total order for integers

Johan G. F. Belinfante
2012 November 10

```
In[1]:= SetDirectory["l:"]; << goedel.12nov08a
```

```
:Package Title: goedel.12nov08a              2012 November 8 at 7:40 a.m.

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

It is now:  2012 Nov 10 at 5:58

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 Nov 10 at 6:14
```

summary

Multiplication by a non-negative integer is monotone with respect to the relation **INTLEQ**.

non-negative differences

An integer **x** is less than or equal to an integer **y** if and only if the difference **y – x** is non-negative. In this section, a wrapper-free rewrite rule for this fact is derived.

Lemma.

```
In[3]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 -> member[pair[x, y], INTLEQ],
            p2 → member[x, Z], p3 → subclass[x, cart[V, V]]}]] // Reverse
```

```
Out[3]= or[not[member[pair[x, y], INTLEQ]], subclass[x, cart[V, V]]] == True
```

```
In[4]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma

```
In[5]:= Map[implies[member[pair[x, y], INTLEQ], #] &, SubstTest[member, pair[x, y],
           composite[image[inverse[INTADD], t], INVERSE], t → range[PLUS]]] // MapNotNot
```

```
Out[5]= or[member[intadd[y, inverse[x]], range[PLUS]], not[member[pair[x, y], INTLEQ]]] == True
```

```
In[6]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary.

```
In[7]:= SubstTest[implies, member[pair[t, y], INTLEQ],
          member[intadd[y, inverse[t]], range[PLUS]], t → composite[Id, x]] // Reverse

Out[7]= or[member[intadd[y, inverse[x]], range[PLUS]],
          not[member[pair[composite[Id, x], y], INTLEQ]]] == True
```

```
In[8]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (This result has two redundant literals.)

```
In[9]:= Map[implies[#, member[pair[composite[Id, x], y], INTLEQ]] &,
          SubstTest[member, pair[composite[Id, x], y],
           composite[image[inverse[INTADD], t], INVERSE], t → range[PLUS]]] // MapNotNot

Out[9]= or[member[pair[composite[Id, x], y], INTLEQ], not[member[domain[x], V]],
          not[member[intadd[y, inverse[x]], range[PLUS]]], not[member[range[x], V]]] == True
```

```
In[10]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[11]:= (Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
              not[implies[p1, p3]], {p1 -> member[intadd[y, t], range[PLUS]],
               p2 → member[t, Z], p3 → member[t, V]}]] // Reverse) /. t → inverse[x]

Out[11]= or[and[member[domain[x], V], member[range[x], V]],
           not[member[intadd[y, inverse[x]], range[PLUS]]]] == True
```

```
In[12]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Eliminating two redundant literals.)

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
             not[implies[p1, p3]], {p1 -> member[intadd[y, inverse[x]], range[PLUS]],
              p2 -> and[member[domain[x], V], member[range[x], V]],
              p3 -> member[pair[composite[Id, x], y], INTLEQ]}]] // Reverse

Out[13]= or[member[pair[composite[Id, x], y], INTLEQ],
           not[member[intadd[y, inverse[x]], range[PLUS]]]] == True
```

```
In[14]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Main Theorem. A wrapper-free rewrite rule concerning non-negative differences.

```
In[15]:= equiv[member[intadd[y, inverse[x]], range[PLUS]],
             member[pair[composite[Id, x], y], INTLEQ]] // not // not

Out[15]= True
```

```
In[16]:= member[intadd[y_, inverse[x_]], range[PLUS]] :=
          member[pair[composite[Id, x], y], INTLEQ]
```

Corollary.

```
In[17]:= SubstTest[member, intadd[x, inverse[t]], range[PLUS], t → id[omega]]

Out[17]= member[pair[id[omega], x], INTLEQ] == member[x, range[PLUS]]

In[18]:= member[pair[id[omega], x_], INTLEQ] := member[x, range[PLUS]]
```

monotonicity theorem

Theorem. The product of non-negative integers is non-negative.

```
In[20]:= SubstTest[implies, and[member[t, u], subclass[u, v]], member[t, v], {t → pair[x, y],
            u → cartsq[range[PLUS]], v → image[inverse[INTMUL], range[PLUS]]}] // Reverse

Out[20]= or[member[intmul[x, y], range[PLUS]],
           not[member[x, range[PLUS]]], not[member[y, range[PLUS]]]] == True

In[21]:= or[member[intmul[x_, y_], range[PLUS]],
           not[member[x_, range[PLUS]]], not[member[y_, range[PLUS]]]] := True
```

Lemma. The monotonicity result with wrapped variables.

```
In[29]:= SubstTest[implies, and[member[t, range[PLUS]], member[w, range[PLUS]]],
           member[intmul[t, w], range[PLUS]],
           {t → intadd[int[y], inverse[int[x]]], w → int[z]}] // Reverse

Out[29]= or[member[pair[intmul[int[x], int[z]], intmul[int[y], int[z]]], INTLEQ],
           not[member[int[z], range[PLUS]]], not[member[pair[int[x], int[y]], INTLEQ]]] == True

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

The variables can be eliminated. To prepare for this, the **int** wrappers need to be removed.

Lemma. Simplification rule.

```
In[24]:= equiv[and[member[x, Z], member[pair[x, y], INTLEQ]], member[pair[x, y], INTLEQ]]

Out[24]= True

In[25]:= and[member[x_, Z], member[pair[x_, y_], INTLEQ]] := member[pair[x, y], INTLEQ]
```

Lemma. Simplification rule.

```
In[26]:= equiv[and[member[y, Z], member[pair[x, y], INTLEQ]], member[pair[x, y], INTLEQ]]

Out[26]= True
```

```
In[27]:= and[member[y_, Z], member[pair[x_, y_], INTLEQ]] := member[pair[x, y], INTLEQ]
```

Theorem. (Eliminating the **int** wrapper.)

```
In[31]:= SubstTest[implies, and[equal[x, int[u]], equal[y, int[v]], equal[z, int[w]]],
           or[member[pair[intmul[x, z], intmul[y, z]], INTLEQ], not[member[z, range[PLUS]]],
            not[member[pair[x, y], INTLEQ]]], {u → x, v → y, w → z}] // Reverse // MapNotNot

Out[31]= or[member[pair[intmul[x, z], intmul[y, z]], INTLEQ],
           not[member[z, range[PLUS]]], not[member[pair[x, y], INTLEQ]]] == True

In[32]:= or[member[pair[intmul[x_, z_], intmul[y_, z_]], INTLEQ],
           not[member[z_, range[PLUS]]], not[member[pair[x_, y_], INTLEQ]]] := True
```

Lemma. A membership rule for composites with **inttimes[z]**.

```
In[33]:= (member[pair[x, y], composite[w, funpart[t]]] // AssertTest) /. t → inttimes[z]

Out[33]= member[pair[x, y], composite[w, inttimes[z]]] ==
          and[member[x, Z], member[y, V], member[z, Z], member[pair[intmul[x, z], y], w]]

In[34]:= member[pair[x_, y_], composite[w_, inttimes[z_]]] :=
          and[member[x, Z], member[y, V], member[z, Z], member[pair[intmul[x, z], y], w]]
```

Lemma. A membership rule for composites with **inverse[inttimes[z]]**.

```
In[35]:= (member[pair[x, y], composite[inverse[funpart[t]], w]] // AssertTest) /. t → inttimes[z]

Out[35]= member[pair[x, y], composite[inverse[inttimes[z]], w]] ==
          and[member[x, V], member[y, Z], member[z, Z], member[pair[x, intmul[y, z]], w]]

In[36]:= member[pair[x_, y_], composite[inverse[inttimes[z_]], w_]] :=
          and[member[x, V], member[y, Z], member[z, Z], member[pair[x, intmul[y, z]], w]]
```

Lemma.

```
In[37]:= or[and[member[x, Z], member[y, Z],
            member[z, Z], member[pair[intmul[x, z], intmul[y, z]], INTLEQ]],
           not[member[z, range[PLUS]]], not[member[pair[x, y], INTLEQ]]] // NotNotTest

Out[37]= or[and[member[x, Z], member[y, Z],
            member[z, Z], member[pair[intmul[x, z], intmul[y, z]], INTLEQ]],
           not[member[z, range[PLUS]]], not[member[pair[x, y], INTLEQ]]] == True

In[38]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Temporary Lemma. Simplification rule.

```
In[41]:= SubstTest[subclass, inverse[u], inverse[v],
          {u -> composite[INVERSE, inttimes[z], INTLEQ, inverse[inttimes[z]], INVERSE],
           v -> composite[INVERSE, INTLEQ, INVERSE]}]

Out[41]= subclass[composite[INVERSE, inttimes[z], INTLEQ, inverse[inttimes[z]], INVERSE],
           composite[INVERSE, INTLEQ, INVERSE]] ==
         subclass[composite[inttimes[z], INTLEQ, inverse[inttimes[z]]], INTLEQ]

In[42]:= subclass[composite[INVERSE, inttimes[z_], INTLEQ, inverse[inttimes[z_]], INVERSE],
           composite[INVERSE, INTLEQ, INVERSE]] :=
         subclass[composite[inttimes[z], INTLEQ, inverse[inttimes[z]]], INTLEQ]
```

Theorem. Monotonicity of multiplication by a non-negative integer.

```
In[43]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[y, z],
           implies[and[member[pair[y, z], u], member[x, w]], member[pair[y, z], v]], {u → INTLEQ,
            v → composite[inverse[inttimes[x]], INTLEQ, inttimes[x]], w → range[PLUS]}]]

Out[43]= or[not[member[x, range[PLUS]]],
           subclass[composite[inttimes[x], INTLEQ, inverse[inttimes[x]]], INTLEQ] == True

In[44]:= or[not[member[x_, range[PLUS]]],
           subclass[composite[inttimes[x_], INTLEQ, inverse[inttimes[x_]]], INTLEQ]] := True
```

Theorem. A variable-free restatement of the monotonicity theorem.

```
In[45]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[implies[member[x, u], member[inttimes[x], v]]],
           {u → range[PLUS], v → monotone[INTLEQ, INTLEQ]}]]

Out[45]= subclass[image[INTTIMES, range[PLUS]], monotone[INTLEQ, INTLEQ]] == True

In[46]:= subclass[image[INTTIMES, range[PLUS]], monotone[INTLEQ, INTLEQ]] := True
```

serendipity

A similar result holds for natural number arithmetic.

Theorem. A variable-free statement of the monotonicity of multiplication by natural numbers.

```
In[47]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[member[times[x], t]], t -> monotone[S, S]]]

Out[47]= subclass[binhom[NATADD, NATADD], monotone[S, S]] == True

In[48]:= subclass[binhom[NATADD, NATADD], monotone[S, S]] := True
```