inequalities for multiplication

Johan G. F. Belinfante
revised 2002 October 12

```
<< goedel52.p82; << tools.m

:Package Title: goedel52.p82          2002 October 8 at 1:50 p.m.

It is now:  2002 Oct 12 at 12:35

Loading Simplification Rules

TOOLS.M                   Revised 2002 September 16

weightlimit = 40
```

■ summary

It is shown in this notebook that a nonzero product of natural numbers is not less than its factors. Several versions are given, some with variables, some without. These theorems are the counterpart of Quaife's theorem **(O16)**.

```
Art Quaife, "Automated Development of Fundamental Mathematical Theories,"
Kluwer Academic Publishers, Dordrecht, 1992. (See page 189.)
```

■ two general lemmas

Two new rewrite rules are used in the sequel which appear to be generally useful. The first is this:

```
or[not[equal[y, z]], subclass[intersection[x, y], z]] // AssertTest

or[not[equal[y, z]], subclass[intersection[x, y], z]] == True

or[not[equal[y_, z_]], subclass[intersection[x_, y_], z_]] := True
```

The other result of general interest is:

```
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]], w -> V]

or[equal[0, u], not[equal[0, v]], not[subclass[u, v]]] == True

or[equal[0, u_], not[equal[0, v_]], not[subclass[u_, v_]]] := True
```

■ a special rule

In this section a special formula is derived which requires several temporary lemmas. This is the first:

```
Map[or[equal[y, z], #] &,
 SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
    {u -> y, v -> intersection[x, singleton[y]], w -> singleton[z]}]]
```

```
or[equal[y, z], not[member[y, x]],
  not[subclass[intersection[x, singleton[y]], singleton[z]]]] == True
```

```
or[equal[y_, z_], not[member[y_, x_]],
  not[subclass[intersection[x_, singleton[y_]], singleton[z_]]]] := True
```

Temporary lemma 2.

```
SubstTest[implies, equal[u, 0], subclass[u, v],
 {u -> intersection[x, singleton[y]], v -> singleton[z]}]
```

```
or[member[y, x], subclass[intersection[x, singleton[y]], singleton[z]]] == True
```

```
or[member[y_, x_], subclass[intersection[x_, singleton[y_]], singleton[z_]]] := True
```

Temporary lemma 3.

```
Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
    {p1 -> equal[y, z], p2 -> equal[singleton[y], singleton[z]],
        p3 -> subclass[intersection[x, singleton[y]], singleton[z]]}]]
```

```
or[not[equal[y, z]], subclass[intersection[x, singleton[y]], singleton[z]]] == True
```

```
or[not[equal[y_, z_]], subclass[intersection[x_, singleton[y_]], singleton[z_]]] := True
```

Double negation is used to combine lemmas 2 and 3.

```
or[and[member[y, x], not[equal[y, z]]],
  subclass[intersection[x, singleton[y]], singleton[z]]] // NotNotTest
```

```
or[and[member[y, x], not[equal[y, z]]],
  subclass[intersection[x, singleton[y]], singleton[z]]] == True
```

```
or[and[member[y_, x_], not[equal[y_, z_]]],
  subclass[intersection[x_, singleton[y_]], singleton[z_]]] := True
```

This observation...

```
equiv[subclass[intersection[x, singleton[y]], singleton[z]],
 or[equal[y, z], not[member[y, x]]]]
```

```
True
```

... justifies this somewhat special rewrite rule, which we nonetheless intend to add as a permanent rule.

```
subclass[intersection[x_, singleton[y_]], singleton[z_]] :=
 or[equal[y, z], not[member[y, x]]]
```

■ an application to multiplication

The newly derived special rewrite rule is needed for this application:

```
Map[subclass[y, #] &,
 SubstTest[natmul, y, natadd[u, v], {u → singleton[0], v → natsub[x, singleton[0]]}]]
```

```
or[equal[0, x], not[member[x, omega]], subclass[y, natmul[x, y]]] == True
```

```
or[equal[0, x_], not[member[x_, omega]], subclass[y_, natmul[x_, y_]]] := True
```

This result can be improved. First we note that:

```
Map[or[#, member[x, omega]] &, SubstTest[implies, equal[v, V], subclass[u, v],
    {u -> y, v -> natmul[x, y]}]]
```

```
or[member[x, omega], subclass[y, natmul[x, y]]] == True
```

```
or[member[x_, omega], subclass[y_, natmul[x_, y_]]] := True
```

This already yields a simpler rule:

```
Map[not,
 SubstTest[and, implies[and[p1, p2], p3], implies[not[p1], p3], not[implies[p2, p3]],
    {p1 -> member[x, omega], p2 -> not[equal[0, x]], p3 -> subclass[y, natmul[x, y]]}]]
```

```
or[equal[0, x], subclass[y, natmul[x, y]]] == True
```

```
or[equal[0, x_], subclass[y_, natmul[x_, y_]]] := True
```

This will be further improved by adding a converse.

■ deriving a converse

First step:

```
Map[or[#, equal[0, y]] &, SubstTest[and, subclass[u, v], equal[0, v],
  {u → y, v → natmul[x, y]}]]
```

```
or[and[equal[0, x], member[y, omega], subclass[y, natmul[x, y]]], equal[0, y]] ==
 equal[0, y]
```

```
Map[implies[First[%], #] &, %] // Reverse
```

```
or[equal[0, y], not[equal[0, x]],
  not[member[y, omega]], not[subclass[y, natmul[x, y]]]] == True
```

```
or[equal[0, y_], not[equal[0, x_]],
  not[member[y_, omega]], not[subclass[y_, natmul[x_, y_]]]] := True
```

Second step:

```
Map[or[member[y, omega], #] &,
 SubstTest[implies, equal[V, v], subclass[y, v], v → natmul[x, y]]]
```

```
or[member[y, omega], subclass[y, natmul[x, y]]] == True
```

```
or[member[y_, omega], subclass[y_, natmul[x_, y_]]] := True
```

Third step:

```
or[and[equal[0, x], member[y, omega], not[equal[0, y]]],
  subclass[y, natmul[x, y]]] // NotNotTest
```

```
or[and[equal[0, x], member[y, omega], not[equal[0, y]]],
  subclass[y, natmul[x, y]]] == True
```

```
or[and[equal[0, x_], member[y_, omega], not[equal[0, y_]]],
  subclass[y_, natmul[x_, y_]]] := True
```

This obervation...

```
equiv[subclass[y, natmul[x, y]],
 implies[equal[0, x], or[not[member[y, omega]], equal[0, y]]]]
```

```
True
```

... justifies the final rewrite rule:

```
subclass[y_, natmul[x_, y_]] := or[equal[0, y], not[equal[0, x]], not[member[y, omega]]]
```

Only this will be made permanent since the other inequality rules for multiplication are subsumed by this one.

■ a corollary

Quaife has noted the following corollary, which does not require introducing any additional rewrite rules.

```
subclass[x, natmul[x, x]]
```

```
True
```

■ eliminating the variable y.

The elimination of the variable **y** is done as follows:

```
SubstTest[assert, forall[y, or[equal[0, x], subclass[y, APPLY[w, y]]]],
  w -> composite[NATMUL, LEFT[x]]] // Reverse
```

```
or[equal[0, x], subclass[composite[NATMUL, LEFT[x]], S]] == True
```

```
or[equal[0, x_], subclass[composite[NATMUL, LEFT[x_]], S]] := True
```

Conversely,

```
SubstTest[implies, subclass[u, v],
 subclass[image[inverse[u], w], image[inverse[v], w]],
    {u -> composite[NATMUL, LEFT[x]], v -> S, w -> singleton[0]}]
```

```
or[not[equal[0, x]], not[subclass[composite[NATMUL, LEFT[x]], S]]] == True
```

```
or[not[equal[0, x_]], not[subclass[composite[NATMUL, LEFT[x_]], S]]] := True
```

This equivalence ...

```
equiv[subclass[composite[NATMUL, LEFT[x]], S], not[equal[0, x]]]

True
```

... justifies adding the following rewrite rule:

```
subclass[composite[NATMUL, LEFT[x_]], S] := not[equal[0, x]]
```

■ another formulation

The result found can be written using a single literal. To do this, we need this result:

```
subclass[composite[id[image[V, x]], y], z] // AssertTest

subclass[composite[id[image[V, x]], y], z] ==
 or[equal[0, x], subclass[composite[Id, y], z]]

subclass[composite[id[image[V, x_]], y_], z_] :=
 or[equal[0, x], subclass[composite[Id, y], z]]
```

The reformulation as a single literal is this:

```
subclass[composite[id[image[V, x]], NATMUL, LEFT[x]], S]

True
```

This reformulation may be useful for deducing consequences of this inequality.

■ eliminating the variable x

One can eliminate the other variable by a similar technique, but the result one obtains is not very pretty.

```
Map[complement, SubstTest[class, x,
   subclass[composite[z, LEFT[x]], s], {s -> S, z -> NATMUL}] // InvertFix]

singleton[0] == intersection[omega,
  image[complement[inverse[fix[composite[inverse[NATMUL], S, SECOND]]]], omega]]
```

One can obtain a better variable−freeformula that expresses the inequality derived in this notebook. Note that:

```
class[pair[pair[x, y], z], and[subclass[x, z], subclass[y, z]]]

composite[S, CUP]
```

The formula we want will be derived using **dif** and **VSNormality**.

```
dif[composite[id[complement[singleton[0]]], NATMUL], composite[S, CUP]] // VSNormality

union[composite[id[complement[singleton[0]]],
   intersection[NATMUL, composite[complement[S], FIRST]]],
  composite[id[complement[singleton[0]]],
   intersection[NATMUL, composite[complement[S], SECOND]]]] == 0
```

```
union[composite[id[complement[singleton[0]]],
   intersection[NATMUL, composite[complement[S], FIRST]]],
  composite[id[complement[singleton[0]]],
   intersection[NATMUL, composite[complement[S], SECOND]]]] := 0
```

The final result is this:

```
SubstTest[equal, 0, dif[u, v],
    {u -> composite[id[complement[singleton[0]]], NATMUL], v -> composite[S, CUP]}] //
 Reverse
```

```
subclass[composite[id[complement[singleton[0]]], NATMUL], composite[S, CUP]] == True
```

```
subclass[composite[id[complement[singleton[0]]], NATMUL], composite[S, CUP]] := True
```