# composability rule for categories

Johan G. F. Belinfante
2009 January 9

```
In[1]:= SetDirectory["l:"]; << goedel.09jan07a;<< tools.m

        :Package Title: goedel.09jan07a         2009 January 7 at 11:35 a.m.

        It is now:  2009 Jan 9 at 8:44

        Loading Simplification Rules

        TOOLS.M                                      Revised 2008 December 26

        weightlimit = 40
```

## summary

The composability criterion for morphisms in a category is assumed as an axiom in the objects-and-arrows approach to category theory. In the arrows-only approach, it is instead a theorem which is derived in this notebook. In the **GOEDEL** program, categories are defined by a variant of MacLane's arrows-only axioms. For convenience, the following terminology will be used here. Morphisms are defined to be elements of **range[cat[x]]**. The **dom** of a morphism **w** in a category **cat[x]** is the unique identity morphism that is composable with **w** on the right. Similarly, the **cod** of a morphism **w** in a category **cat[x]** is the unique identity morphism that is composable with **w** on the left. The composability criterion to be established says that morphisms **u** and **v** in a category **cat[x]** are composable if and only if the **dom** of **u** is equal to the **cod** of **v**. Eliminating the variables **u** and **v** yields the equivalent statement that the domain of **cat[x]** is the composite of the inverse of the function **cod[cat[x]]** with the function **dom[cat[x]]**.

## simplification rules

Theorem.

```
In[2]:= equiv[and[member[u, range[cat[x]]], member[pair[u, v], domain[cat[x]]]],
          member[pair[u, v], domain[cat[x]]]]

Out[2]= True

In[3]:= and[member[u_, range[cat[x_]]], member[pair[u_, v_], domain[cat[x_]]]] :=
          member[pair[u, v], domain[cat[x]]]
```

The dual of this also holds:

```
In[4]:= SubstTest[and, member[v, range[cat[t]]],
          member[pair[v, u], domain[cat[t]]], t → flip[cat[x]]] // Reverse

Out[4]= and[member[v, range[cat[x]]], member[pair[u, v], domain[cat[x]]]] ==
         member[pair[u, v], domain[cat[x]]]

In[5]:= and[member[v_, range[cat[x_]]], member[pair[u_, v_], domain[cat[x_]]]] :=
         member[pair[u, v], domain[cat[x]]]
```

## composability with cod and dom

Theorem. Any morphism is composable with its dom.

```
In[6]:= SubstTest[member, pair[u, APPLY[funpart[t], u]], funpart[t], t → dom[cat[x]]] // Reverse

Out[6]= member[pair[u, APPLY[dom[cat[x]], u]], domain[cat[x]]] == member[u, range[cat[x]]]

In[7]:= member[pair[u_, APPLY[dom[cat[x_]], u_]], domain[cat[x_]]] := member[u, range[cat[x]]]
```

A similar result holds for cod.

```
In[8]:= SubstTest[member, pair[u, APPLY[funpart[t], u]], funpart[t], t → cod[cat[x]]] // Reverse

Out[8]= member[pair[APPLY[cod[cat[x]], u], u], domain[cat[x]]] == member[u, range[cat[x]]]

In[9]:= member[pair[APPLY[cod[cat[x_]], u_], u_], domain[cat[x_]]] := member[u, range[cat[x]]]
```

## equality of cod and dom

Lemma.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
           {p1 → and[equal[APPLY[funpart[x], v], APPLY[funpart[y], u]],
              member[v, domain[funpart[x]]]],
            p2 -> member[u, domain[funpart[y]]], p3 → member[u, V]}]] // Reverse

Out[10]= or[member[u, V], not[equal[APPLY[funpart[x], v], APPLY[funpart[y], u]]],
          not[member[v, domain[funpart[x]]]]] == True

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

Theorem. Equations of the form **f(u) = g(v)**.

```
In[12]:= equiv[member[pair[u, v], composite[inverse[funpart[x]], funpart[y]]],
           and[equal[APPLY[funpart[x], v], APPLY[funpart[y], u]],
            member[v, domain[funpart[x]]]]] // assert

Out[12]= True
```

```
In[13]:= member[pair[u_, v_], composite[inverse[funpart[x_]], funpart[y_]]] :=
          and[equal[APPLY[funpart[x], v], APPLY[funpart[y], u]], member[v, domain[funpart[x]]]]
```

Corollary. Application to the case of cod and dom.

```
In[14]:= SubstTest[member, pair[u, v], composite[inverse[funpart[r]], funpart[s]],
           {r → cod[cat[x]], s → dom[cat[x]]}] // Reverse

Out[14]= member[pair[u, v], composite[inverse[cod[cat[x]]], dom[cat[x]]]] ==
          and[equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]], member[v, range[cat[x]]]]

In[15]:= member[pair[u_, v_], composite[inverse[cod[cat[x_]]], dom[cat[x_]]]] :=
          and[equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]], member[v, range[cat[x]]]]
```

## implication in one direction

Lemma.

```
In[16]:= SubstTest[implies, and[subclass[u, y], subclass[v, z]],
           subclass[composite[u, v], composite[y, z]],
           {v → dom[cat[x]], u → inverse[cod[cat[x]]], y → domain[cat[x]],
            z → composite[SECOND, inverse[cat[x]]]}] // Reverse

Out[16]= subclass[composite[inverse[cod[cat[x]]], dom[cat[x]]],
           composite[domain[cat[x]], SECOND, inverse[cat[x]]]] == True

In[17]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[18]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
           subclass[u, w], {u -> composite[inverse[cod[cat[x]]], dom[cat[x]]], v ->
             composite[domain[cat[x]], SECOND, inverse[cat[x]]], w → domain[cat[x]]}] // Reverse

Out[18]= subclass[composite[inverse[cod[cat[x]]], dom[cat[x]]], domain[cat[x]]] == True

In[19]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. If the dom of a morphism **u** is composable with the cod of a morphism **v**, then **u** and **v** are composable.

```
In[20]:= SubstTest[implies, and[member[t, y], subclass[y, z]], member[t, z],
            {t → pair[u, v], y -> composite[inverse[cod[cat[x]]], dom[cat[x]]],
             z → domain[cat[x]]}] // Reverse // MapNotNot

Out[20]= or[member[pair[u, v], domain[cat[x]]],
           not[equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]]],
           not[member[v, range[cat[x]]]]] == True

In[21]:= or[member[pair[u_, v_], domain[cat[x_]]],
           not[equal[APPLY[cod[cat[x_]], v_], APPLY[dom[cat[x_]], u_]]],
           not[member[v_, range[cat[x_]]]]] := True
```

The remainder of this notebook is devoted to deriving the converse statement, as well as some related results.

## composability with dom-identities

Theorem.

```
In[22]:= SubstTest[implies, member[pair[APPLY[cat[x], PAIR[u, t]], v], domain[cat[x]]],
           member[pair[t, v], domain[cat[x]]], t → APPLY[dom[cat[x]], u]] // Reverse // MapNotNot

Out[22]= or[member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]],
           not[member[pair[u, v], domain[cat[x]]]]] == True

In[23]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Some lemmas are needed to derive the converse.

Lemma.

```
In[24]:= SubstTest[implies, member[pair[t, v], domain[cat[x]]],
           member[t, V], t → APPLY[dom[cat[x]], u]] // Reverse

Out[24]= or[member[u, range[cat[x]]],
           not[member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]]]] == True

In[25]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Temporary rewrite rule.

```
In[26]:= equiv[and[member[u, range[cat[x]]],
           member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]]],
          member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]]]

Out[26]= True

In[27]:= and[member[u_, range[cat[x_]]],
           member[pair[APPLY[dom[cat[x_]], u_], v_], domain[cat[x_]]]] :=
          member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]]
```

Converse.

```
In[28]:= SubstTest[implies,
            and[member[pair[u, t], domain[cat[x]]], member[pair[t, v], domain[cat[x]]]],
            member[pair[APPLY[cat[x], PAIR[u, t]], v], domain[cat[x]]],
            t → APPLY[dom[cat[x]], u]] // Reverse // MapNotNot

Out[28]= or[member[pair[u, v], domain[cat[x]]],
           not[member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]]]] == True

In[29]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

The following rewrite rule is obtained by combining the above theorem and its converse. Morphisms **u** and **v** are composable if and only if the dom of **u** is composable with **v**.

```
In[30]:= equiv[member[pair[APPLY[dom[cat[x]], u], v], domain[cat[x]]],
           member[pair[u, v], domain[cat[x]]]]

Out[30]= True

In[31]:= member[pair[APPLY[dom[cat[x_]], u_], v_], domain[cat[x_]]] :=
           member[pair[u, v], domain[cat[x]]]
```

## composability with cod-identities

Lemma. The dual of a lemma derived in the preceding section.

```
In[32]:= SubstTest[implies, member[pair[u, t], domain[cat[x]]],
           member[t, V], t → APPLY[cod[cat[x]], v]] // Reverse

Out[32]= or[member[v, range[cat[x]]],
           not[member[pair[u, APPLY[cod[cat[x]], v]], domain[cat[x]]]]] == True

In[33]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Temporary Lemma.

```
In[34]:= equiv[and[member[v, range[cat[x]]],
           member[pair[u, APPLY[cod[cat[x]], v]], domain[cat[x]]]],
          member[pair[u, APPLY[cod[cat[x]], v]], domain[cat[x]]]]

Out[34]= True

In[35]:= and[member[v_, range[cat[x_]]],
           member[pair[u_, APPLY[cod[cat[x_]], v_]], domain[cat[x_]]]] :=
          member[pair[u, APPLY[cod[cat[x]], v]], domain[cat[x]]]
```

Corollary. Morphisms **u** and **v** are composable if and only if **u** is composable with the cod of **v**.

```
In[36]:= SubstTest[member, pair[APPLY[dom[cat[t]], v], u],
           domain[cat[t]], t → flip[cat[x]]] // Reverse

Out[36]= member[pair[u, APPLY[cod[cat[x]], v]], domain[cat[x]]] ==
          member[pair[u, v], domain[cat[x]]]

In[37]:= member[pair[u_, APPLY[cod[cat[x_]], v_]], domain[cat[x_]]] :=
          member[pair[u, v], domain[cat[x]]]
```

## the composability criterion

Theorem. If **u** and **v** are composable, then the **dom** of **u** is equal to the **cod** of **v**.

```
In[38]:= Map[implies[#, equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]]] &,
          SubstTest[member, pair[r, s], composite[id[t], w, id[t]], {r → APPLY[dom[cat[x]], u],
            s -> APPLY[cod[cat[x]], v], t → ids[cat[x]], w → domain[cat[x]]}]]

Out[38]= or[equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]],
            not[member[pair[u, v], domain[cat[x]]]]] == True

In[39]:= or[equal[APPLY[cod[cat[x_]], v_], APPLY[dom[cat[x_]], u_]],
            not[member[pair[u_, v_], domain[cat[x_]]]]] := True
```

The next two lemmas are needed to eliminate the variables **u** and **v**.

Lemma.

```
In[40]:= implies[member[pair[u, v], domain[cat[x]]],
            member[pair[u, v], composite[inverse[cod[cat[x]]], dom[cat[x]]]]] // NotNotTest

Out[40]= or[and[equal[APPLY[cod[cat[x]], v], APPLY[dom[cat[x]], u]], member[v, range[cat[x]]]],
            not[member[pair[u, v], domain[cat[x]]]]] == True

In[41]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[42]:= Map[composite[Id, complement[#]] &,
          SubstTest[class, pair[u, v], or[member[pair[u, v], s], not[member[pair[u, v], t]]],
           {s → composite[inverse[cod[cat[x]]], dom[cat[x]]], t → domain[cat[x]]}]]

Out[42]= intersection[
            composite[complement[inverse[cod[cat[x]]]], dom[cat[x]]], domain[cat[x]]] == 0

In[43]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[44]:= SubstTest[empty, dif[u, v],
           {u -> domain[cat[x]], v -> composite[inverse[cod[cat[x]]], dom[cat[x]]]}]

Out[44]= subclass[domain[cat[x]], composite[inverse[cod[cat[x]]], dom[cat[x]]]] == True

In[45]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. Composability criterion for morphisms in a category.

```
In[46]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> composite[inverse[cod[cat[x]]], dom[cat[x]]], v -> domain[cat[x]]}]
Out[46]= equal[composite[inverse[cod[cat[x]]], dom[cat[x]]], domain[cat[x]]] == True

In[47]:= composite[inverse[cod[cat[x_]]], dom[cat[x_]]] := domain[cat[x]]
```

---

## some corollaries

The rewrite rules in this section follow immediately from the composability criterion derived above.

Corollary.

```
In[48]:= composite[inverse[dom[cat[x]]], cod[cat[x]]] // DoubleInverse
Out[48]= composite[inverse[dom[cat[x]]], cod[cat[x]]] == inverse[domain[cat[x]]]

In[49]:= composite[inverse[dom[cat[x_]]], cod[cat[x_]]] := inverse[domain[cat[x]]]
```

Corollary.

```
In[50]:= Assoc[cod[cat[x]], inverse[cod[cat[x]]], dom[cat[x]]]
Out[50]= composite[cod[cat[x]], domain[cat[x]]] == dom[cat[x]]

In[51]:= composite[cod[cat[x_]], domain[cat[x_]]] := dom[cat[x]]
```

Corollary.

```
In[52]:= Assoc[inverse[cod[cat[x]]], dom[cat[x]], inverse[dom[cat[x]]]] // Reverse
Out[52]= composite[domain[cat[x]], inverse[dom[cat[x]]]] == inverse[cod[cat[x]]]

In[53]:= composite[domain[cat[x_]], inverse[dom[cat[x_]]]] := inverse[cod[cat[x]]]
```

Corollary.

```
In[54]:= Assoc[inverse[cod[cat[x]]], dom[cat[x]], dom[cat[x]]] // Reverse
Out[54]= composite[domain[cat[x]], dom[cat[x]]] == domain[cat[x]]

In[55]:= composite[domain[cat[x_]], dom[cat[x_]]] := domain[cat[x]]
```

Corollary.

```
In[56]:= Map[inverse, Assoc[inverse[dom[cat[x]]], cod[cat[x]], cod[cat[x]]] // Reverse]
Out[56]= composite[inverse[cod[cat[x]]], domain[cat[x]]] == domain[cat[x]]

In[57]:= composite[inverse[cod[cat[x_]]], domain[cat[x_]]] := domain[cat[x]]
```

Corollary.

```
In[58]:= Assoc[inverse[cod[cat[x]]], dom[cat[x]], cod[cat[x]]] // Reverse

Out[58]= composite[domain[cat[x]], cod[cat[x]]] == composite[inverse[cod[cat[x]]], cod[cat[x]]]

In[59]:= composite[domain[cat[x_]], cod[cat[x_]]] :=
          composite[inverse[cod[cat[x]]], cod[cat[x]]]
```

Corollary.

```
In[60]:= Map[inverse, Assoc[inverse[dom[cat[x]]], cod[cat[x]], dom[cat[x]]] // Reverse]

Out[60]= composite[inverse[dom[cat[x]]], domain[cat[x]]] ==
          composite[inverse[dom[cat[x]]], dom[cat[x]]]

In[61]:= composite[inverse[dom[cat[x_]]], domain[cat[x_]]] :=
          composite[inverse[dom[cat[x]]], dom[cat[x]]]
```

Corollary.

```
In[62]:= Assoc[inverse[dom[cat[x]]], cod[cat[x]], domain[cat[x]]] // Reverse

Out[62]= composite[inverse[domain[cat[x]]], domain[cat[x]]] ==
          composite[inverse[dom[cat[x]]], dom[cat[x]]]

In[63]:= composite[inverse[domain[cat[x_]]], domain[cat[x_]]] :=
          composite[inverse[dom[cat[x]]], dom[cat[x]]]
```

```
In[64]:= SubstTest[composite, inverse[domain[cat[t]]],
           domain[cat[t]], t → flip[cat[x]]] // Reverse

Out[64]= composite[domain[cat[x]], inverse[domain[cat[x]]]] ==
          composite[inverse[cod[cat[x]]], cod[cat[x]]]

In[65]:= composite[domain[cat[x_]], inverse[domain[cat[x_]]]] :=
          composite[inverse[cod[cat[x]]], cod[cat[x]]]
```