rational multiplication is commutative

Johan G. F. Belinfante
2012 August 16

In[1]:=
```
SetDirectory["l:"]; << goedel.12aug11a
```

```
:Package Title: goedel.12aug11a                2012 August 11 at 8:25 p.m.

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

It is now:  2012 Aug 16 at 14:48

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 Aug 16 at 15:3
```

summary

An explicit formula for the product of two rational number is used to show that rational multiplication is commutative. It is also shown that **id[Z]** is the neutral element for rational multiplication.

temporary abbreviations

It is suggestive to write fractions as **d \ n** instead of **n / d**. The following temporary abbreviation is introduced for the fraction **x \ y**.

In[2]:=
```
frac[x_, y_] := composite[inverse[inttimes[x]], inttimes[y]]
```

In addition, the following temporary abbreviation saves some writing.

In[3]:=
```
rat[x_] := frac[first[x], second[x]]
```

derivation

Multiplication of fractions involves two numerators and two denominators. Each rational number is a straight line through the origin in the integer plane **Z × Z**. To reduce the number of variables, it is convenient to introduce just one variable for each fraction, representing some point **x** other than the origin on the straight line for that fraction. The denominator is then written as **first[x]**, and the numerator as **second[x]**. The fraction itself can be written as **APPLY[RATIO, x]** or as **rat[x]**. These are equal only when **x ∈ domain[RATIO]**. The following two lemmas reflect the definition of rational multiplication, using each of these two ways of writing a fraction.

Lemma. (Rewriting the hull of a composite as a rational product.

```
In[4]:= SubstTest[implies, and[member[u, RATS], member[v, RATS]],
          equal[hull[RATS, composite[u, v]], ratmul[u, v]],
          {u → APPLY[RATIO, x], v → APPLY[RATIO, y]}] // Reverse

Out[4]= or[equal[first[x], id[omega]], equal[first[y], id[omega]],
          equal[hull[RATS, composite[APPLY[RATIO, x], APPLY[RATIO, y]]],
           ratmul[APPLY[RATIO, x], APPLY[RATIO, y]]], not[member[first[x], Z]],
          not[member[first[y], Z]], not[member[second[x], Z]], not[member[second[y], Z]]] == True

In[5]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Rewriting the hull of a composite as a rational product.)

```
In[6]:= SubstTest[implies, and[member[u, RATS], member[v, RATS]],
          equal[hull[RATS, composite[u, v]], ratmul[u, v]], {u → rat[x], v → rat[y]}] // Reverse

Out[6]= or[equal[first[x], id[omega]], equal[first[y], id[omega]],
          equal[hull[RATS, composite[inverse[inttimes[first[x]]],
             inttimes[second[x]], inverse[inttimes[first[y]]], inttimes[second[y]]]],
           ratmul[composite[inverse[inttimes[first[x]]], inttimes[second[x]]],
            composite[inverse[inttimes[first[y]]], inttimes[second[y]]]]],
          not[member[first[x], Z]], not[member[first[y], Z]],
          not[member[second[x], Z]], not[member[second[y], Z]]] == True

In[7]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The following result relates rational multiplication to integer multiplication. It says that **(a \ b) · (c \ d) = (a · c) \ (b · d)**.

Theorem. An explicit formula for the product of two rational numbers.

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[p1, p4]], {p1 -> and[member[x, domain[RATIO]], member[y, domain[RATIO]]],
           p2 -> equal[hull[RATS, composite[rat[x], rat[y]]],
             frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]],
           p3 -> equal[hull[RATS, composite[rat[x], rat[y]]], ratmul[rat[x], rat[y]]],
           p4 -> equal[ratmul[rat[x], rat[y]],
             frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]]}]] // Reverse

Out[8]= or[equal[composite[inverse[inttimes[intmul[first[x], first[y]]]],
           inttimes[intmul[second[x], second[y]]]],
          ratmul[composite[inverse[inttimes[first[x]]], inttimes[second[x]]],
           composite[inverse[inttimes[first[y]]], inttimes[second[y]]]]],
        equal[first[x], id[omega]], equal[first[y], id[omega]],
        not[member[first[x], Z]], not[member[first[y], Z]],
        not[member[second[x], Z]], not[member[second[y], Z]]] == True

In[9]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The following corollary restates theorem using **APPLY[RATIO, x]** instead of **rat[x]** for each fraction. This makes it easier to later to eliminate the variables.

Corollary. Another explicit formula for the product of two rational numbers.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[p1, p4], implies[and[p2, p3, p4], p5], not[implies[p1, p5]],
          {p1 -> and[member[x, domain[RATIO]], member[y, domain[RATIO]]],
           p2 -> equal[ratmul[rat[x], rat[y]],
             frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]],
           p3 -> equal[APPLY[RATIO, x], rat[x]], p4 -> equal[APPLY[RATIO, y], rat[y]],
           p5 → equal[ratmul[APPLY[RATIO, x], APPLY[RATIO, y]],
             frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]]}]] // Reverse

Out[10]= or[equal[composite[inverse[inttimes[intmul[first[x], first[y]]]],
            inttimes[intmul[second[x], second[y]]]], ratmul[APPLY[RATIO, x], APPLY[RATIO, y]]],
          equal[first[x], id[omega]], equal[first[y], id[omega]], not[member[first[x], Z]],
          not[member[first[y], Z]], not[member[second[x], Z]], not[member[second[y], Z]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

It is possible, but difficult, to eliminate the variables **x** and **y** in this result. This will be done in a separate notebook. For now, a corollary will be derived that does not involve the constructor **intmul**. It depends only on the fact that integer multiplication is commutative.

Theorem. A form of the commutative law for multiplying fractions that involves two variables.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
            not[implies[p1, p4]], {p1 -> and[member[x, domain[RATIO]], member[y, domain[RATIO]]],
             p2 -> equal[ratmul[APPLY[RATIO, x], APPLY[RATIO, y]],
               frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]],
             p3 -> equal[ratmul[APPLY[RATIO, y], APPLY[RATIO, x]],
               frac[intmul[first[x], first[y]], intmul[second[x], second[y]]]],
             p4 → equal[ratmul[APPLY[RATIO, x], APPLY[RATIO, y]],
               ratmul[APPLY[RATIO, y], APPLY[RATIO, x]]]}]] // Reverse

Out[12]= or[equal[first[x], id[omega]], equal[first[y], id[omega]],
          equal[ratmul[APPLY[RATIO, x], APPLY[RATIO, y]],
           ratmul[APPLY[RATIO, y], APPLY[RATIO, x]]], not[member[first[x], Z]],
          not[member[first[y], Z]], not[member[second[x], Z]], not[member[second[y], Z]]] == True

In[13]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A simplification rule. (The rational product of two classes is a set exactly when both classes are rational numbers.)

```
In[14]:= image[V, set[ratmul[x, y]]] // Normality

Out[14]= image[V, set[ratmul[x, y]]] == intersection[
          image[V, intersection[RATS, set[x]]], image[V, intersection[RATS, set[y]]]]

In[15]:= image[V, set[ratmul[x_, y_]]] := intersection[
          image[V, intersection[RATS, set[x]]], image[V, intersection[RATS, set[y]]]]
```

Theorem. A simplification rule. (The fraction **APPLY[RATIO, x]** is a rational number exactly when **x ∈ domain[RATIO]**.)

```
In[16]:= SubstTest[case, member[t, RATS], t → APPLY[RATIO, x]]

Out[16]= image[V, intersection[RATS, set[APPLY[RATIO, x]]]] ==
         case[and[member[first[x], Z], member[second[x], Z], not[equal[first[x], id[omega]]]]]

In[17]:= image[V, intersection[RATS, set[APPLY[RATIO, x_]]]] :=
         case[and[member[first[x], Z], member[second[x], Z], not[equal[first[x], id[omega]]]]]
```

Lemma. (Eliminating both variables at the same time. This takes a while.)

```
In[18]:= Map[equal[V, domain[#]] &,
         SubstTest[reify, x, case[implies[member[x, u], equal[APPLY[funpart[v],
              PAIR[APPLY[funpart[w], first[x]], APPLY[funpart[w], second[x]]]], APPLY[
              funpart[v], PAIR[APPLY[funpart[w], second[x]], APPLY[funpart[w], first[x]]]]]]],
          {u → cartsq[domain[RATIO]], v → RATMUL, w → RATIO}]]

Out[18]= subclass[cart[cart[intersection[Z, complement[set[id[omega]]]], Z],
           cart[intersection[Z, complement[set[id[omega]]]], Z]], composite[
           inverse[RATIO], fix[composite[inverse[RATMUL], RATMUL, SWAP]], RATIO]] == True

In[19]:= % /. Equal → SetDelayed
```

Lemma. A better variable-free statement.

```
In[20]:= SubstTest[implies, subclass[u, v],
           subclass[image[t, u], image[t, v]], {t → cross[RATIO, RATIO],
            u -> cart[cart[intersection[Z, complement[set[id[omega]]]], Z],
              cart[intersection[Z, complement[set[id[omega]]]], Z]], v -> composite[
              inverse[RATIO], fix[composite[inverse[RATMUL], RATMUL, SWAP]], RATIO]}] // Reverse

Out[20]= subclass[cart[RATS, RATS], fix[composite[inverse[RATMUL], RATMUL, SWAP]]] == True

In[21]:= % /. Equal → SetDelayed
```

Lemma. An inclusion of functions.

```
In[22]:= SubstTest[subclass, domain[funpart[u]],
          fix[composite[inverse[funpart[u]], funpart[v]]], {u → RATMUL, v → flip[RATMUL]}]

Out[22]= subclass[RATMUL, composite[RATMUL, SWAP]] == True

In[23]:= % /. Equal → SetDelayed
```

Inclusions between functions can always be rewritten as equations.

Theorem. A variable-free statement that rational multiplication is commutative.

```
In[24]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u → RATMUL, v → flip[RATMUL]}] // Reverse

Out[24]= equal[RATMUL, composite[RATMUL, SWAP]] == True

In[25]:= composite[RATMUL, SWAP] := RATMUL
```

Corollary. (Reintroducing variables.)

```
In[26]:= Map[equal[ratmul[x, y], #] &, ApComp[RATMUL, SWAP, PAIR[x, y]]]

Out[26]= equal[ratmul[x, y], ratmul[y, x]] == True

In[27]:= equal[ratmul[x_, y_], ratmul[y_, x_]] := True
```

---

the neutral element for rational multiplication

It has been shown earlier that **id[Z]** is right-neutral for rational multiplication. A variable-free statement of this will now be derived.

Lemma. Eliminating a variable.

```
In[28]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[implies[member[x, RATS], equal[APPLY[funpart[t], x], x]]],
           t → composite[RATMUL, RIGHT[id[Z]]]]]

Out[28]= subclass[RATS,
           image[inverse[fix[composite[inverse[FIRST], RATMUL]]], set[id[Z]]]] == True
```

```
In[29]:= % /. Equal → SetDelayed
```

Theorem. A better formulation.

```
In[30]:= SubstTest[subclass, domain[funpart[t]],
          fix[funpart[t]], t → composite[RATMUL, RIGHT[id[Z]]]]

Out[30]= subclass[composite[RATMUL, RIGHT[id[Z]]], Id] == True

In[31]:= % /. Equal → SetDelayed
```

A still better rule can be obtained by replacing the above inclusion by an equation.

Corollary. Right-neutrality of **id[Z]**.

```
In[32]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> composite[RATMUL, RIGHT[id[Z]]], v -> Id}] // Reverse

Out[32]= equal[composite[RATMUL, RIGHT[id[Z]]], id[RATS]] == True

In[33]:= composite[RATMUL, RIGHT[id[Z]]] := id[RATS]
```

Since **RATMUL** is commutative, **id[Z]** is also left-neutral.

Corollary.

```
In[34]:= Assoc[RATMUL, SWAP, RIGHT[id[Z]]]

Out[34]= composite[RATMUL, LEFT[id[Z]]] == id[RATS]

In[35]:= composite[RATMUL, LEFT[id[Z]]] := id[RATS]
```

Theorem. The rational number **id[Z]** is a neutral element for rational multiplication.

```
In[36]:= member[id[Z], ids[RATMUL]] // AssertTest

Out[36]= member[id[Z], ids[RATMUL]] == True

In[37]:= % /. Equal → SetDelayed
```

Since **RATMUL** is a binary operation, **id[Z]** is the only neutral element.

Theorem.

```
In[38]:= SubstTest[implies, and[member[x, BINOPS], member[y, ids[x]]],
           equal[ids[x], set[y]], {x → RATMUL, y → id[Z]}] // Reverse

Out[38]= equal[ids[RATMUL], set[id[Z]]] == True

In[39]:= ids[RATMUL] := set[id[Z]]
```

Corollary. The neutral element for rational multiplication is **id[Z]**.

```
In[40]:= SubstTest[A, ids[x], x → RATMUL]

Out[40]= e[RATMUL] == id[Z]

In[41]:= e[RATMUL] := id[Z]
```