covariant power functor

Johan G. F. Belinfante
2009 September 12

```
In[1]:= SetDirectory["l:"]; << goedel.09sep10a; << tools.m
```

```
:Package Title: goedel.09sep10a              2009 September 10 at 2:55 p.m.

It is now:  2009 Sep 12 at 14:43

Loading Simplification Rules

TOOLS.M                                    Revised 2009 September 3

weightlimit = 40
```

summary

The function **(IPD ⊗ POWER) ∘ id[range[CATOFUNS]]** is a functor from the category of sets to itself. In the literature on category theory this functor is called the covariant power functor. (A brief discussion is given on page 13 in Mac Lane's book.)

```
In[2]:= "Saunders Mac Lane, Categories for the
          Working Mathematician, Springer-Verlag, New York, 1971.";
```

A corresponding functor from the category **CATORELN** to itself is first derived, and used to derive the results about **CATOFUNS** as a corollary.

abbreviations

It is convenient to introduce some abbreviations. The function **IPD** maps any set **x** to the restriction **f[x]** of the function **IMAGE[x]** to the the power class of the domain of **x**.

```
In[3]:= f[x_] := composite[IMAGE[x], id[P[domain[x]]]]
```

The class **MOR** of morphisms in the category of relations is **range[CATORELN]**.

```
In[4]:= MOR := composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]
```

The power functor for the category of relations is abbreviated to **PF = (IPD ⊗ POWER) ∘ id[MOR]**.

```
In[5]:= PF := composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]
```

simplification rules

Lemma.

```
In[6]:= Assoc[id[cart[V, V]], id[range[CATORELN]], CATORELN]

Out[6]= composite[id[cart[V, V]], CATORELN] == CATORELN
```

```
In[7]:= composite[id[cart[V, V]], CATORELN] := CATORELN
```

Lemma.

```
In[8]:= Assoc[CATORELN, id[domain[CATORELN]], id[cartsq[cart[V, V]]]] // Reverse

Out[8]= composite[CATORELN, id[cart[cart[V, V], cart[V, V]]]] == CATORELN
```

```
In[9]:= composite[CATORELN, id[cart[cart[V, V], cart[V, V]]]] := CATORELN
```

PF preserves CATORELN

In this section if is shown that if an ordered triple of morphisms belongs to **CATORELN**, then so does the ordered triple obtained by applying **PF** to each argument. Each morphism is itself an ordered pair of a relation and a set that contains the range of that relation. So in all, one has to introduce six variables for this. To keep expressions simple, it is convenient to wrap each of the six variables with **setpart**. In the next section all six variables will be eliminated, leading to more transparent expressions.

Lemma. For a relation **u**, the condition **range[u] ⊂ x** can be rewritten as **u ⊂ V × x**.

```
In[10]:= or[and[subclass[range[setpart[u]], setpart[x]],
             subclass[range[setpart[v]], setpart[y]]],
           not[equal[domain[setpart[u]], setpart[y]]],
           not[subclass[setpart[u], cart[V, setpart[x]]]],
           not[subclass[setpart[v], cart[V, setpart[y]]]]] // NotNotTest

Out[10]= or[and[subclass[range[setpart[u]], setpart[x]],
             subclass[range[setpart[v]], setpart[y]]],
           not[equal[domain[setpart[u]], setpart[y]]],
           not[subclass[setpart[u], cart[V, setpart[x]]]],
           not[subclass[setpart[v], cart[V, setpart[y]]]]] == True
```

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

Technical Lemma.

```
In[12]:= IMAGE[image[V, x]] // RelnNormality

Out[12]= IMAGE[image[V, x]] ==
          union[cart[V, intersection[complement[image[V, x]], set[0]]], cart[set[0], set[0]]]
```

```
In[13]:= IMAGE[image[V, x_]] :=
          union[cart[V, intersection[complement[image[V, x]], set[0]]], cart[set[0], set[0]]]
```

Lemma. The function **IPD** preserves composition of relations **u** and **v** if **range[v] ⊂ domain[u]**.

```
In[14]:= Map[implies[subclass[range[setpart[v]], domain[setpart[u]]],
            equal[#, composite[IMAGE[setpart[u]], id[P[domain[setpart[u]]]], IMAGE[setpart[v]],
              id[P[domain[setpart[v]]]]]] &, ApComp[composite[IPD, COMPOSE],
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
           pair[setpart[u], setpart[v]]]]

Out[14]= or[equal[composite[IMAGE[setpart[u]], IMAGE[setpart[v]],
             id[P[image[inverse[setpart[v]], domain[setpart[u]]]]]],
            composite[IMAGE[setpart[u]], id[P[domain[setpart[u]]]],
             IMAGE[setpart[v]], id[P[domain[setpart[v]]]]],
           not[subclass[range[setpart[v]], domain[setpart[u]]]]] == True

In[15]:= or[equal[composite[IMAGE[setpart[u_]], IMAGE[setpart[v_]],
             id[P[image[inverse[setpart[v_]], domain[setpart[u_]]]]]],
            composite[IMAGE[setpart[u_]], id[P[domain[setpart[u_]]]],
             IMAGE[setpart[v_]], id[P[domain[setpart[v_]]]]],
           not[subclass[range[setpart[v_]], domain[setpart[u_]]]]] := True
```

Lemma. The preceding lemmas are combined into a single statement.

```
In[16]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 -> subclass[setpart[v], cart[V, domain[setpart[u]]]],
             p2 -> subclass[range[setpart[v]], domain[setpart[u]]],
             p3 -> equal[composite[IMAGE[setpart[u]], IMAGE[setpart[v]],
                id[P[image[inverse[setpart[v]], domain[setpart[u]]]]]],
               composite[IMAGE[setpart[u]], id[P[domain[setpart[u]]]],
                IMAGE[setpart[v]], id[P[domain[setpart[v]]]]]]}]] // Reverse

Out[16]= or[equal[composite[IMAGE[setpart[u]], IMAGE[setpart[v]],
             id[P[image[inverse[setpart[v]], domain[setpart[u]]]]]],
            composite[IMAGE[setpart[u]], id[P[domain[setpart[u]]]],
             IMAGE[setpart[v]], id[P[domain[setpart[v]]]]],
           not[subclass[setpart[v], cart[V, domain[setpart[u]]]]]] == True

In[17]:= or[equal[composite[IMAGE[setpart[u_]], IMAGE[setpart[v_]],
             id[P[image[inverse[setpart[v_]], domain[setpart[u_]]]]]],
            composite[IMAGE[setpart[u_]], id[P[domain[setpart[u_]]]],
             IMAGE[setpart[v_]], id[P[domain[setpart[v_]]]]],
           not[subclass[setpart[v_], cart[V, domain[setpart[u_]]]]]] := True
```

The following lemma takes a minute or so. One needs to be patient.

Lemma. If an ordered triple of morphisms belongs to **CATORELN**, then so does the ordered triple obtained by applying **PF** to each argument.

```
In[18]:= implies[member[pair[pair[pair[setpart[u], setpart[x]], pair[setpart[v], setpart[y]]],
             pair[setpart[w], setpart[z]]], CATORELN],
           member[pair[pair[pair[f[setpart[u]], P[setpart[x]]], pair[f[setpart[v]],
              P[setpart[y]]]], pair[f[setpart[w]], P[setpart[z]]]], CATORELN]] // NotNotTest

Out[18]= or[and[equal[composite[IMAGE[setpart[w]], id[P[domain[setpart[w]]]]],
            composite[IMAGE[setpart[u]], id[P[domain[setpart[u]]]],
             IMAGE[setpart[v]], id[P[domain[setpart[v]]]]]],
           subclass[range[setpart[u]], setpart[x]], subclass[range[setpart[v]], setpart[y]]],
          not[equal[composite[setpart[u], setpart[v]], setpart[w]]],
          not[equal[domain[setpart[u]], setpart[y]]],
          not[equal[setpart[x], setpart[z]]],
          not[subclass[setpart[u], cart[V, setpart[x]]]],
          not[subclass[setpart[v], cart[V, setpart[y]]]]] == True

In[19]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

eliminating variables

In this section all six variables introduced in the preceding section are eliminated, yielding a simple-variable-free inclusion.

Lemma. Temporary membership rule.

```
In[20]:= (member[pair[setpart[x], P[setpart[y]]], composite[z, funpart[t]]] // AssertTest) /.
          t → IPD

Out[20]= member[pair[setpart[x], P[setpart[y]]], composite[z, IPD]] == member[
           pair[composite[IMAGE[setpart[x]], id[P[domain[setpart[x]]]]], P[setpart[y]]], z]

In[21]:= member[pair[setpart[x_], P[setpart[y_]]], composite[z_, IPD]] := member[
           pair[composite[IMAGE[setpart[x]], id[P[domain[setpart[x]]]]], P[setpart[y]]], z]
```

Lemma. (Eliminating variables is done in a standard fashion, but here it is best to omit the customary mapping with **complement**, because doing so is not only time consuming, but rewrite rules unexpectedly spoil the outcome.)

```
In[22]:= SubstTest[class, pair[pair[pair[u, x], pair[v, y]], pair[w, z]],
          implies[member[pair[pair[pair[setpart[u], setpart[x]], pair[setpart[v], setpart[y]]],
             pair[setpart[w], setpart[z]]], s],
           member[pair[pair[pair[setpart[u], setpart[x]], pair[setpart[v], setpart[y]]],
             pair[setpart[w], setpart[z]]], t]],
          {s → CATORELN, t -> composite[cross[inverse[IPD], inverse[POWER]],
             CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]]}]

Out[22]= union[composite[cross[inverse[IPD], inverse[POWER]],
            CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]],
           composite[id[cart[V, V]], complement[CATORELN], id[cart[cart[V, V], cart[V, V]]]]] ==
          cart[cart[cart[V, V], cart[V, V]], cart[V, V]]

In[23]:= % /. Equal→ SetDelayed
```

Lemma. (The intention of this lemma is to serve as a replacement for the mapping with **complement**.)

```
In[24]:= or[not[equal[cart[u, v], union[x, composite[id[v], complement[y], id[u]]]]],
           subclass[composite[id[v], y, id[u]], x]] // AssertTest

Out[24]= or[not[equal[cart[u, v], union[x, composite[id[v], complement[y], id[u]]]]],
           subclass[composite[id[v], y, id[u]], x]] == True

In[25]:= or[not[equal[cart[u_, v_], union[x_, composite[id[v_], complement[y_], id[u_]]]]],
           subclass[composite[id[v_], y_, id[u_]], x_]] := True
```

Theorem. The function **IPD ⊗ POWER** preserves **CATORELN**.

```
In[26]:= SubstTest[implies, equal[cart[u, v], union[x, composite[id[v], complement[y], id[u]]]],
           subclass[composite[id[v], y, id[u]], x], {u -> cart[cart[V, V], cart[V, V]],
            v -> cart[V, V], x -> composite[cross[inverse[IPD], inverse[POWER]], CATORELN,
              cross[cross[IPD, POWER], cross[IPD, POWER]]], y → CATORELN}] // Reverse

Out[26]= subclass[CATORELN, composite[cross[inverse[IPD], inverse[POWER]],
            CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]]] == True

In[27]:= subclass[CATORELN, composite[cross[inverse[IPD], inverse[POWER]],
            CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]]] := True
```

Corollary. An intertwine-style inclusion

```
In[28]:= SubstTest[implies, subclass[u, v], subclass[composite[t, u], composite[t, v]],
           {t → cross[IPD, POWER], u → CATORELN,
            v -> composite[cross[inverse[IPD], inverse[POWER]], CATORELN,
              cross[cross[IPD, POWER], cross[IPD, POWER]]]}] // Reverse

Out[28]= subclass[composite[cross[IPD, POWER], CATORELN],
           composite[CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]]] == True

In[29]:= % /. Equal→ SetDelayed
```

Corollary. A similar intertwine-style inclusion involving **CATORELN** and **PF**.

```
In[30]:= SubstTest[subclass, t, intersection[u, v], {t -> composite[cross[IPD, POWER], CATORELN],
            u -> composite[CATORELN, cross[cross[IPD, POWER], cross[IPD, POWER]]],
            v → cart[cartsq[MOR], V]}] // Reverse

Out[30]= subclass[composite[cross[IPD, POWER], CATORELN],
           composite[CATORELN, cross[composite[cross[IPD, POWER],
              id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], composite[
              cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]] == True

In[31]:= % /. Equal→ SetDelayed
```

Restatement.

```
In[32]:= subclass[composite[PF, CATORELN], composite[CATORELN, cross[PF, PF]]]

Out[32]= True
```

an intertwine equation

In this section the inclusion of the last section is strengthened to an equation. To do so, one needs an inclusion relating the domains of the two functions that occur in the inclusion. The domain of the function **PF ∘ CATORELN** is the same as the domain of **CATORELN**. The domain of the function **CATORELN ∘ (PF ⊗ PF)** is more difficult to deal with, but all one needs is an inclusion. The trick here is to show that for ternary relations, the inclusion **x ⊂ y** follows if one can show that **domain[domain[dif[x, y]]** is empty.

Lemma. Simplification rule.

```
In[33]:= composite[inverse[PF], id[MOR]] // DoubleInverse

Out[33]= composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
           cross[inverse[IPD], inverse[POWER]],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]] ==
         composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
           cross[inverse[IPD], inverse[POWER]]]

In[34]:= % /. Equal→ SetDelayed
```

It helps here to clear the **simplify** flag.

```
In[35]:= simplify = False; cond = True;
```

Lemma.

```
In[36]:= SubstTest[empty, domain[domain[dif[u, v]]],
          {u -> composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             intersection[composite[inverse[SECOND], inverse[POWER]], composite[inverse[FIRST],
               inverse[IPD], id[P[cart[V, V]]], inverse[IMAGE[SECOND]], inverse[S]]],
             IMAGE[FIRST], id[P[cart[V, V]]], intersection[composite[IPD, FIRST],
              composite[inverse[IMAGE[SECOND]], inverse[S], POWER, SECOND]],
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]],
           v -> composite[inverse[SECOND], IMAGE[FIRST], FIRST]}]

Out[36]= subclass[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
            intersection[composite[inverse[SECOND], inverse[POWER]], composite[inverse[FIRST],
              inverse[IPD], id[P[cart[V, V]]], inverse[IMAGE[SECOND]], inverse[S]]],
            IMAGE[FIRST], id[P[cart[V, V]]], intersection[composite[IPD, FIRST],
             composite[inverse[IMAGE[SECOND]], inverse[S], POWER, SECOND]],
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]],
           composite[inverse[SECOND], IMAGE[FIRST], FIRST]] == True

In[37]:= % /. Equal→ SetDelayed
```

Theorem. (It is not clear how best to orient this rewrite rule.)

```
In[38]:= SubstTest[implies, and[subclass[x, y], FUNCTION[y], subclass[domain[y], domain[x]]],
           equal[x, y], {x -> composite[cross[IPD, POWER], CATORELN],
            y -> composite[CATORELN, cross[PF, PF]]}] // Reverse

Out[38]= equal[composite[CATORELN, cross[
             composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]],
             composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]],
           composite[cross[IPD, POWER], CATORELN]] == True

In[39]:= composite[CATORELN, cross[composite[cross[IPD, POWER],
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], composite[
            cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]] :=
          composite[cross[IPD, POWER], CATORELN]
```

Restatement.

```
In[40]:= intertwine[CATORELN, cross[PF, PF], PF]

Out[40]= True
```

Theorem. The function **PF** is a functor from the category of relations to itself.

```
In[41]:= SubstTest[implies,
           and[FUNCTION[t], equal[domain[t], range[u]], subclass[image[t, ids[u]], ids[v]],
            equal[composite[t, u], composite[v, cross[t, t]]]],
           functor[t, u, v], {t → composite[cross[IPD, POWER], id[range[CATORELN]]],
            u → CATORELN, v → CATORELN}] // Reverse

Out[41]= functor[composite[cross[IPD, POWER],
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], CATORELN, CATORELN] == True

In[42]:= functor[composite[cross[IPD, POWER],
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], CATORELN, CATORELN] := True
```

category of sets

Some simplification rules are derived in this section to facilitate relating the category of relations to the category of sets.

Lemma.

```
In[43]:= subclass[IPD, cart[V, FUNS]] // AssertTest

Out[43]= subclass[IPD, cart[V, FUNS]] == True

In[44]:= subclass[IPD, cart[V, FUNS]] := True
```

Theorem. Simplification rule.

```
In[45]:= equal[composite[id[FUNS], IPD], IPD]

Out[45]= True
```

```
In[46]:= composite[id[FUNS], IPD] := IPD
```

Corollary. Automatic replacement of **CATORELN** by **CATOFUNS**.

```
In[47]:= Assoc[CATORELN, id[cartsq[cart[FUNS, V]]],
          cross[composite[cross[IPD, u], v], composite[cross[IPD, x], y]]]

Out[47]= composite[CATORELN, cross[composite[cross[IPD, u], v], composite[cross[IPD, x], y]]] ==
          composite[CATOFUNS, cross[composite[cross[IPD, u], v], composite[cross[IPD, x], y]]]

In[48]:= composite[CATORELN,
           cross[composite[cross[IPD, u_], v_], composite[cross[IPD, x_], y_]]] :=
          composite[CATOFUNS, cross[composite[cross[IPD, u], v], composite[cross[IPD, x], y]]]
```

the covariant power functor

Lemma. Intertwine equation for the category of sets.

```
In[49]:= Assoc[CATORELN, cross[composite[cross[IPD, POWER], id[range[CATORELN]]],
           composite[cross[IPD, POWER], id[range[CATORELN]]]], id[cartsq[cart[FUNS, V]]]]

Out[49]= composite[CATOFUNS,
           cross[composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]],
            composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]]]] ==
          composite[cross[IPD, POWER], CATOFUNS]

In[50]:= composite[CATOFUNS,
           cross[composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]],
            composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]]]] :=
          composite[cross[IPD, POWER], CATOFUNS]
```

Theorem. The covariant power functor for the category of sets.

```
In[51]:= SubstTest[implies,
           and[FUNCTION[t], equal[domain[t], range[u]], subclass[image[t, ids[u]], ids[v]],
            equal[composite[t, u], composite[v, cross[t, t]]]],
           functor[t, u, v], {t → composite[cross[IPD, POWER], id[range[CATOFUNS]]],
            u → CATOFUNS, v → CATOFUNS}] // Reverse

Out[51]= or[functor[composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]],
            CATOFUNS, CATOFUNS], not[equal[composite[cross[IPD, POWER], CATOFUNS], composite[
              cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]], CATOFUNS]]] == True

In[52]:= functor[composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[FUNS]]]],
           CATOFUNS, CATOFUNS] := True
```

Lemma.

```
In[53]:= Assoc[CATORELN, id[cartsq[cart[FUNS, V]]], cross[composite[cross[IPD, POWER],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], composite[cross[IPD, POWER],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]] // Reverse

Out[53]= composite[CATOFUNS, cross[
           composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]],
           composite[cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]] ==
         composite[cross[IPD, POWER], CATORELN]

In[54]:= composite[CATOFUNS, cross[composite[cross[IPD, POWER],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]], composite[
           cross[IPD, POWER], id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]] :=
         composite[cross[IPD, POWER], CATORELN]
```

Theorem. The power functor from **CATORELN** to **CATOFUNS**.

```
In[55]:= SubstTest[implies,
           and[FUNCTION[t], equal[domain[t], range[u]], subclass[image[t, ids[u]], ids[v]],
            equal[composite[t, u], composite[v, cross[t, t]]]],
           functor[t, u, v], {t → composite[cross[IPD, POWER], id[range[CATORELN]]],
            u → CATORELN, v → CATOFUNS}] // Reverse

Out[55]= functor[composite[cross[IPD, POWER],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], CATORELN, CATOFUNS] == True

In[56]:= functor[composite[cross[IPD, POWER],
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], CATORELN, CATOFUNS] := True
```