a transitivity property for integer equi-ratios

Johan G. F. Belinfante
2012 June 26

```
In[1]:= SetDirectory["l:"]; << goedel.12jun25a

:Package Title: goedel.12jun25a                    2012 June 25 at 1:45 a.m.

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

It is now:  2012 Jun 26 at 2:9

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 Jun 26 at 2:26
```

summary

Equality of fractions satisfies a transitive property: if **a/b = c/d** and **c/d = e/f**, then **a/b = e/f**. The two equations in the hypothesis imply that **ad = bc** and **cf = de**, from which one can deduce **adf = bcf = bde**. If **d** is nonzero, one can cancel out the middle factor **d** to obtain **af = be**. In this notebook a wrapper-free statement involving six integer variables is derived that amounts to this transitivity statement about equal integer ratios.

wrapped version

At first blush it would appear that the complete statement that needs to be derived must have an integer-hood hypothesis for each of the six variables in addition to the three equations about products and the condition that the middle variable be nonzero, for a total of ten literals. The **GOEDEL** program automatically scans any statement made to see whether any of the many available rewrite rules can be used to simplify it, and this would take a long time for a statement with ten literals. The number of literals can be reduced to four if each of the variables is wrapped with **int**. It is easy to derive a transitivity statement involving six variables if all of them are wrapped with **int**. This is done in the present section. One lemma is needed.

Lemma. A cancellation law for products of three integers.

```
In[2]:= SubstTest[equal, intmul[int[r], int[w]], intmul[int[s], int[w]],
          {r → intmul[int[u], int[x]], s -> intmul[int[v], int[y]]}] // Reverse

Out[2]= equal[intmul[int[u], int[w], int[x]], intmul[int[v], int[w], int[y]]] ==
         or[equal[id[omega], int[w]], equal[intmul[int[u], int[x]], intmul[int[v], int[y]]]]

In[3]:= equal[intmul[int[u_], int[w_], int[x_]], intmul[int[v_], int[w_], int[y_]]] :=
         or[equal[id[omega], int[w]], equal[intmul[int[u], int[x]], intmul[int[v], int[y]]]]
```

Theorem. A wrapped version of the transitive law for integer equi-ratios.

```
In[4]:= SubstTest[implies, and[equal[r, s], equal[s, t]], equal[r, t],
           {r → intmul[int[u], int[x], int[z]], s → intmul[int[v], int[w], int[z]],
            t → intmul[int[v], int[x], int[y]]}] // Reverse // MapNotNot

Out[4]= or[equal[id[omega], int[x]], equal[intmul[int[u], int[z]], intmul[int[v], int[y]]],
          not[equal[intmul[int[u], int[x]], intmul[int[v], int[w]]]],
          not[equal[intmul[int[w], int[z]], intmul[int[x], int[y]]]]] == True

In[5]:= or[equal[id[omega], int[x_]],
          equal[intmul[int[u_], int[z_]], intmul[int[v_], int[y_]]],
          not[equal[intmul[int[u_], int[x_]], intmul[int[v_], int[w_]]]],
          not[equal[intmul[int[w_], int[z_]], intmul[int[x_], int[y_]]]]] := True
```

Experiments using **TimeConstrained** were done to verify that the preceding theorem would take over a minute if all the **int** wrappers were to be omitted. However, one does need to somehow manage to remove these **int** wrappers if one wishes to eliminate the variables.

---

removing four int wrappers

Although it is difficult to remove all the **int** wrappers, four can be removed. This will be done in two steps.

Theorem. Removing the first pair does not take long.

```
In[6]:= SubstTest[implies, and[equal[u, int[s]], equal[v, int[t]]],
          or[equal[id[omega], int[x]], equal[intmul[u, int[z]], intmul[v, int[y]]],
           not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
           not[equal[intmul[int[w], int[z]], intmul[int[x], int[y]]]]], {s → u, t → v}] // Reverse

Out[6]= or[equal[id[omega], int[x]], equal[intmul[u, int[z]], intmul[v, int[y]]],
          not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
          not[equal[intmul[int[w], int[z]], intmul[int[x], int[y]]]],
          not[member[u, Z]], not[member[v, Z]]] == True

In[7]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. Removing a second pair takes a little longer.

```
In[8]:= SubstTest[implies, and[equal[y, int[s]], equal[z, int[t]]], or[equal[id[omega], int[x]],
           equal[intmul[u, z], intmul[v, y]], not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
           not[equal[intmul[int[w], z], intmul[int[x], y]]],
           not[member[u, Z]], not[member[v, Z]]], {s → y, t → z}] // Reverse

Out[8]= or[equal[id[omega], int[x]], equal[intmul[u, z], intmul[v, y]],
          not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
          not[equal[intmul[y, int[x]], intmul[z, int[w]]]], not[member[u, Z]],
          not[member[v, Z]], not[member[y, Z]], not[member[z, Z]]] == True

In[9]:= (% /. {u → u_, v → v_, w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

eliminating four literals

Before attempting to remove the remaining two **int** wrappers, it is desirable to reduce the number of literals. The following lemma shows that at least two integer-hood literals are redundant. The lemma implies that if **u** is an integer, then so is **v**, and vice versa. The same goes for **y** and **z**.

Lemma.

```
In[10]:= SubstTest[implies, and[equal[r, s], member[s, Z]],
            member[r, Z], {r → intmul[u, int[x]], s → intmul[v, int[w]]}] // Reverse

Out[10]= or[member[u, Z],
            not[equal[intmul[u, int[x]], intmul[v, int[w]]]], not[member[v, Z]]] == True

In[11]:= or[member[u_, Z],
            not[equal[intmul[int[w_], v_], intmul[int[x_], u_]]], not[member[v_, Z]]] := True
```

A further reduction of integer-hood literals is possible.

Lemma. Either **u** or **y** must be an integer.

```
In[12]:= Map[or[member[u, Z], member[y, Z], #] &,
            SubstTest[implies, and[equal[r, t], equal[s, t]], equal[r, s],
             {r -> intmul[u, z], s -> intmul[v, y], t → V}]] // Reverse

Out[12]= or[equal[intmul[u, z], intmul[v, y]], member[u, Z], member[y, Z]] == True

In[13]:= or[equal[intmul[u_, z_], intmul[v_, y_]], member[u_, Z], member[y_, Z]] := True
```

Theorem. The literal **member[z, Z]** is redundant.

```
In[14]:= SubstTest[and, implies[p, q], or[p, q],
           {p → member[u, Z], q → implies[and[equal[intmul[u, int[x]], intmul[v, int[w]]],
              not[equal[intmul[u, z], intmul[v, y]]]], member[v, Z]]}]

Out[14]= or[equal[intmul[u, z], intmul[v, y]], member[v, Z],
            not[equal[intmul[u, int[x]], intmul[v, int[w]]]]] == True
```

```
In[15]:= or[equal[intmul[u_, z_], intmul[v_, y_]], member[v_, Z],
           not[equal[intmul[int[w_], v_], intmul[int[x_], u_]]]] := True
```

By symmetry, the same goes for the other integer-hood literals. These redundant literals can be removed all at once.

Theorem. Elimination of four integer-hood literals.

```
In[16]:= SubstTest[and, implies[p, q], or[p, q],
           {p → and[member[u, Z], member[v, Z], member[y, Z], member[z, Z]],
            q -> or[equal[id[omega], int[x]], equal[intmul[u, z], intmul[v, y]],
              not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
              not[equal[intmul[y, int[x]], intmul[z, int[w]]]]]}] // MapNotNot

Out[16]= or[equal[id[omega], int[x]], equal[intmul[u, z], intmul[v, y]],
           not[equal[intmul[u, int[x]], intmul[v, int[w]]]],
           not[equal[intmul[y, int[x]], intmul[z, int[w]]]]] == True

In[17]:= or[equal[id[omega], int[x_]], equal[intmul[u_, z_], intmul[v_, y_]],
           not[equal[intmul[int[w_], v_], intmul[int[x_], u_]]],
           not[equal[intmul[int[w_], z_], intmul[int[x_], y_]]]] := True
```

The final two **int** wrappers can now be removed, introducing two new integer-hood literals.

Theorem. A transitivity statement with six literals, and six variables, but no **int** wrappers.

```
In[18]:= SubstTest[implies, and[equal[w, int[s]], equal[x, int[t]]], or[equal[id[omega], x],
            equal[intmul[u, z], intmul[v, y]], not[equal[intmul[u, x], intmul[v, w]]],
            not[equal[intmul[y, x], intmul[z, w]]]], {s → w, t → x}] // Reverse

Out[18]= or[equal[x, id[omega]], equal[intmul[u, z], intmul[v, y]],
           not[equal[intmul[u, x], intmul[v, w]]], not[equal[intmul[w, z], intmul[x, y]]],
           not[member[w, Z]], not[member[x, Z]]] == True

In[19]:= or[equal[id[omega], x_], equal[intmul[u_, z_], intmul[v_, y_]],
           not[equal[intmul[u_, x_], intmul[v_, w_]]],
           not[equal[intmul[w_, z_], intmul[x_, y_]]],
           not[member[w_, Z]], not[member[x_, Z]]] := True
```