inv[RATMUL]

Johan G. F. Belinfante
2012 September 24

```
In[1]:= SetDirectory["l:"]; << goedel.12sep23a
```

```
:Package Title: goedel.12sep23a           2012 September 23 at 10:20 p.m.

Loading takes about sixteen minutes, half that time due to builtin pauses.

It is now:  2012 Sep 24 at 16:35

Loading Simplification Rules

TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

weightlimit = 40

Loading completed.

It is now:  2012 Sep 24 at 16:51
```

summary

The inversion function **inv[RATMUL]** for rational multiplication is the two-sided restriction of **INVERSE** to **RATS**.

derivation

The rational number zero is the constant function **Z** × **{id[ω]}**. Its range is a singleton. All other rational numbers have infinite ranges.

Lemma. The only rational number with trivial range is the rational number zero.

```
In[3]:= equiv[subclass[range[rat[x]], set[id[omega]]], equal[cart[Z, set[id[omega]]], rat[x]]]
```

```
Out[3]= True
```

```
In[4]:= subclass[range[rat[x_]], set[id[omega]]] := equal[cart[Z, set[id[omega]]], rat[x]]
```

If a set with non-trivial domain is a subclass of a rational number, then its rational hull is that rational number. The following lemma is an application of this to the case of the identity function restricted to the range of a rational number. These are subsets of the rational number one, which is **id[Z]**.

Lemma. The rational hull of **id[range[rat[x]]]** for any non-zero rational number **rat[x]** is **id[Z]**.

```
In[5]:= SubstTest[implies, and[not[subclass[domain[u], set[id[omega]]]], subclass[u, rat[t]]],
          equal[hull[RATS, u], rat[t]], {u → id[range[rat[x]]], t → id[Z]}] // Reverse

Out[5]= or[equal[cart[Z, set[id[omega]]], rat[x]],
          equal[hull[RATS, id[range[rat[x]]]], id[Z]]] == True

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

Lemma. Simplification rule.

```
In[7]:= equiv[equal[cart[set[x], set[x]], id[y]], equal[y, set[x]]] // assert

Out[7]= True

In[8]:= equal[cart[set[x_], set[x_]], id[y_]] := equal[y, set[x]]
```

Theorem. The rational hull of the identity on the range of any non-zero rational number is **id[Z]**.

```
In[9]:= equiv[equal[hull[RATS, id[range[rat[x]]]], id[Z]],
         not[equal[cart[Z, set[id[omega]]], rat[x]]]]

Out[9]= True

In[10]:= equal[hull[RATS, id[range[rat[x_]]]], id[Z]] :=
          not[equal[cart[Z, set[id[omega]]], rat[x]]]
```

Lemma. A formula for the product of any non-zero rational number and its inverse.

```
In[11]:= SubstTest[implies, and[member[u, RATS], member[v, RATS]],
           equal[ratmul[u, v], hull[RATS, composite[u, v]]],
           {u → rat[x], v → inverse[rat[x]]}] // Reverse

Out[11]= or[equal[cart[Z, set[id[omega]]], rat[x]],
           equal[hull[RATS, id[range[rat[x]]]], ratmul[rat[x], inverse[rat[x]]]]] == True

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

Theorem. The product of any non-zero rational number and its inverse is equal to **id[Z]**.

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2],
            not[implies[p1, p3]], {p1 -> not[equal[cart[Z, set[id[omega]]], rat[x]]],
             p2 -> equal[hull[RATS, id[range[rat[x]]]], ratmul[rat[x], inverse[rat[x]]]],
             p3 -> equal[id[Z], ratmul[rat[x], inverse[rat[x]]]]}]] // Reverse

Out[13]= or[equal[cart[Z, set[id[omega]]], rat[x]],
           equal[id[Z], ratmul[rat[x], inverse[rat[x]]]]] == True

In[14]:= (% /. x → x_) /. Equal → SetDelayed
```

A slightly better rewrite rule for the preceding result can be derived by combining it with its converse. A rewrite rule will be derived that does not involve the **rat** wrapper.

Lemma. A simplification rule.

```
In[15]:= equal[ratmul[x, cart[set[y], z]], V]

Out[15]= True

In[16]:= ratmul[x_, cart[set[y_], z_]] := V
```

Corollary. (Remove the **rat** wrapper.)

```
In[17]:= SubstTest[implies, equal[x, rat[t]], or[equal[cart[Z, set[id[omega]]], x],
           equal[id[Z], ratmul[x, inverse[x]]]], t → x] // Reverse

Out[17]= or[equal[x, cart[Z, set[id[omega]]]],
          equal[id[Z], ratmul[x, inverse[x]]], not[member[x, RATS]]] == True

In[18]:= or[equal[x_, cart[Z, set[id[omega]]]],
          equal[id[Z], ratmul[x_, inverse[x_]]], not[member[x_, RATS]]] := True
```

Corollary. (A similar result for the product in the reverse order.)

```
In[19]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]],
            p2 -> equal[id[Z], ratmul[x, inverse[x]]],
            p3 -> equal[id[Z], ratmul[inverse[x], x]]}]] // Reverse

Out[19]= or[equal[x, cart[Z, set[id[omega]]]],
          equal[id[Z], ratmul[inverse[x], x]], not[member[x, RATS]]] == True

In[20]:= or[equal[x_, cart[Z, set[id[omega]]]],
          equal[id[Z], ratmul[inverse[x_], x_]], not[member[x_, RATS]]] := True
```

Lemma. A converse.

```
In[21]:= Map[or[member[x, RATS], #] &, SubstTest[implies,
           equal[t, id[Z]], member[t, V], t → ratmul[x, inverse[x]]]] // Reverse

Out[21]= or[member[x, RATS], not[equal[id[Z], ratmul[x, inverse[x]]]]] == True

In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. A wrapper-free rewrite rule concerning the product of a rational number and its inverse.

```
In[23]:= equiv[equal[id[Z], ratmul[x, inverse[x]]],
           and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]] // not // not

Out[23]= True

In[24]:= equal[id[Z], ratmul[x_, inverse[x_]]] :=
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]
```

A similar result will be derived for the product in the reverse order.

Lemma.

```
In[25]:= Map[or[member[x, RATS], #] &, SubstTest[implies,
           equal[t, id[Z]], member[t, V], t → ratmul[inverse[x], x]]] // Reverse

Out[25]= or[member[x, RATS], not[equal[id[Z], ratmul[inverse[x], x]]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[27]:= equal[ratmul[cart[set[x], y], z], V]

Out[27]= True

In[28]:= ratmul[cart[set[x_], y_], z_] := V
```

Corollary. A wrapper-free rule for the product of a rational number and its inverse in the reverse order.

```
In[29]:= equiv[equal[id[Z], ratmul[inverse[x], x]],
           and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]] // not // not

Out[29]= True

In[30]:= equal[id[Z], ratmul[inverse[x_], x_]] :=
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]
```

Lemma. A temporary simplification rule.

```
In[32]:= equiv[and[member[x, RATS],
            member[inverse[x], RATS], not[equal[x, cart[Z, set[id[omega]]]]]],
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]]

Out[32]= True

In[33]:= and[member[x_, RATS], member[inverse[x_], RATS],
            not[equal[x_, cart[Z, set[id[omega]]]]]] :=
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]
```

Lemma.

```
In[34]:= SubstTest[member, pair[inverse[x], x],
            image[inverse[RATMUL], y], y → set[id[Z]]] // Reverse

Out[34]= member[pair[pair[inverse[x], x], id[Z]], RATMUL] ==
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]

In[36]:= member[pair[pair[inverse[x_], x_], id[Z]], RATMUL] :=
          and[member[x, RATS], not[equal[x, cart[Z, set[id[omega]]]]]]
```

Lemma.

```
In[37]:= fix[composite[image[inverse[RATMUL], set[id[Z]]], INVERSE]] // Normality

Out[37]= fix[composite[image[inverse[RATMUL], set[id[Z]]], INVERSE]] ==
          intersection[RATS, complement[set[cart[Z, set[id[omega]]]]]]
```

```
In[38]:= fix[composite[image[inverse[RATMUL], set[id[Z]]], INVERSE]] :=
          intersection[RATS, complement[set[cart[Z, set[id[omega]]]]]]
```

Lemma.

```
In[39]:= SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
           {u -> id[dif[RATS, set[cart[Z, set[id[omega]]]]]],
            v -> composite[image[inverse[RATMUL], set[id[Z]]], INVERSE],
            w → composite[id[P[cart[V, V]]], INVERSE]}] // Reverse
```

```
Out[39]= subclass[image[RATMUL, INVERSE], set[id[Z]]] == True
```

```
In[40]:= % /. Equal→ SetDelayed
```

Lemma. Inversion is a function for any semigroup.

```
In[41]:= SubstTest[FUNCTION, inv[semigp[t]], t → RATMUL] // Reverse
```

```
Out[41]= FUNCTION[inv[RATMUL]] == True
```

```
In[42]:= % /. Equal→ SetDelayed
```

Lemma.

```
In[43]:= Map[FUNCTION, inv[RATMUL] // Normality] // Reverse
```

```
Out[43]= FUNCTION[image[inverse[RATMUL], set[id[Z]]]] == True
```

```
In[44]:= % /. Equal→ SetDelayed
```

Lemma. An upper bound for the domain of the function **image[inverse[RATMUL], {id[Z]}]**.

```
In[45]:= SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
           {u -> image[inverse[RATMUL], set[id[Z]]], v -> cartsq[RATS]}] // Reverse
```

```
Out[45]= subclass[domain[image[inverse[RATMUL], set[id[Z]]], RATS] == True
```

```
In[46]:= % /. Equal→ SetDelayed
```

Lemma. A lower bound for the same function.

```
In[47]:= SubstTest[implies, subclass[u, v],
           subclass[domain[u], domain[v]], {u -> composite[id[RATS], INVERSE, id[RATS]],
            v -> image[inverse[RATMUL], set[id[Z]]]}] // Reverse
```

```
Out[47]= subclass[RATS, union[
            domain[image[inverse[RATMUL], set[id[Z]]]], set[cart[Z, set[id[omega]]]]]] == True
```

```
In[48]:= % /. Equal→ SetDelayed
```

Lemma. A simplification rule.

```
In[49]:= equal[cart[x, set[y]], id[x]] // AssertTest
```

```
Out[49]= equal[cart[x, set[y]], id[x]] == subclass[x, set[y]]
```

```
In[50]:= equal[cart[x_, set[y_]], id[x_]] := subclass[x, set[y]]
```

Theorem. Zero does not divide one.

```
In[51]:= Map[member[id[Z], #] &, SubstTest[range, composite[binop[t], LEFT[x]], t → RATMUL]] /.
          x → cart[Z, set[id[omega]]]
```

```
Out[51]= member[pair[cart[Z, set[id[omega]]], id[Z]], composite[RATMUL, inverse[FIRST]]] == False
```

```
In[52]:= % /. Equal→ SetDelayed
```

Theorem. An equation for the domain of the function **image[inverse[RATMUL], {id[Z]}]**.

```
In[53]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> domain[image[inverse[RATMUL], set[id[Z]]]],
            v -> intersection[RATS, complement[set[cart[Z, set[id[omega]]]]]]}]
```

```
Out[53]= equal[domain[image[inverse[RATMUL], set[id[Z]]]],
            intersection[RATS, complement[set[cart[Z, set[id[omega]]]]]]] == True
```

```
In[54]:= domain[image[inverse[RATMUL], set[id[Z]]]] :=
          intersection[RATS, complement[set[cart[Z, set[id[omega]]]]]]
```

Theorem. An equation for the function **image[inverse[RATMUL], {id[Z]}]**.

```
In[55]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
            equal[u, composite[v, id[domain[u]]]], {u -> composite[id[RATS], INVERSE, id[RATS]],
             v -> image[inverse[RATMUL], set[id[Z]]]}] // Reverse
```

```
Out[55]= equal[composite[id[RATS], INVERSE, id[RATS]],
            image[inverse[RATMUL], set[id[Z]]]] == True
```

```
In[56]:= image[inverse[RATMUL], set[id[Z]]] := composite[id[RATS], INVERSE, id[RATS]]
```

Theorem. An equation for the inversion function **inv[RATMUL]**.

```
In[57]:= SubstTest[intersection, image[inverse[t], ids[t]],
          inverse[image[inverse[t], ids[t]]], t → RATMUL] // Reverse
```

```
Out[57]= inv[RATMUL] == composite[id[RATS], INVERSE, id[RATS]]
```

```
In[58]:= inv[RATMUL] := composite[id[RATS], INVERSE, id[RATS]]
```

Corollary. An equation for **image[RATMUL, INVERSE]**.

```
In[59]:= SubstTest[image, binop[t], inv[binop[t]], t → RATMUL] // Reverse
```

```
Out[59]= image[RATMUL, INVERSE] == set[id[Z]]
```

```
In[60]:= image[RATMUL, INVERSE] := set[id[Z]]
```