case-wrapped definition of functor

Johan G. F. Belinfante
2011 December 27

```
In[1]:= SetDirectory["l:"]; << goedel.11dec27a
```

```
:Package Title: goedel.11dec27a              2011 December 27 at 8:45 a.m.

Loading takes about thirteen minutes, half that time due to builtin pauses.

It is now:  2011 Dec 27 at 9:47

Loading Simplification Rules

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

weightlimit = 40

Loading completed.

It is now:  2011 Dec 27 at 10:0
```

summary

The definition of functor that was introduced 2009 January 16 in the notebook **functor.nb** is too strict for category theory. A weaker concept of functor is studied here, replacing the equation **w ∘ x = y ∘ (w ⊗ w)** with the inclusion **w ∘ x ⊂ y ∘ (w ⊗ w)**. To distinguish these two concepts, the term **strong functor** will be used when the equation holds, and the term **functor** will be used when only an inclusion is assumed. All the old theorems have been retained, except that the term **functor** in them has been replaced with **strongfunctor** to avoid confusion.

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 is here 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 new predicate **functor[w, x, y]** are derived. Many of the theorems for strong functors generalize to the new weaker definition of functor.

strong functors

Originally the **strongfunctor** predicate was defined in the **GOEDEL** program by a **class**-wrapped rule:

class[t_, strongfunctor[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]]]]

Wrapping the definition of the predicate **strongfunctor** with **class** prevents the definition from forever being automatically expanded into its four constituent literals. After all, the whole purpose of definitions is for them to serve as abbreviations for complex expressions. Nonetheless, to facilitate reasoning about strong functors unwrapped versions of its definition were derived using **AssertTest**. Since breaking into **class**-wrapped definitions is unacceptably slow, a different approach is used here. The old **class** rewrite rule has been removed, and will now be replaced with the following equivalent **case**-wrapped rewrite rule.

Theorem. A rewrite rule for **case[strongfunctor[w, x, y]]**.

```
In[2]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> case[strongfunctor[w, x, y]],
          v -> case[and[equal[domain[w], range[x]], FUNCTION[w], equal[composite[w, x],
              composite[y, cross[w, w]]], subclass[image[w, ids[x]], ids[y]]]]}] // MapNotNot

Out[2]= equal[case[and[equal[composite[w, x], composite[y, cross[w, w]]],
            equal[domain[w], range[x]], FUNCTION[w], subclass[image[w, ids[x]], ids[y]]]],
          case[strongfunctor[w, x, y]]] == True

In[3]:= case[strongfunctor[w_, x_, y_]] :=
         case[and[equal[composite[w, x], composite[y, cross[w, w]]],
           equal[domain[w], range[x]], FUNCTION[w], subclass[image[w, ids[x]], ids[y]]]]
```

definition of functors

A new **case**-wrapped definition of **functor** will now be given that differs from the rule for **strongfunctor** by modifying just one literal, replacing the equation $\mathbf{w \circ x = y \circ (w \otimes w)}$ with the inclusion $\mathbf{w \circ x \subset y \circ (w \otimes w)}$.

Definition.

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

Theorem. Strong functors are functors.

```
In[5]:= SubstTest[subclass, case[p], case[q], {p → strongfunctor[w, x, y], q → functor[w, x, y]}]

Out[5]= or[functor[w, x, y], not[strongfunctor[w, x, y]]] == True
```

```
In[6]:= or[functor[w_, x_, y_], not[strongfunctor[w_, x_, y_]]] := True
```

A counterexample that shows that a functor between categories need not be strong will now be given.

Theorem.

```
In[9]:= functor[composite[FIRST, id[cart[V, V]]],
          composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] // AssertTest
Out[9]= functor[FIRST, composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] == True

In[10]:= functor[FIRST, composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] := True
```

Theorem.

```
In[11]:= strongfunctor[composite[FIRST, id[cart[V, V]]],
           composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] // AssertTest
Out[11]= strongfunctor[FIRST, composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] == False

In[12]:= strongfunctor[FIRST, composite[id[cart[V, V]], inverse[DUP]], inverse[DUP]] := False
```

basic properties

Unwrapped rewrite rules corresponding to the definition of **functor** will now be derived. In one direction, one needs rewrite rules corresponding to each of the four literals in the definition. In the opposite direction, only one rewrite rule is needed.

Theorem. Functors are functions.

```
In[13]:= SubstTest[subclass, case[p], case[q], {p → functor[w, x, y], q → FUNCTION[w]}]
Out[13]= or[FUNCTION[w], not[functor[w, x, y]]] == True

In[14]:= or[FUNCTION[w_], not[functor[w_, x_, y_]]] := True
```

Theorem. The domain of a functor **w** from **x** to **y** is equal to the range of **x**.

```
In[15]:= SubstTest[subclass, case[p], case[q],
           {p → functor[w, x, y], q → equal[domain[w], range[x]]}]
Out[15]= or[equal[domain[w], range[x]], not[functor[w, x, y]]] == True

In[16]:= or[equal[domain[w_], range[x_]], not[functor[w_, x_, y_]]] := True
```

Theorem. Functors preserve identities.

```
In[17]:= SubstTest[subclass, case[p], case[q],
           {p → functor[w, x, y], q → subclass[image[w, ids[x]], ids[y]]}]
Out[17]= or[not[functor[w, x, y]], subclass[image[w, ids[x]], ids[y]]] == True
```

```
In[18]:= or[not[functor[w_, x_, y_]], subclass[image[w_, ids[x_]], ids[y_]]] := True
```

Theorem. Functors preserve composition.

```
In[19]:= SubstTest[subclass, case[p], case[q],
          {p → functor[w, x, y], q → subclass[composite[w, x], composite[y, cross[w, w]]]}]

Out[19]= or[not[functor[w, x, y]], subclass[composite[w, x], composite[y, cross[w, w]]]] == True
```

```
In[20]:= or[not[functor[w_, x_, y_]],
            subclass[composite[w_, x_], composite[y_, cross[w_, w_]]]] := True
```

Theorem. Implication in the opposite direction.

```
In[21]:= SubstTest[subclass, case[p], case[q],
          {p -> and[equal[domain[w], range[x]], FUNCTION[w], subclass[composite[w, x], composite[
               y, cross[w, w]]], subclass[image[w, ids[x]], ids[y]]], q → functor[w, x, y]}]

Out[21]= or[functor[w, x, y], not[equal[domain[w], range[x]]],
            not[FUNCTION[w]], not[subclass[composite[w, x], composite[y, cross[w, w]]]],
            not[subclass[image[w, ids[x]], ids[y]]]] == True
```

```
In[23]:= or[functor[w_, x_, y_], not[equal[domain[w_], range[x_]]], not[FUNCTION[w_]],
            not[subclass[composite[w_, x_], composite[y_, cross[w_, 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.

```
In[30]:= functor[id[range[x]], x, x] // AssertTest

Out[30]= functor[id[range[x]], x, x] == subclass[domain[x], cart[range[x], range[x]]]
```

```
In[31]:= functor[id[range[x_]], x_, x_] := subclass[domain[x], cart[range[x], range[x]]]
```

Observation. This condition is true by definition when **x** is a category.

```
In[32]:= implies[category[x], functor[id[range[x]], x, x]]

Out[32]= True
```

empty functors

Theorem. The empty function is a functor from the empty set to any class. More generally:

```
In[33]:= functor[0, x, y] // AssertTest
Out[33]= functor[0, x, y] == equal[0, domain[x]]

In[34]:= functor[0, x_, y_] := equal[0, domain[x]]
```

Lemma.

```
In[48]:= subclass[composite[w, x], 0] // AssertTest
Out[48]= subclass[composite[w, x], 0] == equal[0, intersection[domain[w], range[x]]]

In[49]:= subclass[composite[w_, x_], 0] := equal[0, intersection[domain[w], range[x]]]
```

Theorem. The only functor to the empty set from any class is the empty set.

```
In[50]:= implies[functor[w, x, 0], equal[0, w]] // AssertTest
Out[50]= or[equal[0, w], not[functor[w, x, 0]]] == True

In[51]:= or[equal[0, w_], not[functor[w_, x_, 0]]] := True
```

---

preservation of composability

The domain of a category is called its **composability** relation. For functors, the theorem about the preservation of composability is only an inclusion. (For strong functors, an equation holds.)

Lemma.

```
In[52]:= SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
           {u -> composite[w, x], v -> composite[y, cross[w, w]]}] // Reverse
Out[52]= or[not[subclass[composite[w, x], composite[y, cross[w, w]]]],
           subclass[image[inverse[x], domain[w]], composite[inverse[w], domain[y], w]]] == True

In[53]:= (% /. {w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. functors preserve composability.

```
In[54]:= 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 → subclass[composite[w, x], composite[y, cross[w, w]]],
             p3 → subclass[image[inverse[x], domain[w]], composite[inverse[w], domain[y], w]],
             p4 → equal[domain[w], range[x]],
             p5 → subclass[domain[x], composite[inverse[w], domain[y], w]]}]] // Reverse
Out[54]= or[not[functor[w, x, y]],
           subclass[domain[x], composite[inverse[w], domain[y], w]]] == True

In[55]:= or[not[functor[w_, x_, y_]],
           subclass[domain[x_], composite[inverse[w_], domain[y_], w_]]] := True
```

range of a functor

The range of a functor from **x** to **y** need not be binary closed under **y**. However, a weaker result can be derived.

Lemma.

```
In[56]:= SubstTest[implies, subclass[u, v], subclass[range[u], range[v]],
           {u -> composite[w, x], v -> composite[y, cross[w, w]]}] // Reverse

Out[56]= or[not[subclass[composite[w, x], composite[y, cross[w, w]]]],
           subclass[image[w, range[x]], image[y, cart[range[w], range[w]]]]] == True

In[57]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. An upper bound for the range of a functor **w** from **x** to **y**.

```
In[58]:= 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 → subclass[composite[w, x], composite[y, cross[w, w]]],
              p3 → subclass[image[w, range[x]], image[y, cart[range[w], range[w]]]],
              p4 → equal[domain[w], range[x]],
              p5 → subclass[range[w], image[y, cart[range[w], range[w]]]]}]] // Reverse

Out[58]= or[not[functor[w, x, y]], subclass[range[w], image[y, cart[range[w], range[w]]]]] == True

In[59]:= or[not[functor[w_, x_, y_]],
           subclass[range[w_], image[y_, cart[range[w_], range[w_]]]]] := True
```

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

```
In[60]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 → functor[w, x, y], p2 → subclass[range[w], image[y, cart[range[w], range[w]]]],
              p3 → subclass[range[w], range[y]]}]] // Reverse

Out[60]= or[not[functor[w, x, y]], subclass[range[w], range[y]]] == True

In[61]:= or[not[functor[w_, x_, y_]], subclass[range[w_], range[y_]]] := True
```

a mapping statement

In general **w**, **x** and **y** may all be proper classes. In this section a mapping statement is derived for the special case that **x** is a set.

Theorem. If **w** is a functor from a set **x** to a class **y**, then **w** is also a set.

```
In[62]:= Map[not, SubstTest[and,
           (*implies[p1,p3],implies[p1,p4],implies[p2,p5],*)implies[and[p3, p4, p5], p6],
           not[implies[and[p1, p2], p6]], {p1 → functor[w, x, y], p2 → member[x, z],
            p3 → FUNCTION[w], p4 → equal[domain[w], range[x]],
            p5 → member[range[x], V], p6 → member[w, V]}]] // Reverse

Out[62]= or[member[w, V], not[functor[w, x, y]], not[member[x, z]]] == True

In[63]:= or[member[w_, V], not[functor[w_, x_, y_]], not[member[x_, z_]]] := True
```

Theorem. If **w** is a functor from a set **x** to a class **y**, then **w ∈ map[range[x], range[y]]**.

```
In[64]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p1, p2], p4], implies[p1, p5],
           implies[p1, p6], (*implies[and[p3,p4,p5,p6],p7],*) not[implies[and[p1, p2], p7]],
           {p1 → functor[w, x, y], p2 → member[x, z], p3 → FUNCTION[w], p4 → member[w, V],
            p5 → equal[domain[w], range[x]], p6 → subclass[range[w], range[y]],
            p7 -> member[w, map[range[x], range[y]]]}]] // Reverse

Out[64]= or[member[w, map[range[x], range[y]]], not[functor[w, x, y]], not[member[x, z]]] == True

In[65]:= or[member[w_, map[range[x_], range[y_]]],
           not[functor[w_, x_, y_]], not[member[x_, z_]]] := True
```

---

dom is preserved

Lemma.

```
In[66]:= 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[66]= 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[67]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[68]:= SubstTest[implies, subclass[u, v], subclass[composite[t, u], composite[t, v]],
           {t → id[ids[x]], u -> domain[x], v -> composite[inverse[w], domain[y], w]}] // Reverse

Out[68]= or[not[subclass[domain[x], composite[inverse[w], domain[y], w]]],
           subclass[dom[x], cart[V, ids[x]]]] == True

In[69]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[70]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
           {r -> dom[x], s -> composite[id[ids[x]], inverse[w], domain[y], w],
            t -> composite[inverse[w], dom[y], w]}] // Reverse

Out[70]= or[not[subclass[composite[id[ids[x]], inverse[w], domain[y], w],
             composite[inverse[w], dom[y], w]]], not[subclass[dom[x], cart[V, ids[x]]]],
           not[subclass[dom[x], composite[inverse[w], domain[y], w]]],
           subclass[dom[x], composite[inverse[w], dom[y], w]]] == True

In[71]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. Functors preserve **dom**.

```
In[72]:= 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 → subclass[domain[x], composite[inverse[w], domain[y], w]],
             p3 → subclass[dom[x], composite[id[ids[x]], inverse[w], domain[y], w]],
             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[72]= or[not[functor[w, x, y]], subclass[dom[x], composite[inverse[w], dom[y], w]]] == True

In[73]:= or[not[functor[w_, x_, y_]],
           subclass[dom[x_], composite[inverse[w_], dom[y_], w_]]] := True
```

---

cod is preserved

Lemma.

```
In[74]:= 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[74]= 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[75]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[76]:= SubstTest[implies, subclass[u, v],
           subclass[composite[t, u], composite[t, v]], {t → id[ids[x]], u -> inverse[domain[x]],
            v -> composite[inverse[w], inverse[domain[y]], w]}] // Reverse

Out[76]= or[not[subclass[inverse[domain[x]], composite[inverse[w], inverse[domain[y]], w]]],
           subclass[cod[x], cart[V, ids[x]]]] == True

In[77]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma

```
In[78]:= SubstTest[implies, and[subclass[r, s], subclass[s, t]], subclass[r, t],
           {r → cod[x], s -> composite[id[ids[x]], inverse[w], inverse[domain[y]], w],
            t -> composite[inverse[w], cod[y], w]}] // Reverse

Out[78]= or[not[subclass[cod[x], cart[V, ids[x]]]],
           not[subclass[cod[x], composite[inverse[w], inverse[domain[y]], w]]],
           not[subclass[composite[id[ids[x]], inverse[w], inverse[domain[y]], w],
             composite[inverse[w], cod[y], w]]],
           subclass[cod[x], composite[inverse[w], cod[y], w]]] == True

In[79]:= (% /. {w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. Functors preserve **cod**.

```
In[80]:= 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 → subclass[domain[x], composite[inverse[w], domain[y], w]],
             p2 → subclass[inverse[domain[x]], composite[inverse[w], inverse[domain[y]], w]],
             p3 → subclass[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[80]= or[not[functor[w, x, y]], subclass[cod[x], composite[inverse[w], cod[y], w]]] == True

In[81]:= or[not[functor[w_, x_, y_]],
           subclass[cod[x_], composite[inverse[w_], cod[y_], w_]]] := True
```

restricting y to its ternary relational part

The results of this section are simplification rules that are of interest only when **x** and **y** are not assumed to be categories.

Theorem. A simplification rule.

```
In[83]:= equiv[functor[w, x, composite[y, id[cart[V, V]]]], functor[w, x, y]] // assert

Out[83]= True

In[84]:= functor[w_, x_, composite[y_, id[cart[V, V]]]] := functor[w, x, y]
```

Comment. 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 can however replace **x** with its binary relational part **Id** ∘ **x**.

Theorem. A simplification rule.

```
In[85]:= equiv[functor[w, composite[Id, x], y], functor[w, x, y]] // assert
Out[85]= True

In[86]:= functor[w_, composite[Id, x_], y_] := functor[w, x, y]
```

flip rule

A functor from **x** to **y** is also a functor from **flip[x]** to **flip[y]**. This fact is useful for applications of duality. One can also use this fact transfer any occurrence of flip on **x** to a flip on **y**.

Lemma. A simplification rule.

```
In[97]:= SubstTest[subclass, flip[u], flip[v],
          {u -> composite[w, x, SWAP], v -> composite[y, cross[w, w]]}]
Out[97]= subclass[composite[w, x, SWAP], composite[y, cross[w, w]]] ==
          subclass[composite[w, x, id[cart[V, V]]], composite[y, SWAP, cross[w, w]]]

In[98]:= subclass[composite[w_, x_, SWAP], composite[y_, cross[w_, w_]]] :=
          subclass[composite[w, x, id[cart[V, V]]], composite[y, SWAP, cross[w, w]]]
```

Theorem. The statement that **w** is a functor from **flip[x]** to **y** is equivalent to the statement that **w** is a functor from **x** ∘ **id[V × V]** to **flip[y]**.

```
In[99]:= equiv[functor[w, composite[x, SWAP], y],
           functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]] // assert
Out[99]= True

In[100]:=
        functor[w_, composite[x_, SWAP], y_] :=
         functor[w, composite[x, id[cart[V, V]]], composite[y, SWAP]]
```

Observation. The following simplification is now automatic when **cat** wrappers are used.

```
In[101]:=
        functor[w, flip[cat[x]], flip[cat[y]]]
Out[101]=
        functor[w, cat[x], cat[y]]
```

composites of 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[102]:=
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[102]=
or[FUNCTION[composite[u, v]], not[functor[u, y, z]], not[functor[v, x, y]]] == True

In[103]:=
(% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma 2.

```
In[104]:=
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[104]=
or[equal[image[inverse[v], domain[u]], range[x]],
  not[functor[u, y, z]], not[functor[v, x, y]]] == True

In[105]:=
(% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma.

```
In[106]:=
SubstTest[implies, subclass[r, s], subclass[composite[r, t], composite[s, t]],
  {r -> composite[u, y], s -> composite[z, cross[u, u]], t → cross[v, v]}] // Reverse

Out[106]=
or[not[subclass[composite[u, y], composite[z, cross[u, u]]]],
  subclass[composite[u, y, cross[v, v]],
   composite[z, cross[composite[u, v], composite[u, v]]]]] == True

In[107]:=
(% /. {u → u_, v → v_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma 3.

In[108]:=
```
Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p3, p4],
   implies[p2, p5], implies[and[p2, p4], p5], implies[and[p4, p5], p6],
   not[implies[p1, p6]], {p1 → and[functor[v, x, y], functor[u, y, z]],
    p2 → subclass[composite[u, y], composite[z, cross[u, u]]],
    p3 → subclass[composite[v, x], composite[y, cross[v, v]]],
    p4 → subclass[composite[u, v, x], composite[u, y, cross[v, v]]],
    p5 → subclass[composite[u, y, cross[v, v]], composite[z,
       cross[composite[u, v], composite[u, v]]]], p6 → subclass[composite[u, v, x],
      composite[z, cross[composite[u, v], composite[u, v]]]]}]] // Reverse
```

Out[108]=
```
or[not[functor[u, y, z]], not[functor[v, x, y]], subclass[composite[u, v, x],
   composite[z, cross[composite[u, v], composite[u, v]]]]] == True
```

In[109]:=
```
(% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma 4.

In[110]:=
```
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[110]=
```
or[not[functor[u, y, z]], not[functor[v, x, y]],
  subclass[image[u, image[v, ids[x]]], ids[z]]] == True
```

In[111]:=
```
(% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Lemma 5.

In[112]:=
```
SubstTest[or, functor[t, x, z], not[subclass[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[112]=
```
or[functor[composite[u, v], x, z], not[equal[image[inverse[v], domain[u]], range[x]]],
  not[FUNCTION[composite[u, v]]], not[
   subclass[composite[u, v, x], composite[z, cross[composite[u, v], composite[u, v]]]]],
  not[subclass[image[u, image[v, ids[x]]], ids[z]]]] == True
```

In[113]:=
```
(% /. {u → u_, v → v_, x → x_, z → z_}) /. Equal→ SetDelayed
```

The composite of composable functors is a functor.

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

In[114]:=

```
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 → subclass[
      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[114]=

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

In[115]:=

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