increasing functions

Johan G. F. Belinfante
2010 April 5

```
In[1]:= << goedel.10apr01a; << tools.m

:Package Title: goedel.10apr01a                2010 April 1 at 10:05 p.m.

It is now:  2010 Apr 9 at 9:37

Loading Simplification Rules

TOOLS.M                                      Revised 2010 February 26

weightlimit = 40
```

summary

It is shown in this notebook that an increasing function **t** from one totally ordering **x** to another **y** is monotone. Here an increasing function is defined by the inclusion **inverse[t]** ∘ **y** ∘ **t** ⊂ **x**, while monotonicity means **t** ∘ **x** ∘ **inverse[t]** ⊂ **y**. To take advantage of available rewrite rules, the variable **t** is wrapped with **funpart** while the variables **x** and **y** are both wrapped with **to**. The main challenge in the derivation presented here involves the elimination of variables introduced to allow the use of function application. In the final section the special case of mappings from the set of natural numbers to itself is considered.

reference

The concept of **increasing function** was suggested by a slightly more special case considered on page 105 of the following reference:

```
In[2]:= "Karel Hrbacek and Thomas Jech, Introduction to
           Set Theory, Third edition, Marcel Dekker Inc., New York, 1999.";
```

a general result

Theorem.

```
In[3]:= SubstTest[implies, and[member[t, u], subclass[u, z]],
          member[t, z], {t → APPLY[funpart[x], y], u → range[funpart[x]]}] // Reverse

Out[3]= or[member[APPLY[funpart[x], y], z],
          not[member[y, domain[funpart[x]]]], not[subclass[range[funpart[x]], z]]] == True

In[4]:= or[member[APPLY[funpart[x_], y_], z_],
          not[member[y_, domain[funpart[x_]]]], not[subclass[range[funpart[x_]], z_]]] := True
```

derivation

In this section, two auxilliary variables **u** and **v** are introduced. These will be eliminated in the next section.

Lemma.

In[5]:=
```
Map[implies[#, or[member[pair[u, v], to[x]],
      not[subclass[composite[inverse[funpart[t]], y, funpart[t]], to[x]]]]] &,
  member[pair[u, v], composite[inverse[funpart[t]], y, funpart[t]]] // AssertTest] // Reverse
```

Out[5]=
```
or[member[pair[u, v], to[x]],
  not[member[u, domain[funpart[t]]]], not[member[v, domain[funpart[t]]]],
  not[member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], y]],
  not[subclass[composite[inverse[funpart[t]], y, funpart[t]], to[x]]] == True
```

In[6]:=
```
(% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

In[7]:=
```
Map[not, SubstTest[and, implies[and[p1, p5], p7], implies[and[p1, p6], p8],
   not[implies[and[p1, p3, p4, p13], p11]], {p1 → equal[domain[funpart[t]], fix[to[x]]],
    p3 → subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]],
    p4 → member[pair[u, v], to[x]], p5 → member[u, fix[to[x]]], p6 → member[v, fix[to[x]]],
    p7 → member[u, domain[funpart[t]]], p8 → member[v, domain[funpart[t]]],
    p11 → member[pair[v, u], to[x]], p13 →
     member[pair[APPLY[funpart[t], v], APPLY[funpart[t], u]], to[y]]}]] // Reverse
```

Out[7]=
```
or[member[pair[v, u], to[x]],
  not[equal[domain[funpart[t]], fix[to[x]]]], not[member[pair[u, v], to[x]]],
  not[member[pair[APPLY[funpart[t], v], APPLY[funpart[t], u]], to[y]]],
  not[subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]]] == True
```

In[8]:=
```
(% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

In[9]:=
```
or[member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], to[y]],
  not[member[pair[u, v], to[x]]], not[member[pair[v, u], to[x]]],
  not[member[pair[APPLY[funpart[t], v], APPLY[funpart[t], u]], to[y]]]] // NotNotTest
```

Out[9]=
```
or[member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], to[y]],
  not[member[pair[u, v], to[x]]], not[member[pair[v, u], to[x]]],
  not[member[pair[APPLY[funpart[t], v], APPLY[funpart[t], u]], to[y]]] == True
```

In[10]:=
```
(% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Even with these lemmas, the derivation of the main theorem takes about 84 seconds.

Main Theorem.

In[11]:=
```
Map[not, SubstTest[and, implies[and[p2, p7], p9],
    implies[and[p2, p8], p10], implies[and[p1, p3, p4, p13], p11],
    implies[and[p4, p11, p13], p14], not[implies[and[p1, p2, p3, p4], p14]],
    {p1 → equal[domain[funpart[t]], fix[to[x]]], p2 → subclass[range[funpart[t]], fix[to[y]]],
     p3 → subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]],
     p4 → member[pair[u, v], to[x]], p7 → member[u, domain[funpart[t]]],
     p8 → member[v, domain[funpart[t]]], p9 → member[APPLY[funpart[t], u], fix[to[y]]],
     p10 → member[APPLY[funpart[t], v], fix[to[y]]],
     p11 → member[pair[v, u], to[x]], p13 →
      member[pair[APPLY[funpart[t], v], APPLY[funpart[t], u]], to[y]],
     p14 → member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], to[y]]}]] // MapNotNot //
 Reverse
```

Out[11]=
```
or[member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], to[y]],
  not[equal[domain[funpart[t]], fix[to[x]]]], not[member[pair[u, v], to[x]]],
  not[subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]]],
  not[subclass[range[funpart[t]], fix[to[y]]]]] == True
```

In[12]:=
```
(% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

eliminating variables

The task of removing the variables **u** and **v** from the main theorem is facilitated by first eliminating the **APPLY** constructor.

Lemma. (Needed to eliminate **APPLY** from the main theorem of the preceding section.)

```
In[13]:= Map[implies[and[member[pair[u, v], to[x]], equal[domain[funpart[t]], fix[to[x]]],
           member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], y]], #] &,
         member[pair[u, v], composite[inverse[funpart[t]], y, funpart[t]]] // AssertTest] // MapNotNot

Out[13]= or[member[pair[u, v], composite[inverse[funpart[t]], y, funpart[t]]],
           not[equal[domain[funpart[t]], fix[to[x]]]], not[member[pair[u, v], to[x]]],
           not[member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], y]] == True

In[14]:= (% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Restatement of the main theorem without the constructor **APPLY**.

```
In[15]:= Map[not, SubstTest[and, implies[and[p1, p2, p3], p4], implies[p1, p5],
           implies[and[p2, p4, p5], p6], not[implies[and[p1, p2, p3], p6]],
           {p1 → and[equal[domain[funpart[t]], fix[to[x]]], subclass[range[funpart[t]], fix[to[y]]]],
            p2 -> member[pair[u, v], to[x]],
            p3 -> subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]],
            p4 -> member[pair[APPLY[funpart[t], u], APPLY[funpart[t], v]], to[y]],
            p5 -> equal[domain[funpart[t]], fix[to[x]]],
            p6 -> member[pair[u, v], composite[inverse[funpart[t]], to[y], funpart[t]]]}]] // Reverse

Out[15]= or[member[pair[u, v], composite[inverse[funpart[t]], to[y], funpart[t]]],
           not[equal[domain[funpart[t]], fix[to[x]]]], not[member[pair[u, v], to[x]]],
           not[subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]]],
           not[subclass[range[funpart[t]], fix[to[y]]]]] == True

In[16]:= (% /. {t → t_, u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Since the rewrite rules for the complement of **to[x]** and its inverse interfere with the process of eliminating variables, these are temporarily removed.

```
In[17]:= complement[to[x_]] =.

In[18]:= complement[inverse[to[x_]]] =.
```

Theorem. Increasing functions are monotone.

```
In[19]:= Map[or[subclass[composite[funpart[t], to[x], inverse[funpart[t]]], to[y]],
           empty[domain[complement[#]]]] &, SubstTest[class, pair[u, v],
         or[member[pair[u, v], q], not[equal[domain[funpart[t]], fix[s]]],
          not[member[pair[u, v], s]], not[subclass[r, s]], not[subclass[range[funpart[t]], w]]],
         {q -> composite[inverse[funpart[t]], to[y], funpart[t]],
          r -> composite[inverse[funpart[t]], to[y], funpart[t]], s -> to[x], w → fix[to[y]]}]]

Out[19]= or[not[equal[domain[funpart[t]], fix[to[x]]]],
           not[subclass[composite[inverse[funpart[t]], to[y], funpart[t]], to[x]]],
           not[subclass[range[funpart[t]], fix[to[y]]]],
           subclass[composite[funpart[t], to[x], inverse[funpart[t]]], to[y]]] == True

In[20]:= or[not[equal[domain[funpart[t_]], fix[to[x_]]]],
          not[subclass[composite[inverse[funpart[t_]], to[y_], funpart[t_]], to[x_]]],
          not[subclass[range[funpart[t_]], fix[to[y_]]]],
          subclass[composite[funpart[t_], to[x_], inverse[funpart[t_]]], to[y_]]] := True
```

Corollary. (Wrapper-free corollary.)

```
In[21]:= SubstTest[implies, and[equal[t, funpart[w]], equal[x, to[u]], equal[y, to[v]]],
           or[not[equal[domain[t], fix[x]]],
            not[subclass[composite[inverse[t], y, t], x]], not[subclass[range[t], fix[y]]],
            subclass[composite[t, x, inverse[t]], y]], {w → t, u → x, v → y}] // Reverse

Out[21]= or[not[equal[domain[t], fix[x]]], not[FUNCTION[t]],
           not[subclass[composite[inverse[t], y, t], x]], not[subclass[range[t], fix[y]]],
           not[TOTALORDER[x]], not[TOTALORDER[y]], subclass[composite[t, x, inverse[t]], y]] == True

In[22]:= or[not[equal[domain[t_], fix[x_]]],
           not[FUNCTION[t_]], not[subclass[composite[inverse[t_], y_, t_], x_]],
           not[subclass[range[t_], fix[y_]]], not[TOTALORDER[x_]],
           not[TOTALORDER[y_]], subclass[composite[t_, x_, inverse[t_]], y_]] := True
```

If **t** is a set, one can use mapping terminology.

Corollary. If **x** and **y** are total orders, and if **t: fix[x] → fix[y]** is increasing, then **t** is monotone.

```
In[23]:= SubstTest[implies, and[TOTALORDER[x], TOTALORDER[y], member[t, V], equal[domain[t], fix[x]],
            p, subclass[composite[inverse[t], y, t], x], subclass[range[t], fix[y]]],
           subclass[composite[t, x, inverse[t]], y], p → FUNCTION[t]] // Reverse

Out[23]= or[not[member[t, map[fix[x], fix[y]]]], not[subclass[composite[inverse[t], y, t], x]],
           not[TOTALORDER[x]], not[TOTALORDER[y]], subclass[composite[t, x, inverse[t]], y]] == True

In[24]:= or[not[member[t_, map[fix[x_], fix[y_]]]],
           not[subclass[composite[inverse[t_], y_, t_], x_]], not[TOTALORDER[x_]],
           not[TOTALORDER[y_]], subclass[composite[t_, x_, inverse[t_]], y_]] := True
```

A separate rewrite rule is needed for the special case **y = x**.

Corollary.

```
In[25]:= SubstTest[or, not[member[t, map[fix[x], fix[y]]]],
           not[subclass[composite[inverse[t], y, t], x]], not[TOTALORDER[x]],
           not[TOTALORDER[y]], subclass[composite[t, x, inverse[t]], y], y → x] // Reverse

Out[25]= or[not[member[t, map[fix[x], fix[x]]]], not[subclass[composite[inverse[t], x, t], x]],
           not[TOTALORDER[x]], subclass[composite[t, x, inverse[t]], x]] == True

In[26]:= or[not[member[t_, map[fix[x_], fix[x_]]]], not[subclass[composite[inverse[t_], x_, t_], x_]],
           not[TOTALORDER[x_]], subclass[composite[t_, x_, inverse[t_]], x_]] := True
```

mappings of natural numbers

The special case of mappings from the set of natural numbers to itself is considered in this section.

Lemma. A simplification rule.

```
In[27]:= Map[not,
           SubstTest[and, implies[p1, p2], not[implies[p1, p3]], {p1 → member[x, map[omega, omega]],
             p2 → equal[domain[x], omega], p3 -> subclass[image[inverse[x], y], omega]}]] // Reverse

Out[27]= or[not[member[x, map[omega, omega]]], subclass[image[inverse[x], y], omega]] == True

In[28]:= or[not[member[x_, map[omega, omega]]], subclass[image[inverse[x_], y_], omega]] := True
```

Lemma. (Brute force specialization of the general theorem to the case of natural numbers.)

```
In[29]:= SubstTest[or, not[equal[domain[t], fix[x]]], not[FUNCTION[t]],
           not[subclass[composite[inverse[t], y, t], x]], not[subclass[range[t], fix[y]]],
           not[TOTALORDER[x]], not[TOTALORDER[y]], subclass[composite[t, x, inverse[t]], y],
           {x → restrict[S, omega, omega], y → restrict[S, omega, omega]}] // Reverse // MapNotNot

Out[29]= or[not[member[t, map[omega, omega]]],
          not[subclass[composite[inverse[t], id[omega], S, id[omega], t], S]], not[
           subclass[image[inverse[t], intersection[omega, image[S, intersection[omega, range[t]]]]],
            omega]], not[subclass[image[inverse[t],
             intersection[omega, image[inverse[S], intersection[omega, range[t]]]], omega]],
          subclass[composite[t, id[omega], S, id[omega], inverse[t]], S]] == True

In[30]:= (% /. t → t_) /. Equal → SetDelayed
```

The above result can be simplified considerably.

Theorem. Any increasing mapping **t:** $\omega \to \omega$ is monotone.

```
In[31]:= Map[not, SubstTest[and, implies[p1, p4], implies[p1, p5], implies[p2, p3],
           implies[p1, p7], implies[and[p6, p7], p8], not[implies[and[p1, p2], p8]],
           {p1 -> member[t, map[omega, omega]], p2 -> subclass[composite[inverse[t], S, t], S],
            p3 -> subclass[composite[inverse[t], id[omega], S, id[omega], t], S],
            p4 -> subclass[image[inverse[t], intersection[omega,
                 image[S, intersection[omega, range[t]]]], omega], p5 -> subclass[image[inverse[t],
                intersection[omega, image[inverse[S], intersection[omega, range[t]]]], omega],
            p6 -> subclass[composite[t, id[omega], S, id[omega], inverse[t]], S],
            p7 → equal[domain[t], omega], p8 -> subclass[composite[t, S, inverse[t]], S]}]] // Reverse

Out[31]= or[not[member[t, map[omega, omega]]], not[subclass[composite[inverse[t], S, t], S]],
          subclass[composite[t, S, inverse[t]], S]] == True

In[32]:= or[not[member[t_, map[omega, omega]]], not[subclass[composite[inverse[t_], S, t_], S]],
           subclass[composite[t_, S, inverse[t_]], S]] := True
```