addition of rational numbers

Johan G. F. Belinfante
2012 September 5

```
In[1]:= SetDirectory["l:"]; << goedel.12sep03a
```

```
:Package Title: goedel.12sep03a              2012 September 3 at 4:55 p.m.

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

It is now:  2012 Sep 5 at 10:46

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 5 at 11:1
```

summary

Addition of rational numbers is defined in this notebook. It is shown that rational addition is a commutative binary operation with $\mathbf{Z} \times \{\mathbf{id}[\omega]\}$ as neutral element.

funadd

In the **GOEDEL** program rational numbers are functions whose graphs are maximal straight lines through the origin in the integer plane $\mathbf{Z} \times \mathbf{Z}$. If **f** and **g** are integer-valued functions, then their sum $\mathbf{h} = \mathbf{f} + \mathbf{g}$ is defined by $\mathbf{h(x)} = \mathbf{f(x)} + \mathbf{g(x)}$ for all **x** where this makes sense. In the notebook **funadd.nb** dated 2003 August 19, just over 9 years ago, the formula for **h** was found to be $\mathbf{h} = \mathbf{INTADD} \circ (\mathbf{f} \otimes \mathbf{g}) \circ \mathbf{DUP}$. For convenience, the following temporary abbreviation is used to save some writing.

```
In[2]:= funadd[x_, y_] := composite[INTADD,
          intersection[composite[inverse[FIRST], x], composite[inverse[SECOND], y]]]
```

Addition of rational numbers is not simply the restriction of **funadd** to rational numbers. If **x** and **y** are rational numbers, **funadd[x, y]** is a function whose graph is a straight line through the origin, but it may fail to be maximal. One needs to take the rational hull of **funadd[x, y]** to obtain a rational number.

definition of RATADD

Rational addition is here defined directly by a variable-free equation:

```
In[3]:= composite[HULL[RATS], IMAGE[cross[inverse[DUP], INTADD]],
          CROSS, id[cart[RATS, RATS]]] := RATADD
```

Theorem. Rational addition is a function.

```
In[4]:= SubstTest[FUNCTION, composite[HULL[RATS],
           IMAGE[cross[inverse[DUP], INTADD]], CROSS, id[t]], t -> cart[RATS, RATS]] // Reverse

Out[4]= FUNCTION[RATADD] == True

In[5]:= FUNCTION[RATADD] := True
```

Theorem. Commutative law for rational addition.

```
In[6]:= Assoc[composite[HULL[RATS], IMAGE[cross[inverse[DUP], INTADD]]],
          composite[CROSS, id[cart[RATS, RATS]]], SWAP] // Reverse

Out[6]= composite[RATADD, SWAP] == RATADD

In[7]:= composite[RATADD, SWAP] := RATADD
```

Corollary. Simplification rule.

```
In[8]:= Assoc[RATADD, SWAP, SWAP]

Out[8]= composite[RATADD, id[cart[V, V]]] == RATADD

In[9]:= composite[RATADD, id[cart[V, V]]] := RATADD
```

The main task addressed in this notebook is to show that rational addition is a binary operation. In other words, any pair of rational numbers can be added, and the result is also a rational number. It will be shown that the domain of **RATADD** is **RATS** × **RATS** and that the range is **RATS**. An inclusion for the domain of **RATADD** follows immediately from the definition.

Lemma. An inclusion for the domain.

```
In[10]:= Map[subclass[#, cart[RATS, RATS]] &,
           IminComp[composite[HULL[RATS], IMAGE[cross[inverse[DUP], INTADD]], CROSS],
            id[cart[RATS, RATS]], V]]

Out[10]= subclass[domain[RATADD], cart[RATS, RATS]] == True

In[11]:= % /. Equal→ SetDelayed
```

Corollary.

```
In[12]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → domain[RATADD], v → cart[RATS, RATS], w → cart[V, V]}] // Reverse

Out[12]= subclass[domain[RATADD], cart[V, V]] == True

In[13]:= % /. Equal→ SetDelayed
```

definition of ratadd[x, y]

The **GOEDEL** program has two different ordered pairs, which agree for sets. One of these is the undefined primitive constructor **pair[x, y]**, and the other is the defined constructor **PAIR[x, y] = A[{x} × {y}]**. The class **ratadd[x, y]** is defined to be the result of applying the function **RATADD** to the ordered pair **PAIR[x, y]**.

```
In[14]:= APPLY[RATADD, PAIR[x_, y_]] := ratadd[x, y]
```

Theorem. One gets the same result no matter which ordered pair is used.

```
In[15]:= ApComp[RATADD, id[cart[V, V]], pair[x, y]] // Reverse

Out[15]= APPLY[RATADD, pair[x, y]] == ratadd[x, y]

In[16]:= APPLY[RATADD, pair[x_, y_]] := ratadd[x, y]
```

It is sometimes desirable to use a single variable to denote an ordered pair. There is also an **APPLY** rule for this.

Theorem.

```
In[17]:= ApComp[RATADD, id[cart[V, V]], x] // Reverse

Out[17]= APPLY[RATADD, x] == ratadd[first[x], second[x]]

In[18]:= APPLY[RATADD, x_] := ratadd[first[x], second[x]]
```

An explicit formula for **ratadd[x, y]** can be derived from the definition of **RATADD**. If

Theorem. Simplification rule.

```
In[19]:= ApComp[HULL[RATS], composite[
           IMAGE[cross[inverse[DUP], INTADD]], CROSS, id[cart[RATS, RATS]]], PAIR[x, y]]

Out[19]= hull[RATS, A[image[IMAGE[cross[inverse[DUP], INTADD]], image[CROSS,
             cart[intersection[RATS, set[x]], intersection[RATS, set[y]]]]]]] == ratadd[x, y]

In[20]:= hull[RATS, A[image[IMAGE[cross[inverse[DUP], INTADD]], image[CROSS,
             cart[intersection[RATS, set[x_]], intersection[RATS, set[y_]]]]]]] := ratadd[x, y]
```

Corollary. A rewrite rule.

```
In[21]:= SubstTest[hull, RATS, A[image[IMAGE[cross[inverse[DUP], INTADD]],
             image[CROSS, cart[intersection[RATS, set[u]], intersection[RATS, set[v]]]]]],
           {u → rat[x], v → rat[y]}] // Reverse

Out[21]= hull[RATS, composite[INTADD, intersection[composite[inverse[FIRST], rat[x]],
             composite[inverse[SECOND], rat[y]]]]] == ratadd[rat[x], rat[y]]

In[22]:= hull[RATS, composite[INTADD, intersection[composite[inverse[FIRST], rat[x_]],
             composite[inverse[SECOND], rat[y_]]]]] := ratadd[rat[x], rat[y]]
```

fractions

Every rational number can be written (in infinitely many ways) as a fraction **d \ n** with denominator **d** and numerator **n**. For convenience the following temporary abbreviation will be used for the fraction with denominator **x** and numerator **y**.

```
In[23]:= frac[x_, y_] := composite[inverse[inttimes[x]], inttimes[y]]
```

adding zero

The **rational number zero** is the fraction **1 \ 0**. This is the constant function that assigns the integer zero to any integer. (The integer zero is **id[ω]**.)

```
In[24]:= frac[plus[set[0]], plus[0]]

Out[24]= cart[Z, set[id[omega]]]
```

Theorem. Zero is a neutral element for addition.

```
In[25]:= Map[APPLY[#, rat[x]] &,
           Assoc[composite[HULL[RATS], IMAGE[cross[inverse[DUP], INTADD]]],
            composite[CROSS, id[cart[RATS, RATS]]], RIGHT[inttimes[plus[0]]]]] // Reverse

Out[25]= ratadd[rat[x], cart[Z, set[id[omega]]]] == rat[x]

In[26]:= ratadd[rat[x_], cart[Z, set[id[omega]]]] := rat[x]
```

The **rational number one** is the fraction **1 \ 1**.

```
In[27]:= frac[plus[set[0]], plus[set[0]]]

Out[27]= id[Z]
```

Corollary. A special case: 1 + 0 = 1. Rational addition of the rational number one and the rational number zero yields the rational number one.

```
In[28]:= SubstTest[ratadd, rat[x], cart[Z, set[id[omega]]], x → id[Z]] // Reverse

Out[28]= ratadd[id[Z], cart[Z, set[id[omega]]]] == id[Z]
```

```
In[29]:= ratadd[id[Z], cart[Z, set[id[omega]]]] := id[Z]
```

Lemma. One can add any rational number and the rational number zero.

```
In[30]:= SubstTest[member, APPLY[funpart[t], w],
           V, {t → RATADD, w→ PAIR[rat[x], inttimes[plus[0]]]}]

Out[30]= member[pair[rat[x], cart[Z, set[id[omega]]]], domain[RATADD]] == True

In[31]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma. Temporary rewrite rule. Any rational number belongs to the range of **RATADD**.

```
In[32]:= SubstTest[member, APPLY[funpart[t], w], range[funpart[t]],
           {t → RATADD, w→ PAIR[rat[x], inttimes[plus[0]]]}] // Reverse

Out[32]= member[rat[x], range[RATADD]] == True

In[33]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. A lower bound for the range of **RATADD**.

```
In[34]:= Map[or[subclass[RATS, range[RATADD]], equal[V, domain[#]]] &,
           SubstTest[reify, x, case[member[rat[x], t]], t -> range[RATADD]]]

Out[34]= subclass[RATS, range[RATADD]] == True

In[35]:= % /. Equal→ SetDelayed
```

computing rational hulls

To obtain an explicit formula for rational addition of fractions, one needs to compute a rational hull. In this section some tools are derived to help with this.

Theorem. If **funadd[rat[x], rat[y]] ⊂ rat[z]**, then **ratadd[rat[x], rat[y]] = rat[z]**.

```
In[36]:= SubstTest[implies, and[subclass[t, rat[z]], not[subclass[domain[t], set[id[omega]]]]],
           equal[hull[RATS, t], rat[z]], t → funadd[rat[x], rat[y]]] // Reverse

Out[36]= or[equal[rat[z], ratadd[rat[x], rat[y]]],
           not[subclass[composite[INTADD, intersection[composite[inverse[FIRST], rat[x]],
               composite[inverse[SECOND], rat[y]]], rat[z]]]] == True

In[37]:= (% /. {x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

Corollary. (Eliminating the **rat** wrappers.) If **x**, **y** and **z** are rational numbers, and if **funadd[x, y] ⊂ z**, then **ratadd[x, y] = z**.

```
In[38]:= SubstTest[implies, and[equal[x, rat[u]], equal[y, rat[v]], equal[z, rat[w]],
           subclass[funadd[x, y], z]], equal[ratadd[x, y], z], {u → x, v → y, w → z}] // Reverse

Out[38]= or[equal[z, ratadd[x, y]],
           not[member[x, RATS]], not[member[y, RATS]], not[member[z, RATS]],
           not[subclass[composite[INTADD, intersection[composite[inverse[FIRST], x],
               composite[inverse[SECOND], y]]], z]]] == True

In[39]:= or[equal[z_, ratadd[x_, y_]], not[member[x_, RATS]],
           not[member[y_, RATS]], not[member[z_, RATS]],
           not[subclass[composite[INTADD, intersection[composite[inverse[FIRST], x_],
               composite[inverse[SECOND], y_]]], z_]]] := True
```

Theorem. An inclusion.

```
In[40]:= SubstTest[implies, equal[s, id[Z]], subclass[composite[id[Z], t], composite[s, t]],
           {s → composite[inverse[inttimes[w]], inttimes[w]], t ->
             composite[INTADD, cross[inverse[inttimes[w]], inverse[inttimes[w]]]]}] // Reverse

Out[40]= or[equal[w, id[omega]], not[member[w, Z]],
           subclass[composite[INTADD, cross[inverse[inttimes[w]], inverse[inttimes[w]]]],
            composite[inverse[inttimes[w]], INTADD]]] == True

In[41]:= (% /. w → w_) /. Equal→ SetDelayed
```

Theorem. If $\mathbf{INTADD} \circ (\mathbf{t} \otimes \mathbf{t}) \subset \mathbf{t} \circ \mathbf{INTADD}$, then $\mathbf{funadd}[\mathbf{t} \circ \mathbf{x}, \mathbf{t} \circ \mathbf{y}] \subset \mathbf{t} \circ \mathbf{funadd}[\mathbf{x}, \mathbf{y}]$.

```
In[42]:= SubstTest[implies, subclass[u, v],
           subclass[composite[u, w], composite[v, w]], {u -> composite[INTADD, cross[t, t]],
            v → composite[t, INTADD], w → composite[cross[x, y], DUP]}] // Reverse

Out[42]= or[not[subclass[composite[INTADD, cross[t, t]], composite[t, INTADD]]],
           subclass[composite[INTADD, intersection[composite[inverse[FIRST], t, x],
              composite[inverse[SECOND], t, y]]], composite[t, INTADD,
             intersection[composite[inverse[FIRST], x], composite[inverse[SECOND], y]]]]] == True

In[43]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Restatement. (An inclusion for funadd.)

```
In[44]:= implies[subclass[composite[INTADD, cross[t, t]], composite[t, INTADD]],
          subclass[funadd[composite[t, x], composite[t, y]], composite[t, funadd[x, y]]]]

Out[44]= True
```

Lemma.

```
In[45]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[w, dif[Z, set[id[omega]]]], p2 ->
             subclass[composite[INTADD, cross[inverse[inttimes[w]], inverse[inttimes[w]]]],
              composite[inverse[inttimes[w]], INTADD], p3 -> subclass[
              funadd[composite[inverse[inttimes[w]], x], composite[inverse[inttimes[w]], y]],
              composite[inverse[inttimes[w]], funadd[x, y]]]}]] // Reverse

Out[45]= or[equal[w, id[omega]], not[member[w, Z]], subclass[
           composite[INTADD, intersection[composite[inverse[FIRST], inverse[inttimes[w]], x],
             composite[inverse[SECOND], inverse[inttimes[w]], y]]],
           composite[inverse[inttimes[w]], INTADD,
            intersection[composite[inverse[FIRST], x], composite[inverse[SECOND], y]]]]] == True

In[46]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. If **w** is a nonzero integer, then **funadd[w \ x, w \ y] ⊂ w \ intadd[x, y]**.

```
In[47]:= SubstTest[implies, member[w, dif[Z, set[id[omega]]]], subclass[
           funadd[composite[inverse[inttimes[w]], u], composite[inverse[inttimes[w]], v]],
           composite[inverse[inttimes[w]], funadd[u, v]]],
          {u → inttimes[x], v → inttimes[y]}] // Reverse

Out[47]= or[equal[w, id[omega]], not[member[w, Z]], subclass[composite[INTADD,
            intersection[composite[inverse[FIRST], inverse[inttimes[w]], inttimes[x]],
             composite[inverse[SECOND], inverse[inttimes[w]], inttimes[y]]]],
           composite[inverse[inttimes[w]], inttimes[intadd[x, y]]]]] == True

In[48]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Restatement. If **w** is a nonzero integer, then **funadd[w \ x, w \ y] ⊂ w \ intadd[x, y]**.

```
In[49]:= implies[member[w, dif[Z, set[id[omega]]]],
          subclass[funadd[frac[w, x], frac[w, y]], frac[w, intadd[x, y]]]]

Out[49]= True
```

adding fractions with a common denominator

Lemma.

```
In[50]:= SubstTest[implies, and[member[u, RATS], member[v, RATS],
           member[t, RATS], subclass[funadd[u, v], t]], equal[ratadd[u, v], t],
          {u → frac[w, x], v → frac[w, y], t -> frac[w, intadd[x, y]]}] // Reverse

Out[50]= or[equal[w, id[omega]], equal[composite[inverse[inttimes[w]], inttimes[intadd[x, y]]],
           ratadd[composite[inverse[inttimes[w]], inttimes[x]],
            composite[inverse[inttimes[w]], inttimes[y]]]], not[member[w, Z]],
          not[member[x, Z]], not[member[y, Z]], not[subclass[composite[INTADD,
             intersection[composite[inverse[FIRST], inverse[inttimes[w]], inttimes[x]],
              composite[inverse[SECOND], inverse[inttimes[w]], inttimes[y]]]],
            composite[inverse[inttimes[w]], inttimes[intadd[x, y]]]]]] == True
```

```
In[51]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Rational addition for fractions with a common denominator.) If **w**, **x** and **y** are integers and **w** is non-zero, then **ratadd[w \ x, w \ y] = w \ intadd[x, y]**.

```
In[52]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 → and[member[w, dif[Z, set[id[omega]]]], member[x, Z], member[y, Z]],
            p2 -> subclass[funadd[frac[w, x], frac[w, y]], frac[w, intadd[x, y]]],
            p3 → equal[ratadd[frac[w, x], frac[w, y]], frac[w, intadd[x, y]]]}]] // Reverse

Out[52]= or[equal[w, id[omega]], equal[composite[inverse[inttimes[w]], inttimes[intadd[x, y]]],
           ratadd[composite[inverse[inttimes[w]], inttimes[x]],
            composite[inverse[inttimes[w]], inttimes[y]]]],
          not[member[w, Z]], not[member[x, Z]], not[member[y, Z]]] == True

In[53]:= or[equal[composite[inverse[inttimes[w_]], inttimes[intadd[x_, y_]]],
           ratadd[composite[inverse[inttimes[w_]], inttimes[x_]],
            composite[inverse[inttimes[w_]], inttimes[y_]]]], equal[id[omega], w_],
          not[member[w_, Z]], not[member[x_, Z]], not[member[y_, Z]]] := True
```

The problem of eliminating the variables will not be addressed here. Instead, the above result will just be used to show that rational addition is a binary operation. For that only the following corollary is needed.

Corollary. If **w**, **x** and **y** are integers, and **w** is not zero, then **ratadd[w \ x, w \ y]** is a rational number.

```
In[54]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 -> and[member[w, dif[Z, set[id[omega]]]], member[x, Z], member[y, Z]],
            p2 → equal[ratadd[frac[w, x], frac[w, y]], frac[w, intadd[x, y]]],
            p3 → member[frac[w, intadd[x, y]], RATS],
            p4 → member[ratadd[frac[w, x], frac[w, y]], RATS]}]] // Reverse

Out[54]= or[equal[w, id[omega]], member[ratadd[composite[inverse[inttimes[w]], inttimes[x]],
            composite[inverse[inttimes[w]], inttimes[y]]], RATS],
          not[member[w, Z]], not[member[x, Z]], not[member[y, Z]]] == True

In[55]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The idea now is to replace fractions with **rat** wrappers. The first attempts to do so took a long time, but reasonable execution times were achieved by breaking up the proof with two lemmas, and commenting out several proof steps. The rewrite rules in the **GOEDEL** program supply these omitted steps.

Lemma 1.

```
In[56]:= Map[not, SubstTest[and, (*implies[and[p0,p1],p2],implies[and[p0,p1],p5],
             implies[and[p0,p1],p6],*) implies[and[p0, p2, p5, p6], p7],
            not[implies[and[p0, p1], p7]], {p0 -> not[equal[w, id[omega]]],
             p1 → and[member[w, domain[rat[x]]], member[w, domain[rat[y]]]], p2 → member[w, Z],
             p5 → member[APPLY[rat[x], w], Z], p6 → member[APPLY[rat[y], w], Z], p7 → member[
               ratadd[frac[w, APPLY[rat[x], w]], frac[w, APPLY[rat[y], w]]], RATS]}]] // Reverse
```

```
Out[56]= or[equal[w, id[omega]],
           member[ratadd[composite[inverse[inttimes[w]], inttimes[APPLY[rat[x], w]]],
             composite[inverse[inttimes[w]], inttimes[APPLY[rat[y], w]]]], RATS],
           not[member[w, domain[rat[x]]]], not[member[w, domain[rat[y]]]]] == True
```

```
In[57]:= (% /. {w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

The following takes about 8 seconds.

Lemma 2.

```
In[58]:= Map[not, SubstTest[and, (* implies[and[p0,p1],p3],implies[and[p0,p1],p4],*)
            implies[and[p0, p1], p7], implies[and[p3, p4, p7], p8],
            not[implies[and[p0, p1], p8]], {p0 -> not[equal[w, id[omega]]],
             p1 → and[member[w, domain[rat[x]]], member[w, domain[rat[y]]]],
             p3 → equal[frac[w, APPLY[rat[x], w]], rat[x]],
             p4 → equal[frac[w, APPLY[rat[y], w]], rat[y]],
             p7 → member[ratadd[frac[w, APPLY[rat[x], w]], frac[w, APPLY[rat[y], w]]], RATS],
             p8 → member[ratadd[rat[x], rat[y]], RATS]}]] // Reverse
```

```
Out[58]= or[equal[w, id[omega]], member[ratadd[rat[x], rat[y]], RATS],
           not[member[w, domain[rat[x]]]], not[member[w, domain[rat[y]]]]] == True
```

```
In[59]:= or[equal[w_, id[omega]], member[ratadd[rat[x_], rat[y_]], RATS],
           not[member[w_, domain[rat[x_]]]], not[member[w_, domain[rat[y_]]]]] := True
```

Theorem. (Eliminate **w**.) The sum of two rational numbers is a rational number.

```
In[60]:= Map[equal[V, domain[#]] &, SubstTest[reify, w,
           case[or[equal[w, id[omega]], member[t, RATS], not[member[w, domain[rat[x]]]],
             not[member[w, domain[rat[y]]]]]], t -> ratadd[rat[x], rat[y]]]]
```

```
Out[60]= member[ratadd[rat[x], rat[y]], RATS] == True
```

```
In[61]:= member[ratadd[rat[x_], rat[y_]], RATS] := True
```

Lemma. Eliminate the **rat** wrapper.

```
In[62]:= SubstTest[implies, and[equal[x, rat[u]], equal[y, rat[v]]],
           member[ratadd[x, y], RATS], {u → x, v → y}] // Reverse
```

```
Out[62]= or[member[ratadd[x, y], RATS], not[member[x, RATS]], not[member[y, RATS]]] == True
```

```
In[63]:= or[member[ratadd[x_, y_], RATS], not[member[x_, RATS]], not[member[y_, RATS]]] := True
```

Lemma. (Eliminating the variables.)

```
In[64]:= Map[equal[V, domain[#]] &, SubstTest[reify, x, case[
           implies[member[x, cartsq[RATS]], member[APPLY[funpart[t], x], RATS]]], t → RATADD]]

Out[64]= subclass[cart[RATS, RATS], image[inverse[RATADD], RATS]] == True

In[65]:= % /. Equal→ SetDelayed
```

Theorem. An equation for the domain of **RATADD**.

```
In[66]:= SubstTest[implies, and[subclass[u, v], subclass[v, w], subclass[w, u]],
           equal[u, w], {u -> cart[RATS, RATS],
            v -> image[inverse[RATADD], RATS], w → domain[RATADD]}] // Reverse

Out[66]= equal[cart[RATS, RATS], domain[RATADD]] == True

In[67]:= domain[RATADD] := cart[RATS, RATS]
```

Lemma. An inclusion for the range of **RATADD**.

```
In[68]:= Map[subclass[#, RATS] &, ImageComp[RATADD, inverse[RATADD], RATS]] // Reverse

Out[68]= subclass[range[RATADD], RATS] == True

In[69]:= % /. Equal→ SetDelayed
```

Theorem. An equation for the range of **RATADD**.

```
In[70]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> range[RATADD], v -> RATS}]

Out[70]= equal[RATS, range[RATADD]] == True

In[71]:= range[RATADD] := RATS
```

Theorem. The function **RATADD** is a set.

```
In[72]:= SubstTest[member, domain[funpart[t]], V, t → RATADD]

Out[72]= member[RATADD, V] == True

In[73]:= member[RATADD, V] := True
```

Theorem. A mapping statement.

```
In[74]:= member[RATADD, map[cart[RATS, RATS], RATS]] // AssertTest

Out[74]= member[RATADD, map[cart[RATS, RATS], RATS]] == True

In[75]:= member[RATADD, map[cart[RATS, RATS], RATS]] := True
```

Corollary. Rational addition is a binary operation.

```
In[76]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
           {x → RATADD, y -> map[cart[RATS, RATS], RATS], z → BINOPS}] // Reverse

Out[76]= member[RATADD, BINOPS] == True

In[77]:= member[RATADD, BINOPS] := True
```

Corollary. An upper bound for **RATADD**.

```
In[78]:= SubstTest[subclass, funpart[t], cart[x, y], t → RATADD] // Reverse

Out[78]= subclass[RATADD, cart[x, y]] == and[subclass[RATS, y], subclass[cart[RATS, RATS], x]]

In[79]:= subclass[RATADD, cart[x_, y_]] := and[subclass[RATS, y], subclass[cart[RATS, RATS], x]]
```

INTTIMES embeds Z into RATS

The orientation of the following rewrite rule was chosen to avoid introducing rational arithmetic into an expression involving only integer arithmetic.

Theorem. A connection between integer addition and rational addition.

```
In[80]:= Assoc[HULL[RATS], composite[IMAGE[cross[inverse[DUP], INTADD]],
           CROSS, id[cart[RATS, RATS]]], cross[INTTIMES, INTTIMES]] // Reverse

Out[80]= composite[RATADD, cross[INTTIMES, INTTIMES]] == composite[INTTIMES, INTADD]

In[81]:= composite[RATADD, cross[INTTIMES, INTTIMES]] := composite[INTTIMES, INTADD]
```

Theorem.

```
In[82]:= (member[t, binhom[x, y]] // AssertTest) /. {t → INTTIMES, x→ INTADD, y→ RATADD}

Out[82]= member[INTTIMES, binhom[INTADD, RATADD]] == True

In[83]:= member[INTTIMES, binhom[INTADD, RATADD]] := True
```

zero is a neutral element

Theorem.

```
In[84]:= Map[composite[VERTSECT[#], id[RATS]] &, SubstTest[reify, x,
           APPLY[funpart[t], PAIR[rat[x], cart[Z, set[id[omega]]]]], t → RATADD]]

Out[84]= composite[RATADD, RIGHT[cart[Z, set[id[omega]]]]] == id[RATS]

In[85]:= composite[RATADD, RIGHT[cart[Z, set[id[omega]]]]] := id[RATS]
```

Theorem.

```
In[86]:= Assoc[RATADD, SWAP, RIGHT[cart[Z, set[id[omega]]]]]

Out[86]= composite[RATADD, LEFT[cart[Z, set[id[omega]]]]] == id[RATS]

In[87]:= composite[RATADD, LEFT[cart[Z, set[id[omega]]]]] := id[RATS]
```

Lemma.

```
In[88]:= member[inttimes[plus[0]], ids[RATADD]] // AssertTest

Out[88]= member[cart[Z, set[id[omega]]], ids[RATADD]] == True

In[89]:= member[cart[Z, set[id[omega]]], ids[RATADD]] := True
```

Theorem.

```
In[90]:= SubstTest[implies, and[member[t, BINOPS], member[u, ids[t]]],
           equal[ids[t], set[u]], {u -> inttimes[plus[0]], t → RATADD}] // Reverse

Out[90]= equal[ids[RATADD], set[cart[Z, set[id[omega]]]]] == True

In[91]:= ids[RATADD] := set[cart[Z, set[id[omega]]]]
```

Corollary.

```
In[92]:= SubstTest[A, ids[t], t → RATADD]

Out[92]= e[RATADD] == cart[Z, set[id[omega]]]

In[93]:= e[RATADD] := cart[Z, set[id[omega]]]
```

Theorem. The function **INTTIMES** is a functor from **INTADD** to **RATADD**.

```
In[102]:=
        functor[INTTIMES, INTADD, RATADD] // AssertTest

Out[102]=
        functor[INTTIMES, INTADD, RATADD] == True

In[103]:=
        functor[INTTIMES, INTADD, RATADD] := True
```