arrows-only category axioms

Johan G. F. Belinfante
2008 December 30

```
In[1]:= SetDirectory["l:"]; << goedel.08dec29a;<< tools.m

        :Package Title: goedel.08dec29a         2008 December 29 at 6:20 a.m.

        It is now:  2008 Dec 30 at 8:58

        Loading Simplification Rules

        TOOLS.M                                      Revised 2008 December 26

        weightlimit = 40
```

summary

On pages 7-9 of his book, Saunders MacLane outlines two axiomatizations of category theory. In this notebook a variant of the second of these sets of axioms, which MacLane calls the **arrows-only axioms**, is presented within the framework of Gödel's class theory.

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

For simplicity the class of **morphisms** for a category is here defined to be the range of its composition law, so that only a single class variable **x** is needed to define a category. In this notebook an (abstract arrows-only) **category** is defined as a composition law **x** that satisfies conditions corresponding to MacLane's three axioms. MacLane's first axiom is taken to be the requirement that the composition law of a category be an associative function. MacLane's second axiom, which requires that the composite $\mathbf{k \cdot g \cdot f}$ of three morphisms be defined if $\mathbf{k \cdot g}$ and $\mathbf{g \cdot f}$ are defined, is replaced by an equivalent condition relating composability to left-divisibility. An explicit example is presented which shows that axiom 2 is not a consequence of the associative law. MacLane's third axiom, which says that every morphism is composable on either side with some identity morphism, depends on the precise definition of an identity morphism. For convenience, to the definition of an identity morphism **u** is added the requirement that **u** be composable with itself.

MacLane's first axiom

Since the class of morphisms of a category **x** is here being defined as **range[x]**, it is natural to add to MacLane's first axiom that **x** be an associative function the further requirement that **domain[x]** be contained in the cartesian square of **range[x]**.

MacLane's second axiom

MacLane's second axiom for arrows-only categories says that the composites **k · (g · f)** and **(k · g) · f** of three morphisms are both defined if and only if **k · g** and **g · f** are defined. It suffices to postulate a weaker result. To begin with, since the associative law implies that **k · (g · f)** is defined if and only if **(k · g) · f** is defined, one need only postulate that one of these is defined. The associative law also implies the **only if** part of the statement. It is convenient to restate the second axiom as a connection between divisibility and composability. To every associative (ternary) relation **x** there correspond two transitive (binary) relations **composite[x, inverse[FIRST]]** and **composite[x, inverse[SECOND]]** for left and right divisibility, respectively . More explicitly, if **p** is the product morphism **g · f = p**, let us say that **g** is a **left-divisor** of **p** and **f** is a **right-divisor** of **p**. The **composability relation** for **x** is the binary relation **domain[x]**. MacLane's second axiom therefore translates into the statement that if a morphism **k** is composable with some left-divisor **g** of a morphism **p**, then **k** is composable with **p**. Eliminating the morphism variables yields a reformulation of the second axiom as the requirement that the composite of the left-divisibility relation and the composability relation be a subclass of the composability relation.

MacLane's third axiom

The third axiom requires the existence of identities composable on either side with any morphism. MacLane defines an identity morphism to be a morphism **u** such that **f · u = f** whenever **f** and **u** are composable, and **u · g = g** whenever **u** and **g** are composable. If is convenient to strengthen the requirement that identities be morphisms to the requirement that an identity **u** be composable with itself, that is: **u · u = u**. In the **GOEDEL** program, the class **ids[x]** of identities for **x** is defined as the class of elements **u** in **fix[domain[x]]** such that left multiplication **composite[x, LEFT[u]]** by **u** and right multiplication **composite[x, RIGHT[u]]** by **u** are both contained in the identity relation **Id**. Closely related to the class **ids[x]** of identity morphisms are the relations **cod[x]** and **dom[x]**, defined as follows:

```
In[3]:= composite[id[ids[x]], domain[x]]

Out[3]= dom[x]

In[4]:= composite[id[ids[x]], inverse[domain[x]]]

Out[4]= cod[x]
```

Eliminating quantifiers for the morphism variables, MacLane's third axiom requires that the class **range[x]** of morphisms be contained in the domains of **cod[x]** and **dom[x]**.

category definition

The definition of the predicate **category[x]** is wrapped with **class** to prevent it from being automatically expanded.

```
In[5]:= class[t_, category[x_]] := class[t, and[associative[x],
           FUNCTION[x], subclass[composite[x, inverse[FIRST], domain[x]], domain[x]],
           subclass[domain[x], cart[range[x], range[x]]],
           subclass[range[x], domain[cod[x]]], subclass[range[x], domain[dom[x]]]]]
```

empty category

The simplest category is the empty category:

```
In[6]:= category[0] // AssertTest

Out[6]= category[0] == True

In[7]:= category[0] := True
```

Many categories are proper classes. For instance:

```
In[8]:= category[CUP] // AssertTest

Out[8]= category[CUP] == True

In[9]:= category[CUP] := True
```

normalization

Theorem. (Normalization for the predicate **category**.)

```
In[10]:= category[x] // AssertTest// Reverse

Out[10]= and[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
           FUNCTION[x], subclass[x,
            cart[cart[range[x], range[x]], intersection[domain[cod[x]], domain[dom[x]]]]],
           subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] == category[x]

In[11]:= and[equal[composite[x_, cross[x_, Id]], composite[x_, cross[Id, x_], ASSOC]],
           FUNCTION[x_], subclass[x_,
            cart[cart[range[x_], range[x_]], intersection[domain[cod[x_]], domain[dom[x_]]]]],
           subclass[composite[x_, inverse[FIRST], domain[x_]], domain[x_]]] := category[x]
```

properties of categories

Theorem. Categories are associative.

```
In[12]:= SubstTest[implies,
           and[p, equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]], subclass[
             x, cart[cart[range[x], range[x]], intersection[domain[cod[x]], domain[dom[x]]]]],
           associative[x], p → and[FUNCTION[x],
             subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]]] // Reverse

Out[12]= or[associative[x], not[category[x]]] == True

In[13]:= or[associative[x_], not[category[x_]]] := True
```

Theorem. Categories are functions.

```
In[14]:= SubstTest[implies, and[p, FUNCTION[x]], FUNCTION[x], p →
            and[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]], subclass[x,
              cart[cart[range[x], range[x]], intersection[domain[cod[x]], domain[dom[x]]]]],
             subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] // Reverse

Out[14]= or[FUNCTION[x], not[category[x]]] == True

In[15]:= or[FUNCTION[x_], not[category[x_]]] := True
```

Theorem. The domain of a category is contained in the cartesian square of the class of morphisms.

```
In[16]:= SubstTest[implies, and[p,
            subclass[x, cart[cartsq[range[x]], intersection[domain[cod[x]], domain[dom[x]]]]],
           subclass[domain[x], cartsq[range[x]]], p -> and[
             equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]], FUNCTION[x],
             subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] // Reverse

Out[16]= or[not[category[x]], subclass[domain[x], cart[range[x], range[x]]]] == True

In[17]:= or[not[category[x_]], subclass[domain[x_], cart[range[x_], range[x_]]]] := True
```

Theorem. Category axiom 2.

```
In[18]:= SubstTest[implies,
           and[p, subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]],
           subclass[composite[x, inverse[FIRST], domain[x]], domain[x]],
           p -> and[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
             FUNCTION[x], subclass[x, cart[cart[range[x], range[x]],
               intersection[domain[cod[x]], domain[dom[x]]]]]] // Reverse

Out[18]= or[not[category[x]],
           subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] == True

In[19]:= or[not[category[x_]],
           subclass[composite[x_, inverse[FIRST], domain[x_]], domain[x_]]] := True
```

Lemma.

```
In[20]:= Map[or[subclass[range[w], y], #] &, SubstTest[implies,
           subclass[w, cart[x, t]], subclass[range[w], t], t → intersection[y, z]]] // Reverse

Out[20]= or[not[subclass[w, cart[x, intersection[y, z]]]], subclass[range[w], y]] == True

In[21]:= or[not[subclass[w_, cart[x_, intersection[y_, z_]]]], subclass[range[w_], y_]] := True
```

Theorem. Composability with identities on the right.

```
In[22]:= SubstTest[implies, and[p, subclass[x,
           cart[cart[range[x], range[x]], intersection[domain[cod[x]], domain[dom[x]]]]]],
         subclass[range[x], domain[dom[x]]], p -> and[
           equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]], FUNCTION[x],
           subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]]] // Reverse

Out[22]= or[not[category[x]], subclass[range[x], domain[dom[x]]]] == True

In[23]:= or[not[category[x_]], subclass[range[x_], domain[dom[x_]]]] := True
```

Theorem. Composability with identities on the left.

```
In[24]:= SubstTest[implies, and[p, subclass[x,
           cart[cart[range[x], range[x]], intersection[domain[cod[x]], domain[dom[x]]]]]],
         subclass[range[x], domain[cod[x]]], p -> and[
           equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]], FUNCTION[x],
           subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]]] // Reverse

Out[24]= or[not[category[x]], subclass[range[x], domain[cod[x]]]] == True

In[25]:= or[not[category[x_]], subclass[range[x_], domain[cod[x_]]]] := True
```

Lemma.

```
In[26]:= SubstTest[implies, and[equal[x, funpart[t]], subclass[domain[x], y],
           subclass[range[x], z]], subclass[x, cart[y, z]], t → x] // Reverse

Out[26]= or[not[FUNCTION[x]], not[subclass[domain[x], y]],
           not[subclass[range[x], z]], subclass[x, cart[y, z]]] == True

In[27]:= or[not[FUNCTION[x_]], not[subclass[domain[x_], y_]],
           not[subclass[range[x_], z_]], subclass[x_, cart[y_, z_]]] := True
```

Lemma.

```
In[28]:= SubstTest[or, not[FUNCTION[w]], not[subclass[domain[w], x]],
           not[subclass[range[w], t]], subclass[w, cart[x, t]], t → intersection[y, z]] // Reverse

Out[28]= or[not[FUNCTION[w]], not[subclass[domain[w], x]], not[subclass[range[w], y]],
           not[subclass[range[w], z]], subclass[w, cart[x, intersection[y, z]]]] == True

In[29]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Technical Lemma.

```
In[30]:= ((or[category[x], not[associative[x]],
              not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
              not[FUNCTION[x]], not[subclass[x, cart[cart[range[x], range[x]],
                 intersection[domain[cod[x]], domain[dom[x]]]]]],
              not[subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]],
              not[subclass[domain[x], cart[range[x], range[x]]]], not[
               subclass[range[x], domain[cod[x]]]], not[subclass[range[x], domain[dom[x]]]]] //
            NotNotTest) /. x → x_) /. Equal → SetDelayed

In[31]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p1, p2, p3], p4], not[implies[p1, p4]],
           {p1 → and[associative[x], FUNCTION[x], subclass[composite[x, inverse[FIRST],
                domain[x]], domain[x]], subclass[domain[x], cart[range[x], range[x]]],
              subclass[range[x], domain[cod[x]]], subclass[range[x], domain[dom[x]]]],
            p2 -> equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
            p3 -> subclass[x, cart[cart[range[x], range[x]],
               intersection[domain[cod[x]], domain[dom[x]]]]], p4 → category[x]}]] // Reverse

Out[31]= or[category[x], not[associative[x]], not[FUNCTION[x]],
           not[subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]],
           not[subclass[domain[x], cart[range[x], range[x]]]],
           not[subclass[range[x], domain[cod[x]]]],
           not[subclass[range[x], domain[dom[x]]]] == True

In[32]:= or[category[x_], not[associative[x_]], not[FUNCTION[x_]],
           not[subclass[composite[x_, inverse[FIRST], domain[x_]], domain[x_]]],
           not[subclass[domain[x_], cart[range[x_], range[x_]]]],
           not[subclass[range[x_], domain[cod[x_]]]],
           not[subclass[range[x_], domain[dom[x_]]]] := True
```

flipped version of axiom 2

Except for axiom 2, each of the defining properties for a category **x** implies the corresponding statement for **flip[x]**. In this section it is shown that in the presence of axiom 1, axiom 2 implies its dual statement.

Lemma. A consequence of the associative law.

```
In[33]:= SubstTest[implies, equal[x, assoc[t]], equal[composite[inverse[domain[x]], x],
           composite[rotate[composite[domain[x], x, SWAP]], SWAP]], t → x] // Reverse

Out[33]= or[equal[composite[inverse[domain[x]], x],
           composite[rotate[composite[domain[x], x, SWAP]], SWAP]], not[associative[x]]] == True

In[34]:= or[equal[composite[inverse[domain[x_]], x_], composite[
            rotate[composite[domain[x_], x_, SWAP]], SWAP]], not[associative[x_]]] := True
```

Lemma.

```
In[35]:= subclass[composite[y, id[domain[funpart[x]]]], composite[z, funpart[x]]] // AssertTest

Out[35]= subclass[composite[y, id[domain[funpart[x]]]], composite[z, funpart[x]]] ==
          subclass[composite[y, inverse[funpart[x]]], z]

In[36]:= subclass[composite[y_, id[domain[funpart[x_]]]], composite[z_, funpart[x_]]] :=
          subclass[composite[y, inverse[funpart[x]]], z]
```

A general theorem.

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 -> subclass[composite[y, inverse[x]], z],
             p2 -> subclass[composite[y, inverse[x], x], composite[z, x]],
             p3 -> subclass[composite[y, id[domain[x]]], composite[z, x]]}]] // Reverse

Out[37]= or[not[subclass[composite[y, inverse[x]], z]],
           subclass[composite[y, id[domain[x]]], composite[z, x]]] == True

In[38]:= or[not[subclass[composite[y_, inverse[x_]], z_]],
           subclass[composite[y_, id[domain[x_]]], composite[z_, x_]]] := True
```

Lemma.

```
In[39]:= SubstTest[subclass, rotate[rotate[u]], rotate[rotate[v]],
          {u → composite[x, SECOND, id[y]], v -> rotate[z]}]

Out[39]= subclass[composite[x, SECOND, id[y]], rotate[z]] ==
          subclass[composite[inverse[y], FIRST, id[x]], z]

In[40]:= subclass[composite[x_, SECOND, id[y_]], rotate[z_]] :=
          subclass[composite[inverse[y], FIRST, id[x]], z]
```

Corollary.

```
In[41]:= SubstTest[subclass, composite[x, SECOND, id[t]], rotate[y], t → V] // Reverse

Out[41]= subclass[composite[x, SECOND], rotate[y]] == subclass[cart[composite[Id, x], V], y]

In[42]:= subclass[composite[x_, SECOND], rotate[y_]] := subclass[cart[composite[Id, x], V], y]
```

Lemma.

```
In[43]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
           {u -> composite[y, id[domain[x]]], v → z, w → inverse[x]}] // Reverse

Out[43]= or[not[subclass[composite[y, id[domain[x]]], z]],
           subclass[composite[y, inverse[x]], composite[z, inverse[x]]]] == True

In[44]:= or[not[subclass[composite[y_, id[domain[x_]]], z_]],
           subclass[composite[y_, inverse[x_]], composite[z_, inverse[x_]]]] := True
```

Lemma.

```
In[45]:= SubstTest[implies, equal[x, funpart[t]],
           or[not[subclass[composite[inverse[domain[x]], FIRST, inverse[x]],
              composite[inverse[domain[x]], x, inverse[x]]]],
            subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]], t → x] // Reverse

Out[45]= or[not[FUNCTION[x]], not[subclass[composite[inverse[domain[x]], FIRST, inverse[x]],
           composite[inverse[domain[x]], x, inverse[x]]]],
          subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[47]:= Map[not, SubstTest[and, implies[p0, p3], implies[p0, p4], implies[p0, p5],
            implies[and[p3, p5], p6], implies[p6, p7], implies[and[p0, p7], p8],
            not[implies[p0, p8]], {p0 → and[FUNCTION[x], associative[x],
               subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]],
             p3 -> equal[composite[domain[x], x], rotate[composite[inverse[domain[x]], x]]],
             p4 -> equal[composite[inverse[domain[x]], x],
               composite[rotate[composite[domain[x], x, SWAP]], SWAP]], p5 →
              subclass[composite[domain[x], SECOND, id[domain[x]]], composite[domain[x], x]],
             p6 -> subclass[composite[inverse[domain[x]], FIRST, id[domain[x]]],
               composite[inverse[domain[x]], x]],
             p7 -> subclass[composite[inverse[domain[x]], FIRST, inverse[x]],
               composite[inverse[domain[x]], x, inverse[x]]],
             p8 → subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]}]] // Reverse

Out[47]= or[not[associative[x]], not[FUNCTION[x]],
          not[subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]],
          subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]] == True

In[48]:= or[not[associative[x_]], not[FUNCTION[x_]],
          not[subclass[composite[domain[x_], SECOND, inverse[x_]], domain[x_]]],
          subclass[composite[x_, inverse[FIRST], domain[x_]], domain[x_]]] := True
```

Corollary. (Restatement using **assoc** and **funpart** wrappers.)

```
In[49]:= SubstTest[or, not[associative[t]], not[FUNCTION[t]],
          not[subclass[composite[domain[t], SECOND, inverse[t]], domain[t]]],
          subclass[composite[t, inverse[FIRST], domain[t]], domain[t]],
          t → assoc[funpart[x]]] // Reverse

Out[49]= or[not[subclass[composite[domain[assoc[funpart[x]]],
             SECOND, inverse[assoc[funpart[x]]]], domain[assoc[funpart[x]]]]],
          subclass[composite[assoc[funpart[x]], inverse[FIRST], domain[assoc[funpart[x]]]],
           domain[assoc[funpart[x]]]]] == True

In[50]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[51]:= SubstTest[or, not[subclass[composite[domain[assoc[funpart[t]]],
             SECOND, inverse[assoc[funpart[t]]]], domain[assoc[funpart[t]]]],
           subclass[composite[assoc[funpart[t]], inverse[FIRST], domain[assoc[funpart[t]]]],
            domain[assoc[funpart[t]]]], t → flip[assoc[funpart[x]]]] // Reverse

Out[51]= or[not[subclass[composite[assoc[funpart[x]], inverse[FIRST],
             domain[assoc[funpart[x]]]], domain[assoc[funpart[x]]]],
           subclass[composite[domain[assoc[funpart[x]]], SECOND, inverse[assoc[funpart[x]]]],
            domain[assoc[funpart[x]]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed
```

Main theorem.

```
In[53]:= SubstTest[implies, equal[x, assoc[funpart[t]]],
           or[not[subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]],
            subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]], t → x] // Reverse

Out[53]= or[not[associative[x]], not[FUNCTION[x]],
           not[subclass[composite[x, inverse[FIRST], domain[x]], domain[x]]],
           subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]] == True

In[54]:= or[not[associative[x_]], not[FUNCTION[x_]],
           not[subclass[composite[x_, inverse[FIRST], domain[x_]], domain[x_]]],
           subclass[composite[domain[x_], SECOND, inverse[x_]], domain[x_]]] := True
```

Corollary. Flipped version of axiom 2.

```
In[55]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[p1, p4], implies[and[p2, p3, p4], p5], not[implies[p1, p5]],
            {p1 → category[x], p2 → associative[x], p3 → FUNCTION[x],
             p4 -> subclass[composite[x, inverse[FIRST], domain[x]], domain[x]],
             p5 -> subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]}]] // Reverse

Out[55]= or[not[category[x]],
           subclass[composite[domain[x], SECOND, inverse[x]], domain[x]]] == True

In[56]:= or[not[category[x_]],
           subclass[composite[domain[x_], SECOND, inverse[x_]], domain[x_]]] := True
```

independence of axiom 2

Lemma.

```
In[58]:= subclass[cart[V, V], union[cart[V, set[0]], cart[set[0], V]]] // AssertTest

Out[58]= subclass[cart[V, V], union[cart[V, set[0]], cart[set[0], V]]] == False

In[59]:= % /. Equal → SetDelayed
```

Theorem. The function **x = union[inverse[RIGHT[0]], inverse[LEFT[0]]]** does not satisfy axiom 2, and is therefore not a category.

```
In[60]:= Map[not, SubstTest[implies, category[x],
            subclass[composite[x, inverse[FIRST], domain[x]], domain[x]],
            x → union[inverse[RIGHT[0]], inverse[LEFT[0]]]]] // Reverse

Out[60]= category[union[inverse[LEFT[0]], inverse[RIGHT[0]]]] == False

In[61]:= category[union[inverse[LEFT[0]], inverse[RIGHT[0]]]] := False
```

This function does satisfy the remaining two axioms for a category. It follows that category axiom 2 is independent of axioms 1 and 3.

```
In[62]:= and[associative[x], FUNCTION[x], subclass[domain[x], cart[range[x], range[x]]],
           subclass[range[x], domain[cod[x]]], subclass[range[x], domain[dom[x]]]] /.
          x -> union[inverse[LEFT[0]], inverse[RIGHT[0]]]

Out[62]= True
```