increasing mappings

Johan G. F. Belinfante
2010 October 27

```
In[1]:= SetDirectory["l:"]; << goedel.10oct26a

        :Package Title: goedel.10oct26a            2010 October 26 at 2:40 p.m.

        It is now:  2010 Oct 27 at 14:10

        Loading Simplification Rules

        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

        weightlimit = 40
```

summary

If a function $\mathbf{f} \subset \Omega \times \Omega$ subcommutes with the membership relation $\mathbf{E}$, then $\mathbf{f} \subset \mathbf{S}$.

strict monotonicity

Theorem. If a function $\mathbf{x}$ subcommutes with the membership relation, then $\mathbf{x} \subset \mathbf{S} \circ \mathbf{IMAGE[x]}$.

```
In[2]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], not[implies[and[p1, p2], p4]],
           {p1 → subcommute[x, E], p2 → FUNCTION[x], p3 → subclass[composite[x, E, inverse[x]], E],
            p4 → subclass[x, composite[S, IMAGE[x]]]}]] // Reverse

Out[2]= or[not[FUNCTION[x]], not[subclass[composite[x, E], composite[E, x]]],
          subclass[x, composite[S, IMAGE[x]]]] == True

In[3]:= or[not[FUNCTION[x_]], not[subclass[composite[x_, E], composite[E, x_]]],
          subclass[x_, composite[S, IMAGE[x_]]]] := True
```

Theorem. An **APPLY** rule for the monotonicity property.

```
In[4]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p2, p3, p4, p5], p6],
           not[implies[and[p1, p2, p4, p5], p6]], {p1 → subcommute[t, E], p2 → FUNCTION[t],
            p3 -> subclass[t, composite[S, IMAGE[t]]], p4 → member[x, domain[t]],
            p5 → member[x, y], p6 -> member[APPLY[t, x], APPLY[t, y]]}]] // Reverse

Out[4]= or[member[APPLY[t, x], APPLY[t, y]], not[FUNCTION[t]], not[member[x, y]],
          not[member[x, domain[t]]], not[subclass[composite[t, E], composite[E, t]]]] == True

In[5]:= or[member[APPLY[t_, x_], APPLY[t_, y_]], not[FUNCTION[t_]], not[member[x_, y_]],
          not[member[x_, domain[t_]]], not[subclass[composite[t_, E], composite[E, t_]]]] := True
```

Because a function that subcommutes with **E** has a full domain, one can replace the condition **x ∈ domain[t]** with **y ∈ domain[t]**.

Corollary. An alternate **APPLY** rule for monotonicity.

```
In[6]:= Map[not, SubstTest[and, implies[p1, p5],
           implies[and[p3, p4, p5], p6], implies[and[p1, p2, p3, p6], p7],
           not[implies[and[p1, p2, p3, p4], p7]], {p1 → subcommute[t, E], p2 → FUNCTION[t],
            p3 → member[x, y], p4 → member[y, domain[t]], p5 → full[domain[t]],
            p6 -> member[x, domain[t]], p7 -> member[APPLY[t, x], APPLY[t, y]]}]] // Reverse

Out[6]= or[member[APPLY[t, x], APPLY[t, y]], not[FUNCTION[t]], not[member[x, y]],
          not[member[y, domain[t]]], not[subclass[composite[t, E], composite[E, t]]]] == True

In[7]:= or[member[APPLY[t_, x_], APPLY[t_, y_]],
          not[FUNCTION[t_]], not[member[x_, y_]], not[member[y_, domain[t_]]],
          not[subclass[composite[t_, E_], composite[E_, t_]]]] := True
```

---

the inclusion f ⊂ S

The main idea is to show that **fix[E ∘ f] ⊂ Ω** is empty by applying the well-ordering principle.

Lemma. If **fix[E ∘ f]** is not empty, then it has a least member.

```
In[8]:= SubstTest[implies, subclass[t, OMEGA],
           or[empty[t], member[A[t], t]], t → fix[composite[E, f]]] // Reverse // MapNotNot

Out[8]= or[equal[0, fix[composite[E, f]]],
          member[A[fix[composite[E, f]]], image[inverse[f], A[fix[composite[E, f]]]]],
          not[subclass[fix[composite[E, f]], OMEGA]]] == True

In[9]:= (% /. f → f_) /. Equal→ SetDelayed
```

Note that in the above lemma, the statement **t ∈ fix[E ∘ f]** was automatically rewritten as follows:

```
In[10]:= member[t, fix[composite[E, f]]]

Out[10]= member[t, image[inverse[f], t]]
```

When **f** is a function, this statement is equivalent to **APPLY[f, t] ∈ t**.

Theorem. If **fix[E ∘ f]** is not empty for a function **f ⊂ Ω × Ω**, then the least member **t ∈ fix[E ∘ f]** satisfies **APPLY[f, t] ∈ t**.

```
In[11]:= Map[not,
           SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], implies[and[p1, p3], p4],
            not[implies[p1, p4]], {p1 → and[FUNCTION[f], subclass[f, cart[OMEGA, OMEGA]],
               not[empty[fix[composite[E, f]]]]], p2 -> subclass[fix[composite[E, f]], OMEGA],
             p3 -> member[A[fix[composite[E, f]]], image[inverse[f], A[fix[composite[E, f]]]]],
             p4 -> member[APPLY[f, A[fix[composite[E, f]]]],
               A[fix[composite[E, f]]]]}]] // Reverse

Out[11]= or[equal[0, fix[composite[E, f]]],
           member[APPLY[f, A[fix[composite[E, f]]]], A[fix[composite[E, f]]]],
           not[FUNCTION[f]], not[subclass[f, cart[OMEGA, OMEGA]]]] == True

In[12]:= (% /. f → f_) /. Equal→ SetDelayed
```

Lemma. No element of **fix[E ∘ f]** is less than its least member (if there is one).

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → subclass[f, cartsq[OMEGA]], p2 → subclass[fix[composite[E, f]], OMEGA],
             p3 → disjoint[A[fix[composite[E, f]]], fix[composite[E, f]]]}]] // Reverse

Out[13]= or[equal[0, intersection[A[fix[composite[E, f]]], fix[composite[E, f]]]],
           not[subclass[f, cart[OMEGA, OMEGA]]]] == True

In[14]:= (% /. f → f_) /. Equal→ SetDelayed
```

Lemma.

```
In[15]:= SubstTest[implies, empty[w], not[member[t, w]],
           w -> intersection[A[fix[composite[E, f]]], fix[composite[E, f]]]] // Reverse

Out[15]= or[not[equal[0, intersection[A[fix[composite[E, f]]], fix[composite[E, f]]]]],
           not[member[t, A[fix[composite[E, f]]]], not[member[t, image[inverse[f], t]]]] == True

In[16]:= (% /. {f → f_, t → t_}) /. Equal→ SetDelayed
```

Theorem. If **APPLY[f, t] ∈ t** for a function **f ⊂ Ω × Ω**, then **t** is not less than the least member of **fix[E ∘ f]**.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p2, p3], p4],
            implies[and[p1, p4], p5], not[implies[and[p1, p3], p5]],
            {p1 → and[FUNCTION[f], subclass[f, cart[OMEGA, OMEGA]]],
             p2 -> equal[0, intersection[A[fix[composite[E, f]]], fix[composite[E, f]]]],
             p3 -> member[t, A[fix[composite[E, f]]]], p4 -> not[member[t, image[inverse[f], t]]],
             p5 → not[member[APPLY[f, t], t]]}]] // Reverse

Out[17]= or[not[FUNCTION[f]], not[member[t, A[fix[composite[E, f]]]]],
           not[member[APPLY[f, t], t]], not[subclass[f, cart[OMEGA, OMEGA]]]] == True

In[18]:= (% /. {f → f_, t → t_}) /. Equal→ SetDelayed
```

Theorem. If a function **f ⊂ Ω × Ω** subcommutes with the membership relation **E**, then **fix[E ∘ f] = 0**.

```
In[19]:= (Map[not, SubstTest[and, implies[and[p0, p1], p2],
              implies[and[p0, p2], p3], implies[and[p1, p2, p3], p4],
              implies[and[p0, p1, p2], not[p4]], p0, p1, {p0 → equal[t, A[fix[composite[E, f]]]],
               p1 → and[FUNCTION[f], subclass[f, cart[OMEGA, OMEGA]], subcommute[f, E],
                 not[empty[fix[composite[E, f]]]]], p2 -> member[APPLY[f, t], t],
               p3 → member[t, domain[f]], p4 → member[APPLY[f, APPLY[f, t]], APPLY[f, t]]}]] //
            Reverse) /. t -> A[fix[composite[E, f]]]

Out[19]= or[equal[0, fix[composite[E, f]]],
           not[FUNCTION[f]], not[subclass[f, cart[OMEGA, OMEGA]]],
           not[subclass[composite[f, E], composite[E, f]]]] == True

In[20]:= (% /. f → f_) /. Equal→ SetDelayed
```

The condition **fix[E ∘ f] = 0** is equivalent to **fix[f ∘ E] = 0**.

Corollary.

```
In[21]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 → and[FUNCTION[f], subclass[f, cart[OMEGA, OMEGA]], subcommute[f, E]], p2 ->
              equal[0, fix[composite[E, f]]], p3 -> equal[0, fix[composite[f, E]]]}]] // Reverse

Out[21]= or[equal[0, fix[composite[f, E]]],
           not[FUNCTION[f]], not[subclass[f, cart[OMEGA, OMEGA]]],
           not[subclass[composite[f, E], composite[E, f]]]] == True

In[22]:= (% /. f → f_) /. Equal→ SetDelayed
```

Main Theorem. If a function **f ⊂ Ω × Ω** subcommutes with the membership relation **E**, then **f ⊂ S**.

```
In[23]:= Map[implies[and[FUNCTION[f], subclass[f, cart[OMEGA, OMEGA]], subcommute[f, E]], #] &,
            SubstTest[subclass, f, intersection[u, v],
             {u → cartsq[OMEGA], v → complement[inverse[E]]}]] // Reverse // MapNotNot

Out[23]= or[not[FUNCTION[f]], not[subclass[f, cart[OMEGA, OMEGA]]],
           not[subclass[composite[f, E], composite[E, f]]], subclass[f, S]] == True

In[24]:= or[not[FUNCTION[f_]], not[subclass[f_, cart[OMEGA, OMEGA]]],
           not[subclass[composite[f_, E], composite[E, f_]]], subclass[f_, S]] := True
```

two cases

A function **f** subcommutes with **E** if and only if **P[f] ⊂ monotone[E, E]** and its domain is full. This yields the following corollary of the main theorem derived in the preceding section.

Corollary. If a strictly monotone function **f ⊂ Ω × Ω** has a full domain, then **f ⊂ S**.

```
In[25]:= Map[not, SubstTest[and, implies[and[p1, p3], p5],
           implies[and[p1, p4, p5], p6], implies[and[p1, p2, p6], p7],
           not[implies[and[p1, p2, p3, p4], p7]], {p1 → FUNCTION[f],
            p2 → subclass[f, cart[OMEGA, OMEGA]], p3 → subclass[f, composite[S, IMAGE[f]]],
            p4 → full[domain[f]], p5 → subclass[composite[f, E, inverse[f]], E],
            p6 → subcommute[f, E], p7 → subclass[f, S]}]] // Reverse

Out[25]= or[not[FUNCTION[f]], not[subclass[f, cart[OMEGA, OMEGA]]],
           not[subclass[f, composite[S, IMAGE[f]]]],
           not[subclass[U[domain[f]], domain[f]]], subclass[f, S]] == True

In[26]:= or[not[FUNCTION[f_]], not[subclass[f_, cart[OMEGA, OMEGA]]],
           not[subclass[f_, composite[S, IMAGE[f_]]]],
           not[subclass[U[domain[f_]], domain[f_]]], subclass[f_, S]] := True
```

There are now two cases: a full subset of $\mathbf{\Omega}$ is either an ordinal or the class $\mathbf{\Omega}$ of all ordinals. Consider first the case **domain[f] = Ω**.

Lemma.

```
In[27]:= implies[and[equal[domain[f], x], subclass[range[f], y], subclass[f, cart[V, V]]],
           subclass[f, cart[x, y]]] // AssertTest

Out[27]= or[not[equal[x, domain[f]]],
           not[subclass[f, cart[V, y]]], subclass[f, cart[x, y]]] == True

In[28]:= (% /. {f → f_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[29]:= or[not[equal[OMEGA, domain[f]]], not[subclass[f, cart[V, V]]],
           not[subclass[range[f], OMEGA]], subclass[f, cart[OMEGA, OMEGA]]] // NotNotTest

Out[29]= or[not[equal[OMEGA, domain[f]]], not[subclass[f, cart[V, V]]],
           not[subclass[range[f], OMEGA]], subclass[f, cart[OMEGA, OMEGA]]] == True

In[30]:= (% /. {f → f_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (The case that **domain[f] = Ω**.)

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p1, p2, p3], p4], not[implies[p1, p5]],
           {p1 → and[FUNCTION[f], equal[domain[f], OMEGA], subclass[range[f], OMEGA], subclass[
               f, composite[S, IMAGE[f]]]], p2 → subclass[f, cart[V, V]], p3 → full[domain[f]],
            p4 -> subclass[f, cart[OMEGA, OMEGA]], p5 -> subclass[f, S]}]] // Reverse

Out[31]= or[not[equal[OMEGA, domain[f]]],
           not[FUNCTION[f]], not[subclass[f, composite[S, IMAGE[f]]]],
           not[subclass[range[f], OMEGA]], subclass[f, S]] == True

In[32]:= or[not[equal[OMEGA, domain[f_]]],
           not[FUNCTION[f_]], not[subclass[f_, composite[S, IMAGE[f_]]]],
           not[subclass[range[f_], OMEGA]], subclass[f_, S]] := True
```

Next consider the case that the domain of **f** is an ordinal.

Theorem. (The case that **domain[f] ∈ Ω**.)

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p1, p2, p3], p4], implies[and[p1, p3, p4], p5],
           not[implies[p1, p5]], {p1 → and[FUNCTION[f], member[domain[f], OMEGA],
              subclass[range[f], OMEGA], subclass[f, composite[S, IMAGE[f]]]],
            p2 → subclass[domain[f], OMEGA], p3 → full[domain[f]],
            p4 -> subclass[f, cart[OMEGA, OMEGA]], p5 -> subclass[f, S]}]] // Reverse

Out[33]= or[not[FUNCTION[f]], not[member[domain[f], OMEGA]],
           not[subclass[f, composite[S, IMAGE[f]]]],
           not[subclass[range[f], OMEGA]], subclass[f, S]] == True

In[34]:= or[not[FUNCTION[f_]], not[member[domain[f_], OMEGA]],
           not[subclass[f_, composite[S, IMAGE[f_]]]],
           not[subclass[range[f_], OMEGA]], subclass[f_, S]] := True
```