semigroups for modular arithmetic

Johan G. F. Belinfante
2006 July 11

```
In[1]:= SetDirectory["l:"]; << goedel83.11a; << tools.m

        :Package Title: goedel83.11a         2006 July 11 at 2:40 p.m.

        It is now:  2006 Jul 11 at 17:6

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 July 8

        weightlimit = 40
```

summary

Modular addition and modular multiplication provide examples of semigroups.

modular addition

Lemma 1.

```
In[2]:= Map[equal[0, #] &, dif[nat[x], x] // Renormality]

Out[2]= subclass[nat[x], x] == True

In[3]:= subclass[nat[x_], x_] := True
```

Lemma 2.

```
In[4]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → image[modulo[x], y], v → range[modulo[x]], w → x}]

Out[4]= or[equal[0, x], subclass[image[modulo[x], y], x]] == True

In[5]:= or[equal[0, x_], subclass[image[modulo[x_], y_], x_]] := True
```

Applying a theorem about restrictions of binary operations, one finds:

```
In[6]:= SubstTest[implies, and[member[u, BINOPS], member[v, binclosed[u]]],
         member[composite[u, id[cart[v, v]]], BINOPS],
         {u -> composite[modulo[nat[x]], NATADD], v → nat[x]}]

Out[6]= or[member[composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]], BINOPS], not[
          subclass[image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]], nat[x]]] == True

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

In particular, the hypothesis is satisfied when **nat[x]** is nonzero.

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
          {p1 → not[empty[nat[x]]],
           p2 -> subclass[image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]], nat[x]],
           p3 -> member[composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]], BINOPS]}]]

Out[8]= or[equal[0, nat[x]],
          member[composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]], BINOPS]] == True

In[9]:= (% /. x → x_) /. Equal → SetDelayed
```

Removing the **nat** wrapper, one obtains:

```
In[10]:= SubstTest[implies, equal[x, nat[y]],
          or[equal[0, x], member[composite[modulo[x], NATADD, id[cart[x, x]]], BINOPS]], y → x]

Out[10]= or[equal[0, x], member[composite[modulo[x], NATADD, id[cart[x, x]]], BINOPS],
           not[member[x, omega]]] == True

In[11]:= (% /. x → x_) /. Equal → SetDelayed
```

The same conclusion holds when **x** is not a natural number.

```
In[12]:= SubstTest[implies, equal[0, z],
          member[composite[z, NATADD, id[cart[x, x]]], BINOPS], z → modulo[x]]

Out[12]= or[member[x, omega],
           member[composite[modulo[x], NATADD, id[cart[x, x]]], BINOPS]] == True

In[13]:= (% /. x → x_) /. Equal → SetDelayed
```

A simple rewrite rule results when these two cases are combined:

```
In[14]:= SubstTest[and, implies[p1, p2], or[p1, p2],
             {p1 → and[member[x, omega], not[empty[x]]], p2 -> member[
                composite[modulo[x], NATADD, id[cart[x, x]]], BINOPS]}] // MapNotNot // Reverse

Out[14]= member[composite[modulo[x], NATADD, id[cart[x, x]]], BINOPS] == True

In[15]:= member[composite[modulo[x_], NATADD, id[cart[x_, x_]]], BINOPS] := True
```

Corollary.

```
In[16]:= member[composite[modulo[x], NATADD, id[cart[x, x]]], SEMIGPS] // AssertTest

Out[16]= member[composite[modulo[x], NATADD, id[cart[x, x]]], SEMIGPS] == True

In[17]:= member[composite[modulo[x_], NATADD, id[cart[x_, x_]]], SEMIGPS] := True
```

modular multiplication

The derivation for **NATMUL** goes through in a similar fashion.

```
In[18]:= SubstTest[implies, and[member[u, BINOPS], member[v, binclosed[u]]],
          member[composite[u, id[cart[v, v]]], BINOPS],
          {u -> composite[modulo[nat[x]], NATMUL], v → nat[x]}]

Out[18]= or[member[composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]], BINOPS],
           not[subclass[image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]], nat[x]]] ==
          True

In[19]:= (% /. x → x_) /. Equal → SetDelayed

In[20]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → not[empty[nat[x]]],
             p2 -> subclass[image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]], nat[x]],
             p3 -> member[composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]], BINOPS]}]]

Out[20]= or[equal[0, nat[x]],
           member[composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]], BINOPS]] == True

In[21]:= (% /. x → x_) /. Equal → SetDelayed

In[22]:= SubstTest[implies, equal[x, nat[y]],
          or[equal[0, x], member[composite[modulo[x], NATMUL, id[cart[x, x]]], BINOPS]], y → x]

Out[22]= or[equal[0, x], member[composite[modulo[x], NATMUL, id[cart[x, x]]], BINOPS],
           not[member[x, omega]]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed

In[24]:= SubstTest[implies, equal[0, z],
          member[composite[z, NATMUL, id[cart[x, x]]], BINOPS], z → modulo[x]]

Out[24]= or[member[x, omega],
           member[composite[modulo[x], NATMUL, id[cart[x, x]]], BINOPS]] == True

In[25]:= (% /. x → x_) /. Equal → SetDelayed

In[26]:= SubstTest[and, implies[p1, p2], or[p1, p2],
             {p1 → and[member[x, omega], not[empty[x]]], p2 -> member[
                composite[modulo[x], NATMUL, id[cart[x, x]]], BINOPS]}] // MapNotNot // Reverse

Out[26]= member[composite[modulo[x], NATMUL, id[cart[x, x]]], BINOPS] == True

In[27]:= member[composite[modulo[x_], NATMUL, id[cart[x_, x_]]], BINOPS] := True
```

```
In[28]:= member[composite[modulo[x], NATMUL, id[cart[x, x]]], SEMIGPS] // AssertTest

Out[28]= member[composite[modulo[x], NATMUL, id[cart[x, x]]], SEMIGPS] == True

In[29]:= member[composite[modulo[x_], NATMUL, id[cart[x_, x_]]], SEMIGPS] := True
```

domains

Lemma.

```
In[30]:= Map[or[equal[0, nat[x]], #] &, SubstTest[subclass, w, intersection[x, y],
           {w → intersection[omega, x], y → image[V, intersection[omega, set[x]]]}]]

Out[30]= or[equal[0, nat[x]], subclass[intersection[omega, x], nat[x]]] == True

In[31]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[32]:= equal[cart[intersection[omega, x], nat[x]], cart[nat[x], nat[x]]] // assert

Out[32]= True

In[33]:= cart[intersection[omega, x_], nat[x_]] := cart[nat[x], nat[x]]
```

Both operations have the cartesian square of **nat[x]** as domain.

```
In[34]:= domain[composite[modulo[x], NATADD, id[cart[x, x]]]]

Out[34]= cart[nat[x], nat[x]]

In[35]:= domain[composite[modulo[x], NATMUL, id[cart[x, x]]]]

Out[35]= cart[nat[x], nat[x]]
```

range for modular addition

Observation:

```
In[36]:= abstract[w, image[modulo[x], image[NATADD, cart[w, x]]]]

Out[36]= composite[inverse[E], IMAGE[modulo[x]], IMAGE[SECOND], VERTSECT[
           composite[id[composite[NATADD, id[cart[V, x]]]], inverse[FIRST], inverse[FIRST]]]]
```

Adding zero to any number gives it back.

```
In[39]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → set[0], v → nat[x],
           w -> composite[inverse[E], IMAGE[modulo[nat[x]]], IMAGE[SECOND], VERTSECT[composite[
               id[composite[NATADD, id[cart[V, nat[x]]]]], inverse[FIRST], inverse[FIRST]]]]}]

Out[39]= subclass[nat[x], image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]]] == True

In[40]:= (% /. x → x_) /. Equal → SetDelayed
```

Since **composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]]** is a binary operation, the reverse inclusion also holds:

```
In[37]:= SubstTest[implies, member[w, BINOPS], subclass[range[w], fix[domain[w]]],
          w -> composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]]]

Out[37]= subclass[image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]], nat[x]] == True

In[38]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the two inclusions yields an equation.

```
In[41]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> nat[x],
            v -> image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]]}] // Reverse

Out[41]= equal[image[modulo[nat[x]], image[NATADD, cart[nat[x], nat[x]]]], nat[x]] == True

In[42]:= (% /. x → x_) /. Equal → SetDelayed
```

The **nat** wrappers are removed, except for one.

```
In[43]:= SubstTest[implies, equal[x, nat[y]],
          equal[image[modulo[x], image[NATADD, cart[x, x]]], nat[x]], y → x]

Out[43]= or[equal[image[modulo[x], image[NATADD, cart[x, x]]], nat[x]],
           not[member[x, omega]]] == True

In[44]:= (% /. x → x_) /. Equal → SetDelayed
```

For non-numbers, a similar result holds.

```
In[45]:= SubstTest[implies, and[equal[0, z], not[member[x, omega]]],
          equal[image[z, image[NATADD, cart[x, x]]], nat[x]], z → modulo[x]]

Out[45]= or[equal[image[modulo[x], image[NATADD, cart[x, x]]], nat[x]], member[x, omega]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the cases of numbers and non-numbers yields a simple rewrite rule:

```
In[47]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 → member[x, omega],
           p2 -> equal[image[modulo[x], image[NATADD, cart[x, x]]], nat[x]]}]

Out[47]= True == equal[image[modulo[x], image[NATADD, cart[x, x]]], nat[x]]
```

```
In[48]:= image[modulo[x_], image[NATADD, cart[x_, x_]]] := nat[x]
```

range for modular multiplication

Lemma.

```
In[49]:= ImageComp[NATMUL, LEFT[set[0]], nat[x]] // Reverse

Out[49]= image[NATMUL, cart[set[set[0]], nat[x]]] == nat[x]

In[50]:= image[NATMUL, cart[set[set[0]], nat[x_]]] := nat[x]
```

Multiplying a number by one gives it back.

```
In[51]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → set[set[0]], v → nat[x],
           w -> composite[inverse[E], IMAGE[modulo[nat[x]]], IMAGE[SECOND], VERTSECT[composite[
               id[composite[NATMUL, id[cart[V, nat[x]]]]], inverse[FIRST], inverse[FIRST]]]]}]

Out[51]= or[not[member[set[0], nat[x]]],
           subclass[nat[x], image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed
```

The same conclusion holds when **nat[x]** is zero or one.

```
In[53]:= SubstTest[and, implies[p1, p4], implies[p2, p4],
           implies[p3, p4], {p1 → equal[set[0], nat[x]], p2 → member[set[0], nat[x]],
            p3 → member[nat[x], set[0]], p4 -> subclass[nat[x],
              image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]]]}] // Reverse

Out[53]= subclass[nat[x], image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

Since **composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]** is a binary operation, the reverse inclusion holds.

```
In[55]:= SubstTest[implies, member[w, BINOPS], subclass[range[w], fix[domain[w]]],
          w -> composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]]]

Out[55]= subclass[image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]], nat[x]] == True

In[56]:= (% /. x → x_) /. Equal → SetDelayed
```

Combine the inclusions to obtain an equation.

```
In[57]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]],
            v -> nat[x]}] // Reverse

Out[57]= equal[image[modulo[nat[x]], image[NATMUL, cart[nat[x], nat[x]]]], nat[x]] == True
```

```
In[58]:= (% /. x → x_) /. Equal → SetDelayed
```

Remove all but one **nat** wrapper.

```
In[59]:= SubstTest[implies, equal[x, nat[y]],
          equal[image[modulo[x], image[NATMUL, cart[x, x]]], nat[x]], y → x]

Out[59]= or[equal[image[modulo[x], image[NATMUL, cart[x, x]]], nat[x]],
           not[member[x, omega]]] == True

In[60]:= (% /. x → x_) /. Equal → SetDelayed
```

For non-numbers, one has:

```
In[61]:= SubstTest[implies, and[equal[0, z], not[member[x, omega]]],
          equal[image[z, image[NATMUL, cart[x, x]]], nat[x]], z → modulo[x]]

Out[61]= or[equal[image[modulo[x], image[NATMUL, cart[x, x]]], nat[x]], member[x, omega]] == True

In[62]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining the cases of numbers and non-numbers, one finds:

```
In[63]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 → member[x, omega],
           p2 -> equal[image[modulo[x], image[NATMUL, cart[x, x]]], nat[x]]}]

Out[63]= True == equal[image[modulo[x], image[NATMUL, cart[x, x]]], nat[x]]

In[64]:= image[modulo[x_], image[NATMUL, cart[x_, x_]]] := nat[x]
```

---

map statements

Each of these two binary operations is a mapping from **cart[nat[x], nat[x]]** to **nat[x]**.

```
In[65]:= member[composite[modulo[x], NATADD, id[cart[x, x]]],
           map[cart[nat[x], nat[x]], nat[x]]] // AssertTest

Out[65]= member[composite[modulo[x], NATADD, id[cart[x, x]]],
           map[cart[nat[x], nat[x]], nat[x]]] == True

In[66]:= member[composite[modulo[x_], NATADD, id[cart[x_, x_]]],
           map[cart[nat[x_], nat[x_]], nat[x_]]] := True

In[67]:= member[composite[modulo[x], NATMUL, id[cart[x, x]]],
           map[cart[nat[x], nat[x]], nat[x]]] // AssertTest

Out[67]= member[composite[modulo[x], NATMUL, id[cart[x, x]]],
           map[cart[nat[x], nat[x]], nat[x]]] == True

In[68]:= member[composite[modulo[x_], NATMUL, id[cart[x_, x_]]],
           map[cart[nat[x_], nat[x_]], nat[x_]]] := True
```