idempotents in a category

Johan G. F. Belinfante
2009 August 7

```
In[1]:= SetDirectory["l:"]; << goedel.09aug06a; << tools.m

        :Package Title: goedel.09aug06a              2009 August 6 at 4:30 p.m.

        It is now:  2009 Aug 7 at 13:49

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 July 2

        weightlimit = 40
```

summary

Idempotent morphisms in a category are endomorphisms. An invertible idempotent is an identity morphism.

derivation

In a category, if **u u = u** and **u v = e** is an identity, then **e = u v = u u v = u e = u**. This argument will now be translated into the language of the **GOEDEL** program.

Lemma.

```
In[3]:= SubstTest[implies, and[member[pair[u, w], domain[cat[x]]], member[w, ids[cat[x]]]],
          equal[u, APPLY[cat[x], PAIR[u, w]]], w → APPLY[cat[x], PAIR[u, v]]] // Reverse

Out[3]= or[equal[u, APPLY[cat[x], PAIR[APPLY[cat[x], PAIR[u, u]], v]]],
          not[member[APPLY[cat[x], PAIR[u, v]], ids[cat[x]]]],
          not[member[pair[u, APPLY[cat[x], PAIR[u, v]]], domain[cat[x]]]]] == True

In[4]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[5]:= SubstTest[implies, and[equal[APPLY[cat[x], PAIR[u, v]], w], member[w, range[cat[x]]]],
          member[APPLY[cat[x], PAIR[u, v]], range[cat[x]]], {v → u, w → u}] // Reverse

Out[5]= or[member[pair[u, u], domain[cat[x]]],
          not[equal[u, APPLY[cat[x], PAIR[u, u]]]], not[member[u, range[cat[x]]]]] == True

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

Theorem. An idempotent with a right-inverse is an identity.

```
In[7]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p0, p2], p3], implies[p1, p4],
          implies[and[p3, p4], p5], implies[and[p1, p5], p6], implies[and[p0, p1, p6], p7],
          not[implies[and[p0, p1], p7]], {p0 → equal[APPLY[cat[x], PAIR[u, u]], u],
           p1 → member[APPLY[cat[x], PAIR[u, v]], ids[cat[x]]], p2 → member[u, range[cat[x]]],
           p3 → member[pair[u, u], domain[cat[x]]],
           p4 -> member[pair[u, v], domain[cat[x]]],
           p5 → member[pair[u, APPLY[cat[x], PAIR[u, v]]], domain[cat[x]]],
           p6 -> equal[u, APPLY[cat[x], PAIR[APPLY[cat[x], PAIR[u, u]], v]]],
           p7 -> member[u, ids[cat[x]]]}]] // Reverse

Out[7]= or[member[u, ids[cat[x]]], not[equal[u, APPLY[cat[x], PAIR[u, u]]]],
          not[member[APPLY[cat[x], PAIR[u, v]], ids[cat[x]]]]] == True

In[8]:= or[member[u_, ids[cat[x_]]], not[equal[u_, APPLY[cat[x_], PAIR[u_, u_]]]],
          not[member[APPLY[cat[x_], PAIR[u_, v_]], ids[cat[x_]]]]] := True
```

To facilitate the elimination of the variables **u** and **v**, one of the two **APPLY** constructors is eliminated.

Corollary.

```
In[9]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[and[p1, p2], p4]], {p1 -> member[pair[pair[u, u], u], cat[x]],
           p2 -> member[APPLY[cat[x], PAIR[u, v]], ids[cat[x]]],
           p3 → equal[u, APPLY[cat[x], PAIR[u, u]]], p4 → member[u, ids[cat[x]]]}]] // Reverse

Out[9]= or[member[u, ids[cat[x]]], not[member[APPLY[cat[x], PAIR[u, v]], ids[cat[x]]]],
          not[member[pair[pair[u, u], u], cat[x]]]] == True

In[10]:= or[member[u_, ids[cat[x_]]], not[member[APPLY[cat[x_], PAIR[u_, v_]], ids[cat[x_]]]],
           not[member[pair[pair[u_, u_], u_], cat[x_]]]] := True
```

Eliminating the morphism variables yields this inclusion:

Lemma.

```
In[11]:= Map[empty[domain[complement[#]]] &, SubstTest[class, pair[u, v],
           implies[and[member[u, w], member[pair[u, v], y]], member[u, z]],
           {w -> fix[composite[cat[x], DUP]],
            y -> image[inverse[cat[x]], ids[cat[x]]], z -> ids[cat[x]]}]]

Out[11]= subclass[intersection[domain[image[inverse[cat[x]], ids[cat[x]]]],
            fix[composite[cat[x], DUP]]], ids[cat[x]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

The reverse inclusion also holds, so one obtains an equation that can be made into a rewrite rule.

Theorem. A right-invertible idempotent is an identity.

```
In[15]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> intersection[domain[image[inverse[cat[x]], ids[cat[x]]]],
             fix[composite[cat[x], DUP]]], v -> ids[cat[x]]}]

Out[15]= equal[ids[cat[x]], intersection[
           domain[image[inverse[cat[x]], ids[cat[x]]]], fix[composite[cat[x], DUP]]]] == True

In[16]:= intersection[domain[image[inverse[cat[x_]], ids[cat[x_]]]],
           fix[composite[cat[x_], DUP]]] := ids[cat[x]]
```

Corollary. (Dual result.)

```
In[17]:= SubstTest[intersection, domain[image[inverse[cat[t]], ids[cat[t]]]],
           fix[composite[cat[t], DUP]], t → flip[cat[x]]] // Reverse

Out[17]= intersection[fix[composite[cat[x], DUP]],
           range[image[inverse[cat[x]], ids[cat[x]]]]] == ids[cat[x]]

In[18]:= intersection[fix[composite[cat[x_], DUP]],
           range[image[inverse[cat[x_]], ids[cat[x_]]]]] := ids[cat[x]]
```

Corollary. An invertible idempotent is an identity.

```
In[19]:= Map[equal[#, ids[cat[x]]] &,
          AssInt[fix[composite[cat[x], DUP]], range[image[inverse[cat[x]], ids[cat[x]]]],
           domain[image[inverse[cat[x]], ids[cat[x]]]]]]

Out[19]= equal[ids[cat[x]],
           intersection[domain[inv[cat[x]]], fix[composite[cat[x], DUP]]]] == True

In[21]:= intersection[domain[inv[cat[x_]]], fix[composite[cat[x_], DUP]]] := ids[cat[x]]
```