divisibility relation

Johan G. F. Belinfante
2002 August 30

```
<< goedel52.p21; << tools.m
```

```
:Package Title: goedel52.p21        2002 August 29 at 9:30 p.m.
```

```
It is now:  2002 Aug 30 at 19:47
```

```
Loading Simplification Rules
```

```
TOOLS.M                    Revised 2002 August 22
```

```
weightlimit = 40
```

■ Summary

In this notebook, the divisibility relation for natural numbers is defined, and a few of its basic properties are derived. In our work with **Otter** definitions of classes are generally given as equations, and membership properties are deduced from these equations, whereas with the **GOEDEL** program, our usual practice is to start with a membership rule, and then to derive an equation. One is under no obligation to do this, but it is just more convenient in many cases. Often times the equation needed to define a given class in terms of the primitives is not immediately clear, and one can take advantage of Gödel's algorithm to discover it. In the case of divisibility, this approach did not prove to be feasible because the program just took too long. We therefore deviate from our usual practice by defining the divisibility relation with an equation instead of starting with a membership rule. It turns out to be easier to derive the membership rule from the equation than the other way around.

■ Definition

A formula for the divisibility relation can be discovered quickly by sequestering **NATMUL** from the **class** construction. That is, we replace **NATMUL** with a dummy variable **n** and later replace it with **NATMUL** to shield it from Gödel's algorithm.

```
DIV == class[pair[x, y], exists[z, member[pair[pair[x, z], y], n]]] /. n -> NATMUL
```

```
DIV == composite[NATMUL, inverse[FIRST]]
```

We take this as our definition of the divisibility relation:

```
composite[NATMUL, inverse[SECOND]] := DIV
```

From this equational definition, it is easy to derive a membership rule:

```
SubstTest[assert, member[x, composite[z, inverse[SECOND]]], z -> NATMUL]
```

```
member[x, DIV] == member[first[x], range[image[inverse[NATMUL], singleton[second[x]]]]]
```

```
member[x_, DIV] := member[first[x], range[image[inverse[NATMUL], singleton[second[x]]]]]
```

■ Vertical section rules

Vertical section rules are probably as useful as membership rules, and can be deduced as follows:

```
ImageComp[NATMUL, inverse[SECOND], singleton[x]]
```

```
image[DIV, singleton[x]] == image[NATMUL, cart[V, singleton[x]]]
```

```
image[DIV, singleton[x_]] := image[NATMUL, cart[V, singleton[x]]]
```

```
IminComp[NATMUL, inverse[SECOND], singleton[x]]
```

```
image[inverse[DIV], singleton[x]] == range[image[inverse[NATMUL], singleton[x]]]
```

```
image[inverse[DIV], singleton[x_]] := range[image[inverse[NATMUL], singleton[x]]]
```

■ Applications of the Assoc test.

Since multiplication of natural numbers is commutative, one can replace **SECOND** wth **FIRST** if desired:

```
Assoc[NATMUL, SWAP, inverse[SECOND]]
```

```
composite[NATMUL, inverse[FIRST]] == DIV
```

```
composite[NATMUL, inverse[FIRST]] := DIV
```

Another basic property is this:

```
Assoc[Id, NATMUL, inverse[SECOND]]
```

```
composite[Id, DIV] == DIV
```

```
composite[Id, DIV] := DIV
```

From this, the **GOEDEL** program can deduce that **DIV** is a relation:

```
subclass[DIV, cart[V, V]]
```

```
True
```

■ Domain, range and fix.

The following equation is only needed as a temporary rule:

```
Assoc[id[omega], NATMUL, inverse[SECOND]]
```

```
composite[id[omega], DIV] == DIV
```

```
composite[id[omega], DIV] := DIV
```

From this one deduces an upper bound on the range:

```
Map[subclass[#, omega] &, ImageComp[id[omega], DIV, V]]
```

```
subclass[range[DIV], omega] == True
```

```
subclass[range[DIV], omega] := True
```

A similar technique works for the domain:

```
Assoc[NATMUL, inverse[SECOND], id[omega]] // Reverse
```

```
composite[DIV, id[omega]] == DIV
```

```
composite[DIV, id[omega]] := DIV
```

```
Map[subclass[#, omega] &, IminComp[DIV, id[omega], V]]
```

```
subclass[domain[DIV], omega] == True
```

```
subclass[domain[DIV], omega] := True
```

To get a lower bound, we use this obvservation:

```
SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
    {u -> LEFT[x], v -> inverse[SECOND], w -> NATMUL}]
```

```
subclass[composite[NATMUL, LEFT[x]], DIV] == True
```

```
subclass[composite[NATMUL, LEFT[x_]], DIV] := True
```

Setting **x = 1**, we get

```
SubstTest[subclass, composite[NATMUL, LEFT[x]], DIV, x -> singleton[0]]
```

```
subclass[omega, fix[DIV]] == True
```

```
subclass[omega, fix[DIV]] := True
```

From this one gets also lower bounds on the domain and range:

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> omega, v -> fix[DIV], w -> domain[DIV]}]
```

```
subclass[omega, domain[DIV]] == True
```

```
subclass[omega, domain[DIV]] := True
```

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> omega, v -> fix[DIV], w -> range[DIV]}]
```

```
subclass[omega, range[DIV]] == True
```

```
subclass[omega, range[DIV]] := True
```

Putting together the upper and lower bounds yields equations:

```
SubstTest[and, subclass[u, v], subclass[v, u], {u -> domain[DIV], v -> omega}]
```

```
True == equal[omega, domain[DIV]]
```

```
domain[DIV] := omega
```

```
SubstTest[and, subclass[u, v], subclass[v, u], {u -> range[DIV], v -> omega}]
```

```
True == equal[omega, range[DIV]]
```

```
range[DIV] := omega
```

A similar technique works for the fixed point set:

```
SubstTest[subclass, fix[z], domain[z], z -> DIV]
```

```
subclass[fix[DIV], omega] == True
```

```
subclass[fix[DIV], omega] := True
```

```
SubstTest[and, subclass[u, v], subclass[v, u], {u -> fix[DIV], v -> omega}]
```

```
True == equal[omega, fix[DIV]]
```

```
fix[DIV] := omega
```

■ DIV is a reflexive relation

The facts deduced above imply that **DIV** is a reflexive relation:

```
REFLEXIVE[DIV]
```

```
True
```

One can improve on this by using the fact that **omega** is a set, which implies that **DIV** is also a set:

```
SubstTest[implies, and[subclass[u, v], member[v, V]], member[u, V],
    {u -> DIV, v -> cart[omega, omega]}]
```

```
member[DIV, V] == True
```

```
member[DIV, V] := True
```

Hence **DIV** belongs to the class **RFX** of all small reflexive relations:

```
member[DIV, RFX]
```

```
True
```

■ The relation DIV is transitive.

```
SubstTest[assert, forall[x, y, subclass[composite[z, LEFT[x], z, LEFT[y]], w]],
    {w -> DIV, z -> NATMUL}]
```

```
True == equal[0,
  domain[fix[composite[inverse[SECOND], DIV, complement[inverse[DIV]], NATMUL]]]]
```

We turn this around and make it a temporary rewrite rule:

```
domain[fix[composite[inverse[SECOND], DIV, complement[inverse[DIV]], NATMUL]]] := 0
```

The **domain** wrapper can be peeled off:

```
SubstTest[composite, z, id[domain[z]],
    z -> fix[composite[inverse[SECOND], DIV, complement[inverse[DIV]], NATMUL]]] //
 Reverse
```

```
fix[composite[inverse[SECOND], DIV, complement[inverse[DIV]], NATMUL]] == 0
```

```
fix[composite[inverse[SECOND], DIV, complement[inverse[DIV]], NATMUL]] := 0
```

The following procedure rewrites this as a transitivity property:

```
SubstTest[implies, equal[0, fix[composite[u, v]]], equal[0, fix[composite[v, u]]],
    {u -> inverse[SECOND], v -> composite[DIV, complement[inverse[DIV]], NATMUL]}]
```

```
subclass[composite[DIV, DIV], DIV] == True
```

```
subclass[composite[DIV, DIV], DIV] := True
```

This result can be improved by using the reflexive property:

```
SubstTest[implies, subclass[u, v], subclass[composite[u, w], composite[v, w]],
    {u -> id[omega], v -> DIV, w -> DIV}]
```

```
subclass[DIV, composite[DIV, DIV]] == True
```

```
subclass[DIV, composite[DIV, DIV]] := True
```

```
SubstTest[and, subclass[u, v], subclass[v, u], {u -> DIV, v -> composite[DIV, DIV]}]
```

```
True == equal[DIV, composite[DIV, DIV]]
```

The following rewrite rule is therefore valid:

```
composite[DIV, DIV] := DIV
```