# functors

Johan G. F. Belinfante
2009 January 16

```
In[1]:= SetDirectory["l:"]; << goedel.09jan14x;<< tools.m

        :Package Title: goedel.09jan14x        2009 January 14 at 7:10 a.m.

        It is now:  2009 Jan 16 at 12:39

        Loading Simplification Rules

        TOOLS.M                                        Revised 2008 December 26

        weightlimit = 40
```

## summary

Functors are functions that preserve partial binary operations and identity elements, that is, their (two-sided) neutral elements. Although functors are chiefly of interest for categories, this concept can be defined for arbitrary classes. Because the intended application is to categories, which can be proper classes, the concept of functor must be introduced via a predicate rather than as a class of functions. In this notebook basic properties of a predicate **functor[w, x, y]** are derived.

```
In[2]:= Begin["Goedel`Private`"];
```

```
In[3]:= ?? functor

        functor[w,x,y] is the statement that w is a functor from x to y
```

The predicate **functor[w, x, y]** is defined in the **GOEDEL** program by the following **class**-wrapped rule:

```
In[4]:= FirstMatch[class[t_, HoldPattern[functor[w_, x_, y_]]]]

Out[4]= class[t_, functor[w_, x_, y_]] := class[t, and[
          FUNCTION[w], equal[domain[w], range[x]], equal[composite[w, x],
          composite[y, cross[w, w]]], subclass[image[w, ids[x]], ids[y]]]]
```

The reason for wrapping the definition of the predicate **functor** with **class** is to prevent this definition from forever being automatically expanded into its constituent four literals. After all, the whole purpose of definitions is for them to serve as abbreviations for complex expressions. Nonetheless, to facilitate reasoning about functors it is convenient make available also unwrapped versions of its definition. The needed rewrite rules are easily derived using **AssertTest**, as are some basic examples. Beyond this, a number of general rewrite rules are derived in this notebook that do not require assuming any specialized properties about the classes **x** and **y**.

## unwrapping the definition

Deriving unwrapped rules corresponding to the definition of **functor** is time-consuming. A significant speedup (by a factor of almost three) can be achieved by clearing the **simplify** flag. The **cond** and **equality** flags have little effect on execution time. In one direction, one needs four rewrite rules corresponding to each of the four literals in the definition of functor. In the opposite direction, only one rewrite rule is needed.

```
In[5]:= simplify = False;
```

Theorem. Functors are functions.

```
In[6]:= implies[functor[w, x, y], FUNCTION[w]] // AssertTest

Out[6]= or[FUNCTION[w], not[functor[w, x, y]]] == True
```

```
In[7]:= or[FUNCTION[w_], not[functor[w_, x_, y_]]] := True
```

Theorem. A functor **w** from **x** to **y** is defined on the range of **x**.

```
In[8]:= implies[functor[w, x, y], equal[domain[w], range[x]]] // AssertTest

Out[8]= or[equal[domain[w], range[x]], not[functor[w, x, y]]] == True
```

```
In[9]:= or[equal[domain[w_], range[x_]], not[functor[w_, x_, y_]]] := True
```

Theorem. Functors preserve composition.

```
In[10]:= implies[functor[w, x, y],
           equal[composite[w, x], composite[y, cross[w, w]]]] // AssertTest

Out[10]= or[equal[composite[w, x], composite[y, cross[w, w]]], not[functor[w, x, y]]] == True
```

```
In[11]:= or[equal[composite[w_, x_], composite[y_, cross[w_, w_]]],
           not[functor[w_, x_, y_]]] := True
```

Theorem. Functors preserve identities.

```
In[12]:= implies[functor[w, x, y], subclass[image[w, ids[x]], ids[y]]] // AssertTest

Out[12]= or[not[functor[w, x, y]], subclass[image[w, ids[x]], ids[y]]] == True
```

```
In[13]:= or[not[functor[w_, x_, y_]], subclass[image[w_, ids[x_]], ids[y_]]] := True
```

Theorem. Implication in the opposite direction.

```
In[14]:= Map[implies[#, functor[w, x, y]] &, functor[w, x, y] // AssertTest] // Reverse

Out[14]= or[functor[w, x, y], not[equal[composite[w, x], composite[y, cross[w, w]]]],
           not[equal[domain[w], range[x]]], not[FUNCTION[w]],
           not[subclass[image[w, ids[x]], ids[y]]]] == True
```

```
In[15]:= or[functor[w_, x_, y_], not[equal[composite[w_, x_], composite[y_, cross[w_, w_]]]],
           not[equal[domain[w_], range[x_]]], not[FUNCTION[w_]],
           not[subclass[image[w_, ids[x_]], ids[y_]]]] := True
```

## identity functors

Theorem. The identity function **id[range[x]]** is a functor from **x** to itself provided that the domain of **x** is contained in the cartesian square of its range. (This condition is true by definition when **x** is a category.)

```
In[16]:= functor[id[range[x]], x, x] // AssertTest
Out[16]= functor[id[range[x]], x, x] == subclass[domain[x], cart[range[x], range[x]]]

In[17]:= functor[id[range[x_]], x_, x_] := subclass[domain[x], cart[range[x], range[x]]]
```

## empty functors

Theorem. The empty function is a functor from **0** to any class. More generally:

```
In[18]:= functor[0, x, y] // AssertTest
Out[18]= functor[0, x, y] == equal[0, domain[x]]

In[19]:= functor[0, x_, y_] := equal[0, domain[x]]
```

Theorem. The only functor from **0** to any class is **0**.

```
In[20]:= implies[functor[w, 0, y], equal[0, w]] // AssertTest
Out[20]= or[equal[0, w], not[functor[w, 0, y]]] == True

In[21]:= or[equal[0, w_], not[functor[w_, 0, y_]]] := True
```

Theorem. The only functor to **0** from any class is **0**.

```
In[22]:= implies[functor[w, x, 0], equal[0, w]] // AssertTest
Out[22]= or[equal[0, w], not[functor[w, x, 0]]] == True

In[23]:= or[equal[0, w_], not[functor[w_, x_, 0]]] := True
```

## general properties of functors

Some general properties about functors are derived in this section, making use of the rewrite rules corresponding to the definition. Since **AssertTest** is not used for any of these, execution time is negligible. The **simplify** flag does not need to be cleared for the remainder of this notebook.

```
In[24]:= simplify = True;
```

Lemma.

```
In[25]:= SubstTest[implies, equal[u, v], equal[domain[u], domain[v]],
           {u -> composite[w, x], v -> composite[y, cross[w, w]]}] // Reverse

Out[25]= or[equal[composite[inverse[w], domain[y], w], image[inverse[x], domain[w]]],
           not[equal[composite[w, x], composite[y, cross[w, w]]]]] == True
```

```
In[26]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Functors preserve composability.

```
In[27]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            implies[p1, p4], implies[and[p3, p4], p5], not[implies[p1, p5]],
            {p1 → functor[w, x, y], p2 -> equal[composite[w, x], composite[y, cross[w, w]]],
             p3 -> equal[composite[inverse[w], domain[y], w], image[inverse[x], domain[w]]],
             p4 → equal[domain[w], range[x]],
             p5 → equal[composite[inverse[w], domain[y], w], domain[x]]}]] // Reverse

Out[27]= or[equal[composite[inverse[w], domain[y], w], domain[x]], not[functor[w, x, y]]] == True
```

```
In[28]:= or[equal[composite[inverse[w_], domain[y_], w_], domain[x_]],
           not[functor[w_, x_, y_]]] := True
```

Lemma.

```
In[29]:= SubstTest[implies, equal[u, v], equal[range[u], range[v]],
           {u -> composite[w, x], v -> composite[y, cross[w, w]]}] // Reverse

Out[29]= or[equal[image[w, range[x]], image[y, cart[range[w], range[w]]]],
           not[equal[composite[w, x], composite[y, cross[w, w]]]]] == True
```

```
In[30]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. The range of a functor **w** from **x** to **y** is closed under **y**.

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            implies[p1, p4], implies[and[p3, p4], p5], not[implies[p1, p5]],
            {p1 -> functor[w, x, y], p2 -> equal[composite[w, x], composite[y, cross[w, w]]],
             p3 -> equal[image[w, range[x]], image[y, cart[range[w], range[w]]]],
             p4 → equal[domain[w], range[x]],
             p5 → equal[image[y, cart[range[w], range[w]]], range[w]]}]] // Reverse

Out[31]= or[equal[image[y, cart[range[w], range[w]]], range[w]], not[functor[w, x, y]]] == True
```

```
In[32]:= or[not[functor[w_, x_, y_]],
           equal[image[y_, cart[range[w_], range[w_]]], range[w_]]] := True
```

Corollary. The range of a functor **w** from **x** to **y** is contained in the range of **y**.

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
           {p1 → functor[w, x, y], p2 -> equal[image[y, cart[range[w], range[w]]], range[w]],
            p3 → subclass[range[w], range[y]]}]] // Reverse

Out[33]= or[not[functor[w, x, y]], subclass[range[w], range[y]]] == True

In[34]:= or[not[functor[w_, x_, y_]], subclass[range[w_], range[y_]]] := True
```

In other words, **w** is a mapping from **range[x]** to **range[y]**. In general **w**, **x** and **y** may all be proper classes.

---

## dom is preserved

Lemma.

```
In[35]:= SubstTest[implies, subclass[s, t],
           subclass[composite[s, u], composite[t, u]], {s -> composite[id[ids[x]], inverse[w]],
            t -> composite[inverse[w], id[ids[y]]], u → composite[domain[y], w]}] // Reverse

Out[35]= or[not[subclass[image[w, ids[x]], ids[y]]],
           subclass[composite[id[ids[x]], inverse[w], domain[y], w],
            composite[inverse[w], dom[y], w]]] == True

In[36]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Functors preserve dom.

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p1, p4],
           implies[p4, p5], implies[and[p3, p5], p6], not[implies[p1, p6]],
           {p1 → functor[w, x, y], p2 → equal[composite[inverse[w], domain[y], w], domain[x]],
            p3 -> equal[composite[id[ids[x]], inverse[w], domain[y], w], dom[x]],
            p4 → subclass[image[w, ids[x]], ids[y]], p5 → subclass[composite[
               id[ids[x]], inverse[w], domain[y], w], composite[inverse[w], dom[y], w]],
            p6 → subclass[dom[x], composite[inverse[w], dom[y], w]]}]] // Reverse

Out[37]= or[not[functor[w, x, y]], subclass[dom[x], composite[inverse[w], dom[y], w]]] == True

In[38]:= or[not[functor[w_, x_, y_]],
           subclass[dom[x_], composite[inverse[w_], dom[y_], w_]]] := True
```

---

## cod is preserved

Lemma.

```
In[39]:= SubstTest[implies, subclass[s, t], subclass[composite[s, u], composite[t, u]],
           {s -> composite[id[ids[x]], inverse[w]], t -> composite[inverse[w], id[ids[y]]],
            u → composite[inverse[domain[y]], w]}] // Reverse

Out[39]= or[not[subclass[image[w, ids[x]], ids[y]]],
           subclass[composite[id[ids[x]], inverse[w], inverse[domain[y]], w],
            composite[inverse[w], cod[y], w]]] == True

In[40]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Functors preserve cod.

```
In[41]:= Map[not, SubstTest[and, implies[p0, p1], implies[p1, p2], implies[p2, p3],
              implies[p0, p4], implies[p4, p5], implies[and[p3, p5], p6], not[implies[p0, p6]],
              {p0 → functor[w, x, y], p1 -> equal[composite[inverse[w], domain[y], w], domain[x]],
               p2 → equal[composite[inverse[w], inverse[domain[y]], w], inverse[domain[x]]],
               p3 → equal[cod[x], composite[id[ids[x]], inverse[w], inverse[domain[y]], w]],
               p4 → subclass[image[w, ids[x]], ids[y]], p5 → subclass[composite[id[ids[x]],
                  inverse[w], inverse[domain[y]], w], composite[inverse[w], cod[y], w]],
               p6 → subclass[cod[x], composite[inverse[w], cod[y], w]]}]] // Reverse

Out[41]= or[not[functor[w, x, y]], subclass[cod[x], composite[inverse[w], cod[y], w]]] == True

In[42]:= or[not[functor[w_, x_, y_]],
           subclass[cod[x_], composite[inverse[w_], cod[y_], w_]]] := True
```

---

## restricting y to its ternary relational part

Lemma.

```
In[43]:= SubstTest[implies, functor[w, x, t], equal[composite[w, x], composite[t, cross[w, w]]],
           t → composite[y, id[cart[V, V]]]] // Reverse

Out[43]= or[equal[composite[w, x], composite[y, cross[w, w]]],
           not[functor[w, x, composite[y, id[cart[V, V]]]]]] == True

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

Lemma.

```
In[45]:= SubstTest[implies, functor[w, x, t],
           subclass[image[w, ids[x]], ids[t]], t → composite[y, id[cart[V, V]]]] // Reverse

Out[45]= or[not[functor[w, x, composite[y, id[cart[V, V]]]]],
           subclass[image[w, ids[x]], ids[y]]] == True

In[46]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. An implication in one direction.

```
In[47]:= Map[not,
           SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4], implies[p1, p5],
            not[implies[p1, p6]], {p1 -> functor[w, x, composite[y, id[cart[V, V]]]],
             p2 -> equal[composite[w, x], composite[y, cross[w, w]]],
             p3 -> equal[domain[w], range[x]], p4 -> FUNCTION[w],
             p5 -> subclass[image[w, ids[x]], ids[y]], p6 → functor[w, x, y]}]] // Reverse

Out[47]= or[functor[w, x, y], not[functor[w, x, composite[y, id[cart[V, V]]]]]] == True

In[48]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[49]:= SubstTest[implies, and[equal[composite[w, x], composite[t, cross[w, w]]],
            equal[domain[w], range[x]], FUNCTION[w], subclass[image[w, ids[x]], ids[t]]],
           functor[w, x, t], t → composite[y, id[cart[V, V]]]] // Reverse

Out[49]= or[functor[w, x, composite[y, id[cart[V, V]]]],
           not[equal[composite[w, x], composite[y, cross[w, w]]]],
           not[equal[domain[w], range[x]]], not[FUNCTION[w]],
           not[subclass[image[w, ids[x]], ids[y]]]] == True

In[50]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Converse Theorem. Implication in the opposite direction.

```
In[51]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[p1, p4], implies[p1, p5], not[implies[p1, p6]],
            {p1 -> functor[w, x, y], p2 -> equal[composite[w, x], composite[y, cross[w, w]]],
             p3 -> equal[domain[w], range[x]], p4 -> FUNCTION[w],
             p5 -> subclass[image[w, ids[x]], ids[y]],
             p6 → functor[w, x, composite[y, id[cart[V, V]]]]}]] // Reverse

Out[51]= or[functor[w, x, composite[y, id[cart[V, V]]]], not[functor[w, x, y]]] == True

In[52]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Combining the theorem and its converse yields a logical equivalence that can be made into a rewrite rule.

Corollary. The class **y** can be replaced with its ternary relational part in the statement **functor[w, x, y]**.

```
In[53]:= equiv[functor[w, x, composite[y, id[cart[V, V]]]], functor[w, x, y]]

Out[53]= True

In[54]:= functor[w_, x_, composite[y_, id[cart[V, V]]]] := functor[w, x, y]
```

Comment. It should be noted that there is no analogous theorem for the variable **x**. One cannot replace **x** by its ternary relational part in the statement **functor[w, x, y]** because the condition that **domain[w]** be equal to **range[x]** is not preserved when **x** is replaced with **composite[x, id[cart[V, V]]]**. One is of course free to replace **x** with its binary relational part **composite[Id, x]**.

## flip rule

Lemma.

```
In[55]:= SubstTest[implies, and[equal[composite[v, cross[w, w]], composite[w, u]],
           equal[domain[w], range[u]], FUNCTION[w], subclass[image[w, ids[u]], ids[v]]],
         functor[w, u, v], {u → flip[x], v → y}] // Reverse

Out[55]= or[functor[w, composite[x, SWAP], y],
         not[equal[composite[y, cross[w, w]], composite[w, x, SWAP]]],
         not[equal[domain[w], image[x, cart[V, V]]]], not[FUNCTION[w]],
         not[subclass[image[w, ids[x]], ids[y]]]] == True

In[56]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[57]:= Map[implies[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]], #] &,
         SubstTest[equal, flip[u], flip[v], {u -> composite[flip[y], cross[w, w]],
           v -> composite[w, x, id[cart[V, V]]]}]] // Reverse

Out[57]= or[equal[composite[y, cross[w, w]], composite[w, x, SWAP]],
         not[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]]] == True

In[58]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[59]:= SubstTest[implies, functor[w, t, y],
         equal[domain[w], range[t]], t → composite[x, id[cart[V, V]]]] // Reverse

Out[59]= or[equal[domain[w], image[x, cart[V, V]]],
         not[functor[w, composite[x, id[cart[V, V]]], y]]] == True

In[60]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[61]:= SubstTest[implies, functor[w, u, v], subclass[image[w, ids[u]], ids[v]],
         {u → composite[x, id[cart[V, V]]], v → flip[y]}] // Reverse

Out[61]= or[not[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]],
         subclass[image[w, ids[x]], ids[y]]] == True

In[62]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[63]:= Map[not,
           SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4], implies[p1, p5],
            not[implies[p1, p6]], {p1 → functor[w, composite[x, id[cart[V, V]]], flip[y]],
             p2 → equal[composite[w, x, SWAP], composite[y, cross[w, w]]],
             p3 → equal[domain[w], image[x, cart[V, V]]], p4 → FUNCTION[w],
             p5 → subclass[image[w, ids[x]], ids[y]], p6 → functor[w, flip[x], y]}]] // Reverse

Out[63]= or[functor[w, composite[x, SWAP], y],
           not[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]]] == True

In[64]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The rest of this section is devoted to the reverse implication.

Lemma.

```
In[65]:= SubstTest[implies, and[equal[composite[v, cross[w, w]], composite[w, u]],
            equal[domain[w], range[u]], FUNCTION[w], subclass[image[w, ids[u]], ids[v]]],
           functor[w, u, v], {u → composite[x, id[cart[V, V]]], v → flip[y]}] // Reverse

Out[65]= or[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]],
           not[equal[composite[w, x, id[cart[V, V]]], composite[y, SWAP, cross[w, w]]]],
           not[equal[domain[w], image[x, cart[V, V]]]], not[FUNCTION[w]],
           not[subclass[image[w, ids[x]], ids[y]]]] == True

In[66]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[67]:= Map[implies[functor[w, flip[x], y], #] &, SubstTest[equal, flip[u], flip[v],
             {u -> composite[y, cross[w, w]], v -> composite[w, x, SWAP]}]] // Reverse

Out[67]= or[equal[composite[w, x, id[cart[V, V]]], composite[y, SWAP, cross[w, w]]],
           not[functor[w, composite[x, SWAP], y]]] == True

In[68]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[69]:= SubstTest[implies, functor[w, t, y], equal[domain[w], range[t]], t → flip[x]] // Reverse

Out[69]= or[equal[domain[w], image[x, cart[V, V]]],
           not[functor[w, composite[x, SWAP], y]]] == True

In[70]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[71]:= SubstTest[implies, functor[w, u, v],
           subclass[image[w, ids[u]], ids[v]], {u → flip[x], v → y}] // Reverse

Out[71]= or[not[functor[w, composite[x, SWAP], y]], subclass[image[w, ids[x]], ids[y]]] == True

In[72]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

```
In[73]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4],
           implies[p1, p5], not[implies[p1, p6]], {p1 -> functor[w, composite[x, SWAP], y],
            p2 -> equal[composite[w, x, id[cart[V, V]]], composite[y, SWAP, cross[w, w]]],
            p3 -> equal[domain[w], image[x, cart[V, V]]],
            p4 → FUNCTION[w], p5 -> subclass[image[w, ids[x]], ids[y]],
            p6 -> functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]}]] // Reverse

Out[73]= or[functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]],
          not[functor[w, composite[x, SWAP], y]]] == True

In[74]:= (% /. {w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Combining the theorem and its converse yields a logical equivalence that can be made into a rewrite rule.

Corollary. Flip rule for functors.

```
In[75]:= equiv[functor[w, composite[x, SWAP], y],
           functor[w, composite[x, id[cart[V, V]]], flip[y]]]

Out[75]= True

In[76]:= functor[w_, composite[x_, SWAP], y_] :=
           functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]
```

---

## composite functors

In this section it is shown that the composite of a functor from **y** to **z** and a functor from **x** to **y** is a functor from **x** to **z**. For convenience, lemmas are derived for each of the four defining properties of a functor, plus a fifth lemma to put all this information together.

Lemma 1.

```
In[77]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[functor[v, x, y], functor[u, y, z]],
            p2 → FUNCTION[u], p3 → FUNCTION[v], p4 → FUNCTION[composite[u, v]]}]] // Reverse

Out[77]= or[FUNCTION[composite[u, v]], not[functor[u, y, z]], not[functor[v, x, y]]] == True

In[78]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma 2.

```
In[79]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p1, p2, p3], p4], implies[and[p1, p2, p3, p4], p5], not[implies[p1, p5]],
           {p1 → and[functor[v, x, y], functor[u, y, z]], p2 → equal[domain[u], range[y]],
            p3 → equal[domain[v], range[x]], p4 → subclass[range[v], domain[u]],
            p5 → equal[domain[composite[u, v]], range[x]]}]] // Reverse

Out[79]= or[equal[image[inverse[v], domain[u]], range[x]],
          not[functor[u, y, z]], not[functor[v, x, y]]] == True
```

```
In[80]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma 3.

```
In[81]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p1, p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[functor[v, x, y], functor[u, y, z]],
            p2 → equal[composite[u, y], composite[z, cross[u, u]]], p3 →
             equal[composite[v, x], composite[y, cross[v, v]]], p4 → equal[composite[u, v, x],
              composite[z, cross[composite[u, v], composite[u, v]]]]}]] // Reverse

Out[81]= or[equal[composite[z, cross[composite[u, v], composite[u, v]]], composite[u, v, x]],
          not[functor[u, y, z]], not[functor[v, x, y]]] == True

In[82]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma 4.

```
In[83]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p1, p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[functor[v, x, y], functor[u, y, z]],
            p2 → subclass[image[u, ids[y]], ids[z]], p3 → subclass[image[v, ids[x]], ids[y]],
            p4 → subclass[image[composite[u, v], ids[x]], ids[z]]}]] // Reverse

Out[83]= or[not[functor[u, y, z]], not[functor[v, x, y]],
          subclass[image[u, image[v, ids[x]]], ids[z]]] == True

In[84]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma 5.

```
In[85]:= SubstTest[or, functor[t, x, z], not[equal[composite[t, x], composite[z, cross[t, t]]]],
          not[equal[domain[t], range[x]]], not[FUNCTION[t]],
          not[subclass[image[t, ids[x]], ids[z]]], t → composite[u, v]] // Reverse

Out[85]= or[functor[composite[u, v], x, z],
          not[equal[composite[z, cross[composite[u, v], composite[u, v]]], composite[u, v, x]]],
          not[equal[image[inverse[v], domain[u]], range[x]]], not[FUNCTION[composite[u, v]]],
          not[subclass[image[u, image[v, ids[x]]], ids[z]]]] == True

In[86]:= (% /. {u → u_, v → v_, x → x_, z → z_}) /. Equal → SetDelayed
```

Theorem. If **u** is a functor from **y** to **z**, and if **v** is a functor from **x** to **y**, then **composite[u, v]** is a functor from **x** to **z**.

```
In[87]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4],
           implies[p1, p5], implies[and[p2, p3, p4, p5], p6], not[implies[p1, p6]],
           {p1 -> and[functor[v, x, y], functor[u, y, z]], p2 -> FUNCTION[composite[u, v]],
            p3 -> equal[domain[composite[u, v]], range[x]], p4 ->
             equal[composite[u, v, x], composite[z, cross[composite[u, v], composite[u, v]]]],
            p5 -> subclass[image[u, image[v, ids[x]]], ids[z]],
            p6 -> functor[composite[u, v], x, z]}]] // Reverse

Out[87]= or[functor[composite[u, v], x, z], not[functor[u, y, z]], not[functor[v, x, y]]] == True
```

```
In[88]:= or[functor[composite[u_, v_], x_, z_],
            not[functor[u_, y_, z_]], not[functor[v_, x_, y_]]] := True
```

## comments

The definition of functor resembles that of binary homomorphism. Neither definition makes any explicit assumption about the nature of the classes **x** and **y**, but the conditions on the function relating **x** to **y** differ because of the differing natures of their intended applications.

```
In[89]:= FirstMatch[class[t_, member[w_, HoldPattern[binhom[x_, y_]]]]]
```

```
Out[89]= class[w_, member[z_, binhom[x_, y_]]] := class[w, and[member[z, map[fix[domain[
           x]], fix[domain[y]]]], equal[composite[z, x], composite[y, cross[z, z]]]]]
```

Aside from the new condition that the class **ids** of identities be preserved for functors, it should be noted that the occurrence of **fix[domain[x]]** in the definition of binary homomorphism has been replaced with **range[x]** in the definition of functor. One needs to make this change because for the intended applications of functors, the class **x** will typically be a partial binary operation, that is, a function whose domain need not be a cartesian square, but is only required to be contained in the cartesian square of its range. On the other hand, for the intended application of the concept of binary homomorphism, the class **x** will typically be a binary operation, whose domain is a cartesian square and whose range is a class whose cartesian square may be properly contained in its domain.