monotone and restrictions

Johan G. F. Belinfante
2010 September 29

```
In[1]:= SetDirectory["l:"]; << goedel.10sep28a

        :Package Title: goedel.10sep28a            2010 September 28 at 1:40 p.m.

        It is now:  2010 Sep 29 at 13:48

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

It is sometimes necessary to consider monotonicity conditions involving restrictions of relations, which can easily lead to rather messy expressions. In this notebook some rewrite rules are derived that allow one in certain situations to replace such conditions with monotonicity involving the unrestricted relations. As a simple application, a generalization is derived for the following available rewrite rule about strictly monotone functions involving natural numbers:

```
In[3]:= subclass[intersection[monotone[E, E], FUNS, P[cart[omega, omega]]], BIJ]

Out[3]= True
```

The generalization replaces **map[omega, omega]** with **FUNS ⋂ P[Ω × Ω]**, allowing one to consider functions whose domain and range are arbitrary classes of ordinal numbers, which in fact need not even be sets.

derivation

Theorem. If **x ⊂ u × v**, then **image[x, y] ⊂ v**.

```
In[4]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → subclass[x, cart[u, v]],
            p2 → subclass[range[x], v], p3 -> subclass[image[x, y], v]}]] // Reverse

Out[4]= or[not[subclass[x, cart[u, v]]], subclass[image[x, y], v]] == True

In[5]:= or[not[subclass[x_, cart[u_, v_]]], subclass[image[x_, y_], v_]] := True
```

Lemma.

```
In[6]:= implies[member[w, intersection[P[cart[u, v]], monotone[x, y]]],
          member[w, monotone[restrict[x, u, u], restrict[y, v, v]]]] // NotNotTest

Out[6]= or[and[subclass[w, cart[V, V]],
           subclass[image[w, intersection[u, image[x, intersection[u, domain[w]]]]], v],
           subclass[image[w, intersection[u, image[inverse[x], intersection[u, domain[w]]]],
            v]], not[member[w, V]], not[subclass[w, cart[u, v]]],
          not[subclass[composite[w, x, inverse[w]], y]]] == True

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

A less messy result is obtained when one eliminates the variable **w**.

Lemma. An inclusion obtained when **w** is eliminated.

```
In[8]:= Map[equal[V, #] &, SubstTest[class, w, implies[member[w, r], member[w, s]],
          {r -> intersection[P[cart[u, v]], monotone[x, y]],
           s -> monotone[restrict[x, u, u], restrict[y, v, v]]}]]

Out[8]= subclass[intersection[monotone[x, y], P[cart[u, v]]],
          monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]]] == True

In[9]:= subclass[intersection[monotone[x_, y_], P[cart[u_, v_]]],
          monotone[composite[id[u_], x_, id[u_]], composite[id[v_], y_, id[v_]]]] := True
```

Lemma.

```
In[10]:= SubstTest[or, not[equal[w, composite[t, id[u]]]],
           not[subclass[composite[t, id[u], x, id[u], inverse[w]], y]],
           subclass[composite[w, x, inverse[w]], y], t → w] // Reverse

Out[10]= or[not[subclass[w, cart[u, V]]],
           not[subclass[composite[w, id[u], x, id[u], inverse[w]], y]],
           subclass[composite[w, x, inverse[w]], y]] == True

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

Lemma.

```
In[12]:= implies[member[w,
            intersection[P[cart[u, V]], monotone[restrict[x, u, u], restrict[y, v, v]]]],
           member[w, monotone[x, y]]] // NotNotTest

Out[12]= or[and[subclass[w, cart[V, V]], subclass[composite[w, x, inverse[w]], y]],
           not[member[w, V]], not[subclass[w, cart[u, V]]],
           not[subclass[composite[w, id[u], x, id[u], inverse[w]], y]],
           not[subclass[image[w, intersection[u, image[x, intersection[u, domain[w]]]], v]],
           not[subclass[image[w,
              intersection[u, image[inverse[x], intersection[u, domain[w]]]], v]]] == True

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

Lemma. Elimination of **w** followed by specializing to the case **v** = **V**.

```
In[15]:= Map[equal[V, #] &, SubstTest[class, w, implies[member[w, r], member[w, s]],
           {r -> intersection[P[cart[u, V]], monotone[restrict[x, u, u], restrict[y, v, v]]],
            s → monotone[x, y]}]] /. v → V
```

```
Out[15]= subclass[intersection[monotone[composite[id[u], x, id[u]], y], P[cart[u, V]]],
           monotone[x, y]] == True
```

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

Corollary.

```
In[17]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
           {r → P[cart[u, v]], s → P[cart[u, V]], t → union[monotone[x, y],
              complement[monotone[composite[id[u], x, id[u]], y]]]}] // Reverse
```

```
Out[17]= subclass[intersection[monotone[composite[id[u], x, id[u]], y], P[cart[u, v]]],
           monotone[x, y]] == True
```

```
In[18]:= subclass[intersection[monotone[composite[id[u_], x_, id[u_]], y_], P[cart[u_, v_]]],
           monotone[x_, y_]] := True
```

Lemma.

```
In[19]:= Map[equal[V, #] &, SubstTest[class, w, implies[member[w, r], member[w, s]],
           {r -> intersection[P[cart[u, V]], monotone[restrict[x, u, u], restrict[y, v, v]]],
            s → monotone[x, y]}]]
```

```
Out[19]= subclass[intersection[monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]],
           P[cart[u, V]]], monotone[x, y]] == True
```

```
In[20]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[21]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
           {r → P[cart[u, v]], s → P[cart[u, V]], t → union[monotone[x, y], complement[
               monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]]]]}] // Reverse
```

```
Out[21]= subclass[intersection[monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]],
           P[cart[u, v]]], monotone[x, y]] == True
```

```
In[22]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[23]:= SubstTest[and, subclass[r, s], subclass[s, r],
          {r -> intersection[monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]],
             P[cart[u, v]]], s -> intersection[monotone[x, y], P[cart[u, v]]]}]
```

```
Out[23]= equal[intersection[monotone[x, y], P[cart[u, v]]],
           intersection[monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]],
            P[cart[u, v]]]] == True
```

```
In[24]:= intersection[monotone[composite[id[u_], x_, id[u_]], composite[id[v_], y_, id[v_]]],
           P[cart[u_, v_]]] := intersection[monotone[x, y], P[cart[u, v]]]
```

an application

Lemma. A simplification rule.

```
In[25]:= SubstTest[monotone, inverse[t], inverse[t], t → composite[id[OMEGA], E]] // Reverse

Out[25]= monotone[composite[inverse[E], id[OMEGA]], composite[inverse[E], id[OMEGA]]] ==
          monotone[composite[id[OMEGA], E], composite[id[OMEGA], E]]

In[26]:= monotone[composite[inverse[E], id[OMEGA]], composite[inverse[E], id[OMEGA]]] :=
          monotone[composite[id[OMEGA], E], composite[id[OMEGA], E]]
```

Lemma. A simplification rule.

```
In[30]:= SubstTest[intersection,
           monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]],
           P[cart[u, v]], {x → E, y → E, u → OMEGA, v → OMEGA}] // Reverse

Out[30]= intersection[monotone[composite[id[OMEGA], E], composite[id[OMEGA], E]],
           P[cart[OMEGA, OMEGA]]] == intersection[monotone[E, E], P[cart[OMEGA, OMEGA]]]

In[31]:= intersection[monotone[composite[id[OMEGA], E], composite[id[OMEGA], E]],
           P[cart[OMEGA, OMEGA]]] := intersection[monotone[E, E], P[cart[OMEGA, OMEGA]]]
```

Theorem. Strictly monotone functions involving ordinals are one-to-one.

```
In[34]:= Map[subclass[intersection[FUNS, #], BIJ] &, SubstTest[intersection,
            monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]], P[cart[u, v]],
            {u → OMEGA, v → OMEGA, x → complement[S], y → complement[S]}]] // Reverse

Out[34]= subclass[intersection[FUNS, monotone[E, E], P[cart[OMEGA, OMEGA]]], BIJ] == True

In[35]:= subclass[intersection[FUNS, monotone[E, E], P[cart[OMEGA, OMEGA]]], BIJ] := True
```

The following corollary can be applied even when the function **x** is a proper class.

Corollary.

```
In[37]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w], {u → P[x],
            v -> intersection[FUNS, monotone[E, E], P[cart[OMEGA, OMEGA]]], w -> BIJ}] // Reverse

Out[37]= or[FUNCTION[inverse[x]], not[FUNCTION[x]], not[subclass[x, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x, E, inverse[x]], E]]] == True

In[38]:= or[FUNCTION[inverse[x_]], not[FUNCTION[x_]], not[subclass[x_, cart[OMEGA, OMEGA]]],
           not[subclass[composite[x_, E, inverse[x_]], E]]] := True
```

A rather different, but equivalent, form of this is obtained when one introduces a **funpart** wrapper.

Lemma.

```
In[39]:= SubstTest[implies, and[FUNCTION[t],
           subclass[t, cartsq[OMEGA]], subclass[composite[t, E, inverse[t]], E]],
         FUNCTION[inverse[t]], t → funpart[x]] // Reverse

Out[39]= or[FUNCTION[inverse[funpart[x]]], not[subclass[domain[funpart[x]], OMEGA]],
           not[subclass[funpart[x], composite[S, IMAGE[funpart[x]]]]],
           not[subclass[range[funpart[x]], OMEGA]]] == True

In[40]:= (% /. x → x_) /. Equal→ SetDelayed
```

Eliminating **funpart** yields the following equivalent formulation of the theorem about strictly monotone functions from ordinals to ordinals.

Theorem. An equivalent result.

```
In[41]:= SubstTest[implies, equal[x, funpart[t]], or[FUNCTION[inverse[x]],
           not[subclass[domain[x], OMEGA]], not[subclass[x, composite[S, IMAGE[x]]]],
           not[subclass[range[x], OMEGA]]], t → x] // Reverse

Out[41]= or[FUNCTION[inverse[x]], not[FUNCTION[x]], not[subclass[x, composite[S, IMAGE[x]]]],
           not[subclass[domain[x], OMEGA]], not[subclass[range[x], OMEGA]]] == True

In[42]:= or[FUNCTION[inverse[x_]], not[FUNCTION[x_]],
           not[subclass[x_, composite[S, IMAGE[x_]]]],
           not[subclass[domain[x_], OMEGA]], not[subclass[range[x_], OMEGA]]] := True
```