monoidal categories

Johan G. F. Belinfante
2009 July 9

```
In[1]:= SetDirectory["l:"]; << goedel.09jul08a; << tools.m

        :Package Title: goedel.09jul08a              2009 July 8 at 3:50 p.m.

        It is now:  2009 Jul 9 at 10:0

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 July 2

        weightlimit = 40
```

summary

A **monoidal category** is a category with just one identity morphism. In this notebook it is shown that a category is monoidal if and only if its domain is the cartesian square of its range. For small categories, the following variable-free statement is already available.

```
In[2]:= intersection[CATS, BINOPS]

Out[2]= union[MONOIDS, set[0]]
```

derivation

Lemma. Uniqueness of identities.

```
In[3]:= SubstTest[implies, equal[u, v], equal[image[t, u], image[t, v]],
          {t → id[cartsq[ids[x]]], u → domain[x], v → cartsq[range[x]]}] // Reverse

Out[3]= or[equal[0, ids[x]], member[ids[x], range[SINGLETON]],
          not[equal[cart[range[x], range[x]], domain[x]]]] == True

In[4]:= or[equal[0, ids[x_]], member[ids[x_], range[SINGLETON]],
          not[equal[cart[range[x_], range[x_]], domain[x_]]]] := True
```

For the special case of a category, this statement is rewritten as follows. (A better rewrite rule will be derived below.)

Corollary.

```
In[5]:= SubstTest[or, equal[0, ids[t]], member[ids[t], range[SINGLETON]],
          not[equal[cart[range[t], range[t]], domain[t]]], t → cat[x]] // Reverse

Out[5]= or[equal[0, cat[x]], member[ids[cat[x]], range[SINGLETON]],
          not[equal[cart[range[cat[x]], range[cat[x]]], domain[cat[x]]]]] == True

In[6]:= (% /. x → x_) /. Equal → SetDelayed
```

For the reverse implication the hypothesis that **x** is a category will be needed. (The **cat[x]** wrapper is used to add this hypothesis.)

Lemma.

```
In[7]:= SubstTest[implies, equal[u, v], equal[composite[domain[cat[x]], id[u], domain[cat[x]]],
           composite[domain[cat[x]], id[v], domain[cat[x]]]],
          {u → ids[cat[x]], v → set[t]}] // Reverse

Out[7]= or[equal[cart[image[inverse[domain[cat[x]]], set[t]], image[domain[cat[x]], set[t]]],
           domain[cat[x]]], not[equal[ids[cat[x]], set[t]]]] == True

In[8]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[9]:= SubstTest[implies, equal[t, cart[u, v]],
          equal[t, cart[domain[t], range[t]]], t → domain[cat[x]]] // Reverse

Out[9]= or[equal[cart[range[cat[x]], range[cat[x]]], domain[cat[x]]],
          not[equal[cart[u, v], domain[cat[x]]]]] == True

In[10]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Theorem. If a category has only one identity morphism, then its domain is the cartesian square of its range.

```
In[11]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p3, p4],
            not[implies[p1, p4]], {p1 → member[ids[cat[x]], range[SINGLETON]],
             p2 -> equal[ids[cat[x]], set[e[cat[x]]]],
             p3 -> equal[cart[image[inverse[domain[cat[x]]], set[e[cat[x]]]],
                image[domain[cat[x]], set[e[cat[x]]]]], domain[cat[x]]],
             p4 → equal[domain[cat[x]], cartsq[range[cat[x]]]]}]] // Reverse

Out[11]= or[equal[cart[range[cat[x]], range[cat[x]]], domain[cat[x]]],
           not[member[ids[cat[x]], range[SINGLETON]]]] == True

In[12]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

Corollary. (Restatement without the **cat** wrapper.)

```
In[13]:= SubstTest[implies, equal[x, cat[t]], or[equal[cart[range[x], range[x]], domain[x]],
            not[member[ids[x], range[SINGLETON]]]], t → x] // Reverse

Out[13]= or[equal[cart[range[x], range[x]], domain[x]],
           not[category[x]], not[member[ids[x], range[SINGLETON]]]] == True
```

```
In[14]:= or[equal[cart[range[x_], range[x_]], domain[x_]],
            not[category[x_]], not[member[ids[x_], range[SINGLETON]]]] := True
```

If the **cat[x]** wrapper is used, one can combine the theorem and its converse into a single rewrite rule.

```
In[15]:= equiv[equal[domain[cat[x]], cartsq[range[cat[x]]]],
             or[empty[ids[cat[x]]], member[ids[cat[x]], range[SINGLETON]]]] // not // not

Out[15]= True

In[16]:= equal[cart[range[cat[x_]], range[cat[x_]]], domain[cat[x_]]] :=
           or[equal[0, cat[x]], member[ids[cat[x]], range[SINGLETON]]]
```

A useful lemma derived above is rewritten by this rewrite rule, and will now be restated.

Theorem.

```
In[17]:= SubstTest[implies, equal[t, cart[u, v]],
            equal[t, cart[domain[t], range[t]]], t → domain[cat[x]]] // Reverse

Out[17]= or[equal[0, cat[x]], member[ids[cat[x]], range[SINGLETON]],
            not[equal[cart[u, v], domain[cat[x]]]]] == True

In[18]:= or[equal[0, cat[x_]], member[ids[cat[x_]], range[SINGLETON]],
            not[equal[cart[u_, v_], domain[cat[x_]]]]] := True
```

Corollary. Restatement without the **cat[x]** wrapper.

```
In[19]:= SubstTest[implies, equal[x, cat[t]], or[equal[0, x], member[ids[x], range[SINGLETON]],
             not[equal[cart[u, v], domain[x]]]], t → x] // Reverse

Out[19]= or[equal[0, x], member[ids[x], range[SINGLETON]],
            not[category[x]], not[equal[cart[u, v], domain[x]]]] == True

In[20]:= or[equal[0, x_], member[ids[x_], range[SINGLETON]],
            not[category[x_]], not[equal[cart[u_, v_], domain[x_]]]] := True
```