similarity of transitive closures

Johan G. F. Belinfante
2010 May 11

```
In[1]:= SetDirectory["l:"]; << goedel.10may10a;<< tools.m

        :Package Title: goedel.10may10a               2010 May 10 at 4:30 p.m.

        It is now:  2010 May 11 at 15:13

        Loading Simplification Rules

        TOOLS.M                                    Revised 2010 February 26

        weightlimit = 40
```

summary

Similar relations have similar transitive closures.

derivation

Lemma.

```
In[2]:= SubstTest[implies, equal[x, y], equal[trv[x], trv[y]], y → restrict[x, t, t]] // Reverse

Out[2]= or[equal[trv[x], trv[composite[id[t], x, id[t]]]], not[subclass[x, cart[t, t]]]] == True

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

The next lemma reduces the study of transitive closures of similarity transforms to the transitive closures of restrictions.

Key Lemma.

```
In[4]:= SubstTest[equal, composite[u, trv[composite[v, u]]],
          composite[trv[composite[u, v]], u], {u → inverse[oopart[t]],
           v → composite[oopart[t], x, id[domain[oopart[t]]]]}] // Reverse

Out[4]= equal[composite[inverse[oopart[t]], trv[composite[oopart[t], x, inverse[oopart[t]]]]],
          composite[trv[composite[id[domain[oopart[t]]], x, id[domain[oopart[t]]]]],
           inverse[oopart[t]]]] == True

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

Lemma. A temporary simplification rule.

```
In[6]:= equal[composite[id[range[oopart[t]]]],
          trv[composite[oopart[t], x, inverse[oopart[t]]]]],
         trv[composite[oopart[t], x, inverse[oopart[t]]]]]

Out[6]= True

In[7]:= composite[id[range[oopart[t_]]]],
          trv[composite[oopart[t_], x_, inverse[oopart[t_]]]]] :=
         trv[composite[oopart[t], x, inverse[oopart[t]]]]
```

Lemma.

```
In[8]:= SubstTest[implies, equal[v, w], equal[composite[u, v], composite[u, w]], {u → oopart[t],
           v -> composite[inverse[oopart[t]], trv[composite[oopart[t], x, inverse[oopart[t]]]]],
           w -> composite[trv[composite[id[domain[oopart[t]]], x, id[domain[oopart[t]]]],
             inverse[oopart[t]]]}] // Reverse

Out[8]= equal[
          composite[oopart[t], trv[composite[id[domain[oopart[t]]], x, id[domain[oopart[t]]]],
           inverse[oopart[t]]], trv[composite[oopart[t], x, inverse[oopart[t]]]]] == True

In[9]:= composite[oopart[t_],
          trv[composite[id[domain[oopart[t_]]], x_, id[domain[oopart[t_]]]],
          inverse[oopart[t_]]] := trv[composite[oopart[t], x, inverse[oopart[t]]]]
```

Theorem. The similarity transform of a transitive closure is the transitive closure of its similarity transform.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 → subclass[x, cartsq[domain[oopart[t]]]], p2 →
              equal[trv[x], trv[composite[id[domain[oopart[t]]], x, id[domain[oopart[t]]]]]],
             p3 → equal[trv[composite[oopart[t], x, inverse[oopart[t]]]],
               composite[oopart[t], trv[x], inverse[oopart[t]]]]}]] // Reverse

Out[10]= or[equal[composite[oopart[t], trv[x], inverse[oopart[t]]],
            trv[composite[oopart[t], x, inverse[oopart[t]]]]],
           not[subclass[x, cart[domain[oopart[t]], domain[oopart[t]]]]] == True

In[11]:= or[equal[composite[oopart[t_], trv[x_], inverse[oopart[t_]]],
            trv[composite[oopart[t_], x_, inverse[oopart[t_]]]]],
           not[subclass[x_, cart[domain[oopart[t_]], domain[oopart[t_]]]]] := True
```

Corollary. Restatement without the **oopart** wrapper.

```
In[12]:= SubstTest[implies, equal[t, oopart[y]],
           or[equal[composite[t, trv[x], inverse[t]], trv[composite[t, x, inverse[t]]]],
            not[subclass[x, cart[domain[t], domain[t]]]]], y → t] // Reverse

Out[12]= or[equal[composite[t, trv[x], inverse[t]], trv[composite[t, x, inverse[t]]]],
           not[FUNCTION[t]], not[FUNCTION[inverse[t]]],
           not[subclass[x, cart[domain[t], domain[t]]]] == True
```

```
In[13]:= or[equal[composite[t_, trv[x_], inverse[t_]], trv[composite[t_, x_, inverse[t_]]],
           not[FUNCTION[inverse[t_]]], not[FUNCTION[t_]],
           not[subclass[x_, cart[domain[t_], domain[t_]]]]] := True
```

The next lemma introduces one occurrence of **SIMILAR**. Another occurrence will appear when the variables are eliminated.

Lemma.

```
In[14]:= Map[implies[subclass[x, cartsq[domain[t]]], #] &,
           SubstTest[implies, and[member[t, BIJ], subclass[u, cartsq[domain[t]]]], member[
             pair[u, composite[t, u, inverse[t]]], SIMILAR], u → trv[x]]] // Reverse // MapNotNot

Out[14]= or[member[pair[trv[x], composite[t, trv[x], inverse[t]]], SIMILAR],
           not[FUNCTION[t]], not[FUNCTION[inverse[t]]], not[member[t, V]],
           not[subclass[x, cart[domain[t], domain[t]]]]] == True

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

Lemma. A new variable **y** is introduced here as well as a **setpart** wrapper. This helps prepare for the elimination of variables.

```
In[16]:= (Map[not, SubstTest[and, implies[and[p0, p1, p2], p3], implies[and[p0, p2, p3], p4],
           not[implies[and[p0, p1, p2], p4]], {p0 → member[t, BIJ],
            p1 → equal[y, composite[t, x, inverse[t]]], p2 → subclass[x, cartsq[domain[t]]],
            p3 -> equal[trv[y], composite[t, trv[x], inverse[t]]],
            p4 → member[pair[trv[x], trv[y]], SIMILAR]}]] // Reverse) /. t → setpart[z]

Out[16]= or[member[pair[trv[x], trv[y]], SIMILAR],
           not[equal[y, composite[setpart[z], x, inverse[setpart[z]]]]],
           not[FUNCTION[inverse[setpart[z]]]], not[FUNCTION[setpart[z]]],
           not[subclass[x, cart[domain[setpart[z]], domain[setpart[z]]]]]] == True

In[17]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[18]:= member[pair[u, v], composite[inverse[IMAGE[id[x]]], y]] // AssertTest

Out[18]= member[pair[u, v], composite[inverse[IMAGE[id[x]]], y]] ==
          and[member[u, V], member[v, V], member[pair[u, intersection[v, x]], y]]

In[19]:= member[pair[u_, v_], composite[inverse[IMAGE[id[x_]]], y_]] :=
          and[member[u, V], member[v, V], member[pair[u, intersection[v, x]], y]]
```

Theorem. Eliminating all variables yields an inclusion. (This takes a while.)

```
In[20]:= Map[empty[composite[#, inverse[SECOND]]] &,
          Map[composite[complement[#], id[cart[V, V]]] &, SubstTest[class, pair[pair[t, x], y],
            or[not[member[pair[pair[setpart[t], setpart[x]], setpart[y]], u]],
             member[pair[setpart[x], setpart[y]], v]],
            {u -> composite[IMG, cross[composite[CROSS, DUP, id[BIJ]], Id],
               id[composite[inverse[S], CART, DUP, IMAGE[FIRST]]]],
             v → composite[inverse[IMAGE[id[cart[V, V]]]], inverse[HULL[TRV]],
               SIMILAR, HULL[TRV], IMAGE[id[cart[V, V]]]]}]]]
```

```
Out[20]= subclass[composite[HULL[TRV], SIMILAR, inverse[HULL[TRV]]], SIMILAR] == True
```

```
In[21]:= subclass[composite[HULL[TRV], SIMILAR, inverse[HULL[TRV]]], SIMILAR] := True
```

Corollary.

```
In[22]:= SubstTest[subclass,
           intersection[u, domain[funpart[y]]], image[inverse[funpart[y]], v],
           {y → cross[HULL[TRV], HULL[TRV]], u → SIMILAR, v→ SIMILAR}] // Reverse
```

```
Out[22]= subclass[SIMILAR, composite[inverse[HULL[TRV]], SIMILAR, HULL[TRV]]] == True
```

```
In[23]:= subclass[SIMILAR, composite[inverse[HULL[TRV]], SIMILAR, HULL[TRV]]] := True
```

Lemma. Reintroduction of variables.

```
In[24]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → set[PAIR[composite[Id, setpart[x]], composite[Id, setpart[y]]]],
            v → SIMILAR, w -> composite[inverse[HULL[TRV]], SIMILAR, HULL[TRV]]}] // Reverse
```

```
Out[24]= or[member[pair[trv[setpart[x]], trv[setpart[y]]], SIMILAR], not[
            member[pair[composite[Id, setpart[x]], composite[Id, setpart[y]]], SIMILAR]]] == True
```

```
In[25]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Simplification rule.

```
In[26]:= SubstTest[member, pair[x, y], composite[t, id[u]], {t → SIMILAR, u→ P[cart[V, V]]}]
```

```
Out[26]= and[member[pair[x, y], SIMILAR], subclass[x, cart[V, V]]] == member[pair[x, y], SIMILAR]
```

```
In[27]:= and[member[pair[x_, y_], SIMILAR], subclass[x_, cart[V, V]]] :=
          member[pair[x, y], SIMILAR]
```

Lemma. Simplification rule.

```
In[28]:= SubstTest[member, pair[x, y], composite[id[u], t], {t → SIMILAR, u→ P[cart[V, V]]}]
```

```
Out[28]= and[member[pair[x, y], SIMILAR], subclass[y, cart[V, V]]] == member[pair[x, y], SIMILAR]
```

```
In[29]:= and[member[pair[x_, y_], SIMILAR], subclass[y_, cart[V, V]]] :=
          member[pair[x, y], SIMILAR]
```

Theorem. Similar relations have similar transitive closures.

```
In[30]:= SubstTest[implies,
           and[equal[x, composite[Id, setpart[u]]], equal[y, composite[Id, setpart[v]]]],
           or[member[pair[trv[x], trv[y]], SIMILAR], not[member[pair[x, y], SIMILAR]]],
           {u → x, v → y}] // Reverse // MapNotNot

Out[30]= or[member[pair[trv[x], trv[y]], SIMILAR], not[member[pair[x, y], SIMILAR]]] == True

In[31]:= or[member[pair[trv[x_], trv[y_]], SIMILAR], not[member[pair[x_, y_], SIMILAR]]] := True
```

acyclic relations

An application to acyclic relations is considered in this section.

Lemma.

```
In[32]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
           {u -> composite[HULL[TRV], SIMILAR, inverse[HULL[TRV]]],
            v → SIMILAR, w -> P[Di]}] // Reverse

Out[32]= equal[0, fix[U[image[HULL[TRV], image[SIMILAR, ACYCLIC]]]]] == True

In[33]:= fix[U[image[HULL[TRV], image[SIMILAR, ACYCLIC]]]] := 0
```

Lemma.

```
In[34]:= SubstTest[subclass, intersection[u, P[cart[V, V]]],
           image[inverse[HULL[TRV]], v], {u → image[SIMILAR, ACYCLIC], v → P[Di]}] // Reverse

Out[34]= subclass[image[SIMILAR, ACYCLIC], ACYCLIC] == True

In[35]:= % /. Equal → SetDelayed
```

Theorem. Relations similar to acyclic relations are acyclic.

```
In[36]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> image[SIMILAR, ACYCLIC], v → ACYCLIC}]

Out[36]= equal[ACYCLIC, image[SIMILAR, ACYCLIC]] == True

In[37]:= image[SIMILAR, ACYCLIC] := ACYCLIC
```