distributive law for natmod

Johan G. Belinfante
2005 June 30

```
In[1]:= SetDirectory["i:"]; << goedel70.29a; << tools.m

        :Package Title: goedel70.29a       2005 June 29 at 5:00 p.m.

        It is now:  2005 Jun 30 at 8:57

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 June 17

        weightlimit = 40
```

summary

In this notebook it is shown that **natmul** distributes over **natmod**. The key ideas are the distributivity of **natmul** over **DIV** and a uniqueness theorem for **natmod**.

derivation

Since **natmul** distributes over **natmod**, one has:

```
In[2]:= SubstTest[implies, member[pair[u, v], DIV],
         member[pair[natmul[nat[x], u], natmul[nat[x], v]], DIV],
         {u -> nat[z], v -> natsub[nat[y], natmod[nat[y], nat[z]]]}]

Out[2]= member[pair[natmul[nat[x], nat[z]], natsub[natmul[nat[x], nat[y]],
            natmul[nat[x], natmod[nat[y], nat[z]]]]], DIV] == True

In[3]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

A cancellation law for multiplication implies the following, which is made into a temporary rewrite rule:

```
In[4]:= SubstTest[member, natmul[nat[x], nat[u]],
         natmul[nat[x], nat[z]], u → natmod[nat[y], nat[z]]]

Out[4]= member[natmul[nat[x], natmod[nat[y], nat[z]]], natmul[nat[x], nat[z]]] ==
         and[not[equal[0, nat[x]]], not[equal[0, nat[z]]]]
```

```
In[5]:= member[natmul[nat[x_], natmod[nat[y_], nat[z_]]], natmul[nat[x_], nat[z_]]] :=
         and[not[equal[0, nat[x]]], not[equal[0, nat[z]]]]
```

A uniqueness theorem for **natmod** yields.

```
In[6]:= SubstTest[implies, and[member[pair[nat[v], natsub[nat[u], nat[w]]], DIV],
          member[nat[w], nat[v]]], equal[nat[w], natmod[nat[u], nat[v]]],
         {u → natmul[nat[x], nat[y]], v → natmul[nat[x], nat[z]],
          w → natmul[nat[x], natmod[nat[y], nat[z]]]}]

Out[6]= or[equal[0, nat[x]], equal[0, nat[z]],
          equal[natmod[natmul[nat[x], nat[y]], natmul[nat[x], nat[z]]],
           natmul[nat[x], natmod[nat[y], nat[z]]]]] == True

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

The equality literals are redundant, and can be removed.

```
In[8]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p3],
          or[p1, p2, p3], not[p3], {p1 -> equal[0, nat[x]], p2 -> equal[0, nat[z]],
           p3 -> equal[natmod[natmul[nat[x], nat[y]], natmul[nat[x], nat[z]]],
             natmul[nat[x], natmod[nat[y], nat[z]]]]}]]

Out[8]= equal[natmod[natmul[nat[x], nat[y]], natmul[nat[x], nat[z]]],
          natmul[nat[x], natmod[nat[y], nat[z]]]] == True

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

One can remove the **nat** wrappers in favor of numberhood literals:

```
In[10]:= SubstTest[implies, and[equal[u, nat[x]], equal[v, nat[y]], equal[w, nat[z]]],
          equal[natmod[natmul[u, v], natmul[u, w]], natmul[u, natmod[v, w]]],
          {u → x, v → y, w → z}]

Out[10]= or[equal[natmod[natmul[x, y], natmul[x, z]], natmul[x, natmod[y, z]]],
           not[member[x, omega]], not[member[y, omega]], not[member[z, omega]]] == True

In[11]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

If any variable is not a number, the equation also holds.

```
In[12]:= SubstTest[implies, and[equal[u, V], equal[v, V]], equal[u, v],
          {u -> natmod[natmul[x, y], natmul[x, z]], v -> natmul[x, natmod[y, z]]}]

Out[12]= or[and[member[x, omega], member[y, omega], member[z, omega]],
           equal[natmod[natmul[x, y], natmul[x, z]], natmul[x, natmod[y, z]]]] == True

In[13]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Consequently, the numberhood literals can be omitted. This distributive law is made into a rewrite rule, oriented to factor rather than expand.

```
In[14]:= SubstTest[and, implies[p1, p2], implies[not[p1], p2],
          {p1 -> and[member[x, omega], member[y, omega], member[z, omega]],
           p2 -> equal[natmod[natmul[x, y], natmul[x, z]], natmul[x, natmod[y, z]]]}]

Out[14]= True == equal[natmod[natmul[x, y], natmul[x, z]], natmul[x, natmod[y, z]]]

In[15]:= natmod[natmul[x_, y_], natmul[x_, z_]] := natmul[x, natmod[y, z]]
```

---

removing the variabes

The variables **y** and **z** are easiest to remove, using a standard **symdif** technique:

```
In[16]:= symdif[composite[NATMOD,
            cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
           composite[NATMUL, LEFT[x], NATMOD]] // VSTriNormality

Out[16]= union[composite[intersection[composite[NATMOD,
              cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
             composite[complement[NATMUL], LEFT[x], NATMOD]], id[cart[V, V]]],
           composite[intersection[composite[complement[NATMOD],
              cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
             composite[NATMUL, LEFT[x], NATMOD]], id[cart[V, V]]]] == 0

In[17]:= (% /. x → x_) /. Equal → SetDelayed
```

This can be rewritten as a simpler equation:

```
In[18]:= SubstTest[equal, 0, composite[symdif[u, v], id[cart[V, V]]],
          {u -> composite[NATMOD, cross[composite[NATMUL, LEFT[x]],
              composite[NATMUL, LEFT[x]]]], v -> composite[NATMUL, LEFT[x], NATMOD]}]

Out[18]= True == equal[composite[NATMOD,
            cross[composite[NATMUL, LEFT[x]], composite[NATMUL, LEFT[x]]]],
           composite[NATMUL, LEFT[x], NATMOD]]

In[19]:= composite[NATMOD,
           cross[composite[NATMUL, LEFT[x_]], composite[NATMUL, LEFT[x_]]]] :=
          composite[NATMUL, LEFT[x], NATMOD]
```

The following lemma prepares for eliminating the variable **x**.

```
In[20]:= Map[equal[intersection[composite[inverse[FIRST], NATMUL, cross[Id, FIRST]],
             composite[inverse[SECOND], NATMUL, cross[Id, SECOND]]], #] &,
          (composite[cross[x, x], TWIST, cross[DUP, Id]] // TriNormality) /. x → NATMUL]

Out[20]= equal[composite[cross[NATMUL, NATMUL], TWIST, cross[DUP, Id]],
           intersection[composite[inverse[FIRST], NATMUL, cross[Id, FIRST]],
            composite[inverse[SECOND], NATMUL, cross[Id, SECOND]]]] == True

In[21]:= intersection[composite[inverse[FIRST], NATMUL, cross[Id, FIRST]],
           composite[inverse[SECOND], NATMUL, cross[Id, SECOND]]] :=
          composite[cross[NATMUL, NATMUL], TWIST, cross[DUP, Id]]
```

Reifying with respect to **x** yields a variable-free formulation of the distributive law:

```
In[22]:= Map[rotate[inverse[#]] &,
           SubstTest[reify, x, composite[u, cross[composite[NATMUL, LEFT[x]],
              composite[NATMUL, LEFT[x]]]], u → NATMOD]] // Reverse

Out[22]= composite[NATMOD, cross[NATMUL, NATMUL], TWIST, cross[DUP, Id]] ==
          composite[NATMUL, cross[Id, NATMOD]]

In[23]:= composite[NATMOD, cross[NATMUL, NATMUL], TWIST, cross[DUP, Id]] :=
          composite[NATMUL, cross[Id, NATMOD]]
```