similar ⇒ monotone

Johan G. F. Belinfante
2010 November 5

```
In[1]:= SetDirectory["l:"]; << goedel.10nov03a

        :Package Title: goedel.10nov03a               2010 November 3 at 11:40 a.m.

        It is now:  2010 Nov 5 at 2:51

        Loading Simplification Rules

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

        weightlimit = 40
```

summary

In this notebook it is shown that if **x** and **y** are similar relations, then there is a bijection from the union of the domain and range of **x** to the union of the domain and range of **y** which is monotone in both directions.

monotone ⇒ similar

The converse of the theorem to be proved is an immediate corollary of an available rewrite rule.

Theorem. A special case of a more general rewrite rule.

```
In[2]:= SubstTest[implies, not[empty[
            intersection[bij[u, v], monotone[x, y], monotone[complement[x], complement[y]]]]],
          member[pair[restrict[x, u, u], restrict[y, v, v]], SIMILAR],
          {u → udora[x], v → udora[y]}] // Reverse

Out[2]= or[equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
            monotone[x, y], monotone[complement[x], complement[y]]]],
          member[pair[composite[Id, x], composite[Id, y]], SIMILAR]] == True

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

Comment. The convenient abbreviation **udora[x]** automatically expands out to the union of the domain and range of **x**. For a reflexive relation, this expression can be replaced with the simpler expression **fix[x]**.

derivation

Recall that by definition, relations **x** and **y** are **similar** if there exists a bijection **t** such that **x ⊂ domain[t] × domain[t]** and **y = t ∘ x ∘ inverse[t]**. The strategy is to show that **w = id[udora[y]] ∘ t** satisfies **w ∈ bij[udora[x], udora[y]]** and is monotone in both directions. The variable **y** will be replaced with **t ∘ x ∘ inverse[t]** to avoid equality substitutions that would otherwise lead to excessive execution times. It is often convenient to use the compound wrapper **oopart[setpart[t]]** as a replacement for the three literals **FUNCTION[t]**, **FUNCTION[inverse[t]]** and **t ∈ V**.

Technical Lemma.

```
In[4]:= member[
          composite[id[union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]]],
             image[oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]]],
            oopart[setpart[t]]], bij[union[intersection[domain[oopart[setpart[t]]],
             image[x, domain[oopart[setpart[t]]]], intersection[
             domain[oopart[setpart[t]]], image[inverse[x], domain[oopart[setpart[t]]]]],
            union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]], image[
             oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]]]] // AssertTes

Out[4]= member[
          composite[id[union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]],
             image[oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]],
            oopart[setpart[t]]], bij[union[intersection[domain[oopart[setpart[t]]],
             image[x, domain[oopart[setpart[t]]]], intersection[
             domain[oopart[setpart[t]]], image[inverse[x], domain[oopart[setpart[t]]]]],
            union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]],
             image[oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]]] == True

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

Removing the wrappers and restoring the variable **y** yields a simpler result.

Theorem. (Removing the wrappers.)

```
In[6]:= SubstTest[implies, equal[t, oopart[setpart[z]]], member[composite[
            id[union[image[t, image[x, domain[t]]], image[t, image[inverse[x], domain[t]]]]],
            t], bij[union[intersection[domain[t], image[x, domain[t]]],
             intersection[domain[t], image[inverse[x], domain[t]]]], union[image[t,
              image[x, domain[t]]], image[t, image[inverse[x], domain[t]]]]], z → t] // Reverse

Out[6]= or[member[composite[
            id[union[image[t, image[x, domain[t]]], image[t, image[inverse[x], domain[t]]]]],
            t], bij[union[intersection[domain[t], image[x, domain[t]]],
             intersection[domain[t], image[inverse[x], domain[t]]],
            union[image[t, image[x, domain[t]]], image[t, image[inverse[x], domain[t]]]]],
          not[FUNCTION[t]], not[FUNCTION[inverse[t]]], not[member[t, V]]] == True

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

Theorem. (Restoring the variable **y**.)

```
In[8]:= SubstTest[implies, and[equal[x, restrict[z, domain[t], domain[t]]],
          equal[y, composite[t, x, inverse[t]]]],
         or[member[composite[id[union[image[t, image[z, domain[t]]],
             image[t, image[inverse[z], domain[t]]]]], t], bij[udora[x], udora[y]]],
          not[FUNCTION[t]], not[FUNCTION[inverse[t]]], not[member[t, V]], z → x] // Reverse

Out[8]= or[member[composite[
           id[union[image[t, image[x, domain[t]]], image[t, image[inverse[x], domain[t]]]]],
           t], bij[union[domain[x], range[x]], union[domain[y], range[y]]]],
         not[equal[y, composite[t, x, inverse[t]]]], not[FUNCTION[t]],
         not[FUNCTION[inverse[t]]], not[member[t, V]],
         not[subclass[x, cart[domain[t], domain[t]]]]] == True

In[9]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

The **oopart[setpart[t]]** wrapper is reintroduced here:

Theorem.

```
In[10]:= SubstTest[or, member[composite[
           id[union[image[w, image[x, domain[w]]], image[w, image[inverse[x], domain[w]]]]],
           w], bij[union[domain[x], range[x]], union[domain[y], range[y]]]],
         not[equal[y, composite[w, x, inverse[w]]]], not[FUNCTION[w]],
         not[FUNCTION[inverse[w]]], not[member[w, V]],
         not[subclass[x, cart[domain[w], domain[w]]]], w → oopart[setpart[t]]] // Reverse

Out[10]= or[member[
           composite[id[union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]]],
             image[oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]]]],
           oopart[setpart[t]]], bij[union[domain[x], range[x]], union[domain[y], range[y]]]],
         not[equal[y, composite[oopart[setpart[t]], x, inverse[oopart[setpart[t]]]]], not[
          subclass[x, cart[domain[oopart[setpart[t]]], domain[oopart[setpart[t]]]]]] == True

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

The two monotonicity conditions are automatically satisfied by **w**.

Lemma.

```
In[17]:= SubstTest[implies, member[w, z], not[empty[z]], z → intersection[monotone[x, y],
           monotone[complement[x], complement[y]], bij[udora[x], udora[y]]]] // Reverse

Out[17]= or[not[
           equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
             monotone[x, y], monotone[complement[x], complement[y]]]]],
         not[member[w, bij[union[domain[x], range[x]], union[domain[y], range[y]]]]],
         not[subclass[composite[w, x, inverse[w]], y]],
         not[subclass[composite[inverse[w], y, w], x]]] == True

In[18]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[19]:= Map[not, SubstTest[and, implies[and[p0, p1], p2], implies[and[p0, p1], p3], implies[
           and[p0, p1], p4], implies[and[p2, p3, p4], p5], not[implies[and[p0, p1], p5]],
          {p0 → equal[w, composite[id[union[image[oopart[setpart[t]],
                 image[x, domain[oopart[setpart[t]]]]], image[oopart[setpart[t]],
                 image[inverse[x], domain[oopart[setpart[t]]]]]], oopart[setpart[t]]]],
           p1 → and[equal[y, composite[oopart[setpart[t]], x, inverse[oopart[setpart[t]]]]],
             subclass[x, cart[domain[oopart[setpart[t]]], domain[oopart[setpart[t]]]]],
           p2 -> member[w, bij[union[domain[x], range[x]], union[domain[y], range[y]]]],
           p3 -> subclass[composite[w, x, inverse[w]], y],
           p4 -> subclass[composite[inverse[w], y, w], x], p5 -> not[equal[0,
              intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
               monotone[x, y], monotone[complement[x], complement[y]]]]}] /. w -> composite[
           id[union[image[oopart[setpart[t]], image[x, domain[oopart[setpart[t]]]]],
             image[oopart[setpart[t]], image[inverse[x], domain[oopart[setpart[t]]]]]],
           oopart[setpart[t]]]] // Reverse

Out[19]= or[not[
           equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
             monotone[x, y], monotone[complement[x], complement[y]]]],
          not[equal[y, composite[oopart[setpart[t]], x, inverse[oopart[setpart[t]]]]], not[
           subclass[x, cart[domain[oopart[setpart[t]]], domain[oopart[setpart[t]]]]] == True

In[20]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Eliminate the wrappers.

```
In[21]:= SubstTest[implies, equal[t, oopart[setpart[w]]], or[not[
            equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
              monotone[x, y], monotone[complement[x], complement[y]]]],
           not[equal[y, composite[t, x, inverse[t]]]],
           not[subclass[x, cart[domain[t], domain[t]]]], w → t] // Reverse

Out[21]= or[not[
           equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
             monotone[x, y], monotone[complement[x], complement[y]]]],
          not[equal[y, composite[t, x, inverse[t]]]], not[FUNCTION[t]],
          not[FUNCTION[inverse[t]]], not[member[t, V]],
          not[subclass[x, cart[domain[t], domain[t]]]] == True

In[22]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

---

eliminating the variable t

The function **IPD = λ x . IMAGE[x] ∘ id[P[domain[x]]]** will be used to eliminate the variable **t**.

```
In[48]:= VERTSECT[reify[x, composite[IMAGE[x], id[P[domain[x]]]]]]

Out[48]= IPD
```

The connection with **SIMILAR** is as follows:

```
In[23]:= composite[inverse[E], IPD, CROSS, DUP, OOPART] // range

Out[23]= SIMILAR
```

The following membership rule is available for this composite.

```
In[25]:= member[pair[setpart[t], pair[setpart[x], setpart[y]]],
           composite[inverse[E], IPD, CROSS, DUP, OOPART]]

Out[25]= and[equal[
            composite[oopart[setpart[t]], setpart[x], inverse[oopart[setpart[t]]]], setpart[y]],
           subclass[setpart[x], cart[domain[oopart[setpart[t]]], domain[oopart[setpart[t]]]]]]
```

Theorem. Eliminating the variable **t**.

```
In[33]:= Map[equal[BIJ, image[OOPART, #]] &, SubstTest[class, t,
            implies[member[pair[setpart[t], pair[setpart[x], setpart[y]]], s], not[equal[0, w]]],
            {s -> composite[inverse[E], IPD, CROSS, DUP, OOPART],
             w -> intersection[bij[union[domain[setpart[x]], range[setpart[x]]],
                 union[domain[setpart[y]], range[setpart[y]]]], monotone[complement[setpart[x]],
                 complement[setpart[y]]], monotone[setpart[x], setpart[y]]]}]]

Out[33]= or[not[equal[0, intersection[bij[union[domain[setpart[x]], range[setpart[x]]],
                 union[domain[setpart[y]], range[setpart[y]]]], monotone[complement[setpart[x]],
                 complement[setpart[y]]], monotone[setpart[x], setpart[y]]]]],
            subclass[BIJ, fix[image[inverse[CROSS], image[inverse[IPD],
                 P[complement[cart[set[setpart[x]], set[setpart[y]]]]]]]]] == True

In[34]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Sinplification rule.

```
In[39]:= SubstTest[image, funpart[t],
            image[inverse[funpart[t]], z], t → composite[CROSS, DUP]] // Reverse

Out[39]= image[CROSS, id[fix[image[inverse[CROSS], z]]]] == intersection[z, image[CROSS, Id]]

In[40]:= image[CROSS, id[fix[image[inverse[CROSS], z_]]]] := intersection[z, image[CROSS, Id]]
```

Lemma.

```
In[41]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
            {t → composite[CROSS, DUP], u → BIJ, v -> fix[image[inverse[CROSS], image[inverse[IPD],
                 P[complement[cart[set[setpart[x]], set[setpart[y]]]]]]]]} // Reverse

Out[41]= or[not[member[pair[setpart[x], setpart[y]], SIMILAR]],
            not[subclass[BIJ, fix[image[inverse[CROSS], image[inverse[IPD],
                 P[complement[cart[set[setpart[x]], set[setpart[y]]]]]]]]] == True

In[42]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Combine the lemmas.)

```
In[43]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 -> member[pair[setpart[x], setpart[y]], SIMILAR],
            p2 -> not[subclass[BIJ, fix[image[inverse[CROSS], image[inverse[IPD],
                  P[complement[cart[set[setpart[x]], set[setpart[y]]]]]]]]]],
            p3 -> not[equal[0, intersection[bij[union[domain[setpart[x]], range[setpart[x]]],
                  union[domain[setpart[y]], range[setpart[y]]]],
                 monotone[complement[setpart[x]], complement[setpart[y]]],
                 monotone[setpart[x], setpart[y]]]]}]] // Reverse

Out[43]= or[not[equal[0, intersection[bij[union[domain[setpart[x]], range[setpart[x]]],
             union[domain[setpart[y]], range[setpart[y]]]], monotone[complement[setpart[x]],
             complement[setpart[y]]], monotone[setpart[x], setpart[y]]]],
           not[member[pair[setpart[x], setpart[y]], SIMILAR]] == True

In[44]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The final step is to remove the **setpart** wrappers.

Main Theorem. If **x** and **y** are similar, then there is a bijection from **udora[x]** to **udora[y]** that is monotone in both directions.

```
In[45]:= SubstTest[implies, and[equal[x, setpart[u]], equal[y, setpart[v]]],
           implies[member[pair[x, y], SIMILAR],
            not[empty[intersection[bij[udora[x], udora[y]], monotone[x, y],
               monotone[complement[x], complement[y]]]]], {u → x, v → y}] // Reverse // MapNotNot

Out[45]= or[not[
            equal[0, intersection[bij[union[domain[x], range[x]], union[domain[y], range[y]]],
              monotone[x, y], monotone[complement[x], complement[y]]]],
           not[member[pair[x, y], SIMILAR]]] == True

In[47]:= or[not[equal[0,
             intersection[bij[union[domain[x_], range[x_]], union[domain[y_], range[y_]]],
              monotone[complement[x_], complement[y_]], monotone[x_, y_]]]],
           not[member[pair[x_, y_], SIMILAR]]] := True
```

an application

In this section the special case of restrictions of the subset relation is considered. Some simplification rules for restrictions are needed.

Lemma. A simplification rule.

```
In[95]:= AssInt[bij[u, v], P[cart[u, v]],
           monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]]] // Reverse

Out[95]= intersection[bij[u, v],
           monotone[composite[id[u], x, id[u]], composite[id[v], y, id[v]]]] ==
          intersection[bij[u, v], monotone[x, y]]

In[96]:= intersection[bij[u_, v_],
           monotone[composite[id[u_], x_, id[u_]], composite[id[v_], y_, id[v_]]]] :=
          intersection[bij[u, v], monotone[x, y]]
```

Lemma.

```
In[75]:= Map[image[#, monotone[x, y]] &,
          Assoc[composite[INVERSE, id[P[cart[u, v]]], INVERSE, INVERSE]]

Out[75]= image[INVERSE, intersection[monotone[x, y], P[cart[u, v]]]] ==
          intersection[monotone[complement[y], complement[x]], P[cart[v, u]]]

In[76]:= image[INVERSE, intersection[monotone[x_, y_], P[cart[u_, v_]]]] :=
          intersection[monotone[complement[y], complement[x]], P[cart[v, u]]]
```

Lemma. Simplification rule.

```
Map[intersection[bij[u, v], #] &,
  Map[image[#, monotone[restrict[y, v, v], restrict[x, u, u]]] &,
   Assoc[composite[INVERSE, id[P[cart[v, u]]], INVERSE, INVERSE]]] // Reverse

Out[80]= intersection[bij[u, v], monotone[complement[cart[u, u]],
           union[complement[cart[v, v]], composite[id[v], complement[y], id[v]]]],
          monotone[composite[id[u], complement[x], id[u]],
           union[complement[cart[v, v]], composite[id[v], complement[y], id[v]]]]] ==
          intersection[bij[u, v], monotone[complement[x], complement[y]]]

In[81]:= intersection[bij[u_, v_], monotone[complement[cart[u_, u_]],
           union[complement[cart[v_, v_]], composite[id[v_], complement[y_], id[v_]]]],
          monotone[composite[id[u_], complement[x_], id[u_]],
           union[complement[cart[v_, v_]], composite[id[v_], complement[y_], id[v_]]]]] :=
          intersection[bij[u, v], monotone[complement[x], complement[y]]]
```

Theorem. If the restrictions of the subset relation to two sets **x** and **y** are similar, then there is a bijection from **x** to **y** that is monotone with respect to inclusion in both directions.

```
In[85]:= SubstTest[implies, member[pair[u, v], SIMILAR],
           not[empty[intersection[bij[udora[u], udora[v]],
              monotone[u, v], monotone[complement[u], complement[v]]]]],
           {u → restrict[S, x, x], v → restrict[S, y, y]}] // Reverse

Out[85]= or[not[equal[0,
             intersection[bij[x, y], monotone[S, S], monotone[complement[S], complement[S]]]],
           not[member[pair[composite[id[x], S, id[x]], composite[id[y], S, id[y]]], SIMILAR]]] ==
          True
```

```
In[86]:= or[not[equal[0, intersection[bij[x_, y_],
              monotone[S, S], monotone[complement[S], complement[S]]]]], not[member[
            pair[composite[id[x_], S, id[x_]], composite[id[y_], S, id[y_]], SIMILAR]]] := True
```

Comment. The converse of this is already available.