MacLane's second arrows-only category axiom

Johan G. F. Belinfante
2009 January 5

```
In[1]:= SetDirectory["l:"]; << goedel.09jan04a;<< tools.m

        :Package Title: goedel.09jan04a         2009 January 4 at 3:50 p.m.

        It is now:  2009 Jan 5 at 11:42

        Loading Simplification Rules

        TOOLS.M                                      Revised 2008 December 26

        weightlimit = 40
```

summary

Saunders MacLane's second arrows-only category axiom states that the composites **u · (v · w)** and **(u · v) · w** of three morphisms are both defined if and only if **u · v** and **v · w** are defined. In the **GOEDEL** program, this statement was reformulated as the requirement if a morphism **u** is composable with some left-divisor **v** of a morphism **w**, then **u** is composable with **w**. In this notebook it is shown explicitly (using the **cat[x]** wrapper) that MacLane's version of the second arrows-only category axiom follows from the conditions used to define categories in the **GOEDEL** program.

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

left divisibility

Lemma. Left divisibility.

```
In[3]:= SubstTest[implies, and[member[pair[u, t], composite[Id, z]],
          member[pair[t, w], composite[Id, y]]], member[pair[u, w], composite[y, z]],
         {t → pair[u, v], y → cat[x], z -> inverse[FIRST]}] // Reverse

Out[3]= or[member[pair[u, w], composite[cat[x], inverse[FIRST]]],
          not[member[pair[pair[u, v], w], cat[x]]]] == True

In[4]:= or[member[pair[u_, w_], composite[cat[x_], inverse[FIRST]]],
          not[member[pair[pair[u_, v_], w_], cat[x_]]]] := True
```

Lemma. If **u** is composable with **v**, then **u** is a left-divisor of **u · v**.

```
In[5]:= SubstTest[implies, member[pair[pair[u, v], w], cat[x]],
          member[pair[u, w], composite[cat[x], inverse[FIRST]]],
          w → APPLY[cat[x], PAIR[u, v]]] // Reverse

Out[5]= or[member[pair[u, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[FIRST]]],
          not[member[pair[u, v], domain[cat[x]]]]] == True

In[6]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma. Conversely, if **u** is a left-divisor of **u · v**, then **u** is composable with **v**.

```
In[7]:= SubstTest[implies, member[pair[u, w], composite[y, z]], member[w, range[y]],
          {w → APPLY[cat[x], PAIR[u, v]], y → cat[x], z → inverse[FIRST]}] // Reverse

Out[7]= or[member[pair[u, v], domain[cat[x]]], not[member[
            pair[u, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[FIRST]]]]] == True

In[8]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Theorem.

```
In[9]:= equiv[member[pair[u, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[FIRST]]],
         member[pair[u, v], domain[cat[x]]]]

Out[9]= True

In[10]:= member[pair[u_, APPLY[cat[x_], PAIR[u_, v_]]], composite[cat[x_], inverse[FIRST]]] :=
          member[pair[u, v], domain[cat[x]]]
```

---

right divisibility

Lemma. Right divisibility.

```
In[11]:= SubstTest[or, member[pair[u, w], composite[cat[t], inverse[FIRST]]],
           not[member[pair[pair[u, v], w], cat[t]]], t → flip[cat[x]]] // Reverse

Out[11]= or[member[pair[u, w], composite[cat[x], inverse[SECOND]]],
           not[member[pair[pair[v, u], w], cat[x]]]] == True

In[12]:= or[member[pair[u_, w_], composite[cat[x_], inverse[SECOND]]],
           not[member[pair[pair[v_, u_], w_], cat[x_]]]] := True
```

Theorem. If **u** is composable with **v**, then **v** is a right-divisor of **u · v**.

```
In[13]:= SubstTest[implies, member[pair[pair[u, v], w], cat[x]],
           member[pair[v, w], composite[cat[x], inverse[SECOND]]],
           w → APPLY[cat[x], PAIR[u, v]]] // Reverse

Out[13]= or[member[pair[v, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[SECOND]]],
           not[member[pair[u, v], domain[cat[x]]]]] == True

In[14]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Converse. If **v** is a left-divisor of **u · v**, then **u** is composable with **v**.

```
In[15]:= SubstTest[implies, member[pair[v, w], composite[y, z]], member[w, range[y]],
           {w → APPLY[cat[x], PAIR[u, v]], y → cat[x], z → inverse[SECOND]}] // Reverse

Out[15]= or[member[pair[u, v], domain[cat[x]]], not[member[
             pair[v, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[SECOND]]]]] == True

In[16]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Theorem.

```
In[17]:= equiv[member[pair[v, APPLY[cat[x], PAIR[u, v]]], composite[cat[x], inverse[SECOND]]],
          member[pair[u, v], domain[cat[x]]]]

Out[17]= True

In[18]:= member[pair[v_, APPLY[cat[x_], PAIR[u_, v_]]], composite[cat[x_], inverse[SECOND]]] :=
          member[pair[u, v], domain[cat[x]]]
```

---

category axiom 2

The second axiom used to define the predicate **category[x]** in the **GOEDEL** program is the requirement that the composite of the left-divisibility relation and the composability relation is contained in the composability relation. Using the **cat[x]** wrapper, this translates into the following:

```
In[19]:= subclass[composite[cat[x], inverse[FIRST], domain[cat[x]]], domain[cat[x]]]

Out[19]= True
```

It will be shown in this section that associativity in combination with this condition implies MacLane's two parts of his second arrows-only category axiom.

Lemma.

```
In[20]:= SubstTest[implies, and[member[PAIR[u, v], z], member[PAIR[v, t], y]],
           member[PAIR[u, t], composite[y, z]], {t → APPLY[cat[x], PAIR[v, w]],
            y -> composite[cat[x], inverse[FIRST]], z -> domain[cat[x]]}] // Reverse

Out[20]= or[member[pair[u, APPLY[cat[x], PAIR[v, w]]],
             composite[cat[x], inverse[FIRST], domain[cat[x]]]],
           not[member[pair[u, v], domain[cat[x]]]],
           not[member[pair[v, w], domain[cat[x]]]]] == True

In[21]:= (% /. {u → u_, v → v_, w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. If morphisms **u** and **v** are composable, and if **v** and **w** are composable, then **u** and **v · w** are composable.

```
In[22]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[p3, p4], not[implies[and[p1, p2], p4]],
            {p1 -> member[pair[u, v], domain[cat[x]]], p2 -> member[pair[v, w], domain[cat[x]]],
             p3 -> member[pair[u, APPLY[cat[x], PAIR[v, w]]],
               composite[cat[x], inverse[FIRST], domain[cat[x]]]],
             p4 -> member[pair[u, APPLY[cat[x], PAIR[v, w]]], domain[cat[x]]]}]] // Reverse

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

In[23]:= or[member[pair[u_, APPLY[cat[x_], PAIR[v_, w_]]], domain[cat[x_]]],
           not[member[pair[u_, v_], domain[cat[x_]]]],
           not[member[pair[v_, w_], domain[cat[x_]]]]] := True
```

Corollary. If morphisms **u** and **v** are composable, and if **v** and **w** are composable, then **u · v** and **w** are composable.

```
In[24]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[p3, p4], not[implies[and[p1, p2], p4]],
            {p1 -> member[pair[u, v], domain[cat[x]]], p2 -> member[pair[v, w], domain[cat[x]]],
             p3 -> member[pair[u, APPLY[cat[x], PAIR[v, w]]], domain[cat[x]]],
             p4 → member[pair[APPLY[cat[x], PAIR[u, v]], w], domain[cat[x]]]}]] // Reverse

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

In[25]:= or[member[pair[APPLY[cat[x_], PAIR[u_, v_]], w_], domain[cat[x_]]],
           not[member[pair[u_, v_], domain[cat[x_]]]],
           not[member[pair[v_, w_], domain[cat[x_]]]]] := True
```

Comment. The **only if** parts of MacLane's second arrows-only category axiom have previously been shown to follow from the associative law:

```
In[26]:= implies[member[pair[APPLY[cat[x], PAIR[u, v]], w], domain[cat[x]]],
            and[member[pair[u, v], domain[cat[x]]],
             member[pair[v, w], domain[cat[x]]]]] // not // not

Out[26]= True

In[27]:= implies[member[pair[u, APPLY[cat[x], PAIR[v, w]]], domain[cat[x]]],
            and[member[pair[u, v], domain[cat[x]]],
             member[pair[v, w], domain[cat[x]]]]] // not // not

Out[27]= True
```