membership rules for RATIO and RATS

Johan G. F. Belinfante
2012 July 2

```
In[1]:= SetDirectory["l:"]; << goedel.12jun30a

        :Package Title: goedel.12jun30a                    2012 June 30 at 5:40 p.m.

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

        It is now:  2012 Jul 2 at 11:31

        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 Jul 2 at 11:48
```

summary

Membership rules for the function **RATIO** and the class **RATS** are derived. Rational numbers are not empty. Simplification rules involving **id[Z]** are also derived.

simplification rules

Lemma. A simplification rule.

```
In[2]:= IminComp[RATIO, id[cart[V, V]], x] // Reverse

Out[2]= composite[Id, image[inverse[RATIO], x]] == image[inverse[RATIO], x]

In[3]:= composite[Id, image[inverse[RATIO], x_]] := image[inverse[RATIO], x]
```

Theorem. Simplification rule.

```
In[5]:= Assoc[INTTIMES, id[Z], id[x]]

Out[5]= composite[INTTIMES, id[intersection[x, Z]]] == composite[INTTIMES, id[x]]

In[6]:= composite[INTTIMES, id[intersection[x_, Z]]] := composite[INTTIMES, id[x]]
```

Corollary.

```
In[7]:= composite[id[intersection[x, Z]], inverse[INTTIMES]] // DoubleInverse

Out[7]= composite[id[intersection[x, Z]], inverse[INTTIMES]] ==
         composite[id[x], inverse[INTTIMES]]

In[8]:= composite[id[intersection[x_, Z]], inverse[INTTIMES]] :=
         composite[id[x], inverse[INTTIMES]]
```

the condition that an integer be zero

In this section, rewrite rules are derived for the condition that an integer be zero (or not zero).

Lemma.

```
In[9]:= SubstTest[implies, equal[x, int[t]],
          or[equal[x, id[omega]], not[subclass[omega, fix[x]]]], t → x] // Reverse

Out[9]= or[equal[x, id[omega]], not[member[x, Z]], not[subclass[omega, fix[x]]]] == True

In[10]:= or[equal[x_, id[omega]], not[member[x_, Z]], not[subclass[omega, fix[x_]]]] := True
```

Theorem.

```
In[11]:= equiv[and[member[x, Z], subclass[omega, fix[x]]], equal[x, id[omega]]]

Out[11]= True

In[12]:= and[member[x_, Z], subclass[omega, fix[x_]]] := equal[x, id[omega]]
```

Corollary.

```
In[13]:= SubstTest[and, member[t, Z], subclass[omega, fix[t]], t → int[x]] // Reverse

Out[13]= subclass[omega, fix[int[x]]] == equal[id[omega], int[x]]

In[14]:= subclass[omega, fix[int[x_]]] := equal[id[omega], int[x]]
```

Theorem.

```
In[15]:= equiv[and[member[x, Z], not[subclass[omega, fix[x]]]],
          and[member[x, Z], not[equal[x, id[omega]]]]]

Out[15]= True

In[16]:= and[member[x_, Z], not[subclass[omega, fix[x_]]]] :=
          and[member[x, Z], not[equal[x, id[omega]]]]
```

RATIO rules

The simplification rules derived in a preceding section requires an existing rule for **RATIO** to be replaced.

Lemma. An inclusion.

```
In[17]:= SubstTest[subclass, composite[id[t], x], x, {t → FUNS,
           x -> composite[COMPOSE, cross[composite[INVERSE, INTTIMES], INTTIMES]]}] // Reverse

Out[17]= subclass[RATIO,
           composite[COMPOSE, cross[composite[INVERSE, INTTIMES], INTTIMES]]] == True

In[18]:= subclass[RATIO,
           composite[COMPOSE, cross[composite[INVERSE, INTTIMES], INTTIMES]]] := True
```

Theorem. Replacement rule.

```
In[19]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u → RATIO,
            v -> composite[COMPOSE, cross[composite[INVERSE, INTTIMES], INTTIMES]]}] // Reverse

Out[19]= equal[RATIO, composite[COMPOSE, cross[
             composite[INVERSE, INTTIMES, id[complement[set[id[omega]]]]], INTTIMES]]] == True

In[20]:= composite[COMPOSE, cross[
            composite[INVERSE, INTTIMES, id[complement[set[id[omega]]]]], INTTIMES]] := RATIO
```

a membership rule for RATIO

A membership rule for **RATIO** is derived in this section.

Lemma.

```
In[21]:= Map[implies[#, member[x, Z]] &,
           SubstTest[member, pair[pair[x, y], z], composite[COMPOSE, cross[composite[INVERSE,
               INTTIMES, id[complement[set[t]]]], INTTIMES]], t → id[omega]]] // Reverse

Out[21]= or[member[x, Z], not[member[pair[pair[x, y], z], RATIO]]] == True

In[22]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[23]:= Map[implies[#, not[equal[x, id[omega]]]] &,
           SubstTest[member, pair[pair[x, y], z], composite[COMPOSE, cross[composite[INVERSE,
               INTTIMES, id[complement[set[t]]]], INTTIMES]], t → id[omega]]] // Reverse

Out[23]= or[not[equal[x, id[omega]]], not[member[pair[pair[x, y], z], RATIO]]] == True
```

```
In[24]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[25]:= Map[implies[#, member[y, Z]] &,
           SubstTest[member, pair[pair[x, y], z], composite[COMPOSE, cross[composite[INVERSE,
               INTTIMES, id[complement[set[t]]]], INTTIMES]], t → id[omega]]] // Reverse

Out[25]= or[member[y, Z], not[member[pair[pair[x, y], z], RATIO]]] == True
```

```
In[26]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[27]:= Map[implies[#, equal[z, composite[inverse[inttimes[int[x]]], inttimes[int[y]]]]] &,
           SubstTest[member, pair[pair[int[x], int[y]], z],
            composite[COMPOSE, cross[composite[INVERSE, INTTIMES, id[complement[set[t]]]],
              INTTIMES]], t → id[omega]]] // Reverse

Out[27]= or[equal[z, composite[inverse[inttimes[int[x]]], inttimes[int[y]]]],
           not[member[pair[pair[int[x], int[y]], z], RATIO]]] == True
```

```
In[28]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. (Eliminate the **int** wrappers.)

```
In[29]:= SubstTest[implies, and[equal[x, int[u]], equal[y, int[v]]],
           or[equal[z, composite[inverse[inttimes[x]], inttimes[y]]],
            not[member[pair[pair[x, y], z], RATIO]]], {u → x, v → y}] // Reverse

Out[29]= or[equal[z, composite[inverse[inttimes[x]], inttimes[y]]], not[member[x, Z]],
           not[member[y, Z]], not[member[pair[pair[x, y], z], RATIO]]] == True
```

```
In[30]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. (Eliminate redundant literals.)

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p1, p3], implies[and[p1, p2, p3], p4], not[implies[p1, p4]],
            {p1 -> member[pair[pair[x, y], z], RATIO], p2 → member[x, Z], p3 → member[y, Z],
             p4 -> equal[z, composite[inverse[inttimes[x]], inttimes[y]]]}]] // Reverse

Out[31]= or[equal[z, composite[inverse[inttimes[x]], inttimes[y]]],
           not[member[pair[pair[x, y], z], RATIO]]] == True
```

```
In[32]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Rewrite rules for the converse will be derived next.

Lemma.

```
In[33]:= Map[implies[#, member[pair[pair[x, y], z], RATIO]] &,
           SubstTest[member, pair[pair[x, y], z], composite[COMPOSE,
             cross[composite[INVERSE, INTTIMES, id[complement[set[t]]]], INTTIMES]],
            t → id[omega]]] /. {x → int[u], y → int[v],
           z -> composite[inverse[inttimes[int[u]]], inttimes[int[v]]]}

Out[33]= or[equal[id[omega], int[u]], member[pair[pair[int[u], int[v]],
             composite[inverse[inttimes[int[u]]], inttimes[int[v]]]], RATIO]] == True

In[34]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

Lemma. (Eliminate the **int** wrappers.)

```
In[35]:= SubstTest[implies, and[equal[x, int[u]], equal[y, int[v]]], or[equal[id[omega], x],
            member[pair[pair[x, y], composite[inverse[inttimes[x]], inttimes[y]]], RATIO]],
           {u → x, v → y}] // Reverse

Out[35]= or[equal[x, id[omega]],
           member[pair[pair[x, y], composite[inverse[inttimes[x]], inttimes[y]]], RATIO],
           not[member[x, Z]], not[member[y, Z]]] == True

In[36]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The main theorem of this secction combines all of the above lemmas.

Theorem. A membership rule for **RATIO**.

```
In[37]:= equiv[member[pair[pair[x, y], z], RATIO],
            and[equal[z, composite[inverse[inttimes[x]], inttimes[y]]],
             member[x, Z], member[y, Z], not[equal[x, id[omega]]]]] // not // not

Out[37]= True

In[38]:= member[pair[pair[x_, y_], z_], RATIO] :=
          and[equal[z, composite[inverse[inttimes[x]], inttimes[y]]],
           member[x, Z], member[y, Z], not[equal[x, id[omega]]]]
```

an APPLY rule for RATIO

Theorem. Temporary vertical section rule.

```
In[39]:= SubstTest[image, funpart[t], cart[set[x], set[y]], t → RATIO] // Reverse

Out[39]= image[RATIO, cart[set[x], set[y]]] == set[APPLY[RATIO, PAIR[x, y]]]

In[40]:= image[RATIO, cart[set[x_], set[y_]]] := set[APPLY[RATIO, PAIR[x, y]]]
```

Theorem. A temporary **APPLY** rule for **RATIO**.

```
In[41]:= SubstTest[member, composite[inverse[inttimes[x]], inttimes[y]],
           image[t, cart[set[x], set[y]]], t → RATIO] // Reverse

Out[41]= equal[APPLY[RATIO, PAIR[x, y]], composite[inverse[inttimes[x]], inttimes[y]]] ==
          and[member[x, Z], member[y, Z], not[equal[x, id[omega]]]]

In[42]:= equal[APPLY[RATIO, PAIR[x_, y_]], composite[inverse[inttimes[x_]], inttimes[y_]]] :=
          and[member[x, Z], member[y, Z], not[equal[x, id[omega]]]]
```

Theorem. An **APPLY** rule for **RATIO**.

```
In[43]:= equal[APPLY[RATIO, PAIR[x, y]],
            union[complement[image[V, intersection[omega, complement[fix[x]]]]], complement[
              image[V, intersection[Z, set[x]]]], complement[image[V, intersection[Z, set[y]]]],
             composite[inverse[inttimes[x]], inttimes[y]]] // not // not

Out[43]= True

In[44]:= APPLY[RATIO, PAIR[x_, y_]] :=
          union[complement[image[V, intersection[omega, complement[fix[x]]]]],
           complement[image[V, intersection[Z, set[x]]]],
           complement[image[V, intersection[Z, set[y]]]],
           composite[inverse[inttimes[x]], inttimes[y]]]
```

---

a membership rule for RATS

Theorem.

```
In[45]:= Map[not[empty[#]] &,
          SubstTest[class, pair[x, y], member[pair[pair[x, y], 0], t], t → RATIO]]

Out[45]= member[0, RATS] == False

In[46]:= member[0, RATS] := False
```

Lemma.

```
In[47]:= Map[or[member[x, Z], #] &, SubstTest[implies, and[member[u, v], subclass[v, w]],
            member[u, w], {u -> composite[inverse[inttimes[x]], inttimes[y]],
             v → RATS, w → complement[set[0]]}]] // Reverse

Out[47]= or[member[x, Z],
           not[member[composite[inverse[inttimes[x]], inttimes[y]], RATS]]] == True

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

Lemma.

```
In[49]:= SubstTest[implies, and[member[u, v], subclass[v, w]],
           member[u, w], {u -> composite[inverse[inttimes[x]], inttimes[y]],
            v → RATS, w → complement[set[0]]}] // Reverse // MapNotNot

Out[49]= or[member[y, Z],
          not[member[composite[inverse[inttimes[x]], inttimes[y]], RATS]]] == True

In[50]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[51]:= SubstTest[implies, and[member[u, v], subclass[v, w]],
           member[u, w], {u -> composite[inverse[inttimes[x]], inttimes[y]],
            v → RATS, w → FUNS}] // Reverse // MapNotNot

Out[51]= or[not[equal[x, id[omega]]], not[member[y, Z]],
          not[member[composite[inverse[inttimes[x]], inttimes[y]], RATS]]] == True

In[52]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Remove a redundant literal.)

```
In[53]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 -> member[composite[inverse[inttimes[x]], inttimes[y]], RATS] ,
            p2 -> member[y, Z], p3 -> not[equal[x, id[omega]]]}]] // Reverse

Out[53]= or[not[equal[x, id[omega]]],
          not[member[composite[inverse[inttimes[x]], inttimes[y]], RATS]]] == True

In[54]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. A membership rule for **RATS**.

```
In[55]:= equiv[member[composite[inverse[inttimes[x]], inttimes[y]], RATS],
           and[member[x, Z], member[y, Z], not[equal[x, id[omega]]]]] // not // not

Out[55]= True

In[56]:= member[composite[inverse[inttimes[x_]], inttimes[y_]], RATS] :=
          and[member[x, Z], member[y, Z], not[equal[x, id[omega]]]]
```

serendipity

Theorem. A simplification rule.

```
In[57]:= intersection[complement[image[V, x]],
           complement[image[V, intersection[x, y]]]] // DoubleComplement

Out[57]= intersection[complement[image[V, x]], complement[image[V, intersection[x, y]]]] ==
          complement[image[V, x]]
```

```
In[58]:= intersection[complement[image[V, x_]], complement[image[V, intersection[x_, y_]]]] :=
          complement[image[V, x]]
```