category of sets: functions and relations

Johan G. F. Belinfante
2005 July 30

```
In[1]:= SetDirectory["i:"]; << goedel71.30a; << tools.m

        :Package Title: goedel71.30a          2005 July 30 at 3:50 p.m.

        It is now:  2005 Jul 30 at 23:26

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 July 18

        weightlimit = 40
```

summary

This notebook introduces the category of sets, and a generalization in which functions are replaced with relations. Both of these categories are proper classes. It is shown that the binary composition in both categories are associative functions, which are certain restrictions of the direct product of the associative functions **COMPOSE** and **FIRST**. In each category, the range of the composition is the class of morphisms for that category. The domain of these binary compositions are known as the composability relations. The domain and range of the composability relation is the class of morphisms.

an associative function

The direct product of two associative functions is another. This particular combination will be given a temporary name:

```
In[2]:= DIRPROD := composite[cross[COMPOSE, FIRST], TWIST]
```

The associative property for **DIRPROD** follows from a general result about direct products of associative relations.

```
In[3]:= SubstTest[implies, and[associative[x], associative[y]],
          associative[composite[cross[x, y], TWIST]], {x → COMPOSE, y → FIRST}]

Out[3]= associative[composite[cross[COMPOSE, FIRST], TWIST]] == True
```

```
In[4]:= associative[composite[cross[COMPOSE, FIRST], TWIST]] := True
```

The following rewrite rule will be needed later:

```
In[5]:= SubstTest[implies, associative[x], equal[composite[x, cross[x, Id]],
          composite[x, cross[Id, x], ASSOC]], x → DIRPROD]

Out[5]= equal[composite[cross[COMPOSE, FIRST],
           TWIST, cross[composite[cross[COMPOSE, FIRST], TWIST], Id]],
          composite[cross[COMPOSE, FIRST], TWIST,
           cross[Id, composite[cross[COMPOSE, FIRST], TWIST]], ASSOC]] == True

In[6]:= composite[cross[COMPOSE, FIRST], TWIST,
          cross[Id, composite[cross[COMPOSE, FIRST], TWIST]], ASSOC] :=
         composite[cross[COMPOSE, FIRST], TWIST,
          cross[composite[cross[COMPOSE, FIRST], TWIST], Id]]
```

---

morphisms, domain and codomain

A temporary abbreviation is introduced for the class of morphisms in the category of relations:

```
In[7]:= class[x,
         and[member[x, cart[P[cart[V, V]], V]], subclass[range[first[x]], second[x]]]]

Out[7]= composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]

In[8]:= MOR := composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]
```

The restriction of **MOR** to functions is the class of morphisms for the category of sets. A morphism in the class of relations is an ordered pair of a (small) relation and a set that contains the range of this relation, called the **codomain** of the morphism. In category theory it is conventional to refer to the domain of the relation as the "domain" of the morphism, although this is technically an abuse of language. In the category of relations, the functions that take a morphism to its "domain" and codomain are the following two functions, for which temporary abbreviations are introduced:

```
In[9]:= DOM := composite[IMAGE[FIRST], FIRST, id[MOR]]

In[10]:= COD := composite[SECOND, id[MOR]]
```

For the category of functions, one needs simply to restrict these functions to the class **FUNS** of all (small) functions.

composition of morphisms: the binary operation of the category of relations

The composability condition for a pair of morphisms in the category of relations is that the domain of the first is equal to the codomain of the second. This will be abbreviated:

```
In[11]:= COMPATIBLE := composite[inverse[COD], DOM]
```

The composition in the category of relations is a restriction of the direct product of **COMPOSE** and **FIRST** to the class of composable morphisms. The composite of two composible morphisms is the ordered pair of the composite relation, and the codomain of the first factor.

```
In[12]:= class[pair[pair[pair[u, v], pair[x, y]], pair[w, z]],
           and[member[pair[u, v], s], member[pair[x, y], s],
            member[pair[pair[u, v], pair[x, y]], t],
            equal[w, composite[u, x]], equal[z, v]]] /. {s → MOR, t → COMPATIBLE}

Out[12]= composite[cross[COMPOSE, FIRST], TWIST,
          id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
            inverse[SECOND], IMAGE[FIRST], FIRST,
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]
```

A name will be given to this:

```
In[13]:= composite[cross[COMPOSE, FIRST], TWIST,
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST,
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]] := CATORELN
```

This formula can be restated:

```
In[14]:= CATORELN == composite[DIRPROD, id[COMPATIBLE]]

Out[14]= True
```

Corollary.

```
In[15]:= Assoc[Id, DIRPROD, id[COMPATIBLE]]

Out[15]= composite[Id, CATORELN] == CATORELN

In[16]:= composite[Id, CATORELN] := CATORELN
```

```
In[17]:= IminComp[DIRPROD, id[COMPATIBLE], V]

Out[17]= domain[CATORELN] == composite[
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
           IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]

In[18]:= domain[CATORELN] := composite[
           id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
           IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]
```

Restatement:

```
In[19]:= domain[CATORELN] == COMPATIBLE

Out[19]= True
```

Corollaries:

```
In[20]:= domain[domain[CATORELN]] == MOR

Out[20]= True

In[21]:= range[domain[CATORELN]] == MOR

Out[21]= True
```

The composition is a function:

```
In[22]:= SubstTest[FUNCTION, composite[funpart[x], id[y]],
           {x → DIRPROD, y → COMPATIBLE}]

Out[22]= FUNCTION[CATORELN] == True

In[23]:= FUNCTION[CATORELN] := True
```

---

identities

The class of objects in the category of relations can be identified with the class of identities. This class will be abbreviated:

```
In[24]:= class[x, and[member[x, cart[V, V]],
           subclass[first[x], Id], equal[range[first[x]], second[x]]]]

Out[24]= inverse[IMAGE[DUP]]

In[25]:= IDS := inverse[IMAGE[DUP]]
```

A useful rewrite rule:

```
In[26]:= composite[inverse[IMAGE[DUP]], id[P[cart[V, V]]]] // DoubleInverse
Out[26]= composite[inverse[IMAGE[DUP]], id[P[cart[V, V]]]] == inverse[IMAGE[DUP]]

In[27]:= composite[inverse[IMAGE[DUP]], id[P[cart[V, V]]]] := inverse[IMAGE[DUP]]
```

Theorems about **IDS** are easiest to prove using this one-to-one generating function:

```
In[28]:= lambda[x, PAIR[id[x], x]] == composite[id[IDS], inverse[SECOND]]
Out[28]= True
```

This generator for **IDS** will be abbreviated:

```
In[29]:= GENIDS := composite[id[IDS], inverse[SECOND]]
```

Lemma. Identities are morphisms.

```
In[30]:= subclass[IDS, MOR] // AssertTest
Out[30]= subclass[inverse[IMAGE[DUP]], composite[S, IMAGE[SECOND]]] == True

In[31]:= subclass[inverse[IMAGE[DUP]], composite[S, IMAGE[SECOND]]] := True
```

Observation: the intersection of the classes **IDS** and **MOR** is equal to **IDS**.

```
In[32]:= equal[intersection[composite[S, IMAGE[SECOND]], inverse[IMAGE[DUP]]],
          inverse[IMAGE[DUP]]]
Out[32]= True
```

A corresponding rewrite rule will be introduced:

```
In[33]:= intersection[composite[S, IMAGE[SECOND]], inverse[IMAGE[DUP]]] :=
          inverse[IMAGE[DUP]]
```

Corollary.

```
In[34]:= intersection[composite[inverse[IMAGE[SECOND]], inverse[S]], IMAGE[DUP]] //
          DoubleInverse
Out[34]= intersection[composite[inverse[IMAGE[SECOND]], inverse[S]], IMAGE[DUP]] ==
          IMAGE[DUP]

In[35]:= intersection[composite[inverse[IMAGE[SECOND]], inverse[S]], IMAGE[DUP]] :=
          IMAGE[DUP]
```

the composite of composible morphisms is a morphism

Lemma 1.

```
In[36]:= intersection[
           complement[fix[composite[inverse[FIRST], inverse[IMAGE[SECOND]], S, IMG]]],
           composite[inverse[S], IMAGE[FIRST]]] // Renormality

Out[36]= intersection[
           complement[fix[composite[inverse[FIRST], inverse[IMAGE[SECOND]], S, IMG]]],
           composite[inverse[S], IMAGE[FIRST]]] == 0

In[37]:= % /. Equal → SetDelayed
```

Lemma 2.

```
In[38]:= SubstTest[implies, equal[0, fix[composite[x, y]]],
          equal[0, fix[composite[y, x]]],
          {x -> composite[inverse[FIRST], inverse[IMAGE[SECOND]]], y ->
            composite[complement[S], IMG, id[composite[inverse[S], IMAGE[FIRST]]]]}]

Out[38]= subclass[composite[IMAGE[SECOND], FIRST,
            id[composite[inverse[S], IMAGE[FIRST]]], inverse[IMG]], S] == True

In[39]:= % /. Equal → SetDelayed
```

Lemma 3.

```
In[40]:= SubstTest[composite, id[range[x]], x, x -> dif[composite[IMAGE[SECOND], FIRST,
              id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
              inverse[COMPOSE]], composite[S, IMAGE[SECOND]]]] // Reverse

Out[40]= intersection[composite[complement[S], IMAGE[SECOND]],
           composite[IMAGE[SECOND], FIRST,
            id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
            inverse[COMPOSE]]] == 0

In[41]:= % /. Equal → SetDelayed
```

Theorem.

```
In[42]:= SubstTest[equal, 0, dif[u, v], {u -> composite[IMAGE[SECOND], FIRST,
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
           inverse[COMPOSE]], v -> composite[S, IMAGE[SECOND]]}] // Reverse

Out[42]= subclass[composite[IMAGE[SECOND], FIRST,
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
           inverse[COMPOSE]], composite[S, IMAGE[SECOND]]] == True

In[43]:= % /. Equal → SetDelayed
```

Adding a factor of **S** yields this corollary.

```
In[44]:= SubstTest[implies, subclass[u, v],
          subclass[composite[S, u], composite[S, v]], {u -> composite[IMAGE[SECOND],
            FIRST, id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
            inverse[COMPOSE]], v -> composite[S, IMAGE[SECOND]]}]

Out[44]= subclass[composite[S, IMAGE[SECOND], FIRST,
           id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
           inverse[COMPOSE]], composite[S, IMAGE[SECOND]]] == True

In[45]:= % /. Equal → SetDelayed
```

Adding a restriction to relations yields a further corollary.

```
In[46]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> composite[S, IMAGE[SECOND], FIRST, id[composite[id[P[cart[V, V]]],
               inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], id[P[cart[V, V]]]]],
            inverse[COMPOSE]], v -> composite[S, IMAGE[SECOND], FIRST,
            id[composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]],
            inverse[COMPOSE]], w -> composite[S, IMAGE[SECOND]]}]

Out[46]= subclass[composite[S, IMAGE[SECOND], FIRST, id[composite[id[P[cart[V, V]]],
            inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], id[P[cart[V, V]]]]],
           inverse[COMPOSE]], composite[S, IMAGE[SECOND]]] == True

In[47]:= % /. Equal → SetDelayed

In[48]:= Map[subclass[#, composite[S, IMAGE[SECOND]]] &,
          ImageComp[DIRPROD, id[COMPATIBLE], V]]

Out[48]= subclass[range[CATORELN], composite[S, IMAGE[SECOND]]] == True

In[49]:= % /. Equal → SetDelayed

In[50]:= Map[subclass[#, MOR] &, ImageComp[DIRPROD, id[COMPATIBLE], V]]

Out[50]= subclass[range[CATORELN], cart[P[cart[V, V]], V]] == True

In[51]:= % /. Equal → SetDelayed
```

Restatement.

```
In[52]:= subclass[range[CATORELN], MOR]

Out[52]= True
```

In the next section it will be shown that this inclusion is in fact an equality. The idea is to show that any morphism can be written as the composite of itself and the identity morphism on its domain.

---

restriction of CATORELN to cart[MOR,IDS]

Lemma.

```
In[53]:= Map[inverse, composite[COMPOSE,
           id[composite[IMAGE[DUP], IMAGE[FIRST], id[P[cart[V, V]]]]],
           inverse[FIRST]] // VSNormality]

Out[53]= composite[FIRST, id[composite[IMAGE[DUP], IMAGE[FIRST], id[P[cart[V, V]]]]],
           inverse[COMPOSE]] == id[P[cart[V, V]]]

In[54]:= composite[FIRST, id[composite[IMAGE[DUP], IMAGE[FIRST], id[P[cart[V, V]]]]],
           inverse[COMPOSE]] := id[P[cart[V, V]]]
```

This yields a lower bound on the range of **CATORELN**.

```
In[55]:= Map[subclass[#, range[CATORELN]] &,
           ImageComp[DIRPROD, id[COMPATIBLE], cart[MOR, IDS]]] // Reverse

Out[55]= subclass[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]], range[CATORELN]] ==
          True

In[56]:= % /. Equal → SetDelayed
```

It follows that the range is equal to the class of morphisms.

```
In[57]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → range[CATORELN], v → MOR}]

Out[57]= True == equal[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]], range[CATORELN]]

In[58]:= range[CATORELN] := composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]
```

COD theorem

In this section, a formula is derived for **composite[COD, CATORELN]**. The first lemma just says that **CATORELN** is a subclass of **DIRPROD**.

```
In[59]:= SubstTest[subclass, composite[x, id[y]], x, {x → DIRPROD, y → COMPATIBLE}]

Out[59]= subclass[CATORELN, composite[cross[COMPOSE, FIRST], TWIST]] == True

In[60]:= subclass[CATORELN, composite[cross[COMPOSE, FIRST], TWIST]] := True
```

Corollary.

```
In[61]:= SubstTest[implies, and[subclass[u, v], subclass[x, y]],
          subclass[composite[u, x], composite[v, y]],
          {u → COD, v → SECOND, x → CATORELN, y → DIRPROD}]

Out[61]= subclass[composite[SECOND, CATORELN], composite[SECOND, FIRST]] == True

In[62]:= % /. Equal → SetDelayed
```

Since a subclass of a function is a restriction, one finds an explicit formula for **composite[SECOND,CATORELN]**.

```
In[63]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u → composite[COD, CATORELN], v -> composite[SECOND, FIRST]}]

Out[63]= equal[composite[SECOND, CATORELN], composite[SECOND, FIRST,
            id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
              inverse[SECOND], IMAGE[FIRST], FIRST,
              id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]] == True

In[64]:= composite[SECOND, CATORELN] := composite[SECOND, FIRST,
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST,
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]
```

Restatement of this result:

```
In[65]:= composite[COD, CATORELN] == composite[COD, FIRST, id[COMPATIBLE]]

Out[65]= True
```

DOM theorem

The compatibility condition assures that the domain of a pair of composible morphisms is equal to the domain of the right hand factor. This is Quaife's theorem **(CO9)**, which he calls Boyer's lemma 18. A variable-free version of this result involves the following compatibility condition:

```
In[66]:= BOYER := composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST]]

In[67]:= composite[IMG, SWAP, cross[IMAGE[FIRST], IMAGE[SWAP]], id[BOYER]] ==
          composite[IMAGE[FIRST], SECOND, id[BOYER]]

Out[67]= True
```

A corollary of this will be derived. First, another application of the result about restrictions of functions:

```
In[68]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]], {u -> composite[IMAGE[FIRST],
             intersection[composite[FIRST, SECOND], composite[BOYER, FIRST, FIRST]],
             id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
               inverse[SECOND], IMAGE[FIRST], FIRST,
               id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]],
           v → composite[IMAGE[FIRST], FIRST, SECOND]}]

Out[68]= equal[
           composite[IMAGE[FIRST], intersection[composite[FIRST, SECOND], composite[
              inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], FIRST, FIRST]],
            id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
              inverse[SECOND], IMAGE[FIRST], FIRST,
              id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]],
           composite[IMAGE[FIRST], FIRST, SECOND,
            id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
              inverse[SECOND], IMAGE[FIRST], FIRST,
              id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]] == True

In[69]:= composite[IMAGE[FIRST], intersection[composite[FIRST, SECOND],
            composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], FIRST, FIRST]],
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST,
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]] :=
          composite[IMAGE[FIRST], FIRST, SECOND, id[composite[
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
             IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]
```

This yields:

```
In[70]:= Assoc[composite[IMG, SWAP, cross[IMAGE[FIRST], IMAGE[SWAP]]],
          id[BOYER], cross[FIRST, FIRST]]

Out[70]= composite[IMG, SWAP,
           cross[composite[IMAGE[FIRST], FIRST], composite[IMAGE[SWAP], FIRST]],
           id[composite[inverse[FIRST], inverse[IMAGE[SECOND]],
             inverse[S], IMAGE[FIRST], FIRST]]] ==
          composite[IMAGE[FIRST], intersection[composite[FIRST, SECOND],
           composite[inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], FIRST, FIRST]]]

In[71]:= % /. Equal → SetDelayed

In[72]:= SubstTest[equal, 0, dif[u, v],
           {u → COMPATIBLE, v -> composite[inverse[FIRST], inverse[IMAGE[SECOND]],
              inverse[S], IMAGE[FIRST], FIRST]}] // Reverse

Out[72]= subclass[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
            inverse[SECOND], IMAGE[FIRST], FIRST,
            id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
           composite[inverse[FIRST], inverse[IMAGE[SECOND]],
            inverse[S], IMAGE[FIRST], FIRST]] == True

In[73]:= % /. Equal → SetDelayed
```

Restatement:

```
In[74]:= intersection[COMPATIBLE, composite[inverse[FIRST],
            inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], FIRST]] == COMPATIBLE

Out[74]= True
```

Lemma.

```
In[75]:= Assoc[composite[IMG, SWAP,
           cross[composite[IMAGE[FIRST], FIRST], composite[IMAGE[SWAP], FIRST]]],
          id[composite[inverse[FIRST], BOYER, FIRST]], id[COMPATIBLE]]

Out[75]= composite[IMG, SWAP,
           cross[composite[IMAGE[FIRST], FIRST], composite[IMAGE[SWAP], FIRST]],
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST,
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]] ==
          composite[IMAGE[FIRST], FIRST, SECOND, id[composite[
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
             IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]

In[76]:= % /. Equal → SetDelayed
```

Final result:

```
In[77]:= Assoc[composite[IMAGE[FIRST], FIRST], DIRPROD, id[COMPATIBLE]]

Out[77]= composite[IMAGE[FIRST], FIRST, CATORELN] ==
          composite[IMAGE[FIRST], FIRST, SECOND, id[composite[
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
             IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]

In[78]:= composite[IMAGE[FIRST], FIRST, CATORELN] :=
          composite[IMAGE[FIRST], FIRST, SECOND, id[composite[
             id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]], inverse[SECOND],
             IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]
```

Restatement:

```
In[79]:= composite[DOM, CATORELN] == composite[DOM, SECOND, id[COMPATIBLE]]

Out[79]= True
```

---

associativity

Lemmas:

```
In[80]:= subclass[cross[x, Id], cross[y, Id]] // AssertTest

Out[80]= subclass[cross[x, Id], cross[y, Id]] == subclass[composite[Id, x], y]

In[81]:= subclass[cross[x_, Id], cross[y_, Id]] := subclass[composite[Id, x], y]

In[82]:= subclass[cross[Id, x], cross[Id, y]] // AssertTest

Out[82]= subclass[cross[Id, x], cross[Id, y]] == subclass[composite[Id, x], y]

In[83]:= subclass[cross[Id, x_], cross[Id, y_]] := subclass[composite[Id, x], y]
```

Since **CATORELN** is a subclass of **DIRPROD**, one has:

```
In[84]:= SubstTest[implies, and[subclass[u, v], subclass[x, y]],
          subclass[composite[u, x], composite[v, y]],
          {u → CATORELN, v → DIRPROD, x -> cross[CATORELN, Id], y → cross[DIRPROD, Id]}]

Out[84]= subclass[composite[CATORELN, cross[CATORELN, Id]],
           composite[cross[COMPOSE, FIRST], TWIST,
            cross[composite[cross[COMPOSE, FIRST], TWIST], Id]]] == True

In[85]:= % /. Equal → SetDelayed
```

This inclusion can be replace with an equation by using the fact that a subclass of a function is a restriction:

```
In[86]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u → composite[CATORELN, cross[CATORELN, Id]],
           v -> composite[cross[COMPOSE, FIRST], TWIST,
             cross[composite[cross[COMPOSE, FIRST], TWIST], Id]]}]

Out[86]= equal[composite[CATORELN, cross[CATORELN, Id]],
           composite[cross[COMPOSE, FIRST], TWIST,
            cross[composite[cross[COMPOSE, FIRST], TWIST], Id],
            id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
              inverse[SECOND], IMAGE[FIRST], FIRST, SECOND,
              id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
                inverse[SECOND], IMAGE[FIRST], FIRST,
                id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]]] == True
```

For convenience, this is made into a temporary rewrite rule.

```
In[87]:= composite[CATORELN, cross[CATORELN, Id]] := composite[cross[COMPOSE, FIRST],
           TWIST, cross[composite[cross[COMPOSE, FIRST], TWIST], Id],
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST, SECOND,
             id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
               inverse[SECOND], IMAGE[FIRST], FIRST,
               id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]]]
```

A similar argument can be applied to **composite[CATORELN, cross[Id,CATORELN], ASSOC]**. The associative property of **DIRPROD** kicks in at this point.

```
In[88]:= SubstTest[implies, and[subclass[u, v], subclass[x, y]],
          subclass[composite[u, x], composite[v, y]],
          {u → CATORELN, v → DIRPROD, x → composite[cross[Id, CATORELN], ASSOC],
           y → composite[cross[Id, DIRPROD], ASSOC]}]

Out[88]= subclass[composite[CATORELN, cross[Id, CATORELN], ASSOC],
           composite[cross[COMPOSE, FIRST], TWIST,
            cross[composite[cross[COMPOSE, FIRST], TWIST], Id]]] == True

In[89]:= % /. Equal → SetDelayed
```

Again the inclusion can be made into an equation, which is made into a temporary rewrite rule.

```
In[90]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u → composite[CATORELN, cross[Id, CATORELN], ASSOC],
           v -> composite[cross[COMPOSE, FIRST], TWIST,
             cross[composite[cross[COMPOSE, FIRST], TWIST], Id]]}]

Out[90]= equal[composite[CATORELN, cross[Id, CATORELN], ASSOC],
           composite[cross[COMPOSE, FIRST], TWIST,
            cross[composite[cross[COMPOSE, FIRST], TWIST], Id],
            id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
              inverse[SECOND], IMAGE[FIRST], FIRST, SECOND,
              id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
                inverse[SECOND], IMAGE[FIRST], FIRST,
                id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]]] == True

In[91]:= composite[CATORELN, cross[Id, CATORELN], ASSOC] :=
          composite[cross[COMPOSE, FIRST], TWIST,
           cross[composite[cross[COMPOSE, FIRST], TWIST], Id],
           id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
             inverse[SECOND], IMAGE[FIRST], FIRST, SECOND,
             id[composite[id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]],
               inverse[SECOND], IMAGE[FIRST], FIRST,
               id[composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]]]]]]]
```

Comparing these two expressions reveals them to be the same. It follows that **CATORELN** is an associative function.

```
In[92]:= SubstTest[implies, and[subclass[x, cart[cart[V, V], V]],
           equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
          associative[x], x → CATORELN]

Out[92]= associative[CATORELN] == True

In[93]:= associative[CATORELN] := True
```

---

the category of sets (functions)

The category of sets is similar to the category of relations, except for replacing relations with functions. The corresponding partial binary composition is given the name **CATOFUNS**.

```
In[94]:= composite[cross[COMPOSE, FIRST], TWIST,
           id[composite[id[composite[S, IMAGE[SECOND], id[FUNS]]],
             inverse[SECOND], IMAGE[FIRST], FIRST,
             id[composite[S, IMAGE[SECOND], id[FUNS]]]]]] := CATOFUNS
```

The category of functions is a restriction of the category of relations to the cartesian square of **cart[FUNS,V]**.

```
In[95]:= Assoc[composite[cross[COMPOSE, FIRST], TWIST,
            id[composite[id[composite[S, IMAGE[SECOND]]], inverse[SECOND],
              IMAGE[FIRST], FIRST, id[composite[S, IMAGE[SECOND]]]]]],
          id[cart[cart[P[cart[V, V]], V], cart[P[cart[V, V]], V]]],
          id[cart[cart[FUNS, V], cart[FUNS, V]]]] // Reverse

Out[95]= composite[CATORELN, id[cart[cart[FUNS, V], cart[FUNS, V]]]] == CATOFUNS

In[96]:= composite[CATORELN, id[cart[cart[FUNS, V], cart[FUNS, V]]]] := CATOFUNS
```

Lemma.

```
In[97]:= ImageComp[DIRPROD, id[COMPATIBLE], V] // Reverse

Out[97]= composite[S, IMAGE[SECOND], FIRST,
           id[composite[id[P[cart[V, V]]], inverse[IMAGE[SECOND]],
             inverse[S], IMAGE[FIRST], id[P[cart[V, V]]]]], inverse[COMPOSE]] ==
         composite[S, IMAGE[SECOND], id[P[cart[V, V]]]]

In[98]:= % /. Equal → SetDelayed
```

Restatement:

```
In[99]:= composite[MOR, FIRST, id[composite[id[P[cart[V, V]]], BOYER]],
           inverse[COMPOSE]] == range[CATORELN]

Out[99]= True
```

Corollary:

```
In[100]:=
        composite[COMPOSE, id[composite[id[P[cart[V, V]]], BOYER, id[P[cart[V, V]]]]],
          inverse[FIRST], inverse[IMAGE[SECOND]], inverse[S]] // DoubleInverse

Out[100]=
        composite[COMPOSE, id[composite[id[P[cart[V, V]]],
            inverse[IMAGE[SECOND]], inverse[S], IMAGE[FIRST], id[P[cart[V, V]]]]],
          inverse[FIRST], inverse[IMAGE[SECOND]], inverse[S]] ==
         composite[id[P[cart[V, V]]], inverse[IMAGE[SECOND]], inverse[S]]

In[101]:=
        % /. Equal → SetDelayed
```

The composite of two functions is a function.

In[102]:=
```
Map[subclass[range[#], FUNS] &,
  Assoc[composite[id[FUNS], COMPOSE], id[cart[FUNS, FUNS]],
   composite[id[composite[id[P[cart[V, V]]], BOYER, id[P[cart[V, V]]]]],
    inverse[FIRST], inverse[IMAGE[SECOND]], inverse[S]]]] // Reverse
```

Out[102]=
```
subclass[image[COMPOSE, composite[id[FUNS], inverse[IMAGE[SECOND]],
    inverse[S], IMAGE[FIRST], id[FUNS]]], FUNS] == True
```

In[103]:=
```
subclass[image[COMPOSE, composite[id[FUNS], inverse[IMAGE[SECOND]],
    inverse[S], IMAGE[FIRST], id[FUNS]]], FUNS] := True
```

It follows that **cart[FUNS, V]** is closed under the binary function **CATORELN**:

In[104]:=
```
Map[subclass[#, cart[FUNS, V]] &,
 ImageComp[DIRPROD, id[COMPATIBLE], cart[cart[FUNS, V], cart[FUNS, V]]]]
```

Out[104]=
```
subclass[image[CATORELN, cart[cart[FUNS, V], cart[FUNS, V]]], cart[FUNS, V]] ==
 True
```

In[105]:=
```
subclass[
  image[CATORELN, cart[cart[FUNS, V], cart[FUNS, V]]], cart[FUNS, V]] := True
```

The restriction of an associative relation to a closed subclass is associative:

In[106]:=
```
SubstTest[implies, and[associative[x], subclass[image[x, cart[y, y]], y]],
 associative[composite[x, id[cart[y, y]]]], {x → CATORELN, y → cart[FUNS, V]}]
```

Out[106]=
```
associative[CATOFUNS] == True
```

In[107]:=
```
associative[CATOFUNS] := True
```