preceding section yields an equation.

```
In[22]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u], {p -> member[nat[x], nat[y]],
            u -> natsub[nat[y], nat[x]], v -> image[inverse[plus[nat[x]]], nat[y]]}] // Reverse

Out[22]= or[equal[image[inverse[plus[nat[x]]], nat[y]], natsub[nat[y], nat[x]]],
           not[member[nat[x], nat[y]]]] == True

In[23]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

derivation

Lemma. A class equal to a nonempty class is not empty. Vertical sections are nonempty at points of the domain. Therefore if two vertical sections are equal, and one is at a point in the domain, so is the other.

```
In[24]:= SubstTest[implies, and[not[empty[s]], equal[s, t]],
          not[empty[t]], {s -> image[x, set[u]], t -> image[x, set[v]]}]

Out[24]= or[member[v, domain[x]],
           not[equal[image[x, set[u]], image[x, set[v]]]], not[member[u, domain[x]]]] == True

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

Suppose **domain[iterate[x,y]]** is a natural number, say **nat[w]**, and suppose **iterate[x,y]** has equal vertical sections at two different members **nat[u]** and **nat[v]** of its domain. One may as well assume **nat[u]** is less than **nat[v]**. One then has:

```
In[26]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p4, p5],
           implies[p3, p6], implies[and[p5, p6], p7], not[and[p1, p2, p7]],
           not[implies[and[p1, p2], not[p4]]], {p1 → member[nat[u], nat[v]],
            p2 → member[nat[v], nat[w]], p3 → member[nat[u], nat[w]],
            p4 → equal[image[inverse[plus[nat[u]]], nat[w]],
              image[inverse[plus[nat[v]]], nat[w]]],
            p5 → subclass[image[inverse[plus[nat[u]]], nat[w]], natsub[nat[w], nat[v]]],
            p6 → equal[image[inverse[plus[nat[u]]], nat[w]], natsub[nat[w], nat[u]]],
            p7 -> subclass[natsub[nat[w], nat[u]], natsub[nat[w], nat[v]]]}]]

Out[26]= or[not[
            equal[image[inverse[plus[nat[u]]], nat[w]], image[inverse[plus[nat[v]]], nat[w]]]],
           not[member[nat[u], nat[v]]], not[member[nat[v], nat[w]]]] == True

In[27]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

Remove all wrappers.

```
In[28]:= SubstTest[implies, and[equal[u, nat[x]], equal[v, nat[y]], equal[w, nat[z]]],
           or[not[equal[image[inverse[plus[u]], w], image[inverse[plus[v]], w]]],
            not[member[u, v]], not[member[v, w]]],
           {x → u, y → v, z → w}] // MapNotNot

Out[28]= or[not[equal[image[inverse[plus[u]], w], image[inverse[plus[v]], w]]],
           not[member[u, omega]], not[member[u, v]],
           not[member[v, w]], not[member[w, omega]]] == True

In[29]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

This can be cleaned up further.

```
In[30]:= Map[not, SubstTest[and, implies[and[p2, p3], p4], implies[and[p1, p4], p5],
           implies[and[p1, p2, p3, p5], p6], not[implies[and[p1, p2, p3], p6]], {p1 → member[u, v],
            p2 → member[v, w], p3 → member[w, omega], p4 → member[v, omega], p5 → member[u, omega],
            p6 -> not[equal[image[inverse[plus[u]], w], image[inverse[plus[v]], w]]]}]]

Out[30]= or[not[equal[image[inverse[plus[u]], w], image[inverse[plus[v]], w]]],
           not[member[u, v]], not[member[v, w]], not[member[w, omega]]] == True

In[31]:= (% /. {u → u_, v → v_, w → w_}) /. Equal → SetDelayed
```

Theorem. If the domain of **iterate[x,y]** is a natural number, then at any two different points of its domain, its vertical sections are not equal.

```
In[32]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[p4, p5],
           not[implies[and[p1, p2, p3], p5]], {p1 → member[domain[iterate[x, y]], omega],
            p2 → member[u, v], p3 → member[v, domain[iterate[x, y]]],
            p4 → not[equal[image[inverse[plus[u]], domain[iterate[x, y]]],
               image[inverse[plus[v]], domain[iterate[x, y]]]]],
            p5 → not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]]}]]

Out[32]= or[not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]],
           not[member[u, v]], not[member[v, domain[iterate[x, y]]]],
           not[member[domain[iterate[x, y]], omega]]] == True

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

If **u** and **v** are natural numbers, and neither one is less than the other, then they are equal. So the condition that **u** is less than **v** in the above theorem can be replaced with the condition that they are not equal. (This step takes quite a while. Be patient.)

```
In[34]:= Map[not, SubstTest[and, implies[and[p2, p3], p4],
           implies[and[p1, p2, p3], p5], implies[and[p1, p2, p4], p6],
           implies[p3, p8], implies[p4, p7], implies[and[p5, p6, p7, p8], p9],
           not[implies[and[p1, p2, p3], p9]], {p1 -> member[domain[iterate[x, y]], omega],
            p2 -> equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]],
            p3 -> member[v, domain[iterate[x, y]]], p4 → member[u, domain[iterate[x, y]]],
            p5 → not[member[u, v]], p6 → not[member[v, u]],
            p7 → member[u, omega], p8 → member[v, omega], p9 → equal[u, v]}]]

Out[34]= or[equal[u, v], not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]],
           not[member[v, domain[iterate[x, y]]]],
           not[member[domain[iterate[x, y]], omega]]] == True

In[35]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary. If **iterate[x,y]** has equal vertical sections at two different points of its domain, then the domain is **omega**.

```
In[36]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4],
           implies[p4, p5], not[implies[and[p1, p2, p3], p5]],
           {p1 → equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]],
            p2 -> member[v, domain[iterate[x, y]]], p3 → not[equal[u, v]],
            p4 -> not[member[domain[iterate[x, y]], omega]],
            p5 → equal[domain[iterate[x, y]], omega]}]]

Out[36]= or[equal[omega, domain[iterate[x, y]]], equal[u, v],
           not[equal[image[iterate[x, y], set[u]], image[iterate[x, y], set[v]]]],
           not[member[v, domain[iterate[x, y]]]]] == True

In[37]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

The variables **u** and **v** are now removed for the special case that **x** is a function and **y** is a singleton.

```
In[38]:= (Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[u, v],
              or[equal[u, v], not[equal[image[z, set[u]], image[z, set[v]]]],
               not[member[v, domain[z]]], equal[domain[z], w]],
              {w → omega, z → iterate[s, t]}]] // Reverse) /. {s → funpart[x], t → set[y]}

Out[38]= or[FUNCTION[inverse[iterate[funpart[x], set[y]]]],
           subclass[omega, domain[iterate[funpart[x], set[y]]]]] == True

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

Main theorem:

```
In[40]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[domain[iterate[funpart[x], set[y]]], omega],
            p2 → not[subclass[omega, domain[iterate[funpart[x], set[y]]]]],
            p3 -> FUNCTION[inverse[iterate[funpart[x], set[y]]]]}]]

Out[40]= or[FUNCTION[inverse[iterate[funpart[x], set[y]]]],
           not[member[domain[iterate[funpart[x], set[y]]], omega]]] == True
```

```
In[41]:= or[FUNCTION[inverse[iterate[funpart[x_], set[y_]]]],
           not[member[domain[iterate[funpart[x_], set[y_]]], omega]]] := True
```

Corollary.

```
In[42]:= SubstTest[implies,
          and[equal[x, funpart[z]], member[domain[iterate[x, set[y]]], omega]],
          FUNCTION[inverse[iterate[x, set[y]]]]], z → x]

Out[42]= or[FUNCTION[inverse[iterate[x, set[y]]]],
           not[FUNCTION[x]], not[member[domain[iterate[x, set[y]]], omega]]] == True

In[43]:= or[FUNCTION[inverse[iterate[x_, set[y_]]]], not[FUNCTION[x_]],
           not[member[domain[iterate[x_, set[y_]]], omega]]] := True
```