# increasing functions x: $\omega \to \omega$

Johan G. F. Belinfante
2010 April 25

```
In[1]:= SetDirectory["l:"]; << goedel.10apr24a;<< tools.m

        :Package Title: goedel.10apr24a              2010 April 24 at 9:25 p.m.

        It is now:  2010 Apr 25 at 6:48

        Loading Simplification Rules

        TOOLS.M                                      Revised 2010 February 26

        weightlimit = 40
```

## summary

It is shown in this notebook that if a mapping **x:** $\omega \to \omega$ satisfies **inverse[x]** ∘ **S** ∘ **x** ⊂ **S**, then **x** ⊂ **S**. The main idea is to show that the hypotheses imply that **fix[E** ∘ **x]** is empty. The latter is established by showing that this is a subset of $\omega$ which has no least member. A long series of lemmas is used to derive this, most of which have the form **a ⇒ c, b ⇒ not[c], ∴ not[a & b]**.

## derivation

Lemma. No member of a subset of $\omega$ is less than the least member.

```
In[2]:= SubstTest[implies, and[subclass[t, omega], member[w, A[t]]],
          not[member[w, t]], t -> fix[composite[E, x]]] // Reverse

Out[2]= or[not[member[w, A[fix[composite[E, x]]]]], not[member[w, image[inverse[x], w]]],
          not[subclass[fix[composite[E, x]], omega]]] == True

In[3]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Lemma. Reverse monotonicity of increasing mappings.

```
In[4]:= Map[not, SubstTest[and, implies[and[p1, p2, p4, p6], p7],
          implies[and[p5, p3], not[p7]], p1, p2, p3, p4, p5, p6,
          {p1 → member[x, map[omega, omega]],  p2 → subclass[composite[inverse[x], S, x], S] ,
           p3 → member[v, omega], p4 → member[APPLY[x, v], omega], p5 -> member[APPLY[x, v], v],
           p6 → subclass[APPLY[x, v], APPLY[x, APPLY[x, v]]], p7 → subclass[v, APPLY[x, v]]}]] //
        Reverse

Out[4]= or[not[member[v, omega]], not[member[x, map[omega, omega]]],
          not[member[APPLY[x, v], v]], not[subclass[APPLY[x, v], APPLY[x, APPLY[x, v]]]],
          not[subclass[composite[inverse[x], S, x], S]]] == True

In[5]:= (% /. {v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[6]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p5], not[p8]],
          implies[and[p4, p6, p7], p8], p1, p2, p3, p5, p6, p7,
          {p1 → member[x, map[omega, omega]], p2 → subclass[composite[inverse[x], S, x], S],
           p3 → member[v, omega], p4 → member[APPLY[x, v], omega],
           p5 -> member[APPLY[x, v], v], p6 → not[member[APPLY[x, APPLY[x, v]], APPLY[x, v]]],
           p7 → member[APPLY[x, APPLY[x, v]], omega],
           p8 → subclass[APPLY[x, v], APPLY[x, APPLY[x, v]]]}]] // Reverse

Out[6]= or[member[APPLY[x, APPLY[x, v]], APPLY[x, v]],
          not[member[v, omega]], not[member[x, map[omega, omega]]],
          not[member[APPLY[x, v], v]], not[member[APPLY[x, APPLY[x, v]], omega]],
          not[subclass[composite[inverse[x], S, x], S]]] == True

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

Lemma.  Increasing mappings are strictly monotone.

```
In[8]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p5, p6], not[p7]],
          implies[and[p1, p4], p7], p1, p2, p3, p4, p5, p6,
          {p1 → member[x, map[omega, omega]], p2 → subclass[composite[inverse[x], S, x], S] ,
           p3 → member[v, omega], p4 → member[APPLY[x, v], omega],
           p5 -> member[APPLY[x, v], v], p6 → not[member[APPLY[x, APPLY[x, v]], APPLY[x, v]]],
           p7 → member[APPLY[x, APPLY[x, v]], omega]}]] // Reverse

Out[8]= or[member[APPLY[x, APPLY[x, v]], APPLY[x, v]], not[member[v, omega]],
          not[member[x, map[omega, omega]]], not[member[APPLY[x, v], v]],
          not[subclass[composite[inverse[x], S, x], S]]] == True

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

Lemma.

```
In[10]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p4], not[p6]],
            implies[and[p1, p5], p6], p1, p2, p3, p4, p5,{p1 → member[x, map[omega, omega]],
             p2 → subclass[composite[inverse[x], S, x], S], p3 → member[v, omega],
             p4 -> member[APPLY[x, v], v], p5 → not[member[APPLY[x, v], fix[composite[E, x]]]],
             p6 → not[member[APPLY[x, APPLY[x, v]], APPLY[x, v]]]}]] // Reverse

Out[10]= or[member[APPLY[x, v], image[inverse[x], APPLY[x, v]]],
           not[member[v, omega]], not[member[x, map[omega, omega]]],
           not[member[APPLY[x, v], v]], not[subclass[composite[inverse[x], S, x], S]]] == True

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

Lemma.

```
In[12]:= Map[not, SubstTest[and, not[and[p1, p2, p4, p5, p6]], implies[and[p0, p3, p5], p6],
            p0, p1, p2, p3, p4, p5,{p0 → equal[v, A[fix[composite[E, x]]]],
             p1 → member[x, map[omega, omega]], p2 → subclass[composite[inverse[x], S, x], S],
             p3 → subclass[fix[composite[E, x]], omega],
             p4 → member[v, omega], p5 -> member[APPLY[x, v], v],
             p6 → not[member[APPLY[x, v], fix[composite[E, x]]]]}]] // Reverse

Out[12]= or[not[equal[v, A[fix[composite[E, x]]]]],
           not[member[v, omega]], not[member[x, map[omega, omega]]],
           not[member[APPLY[x, v], v]], not[subclass[composite[inverse[x], S, x], S]],
           not[subclass[fix[composite[E, x]], omega]]] == True

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

Lemma.

```
In[14]:= Map[not, SubstTest[and, not[and[p0, p1, p2, p3, p5, p6]], implies[and[p1, p4], p6],
            p0, p1, p2, p3, p4, p5,{p0 → equal[v, A[fix[composite[E, x]]]],
             p1 → member[x, map[omega, omega]], p2 → subclass[composite[inverse[x], S, x], S],
             p3 → subclass[fix[composite[E, x]], omega], p4 → member[v, fix[composite[E, x]]],
             p5 → member[v, omega], p6 -> member[APPLY[x, v], v]}]] // Reverse

Out[14]= or[not[equal[v, A[fix[composite[E, x]]]]],
           not[member[v, omega]], not[member[v, image[inverse[x], v]]],
           not[member[x, map[omega, omega]]], not[subclass[composite[inverse[x], S, x], S]],
           not[subclass[fix[composite[E, x]], omega]]] == True

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

Lemma.

```
In[16]:= Map[not, SubstTest[and, not[and[p0, p1, p2, p3, p4, p5]], implies[and[p3, p4], p5],
            p0, p1, p2, p3, p4, {p0 → equal[v, A[fix[composite[E, x]]]],
             p1 → member[x, map[omega, omega]],  p2 → subclass[composite[inverse[x], S, x], S] ,
             p3 → subclass[fix[composite[E, x]], omega],
             p4 → member[v, fix[composite[E, x]]], p5 → member[v, omega]}]] // Reverse

Out[16]= or[not[equal[v, A[fix[composite[E, x]]]]], not[member[v, image[inverse[x], v]]],
           not[member[x, map[omega, omega]]], not[subclass[composite[inverse[x], S, x], S]],
           not[subclass[fix[composite[E, x]], omega]]] == True

In[17]:=  (% /. {v → v_, x → x_}) /. Equal → SetDelayed
```

The well-ordering principle says that any non-empty subset of $\omega$ has a least member.

Lemma.  (Special case of the well-ordering principle.)

```
In[18]:= SubstTest[implies, and[not[empty[t]], subclass[t, omega]],
           member[A[t], t], t -> fix[composite[E, x]]] // Reverse

Out[18]= or[equal[0, fix[composite[E, x]]],
           member[A[fix[composite[E, x]]], image[inverse[x], A[fix[composite[E, x]]]]],
           not[subclass[fix[composite[E, x]], omega]]] == True

In[19]:=  (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[20]:= Map[not, SubstTest[and, not[and[p0, p1, p2, p3, p4]],
             implies[p1, p3], implies[and[p0, p3], or[p4, p5]] ,
             not[implies[and[p0, p1, p2], p5]], {p0 → equal[v, A[fix[composite[E, x]]]],
              p1 → member[x, map[omega, omega]],  p2 → subclass[composite[inverse[x], S, x], S] ,
              p3 → subclass[fix[composite[E, x]], omega], p4 → member[v, fix[composite[E, x]]],
              p5 → empty[fix[composite[E, x]]]}] /. v -> A[fix[composite[E, x]]]] // Reverse

Out[20]= or[equal[0, fix[composite[E, x]]], not[member[x, map[omega, omega]]],
           not[subclass[composite[inverse[x], S, x], S]]] == True

In[21]:=  (% /. x → x_) /. Equal → SetDelayed
```

Theorem.  If **x:** $\omega \rightarrow \omega$ is increasing, then **x** ⊂ **S**.

```
In[22]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
            {p1 → and[member[x, map[omega, omega]], subclass[composite[inverse[x], S, x], S]],
             p2 -> equal[0, fix[composite[E, x]]], p3 → subclass[x, S]}]] // Reverse

Out[22]= or[not[member[x, map[omega, omega]]],
           not[subclass[composite[inverse[x], S, x], S]], subclass[x, S]] == True

In[23]:= or[not[member[x_, map[omega, omega]]],
           not[subclass[composite[inverse[x_], S, x_], S]], subclass[x_, S]] := True
```