integer divisibility relation

Johan G. F. Belinfante
2006 October 21

```
In[1]:= SetDirectory["l:"]; << goedel86.21a; << tools.m

:Package Title: goedel86.21a          2006 October 21 at 4:40 p.m.

It is now:  2006 Oct 21 at 19:16

Loading Simplification Rules

TOOLS.M                      Revised 2006 October 21

weightlimit = 40
```

summary

The integer divisibility relation **INTDIV** has been introduced into the **GOEDEL** program by means of a **class**-wrapped membership rule:

```
In[2]:= Begin["Goedel`Private`"];

In[3]:= ? INTDIV

INTDIV is the integer divisibility relation

In[4]:= FirstMatch[class[x_, member[y_, HoldPattern[INTDIV]]]]

Out[4]= class[x_, member[y_, INTDIV]] := ReleaseHold[Module[{z = Unique[]}, class[
          x, exists[z, and[member[z, binhom[INTADD, INTADD]]], member[y, z]]]]]]
```

A novel feature of this definition is that it does not involve explicit mention of integer multiplication. Indeed, at this stage of development, integer multiplication has not yet been defined in the **GOEDEL** program. The definition of **INTDIV** was inspired by a formula derived for the natural number divisibility relation **DIV**. (A technical comment: The expression **binhom[INTADD, INTADD]]** in this definition has been wrapped with **HoldPattern** to prevent the **class** rules from expanding this expression any further.)

U[binhom[NATADD,NATADD]]

The divisibility relation for natural numbers is the union of all **times[x]** functions, a fact which can be succinctly stated as follows:

```
In[5]:= range[reify[x, times[x]]]

Out[5]= DIV
```

The function **times[x] = composite[NATMUL, LEFT[x]]** is the operation of multiplication of natural numbers by **x**. These functions are precisely the binary homomorphisms for natural addition. Using the function **TIMES = lambda[x, times[x]]**, one can write this fact as

```
In[6]:= range[TIMES]

Out[6]= binhom[NATADD, NATADD]
```

Lemma. (Comment. The orientation of this as a rewrite rule is based on an analogy with an existing rule in which the functio **TIMES** is replaced by **PLUS**.)

```
In[7]:= Assoc[SWAP, inverse[E],
          composite[VERTSECT[inverse[rotate[NATMUL]]], id[omega]]] // Reverse

Out[7]= composite[inverse[E], TIMES] == composite[SWAP, inverse[rotate[NATMUL]]]

In[8]:= composite[inverse[E], TIMES] := composite[SWAP, inverse[rotate[NATMUL]]]
```

An immediate corollary is that the divisibility relation is the union of all **times[x]** functions.

```
In[9]:= ImageComp[inverse[E], TIMES, V] // Reverse

Out[9]= U[binhom[NATADD, NATADD]] == DIV

In[10]:= U[binhom[NATADD, NATADD]] := DIV
```

Theorem. One can recover **NATMUL** from **TIMES**.

```
In[11]:= rotate[composite[inverse[TIMES], E]] // VSTriNormality

Out[11]= rotate[composite[inverse[TIMES], E]] == NATMUL

In[12]:= rotate[composite[inverse[TIMES], E]] := NATMUL
```

integer divisibility

Although integer multiplication has yet to be defined, the above results for natural numbers suggests a definition of integer divisibility by simply replacing **NATADD** with **INTADD**. The **class**-wrapped membership rule for **INTDIV** implies the following normalizaton rule:

```
In[13]:= INTDIV // Normality // Reverse

Out[13]= U[binhom[INTADD, INTADD]] == INTDIV

In[14]:= U[binhom[INTADD, INTADD]] := INTDIV
```

INTDIV is a relation

Upper bound.

```
In[15]:= SubstTest[implies, subclass[u, v],
          subclass[U[u], U[v]], {u → binhom[INTADD, INTADD], v -> map[Z, Z]}]

Out[15]= subclass[INTDIV, cart[Z, Z]] == True

In[16]:= subclass[INTDIV, cart[Z, Z]] := True
```

Corollary.

```
In[17]:= SubstTest[implies, subclass[u, cart[Z, Z]], subclass[u, cart[V, V]], u → INTDIV]

Out[17]= subclass[INTDIV, cart[V, V]] == True

In[18]:= subclass[INTDIV, cart[V, V]] := True
```

Corollary.

```
In[19]:= equal[composite[Id, INTDIV], INTDIV]

Out[19]= True

In[20]:= composite[Id, INTDIV] := INTDIV
```

sethood

The relation **INTDIV** is a set:

```
In[21]:= SubstTest[implies, and[subclass[x, y], member[y, V]],
          member[x, V], {x → INTDIV, y → cart[Z, Z]}]

Out[21]= member[INTDIV, V] == True

In[22]:= member[INTDIV, V] := True
```

a general theorem

If a class is closed under composition then the relational part of its union is transitive.

```
In[23]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → image[COMPOSE, cart[x, x]], v → x, w → inverse[E]}]

Out[23]= or[not[subclass[image[COMPOSE, cart[x, x]], x]], TRANSITIVE[composite[Id, U[x]]]] == True
```

```
In[24]:= or[not[subclass[image[COMPOSE, cart[x_, x_]], x_]],
           TRANSITIVE[composite[Id, U[x_]]]] := True
```

A variable-free corollary can be derived which captures this result for the special case of sets. Technical comment: The **setpart** wrapper helps in eliminating the variable **x**.

```
In[25]:= Map[equal[V, #] &,
           SubstTest[class, x, implies[member[setpart[x], u], member[U[setpart[x]], v]],
            {u → binclosed[COMPOSE], v → image[inverse[IMAGE[id[cart[V, V]]]], TRV]}]] // Reverse

Out[25]= subclass[image[IMAGE[id[cart[V, V]]], image[BIGCUP, binclosed[COMPOSE]]], TRV] == True

In[26]:= % /. Equal → SetDelayed
```

The reverse inclusion also holds:

```
In[27]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
          {u → inverse[POWER], v → BIGCUP, w → binclosed[COMPOSE]}]

Out[27]= subclass[image[inverse[IMAGE[id[cart[V, V]]]], TRV],
           image[BIGCUP, binclosed[COMPOSE]]] == True

In[28]:= % /. Equal → SetDelayed
```

These inclusions can be combined into an equation:

```
In[29]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[inverse[IMAGE[id[cart[V, V]]]], TRV],
           v -> image[BIGCUP, binclosed[COMPOSE]]}]

Out[29]= True ==
          equal[image[BIGCUP, binclosed[COMPOSE]], image[inverse[IMAGE[id[cart[V, V]]]], TRV]]

In[30]:= image[BIGCUP, binclosed[COMPOSE]] := image[inverse[IMAGE[id[cart[V, V]]]], TRV]
```

TRANSITIVE

```
In[31]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
          {u -> binhom[INTADD, INTADD], v → binclosed[COMPOSE],
           w → image[inverse[BIGCUP], image[inverse[IMAGE[id[cart[V, V]]]], TRV]]}]

Out[31]= TRANSITIVE[INTDIV] == True

In[32]:= TRANSITIVE[INTDIV] := True
```

REFLEXIVE

Lemma.

```
In[33]:= SubstTest[implies, member[x, y],
          subclass[x, U[y]], {x → id[Z], y → binhom[INTADD, INTADD]}]

Out[33]= subclass[Z, fix[INTDIV]] == True

In[34]:= % /. Equal → SetDelayed

In[35]:= SubstTest[implies, subclass[u, v],
          subclass[fix[u], fix[v]], {u → INTDIV, v → cart[Z, Z]}]

Out[35]= subclass[fix[INTDIV], Z] == True

In[36]:= % /. Equal → SetDelayed
```

Theorem.

```
In[37]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> fix[INTDIV], v -> Z}]

Out[37]= True == equal[Z, fix[INTDIV]]

In[38]:= fix[INTDIV] := Z
```

Corollary.

```
In[39]:= SubstTest[subclass, x, cart[fix[x], fix[x]], x → INTDIV] // Reverse

Out[39]= REFLEXIVE[INTDIV] == True

In[40]:= REFLEXIVE[INTDIV] := True
```

Corollary.

```
In[41]:= SubstTest[domain, rfx[x], x → INTDIV]

Out[41]= domain[INTDIV] == Z

In[42]:= domain[INTDIV] := Z
```

Corollary.

```
In[43]:= SubstTest[range, rfx[x], x → INTDIV]

Out[43]= range[INTDIV] == Z

In[44]:= range[INTDIV] := Z
```

Corollary.

```
In[45]:= SubstTest[idempotent, rfx[trv[x]], x → INTDIV]

Out[45]= equal[INTDIV, composite[INTDIV, INTDIV]] == True

In[46]:= composite[INTDIV, INTDIV] := INTDIV
```

comment

The natural number divisibility relation is a partial order. Integer divisibility is not a partial order because any integer divides its negative.

```
In[47]:= SubstTest[implies, member[x, y], subclass[x, U[y]],
          {x → composite[id[Z], INVERSE], y → binhom[INTADD, INTADD]}]

Out[47]= subclass[composite[id[Z], INVERSE], INTDIV] == True

In[48]:= subclass[composite[id[Z], INVERSE], INTDIV] := True
```

Corollary.

```
In[49]:= SubstTest[implies, subclass[x, y], subclass[composite[w, x], composite[w, y]],
          {w → INTDIV, x → composite[id[Z], INVERSE], y → INTDIV}]

Out[49]= subclass[composite[INTDIV, INVERSE], INTDIV] == True

In[50]:= % /. Equal → SetDelayed
```

Lemma.

```
In[51]:= Assoc[INTDIV, id[Z], id[P[cart[V, V]]]] // Reverse

Out[51]= composite[INTDIV, id[P[cart[V, V]]]] == INTDIV

In[52]:= composite[INTDIV, id[P[cart[V, V]]]] := INTDIV

In[53]:= SubstTest[implies, subclass[x, y], subclass[composite[x, z], composite[y, z]],
          {x → composite[INTDIV, INVERSE], y → INTDIV, z → INVERSE}]

Out[53]= subclass[INTDIV, composite[INTDIV, INVERSE]] == True

In[54]:= % /. Equal → SetDelayed
```

Theorem.

```
In[55]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> composite[INTDIV, INVERSE], v -> INTDIV}]

Out[55]= True == equal[INTDIV, composite[INTDIV, INVERSE]]

In[56]:= composite[INTDIV, INVERSE] := INTDIV
```

A similar result holds in the reverse order.

```
In[57]:= SubstTest[implies, subclass[x, y], subclass[composite[x, z], composite[y, z]],
          {x → composite[id[Z], INVERSE], y → INTDIV, z → INTDIV}]

Out[57]= subclass[composite[INVERSE, INTDIV], INTDIV] == True
```

```
In[58]:= % /. Equal → SetDelayed
```

Lemma.

```
In[59]:= Assoc[id[P[cart[V, V]]], id[Z], INTDIV]

Out[59]= composite[id[P[cart[V, V]]], INTDIV] == INTDIV

In[60]:= composite[id[P[cart[V, V]]], INTDIV] := INTDIV

In[61]:= SubstTest[implies, subclass[x, y], subclass[composite[w, x], composite[w, y]],
           {w → INVERSE, x → composite[INVERSE, INTDIV], y → INTDIV}]

Out[61]= subclass[INTDIV, composite[INVERSE, INTDIV]] == True

In[62]:= % /. Equal → SetDelayed
```

Final result.

```
In[63]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> composite[INVERSE, INTDIV], v -> INTDIV}]

Out[63]= True == equal[INTDIV, composite[INVERSE, INTDIV]]

In[64]:= composite[INVERSE, INTDIV] := INTDIV
```

serendipity

Some additional facts discovered in the course of this work are recorded here. First, a general result:

```
In[65]:= member[x, image[inverse[funpart[y]], z]] // AssertTest

Out[65]= member[x, image[inverse[funpart[y]], z]] == member[APPLY[funpart[y], x], z]

In[66]:= member[x_, image[inverse[funpart[y_]], z_]] := member[APPLY[funpart[y], x], z]
```

Corollary A general inverse-image rule for **TIMES**.

```
In[67]:= SubstTest[member, x, image[inverse[funpart[z]], y], z → TIMES]

Out[67]= member[x, image[inverse[TIMES], y]] == and[member[x, omega], member[times[x], y]]

In[68]:= member[x_, image[inverse[TIMES], y_]] := and[member[x, omega], member[times[x], y]]
```