two cases where IPD preserves composition

Johan G. F. Belinfante
2009 September 8

```
In[1]:= SetDirectory["l:"]; << goedel.09sep07a;<< tools.m

        :Package Title: goedel.09sep07a              2009 September 7 at 4:45 p.m.

        It is now:  2009 Sep 8 at 15:10

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 September 3

        weightlimit = 40
```

summary

The function **IMAGE[x]** is the class of all ordered pairs of sets **u** and **v** such that **v = image[x,u]**.

```
In[2]:= class[pair[u, v], equal[image[x, u], v]]
Out[2]= IMAGE[x]
```

The constructor **IMAGE** does not preserve composition in general, but it does so if the right-hand factor is thin. In particular, **IMAGE** preserves composition for sets:

```
In[3]:= IMAGE[composite[setpart[x], setpart[y]]]
Out[3]= composite[IMAGE[setpart[x]], IMAGE[setpart[y]]]
```

When **x** is a set, the function **IMAGE[x]** is a proper class, and therefore there is no function that takes a set **x** to **IMAGE[x]**. More precisely, the function **λx**. **IMAGE[x]** is empty:

```
In[4]:= lambda[x, IMAGE[x]]
Out[4]= 0
```

One can to some extent get around this problem by factoring the proper class **IMAGE[x]** as the composite of its restriction of the power class of the domain of **x** and the function that maps any set to its intersection with the domain of **x**. Most of the useful information about the function **IMAGE[x]** rests in the first factor, which is a set when **x** is a set. Consequently there does exist a function **IPD** that maps every set **x** to the restriction of **IMAGE[x]** to the power set of the domain of **x**. Unfortunately, this function **IPD = λx**. **IMAGE[x] ∘ id[P[domain[x]]]** does not preserve composition in general. In this notebook it is shown that **IPD** does nonetheless does preserve composition in at least two fairly general situations of interest. One case is for the composite of relations **x** and **y** that satisfy the extra restriction **range[y] ⊂ domain[x]**. The second case is when the right-hand factor **y** is a function.

a counterexample

The following temporary abbreviation will be used in this notebook.

```
In[5]:= f[x_] := composite[IMAGE[x], id[P[domain[x]]]]
```

Counterexample. In general the constructor **f** does not preserve composition, not even for cartesian products of sets.

```
In[6]:= equal[f[composite[x, y]], composite[f[x], f[y]]] /.
         {x → cartsq[set[0]], y → cartsq[succ[set[0]]]}

Out[6]= False
```

In this notebook two situations are identified for which **f** does preserve composition. One case is under the condition that the range of **y** is a subclass of the domain of **x**. The second case is the situation where **y** is a function.

a general inclusion

Observation. The following inclusion does hold without restriction.

```
In[7]:= subclass[composite[f[setpart[x]], f[setpart[y]]], f[composite[setpart[x], setpart[y]]]]

Out[7]= True
```

Before moving on, a variable-free reformulation of this result will be derived. (No use of this is made in the rest of this notebook.)

Lemma.

```
In[8]:= Map[composite[Id, complement[#]] &, SubstTest[reify, x,
          image[t, set[PAIR[setpart[first[x]], setpart[second[x]]]]], t → complement[
            dif[composite[COMPOSE, cross[IPD, IPD]], composite[inverse[S], IPD, COMPOSE]]]]]

Out[8]= composite[intersection[composite[COMPOSE, cross[IPD, IPD]],
           composite[complement[inverse[S]], IPD, COMPOSE]], id[cart[V, V]]] == 0

In[9]:= % /. Equal → SetDelayed
```

Theorem. Variable-free restatement of a general inclusion.

```
In[10]:= SubstTest[empty, composite[dif[u, v], id[cart[V, V]]],
          {u -> composite[COMPOSE, cross[IPD, IPD]], v -> composite[inverse[S], IPD, COMPOSE]}]

Out[10]= subclass[composite[COMPOSE, cross[IPD, IPD]],
           composite[inverse[S], IPD, COMPOSE]] == True

In[11]:= subclass[composite[COMPOSE, cross[IPD, IPD]],
           composite[inverse[S], IPD, COMPOSE]] := True
```

the compatibility relation

It is convenient to introduce a temporary definition: **x** and **y** are said to be **compatible** if the range of **y** is a subclass of the domain of **x**.

```
In[12]:= compatible[x_, y_] := subclass[range[y], domain[x]]
```

The compatibility relation is the class of all pairs that satisfy this condition.

```
In[13]:= class[pair[x, y], compatible[x, y]]
Out[13]= composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]
```

The following temporary abbreviation for this relation will be used in this notebook.

```
In[14]:= COMPATIBLE := composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]
```

Theorem. A simplification rule.

```
In[15]:= Assoc[S, BIGCUP, VERTSECT[composite[IMG, inverse[FIRST]]]] // Reverse
Out[15]= composite[inverse[POWER], S, VERTSECT[composite[IMG, inverse[FIRST]]]] ==
          composite[S, IMAGE[SECOND]]

In[16]:= composite[inverse[POWER], S, VERTSECT[composite[IMG, inverse[FIRST]]]] :=
          composite[S, IMAGE[SECOND]]
```

Corollary. A simplification rule.

```
In[17]:= composite[inverse[VERTSECT[composite[IMG, inverse[FIRST]]]], inverse[S], POWER] //
          DoubleInverse
Out[17]= composite[inverse[VERTSECT[composite[IMG, inverse[FIRST]]]], inverse[S], POWER] ==
          composite[inverse[IMAGE[SECOND]], inverse[S]]

In[18]:= composite[inverse[VERTSECT[composite[IMG, inverse[FIRST]]]], inverse[S], POWER] :=
          composite[inverse[IMAGE[SECOND]], inverse[S]]
```

Theorem. A simplification rule.

```
In[19]:= composite[inverse[IPD], inverse[IMAGE[SECOND]]] // DoubleInverse
Out[19]= composite[inverse[IPD], inverse[IMAGE[SECOND]]] ==
          inverse[VERTSECT[composite[IMG, inverse[FIRST]]]]

In[20]:= composite[inverse[IPD], inverse[IMAGE[SECOND]]] :=
          inverse[VERTSECT[composite[IMG, inverse[FIRST]]]]
```

These simplification rules imply that the function **IPD** preserves compatibility.

```
In[21]:= composite[inverse[IPD], COMPATIBLE, IPD] == COMPATIBLE
```

```
Out[21]= True
```

---

compatible composition

Lemma.

```
In[22]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u → composite[f[setpart[x]], f[setpart[y]]],
            v -> f[composite[setpart[x], setpart[y]]]}] // Reverse
```

```
Out[22]= equal[composite[IMAGE[setpart[x]], IMAGE[setpart[y]], id[P[intersection[
              complement[image[inverse[setpart[y]], complement[domain[setpart[x]]]]],
              image[inverse[setpart[y]], domain[setpart[x]]]]]]],
            composite[IMAGE[setpart[x]], id[P[domain[setpart[x]]]],
             IMAGE[setpart[y]], id[P[domain[setpart[y]]]]]] == True
```

```
In[23]:= composite[IMAGE[setpart[x_]], IMAGE[setpart[y_]], id[P[intersection[
             complement[image[inverse[setpart[y_]], complement[domain[setpart[x_]]]]],
             image[inverse[setpart[y_]], domain[setpart[x_]]]]]]] :=
           composite[IMAGE[setpart[x]], id[P[domain[setpart[x]]]],
            IMAGE[setpart[y]], id[P[domain[setpart[y]]]]]
```

Theorem.

```
In[24]:= SubstTest[implies, subclass[u, v],
           subclass[composite[t, u], composite[t, v]], {t → IMAGE[x],
            u -> composite[IMAGE[setpart[y]], id[P[image[inverse[setpart[y]], domain[x]]]]],
            v -> composite[id[P[domain[x]]], IMAGE[setpart[y]]]}] // Reverse
```

```
Out[24]= or[not[subclass[image[setpart[y], image[inverse[setpart[y]], domain[x]]], domain[x]]],
           subclass[composite[IMAGE[x], IMAGE[setpart[y]],
             id[P[image[inverse[setpart[y]], domain[x]]]],
            composite[IMAGE[x], id[P[domain[x]]], IMAGE[setpart[y]]]]] == True
```

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

Lemma.

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 → compatible[x, setpart[y]], p2 ->
              subclass[image[setpart[y], image[inverse[setpart[y]], domain[x]]], domain[x]],
             p3 -> subclass[composite[IMAGE[x], IMAGE[setpart[y]],
                id[P[image[inverse[setpart[y]], domain[x]]]],
               composite[IMAGE[x], id[P[domain[x]]], IMAGE[setpart[y]]]]}]] // Reverse
```

```
Out[26]= or[not[subclass[range[setpart[y]], domain[x]]], subclass[composite[IMAGE[x],
             IMAGE[setpart[y]], id[P[image[inverse[setpart[y]], domain[x]]]],
            composite[IMAGE[x], id[P[domain[x]]], IMAGE[setpart[y]]]]] == True
```

```
In[27]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[28]:= Map[implies[compatible[setpart[x], setpart[y]], #] &, SubstTest[and,
           subclass[u, v], subclass[v, u], {u → composite[f[setpart[x]], f[setpart[y]]],
            v → f[composite[setpart[x], setpart[y]]]}]] // MapNotNot

Out[28]= or[equal[composite[IMAGE[setpart[x]], IMAGE[setpart[y]],
            id[P[image[inverse[setpart[y]], domain[setpart[x]]]]]],
           composite[IMAGE[setpart[x]], id[P[domain[setpart[x]]]],
            IMAGE[setpart[y]], id[P[domain[setpart[y]]]]]],
          not[subclass[range[setpart[y]], domain[setpart[x]]]]] == True
```

```
In[29]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Restatement.

```
In[30]:= implies[compatible[setpart[x], setpart[y]],
          equal[f[composite[setpart[x], setpart[y]]], composite[f[setpart[x]], f[setpart[y]]]]]

Out[30]= True
```

Lemma. Eliminating variables.

```
In[31]:= SubstTest[reify, x, image[t, set[PAIR[setpart[first[x]], setpart[second[x]]]]],
          t -> dif[composite[IPD, COMPOSE, id[COMPATIBLE]], composite[COMPOSE, cross[IPD, IPD]]]]

Out[31]= composite[intersection[composite[IPD, COMPOSE],
            composite[complement[COMPOSE], cross[IPD, IPD]]],
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]] == 0
```

```
In[32]:= % /. Equal → SetDelayed
```

```
In[33]:= SubstTest[empty, dif[u, v], {u -> composite[IPD, COMPOSE, id[COMPATIBLE]],
           v -> composite[COMPOSE, cross[IPD, IPD]]}]

Out[33]= subclass[composite[IPD, COMPOSE,
            id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]],
           composite[COMPOSE, cross[IPD, IPD]]] == True
```

```
In[34]:= % /. Equal → SetDelayed
```

```
In[35]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u -> composite[IPD, COMPOSE, id[COMPATIBLE]],
            v → composite[COMPOSE, cross[IPD, IPD]]}] // Reverse

Out[35]= equal[composite[COMPOSE, cross[IPD, IPD],
            id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]], composite[IPD,
            COMPOSE, id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]]] == True
```

```
In[36]:= composite[IPD, COMPOSE,
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]] :=
          composite[COMPOSE, cross[IPD, IPD],
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]]]
```

Restatement.

```
In[37]:= equal[composite[IPD, COMPOSE, id[COMPATIBLE]],
          composite[COMPOSE, id[COMPATIBLE], cross[IPD, IPD]]]

Out[37]= True

In[38]:= intertwine[composite[COMPOSE, id[COMPATIBLE]], cross[IPD, IPD], IPD]

Out[38]= True
```

Comment. Although the function **COMPOSE** is associative, the restriction **composite[COMPOSE, id[COMPATIBLE]]** is not associative.

```
In[39]:= associative[composite[COMPOSE, id[COMPATIBLE]]]

Out[39]= False
```

IPD preserves COMPOSE for functions

In this section it is shown that the restriction of **IPD** to functions preserves composition. In fact one only needs to restrict the right-hand factor to functions. Compatibility is not required.

Lemma.

```
In[40]:= symdif[composite[IMAGE[x], IMAGE[funpart[y]],
            id[P[image[inverse[funpart[y]], domain[x]]]]],
           composite[IMAGE[x], id[P[domain[x]]], IMAGE[funpart[y]],
            id[P[domain[funpart[y]]]]]] // FastReifNormality

Out[40]= composite[IMAGE[x], id[P[domain[x]]], IMAGE[funpart[y]], id[intersection[
             complement[P[image[inverse[funpart[y]], domain[x]]]], P[domain[funpart[y]]]]]] == 0

In[41]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

Theorem.

In[42]:= SubstTest[empty, symdif[u, v], {u -> composite[IMAGE[x], IMAGE[funpart[y]],
             id[P[image[inverse[funpart[y]], domain[x]]]]], v -> composite[IMAGE[x],
             id[P[domain[x]]], IMAGE[funpart[y]], id[P[domain[funpart[y]]]]]}]

Out[42]= equal[composite[IMAGE[x], IMAGE[funpart[y]],
            id[P[image[inverse[funpart[y]], domain[x]]]]], composite[IMAGE[x],
            id[P[domain[x]]], IMAGE[funpart[y]], id[P[domain[funpart[y]]]]]] == True
```

```
In[43]:= composite[IMAGE[x_], IMAGE[funpart[y_]],
           id[P[image[inverse[funpart[y_]], domain[x_]]]]] :=
         composite[IMAGE[x], id[P[domain[x]]], IMAGE[funpart[y]], id[P[domain[funpart[y]]]]]
```

Restatement. The constructor **f** preserves composites when the right-hand factor is a function.

```
In[44]:= f[composite[x, funpart[y]]] == composite[f[x], f[funpart[y]]]

Out[44]= True
```

The variables **x** and **y** can be eliminated.

Lemma.

```
In[45]:= symdif[composite[COMPOSE, cross[IPD, composite[IPD, FUNPART]]],
           composite[IPD, COMPOSE, cross[Id, FUNPART]]] // VSTriNormality

Out[45]= union[composite[
            intersection[composite[COMPOSE, cross[IPD, composite[IPD, FUNPART]]], composite[
              complement[IPD], COMPOSE, cross[Id, FUNPART]]], id[cart[V, V]]], composite[
            intersection[composite[complement[COMPOSE], cross[IPD, composite[IPD, FUNPART]]],
             composite[IPD, COMPOSE, cross[Id, FUNPART]]], id[cart[V, V]]]] == 0

In[46]:= % /. Equal→ SetDelayed
```

Theorem. The function **IPD** preserves composition when the right-hand factor is a function.

```
In[47]:= SubstTest[empty, composite[symdif[u, v], id[cart[V, V]]],
          {u -> composite[COMPOSE, cross[IPD, composite[IPD, FUNPART]]],
           v -> composite[IPD, COMPOSE, cross[Id, FUNPART]]}]

Out[47]= equal[composite[COMPOSE, cross[IPD, composite[IPD, FUNPART]]],
           composite[IPD, COMPOSE, cross[Id, FUNPART]]] == True

In[48]:= composite[IPD, COMPOSE, cross[Id, FUNPART]] :=
          composite[COMPOSE, cross[IPD, composite[IPD, FUNPART]]]
```

Corollary.

```
In[49]:= Assoc[composite[IPD, COMPOSE], cross[Id, FUNPART], cross[Id, inverse[FUNPART]]]

Out[49]= composite[IPD, COMPOSE, id[cart[V, FUNS]]] ==
          composite[COMPOSE, cross[IPD, composite[IPD, id[FUNS]]]]

In[50]:= composite[IPD, COMPOSE, id[cart[V, FUNS]]] :=
          composite[COMPOSE, cross[IPD, composite[IPD, id[FUNS]]]]
```

Corollary. The class **image[IPD, FUNS]** is binary-closed under **COMPOSE**.

```
In[51]:= ImageComp[composite[IPD, COMPOSE], id[cart[V, FUNS]], cart[FUNS, V]]

Out[51]= image[COMPOSE, cart[image[IPD, FUNS], image[IPD, FUNS]]] == image[IPD, FUNS]

In[52]:= image[COMPOSE, cart[image[IPD, FUNS], image[IPD, FUNS]]] := image[IPD, FUNS]
```