associativity of modular arithmetic

Johan G. F. Belinfante
2005 July 21

```
In[1]:= SetDirectory["i:"]; << goedel71.21a; << tools.m
```

```
:Package Title: goedel71.21a          2005 July 21 at 9:00 a.m.

It is now:  2005 Jul 22 at 16:47

Loading Simplification Rules

TOOLS.M                       Revised 2005 July 18

weightlimit = 40
```

---

summary

The functional formulations of Quaife's Theorems **(Q10)** and **(Q15)** may be viewed as statements about homomorphisms from the natural number system to modular number systems. In this notebook, this is used to derive formal statements about the associativity of modular arithmetic. The case of modulo **0** is exceptional in many ways, but not with regard to the statements of the theorems in this notebook.

---

modular addition

Lemma.

```
In[2]:= Assoc[composite[modulo[x], NATADD],
          cross[Id, modulo[x]], composite[cross[Id, NATADD], ASSOC]]
```

```
Out[2]= composite[modulo[x], NATADD, cross[Id, composite[modulo[x], NATADD]], ASSOC] ==
          composite[modulo[x], NATADD, cross[NATADD, Id]]
```

```
In[3]:= composite[modulo[x_], NATADD, cross[Id, composite[modulo[x_], NATADD]],
           ASSOC] := composite[modulo[x], NATADD, cross[NATADD, Id]]
```

Lemma.

```
In[4]:= Assoc[composite[modulo[x], NATADD], cross[modulo[x], Id], cross[NATADD, Id]]

Out[4]= composite[modulo[x], NATADD, cross[composite[modulo[x], NATADD], Id]] ==
         composite[modulo[x], NATADD, cross[NATADD, Id]]

In[5]:= composite[modulo[x_], NATADD, cross[composite[modulo[x_], NATADD], Id]] :=
         composite[modulo[x], NATADD, cross[NATADD, Id]]
```

The associativity of modular addition.

```
In[6]:= SubstTest[implies, and[subclass[y, cart[cart[V, V], V]],
          equal[composite[y, cross[Id, y], ASSOC], composite[y, cross[y, Id]]]],
         associative[y], y -> composite[modulo[x], NATADD]]

Out[6]= associative[composite[modulo[x], NATADD]] == True

In[7]:= associative[composite[modulo[x_], NATADD]] := True
```

The restriction of addition modulo **x** to numbers less than **x** is also assocative. Quite a few steps are needed to obtain a clean statement of this.

Lemma.

```
In[8]:= ImageComp[modulo[nat[x]], id[nat[x]], V] // Reverse

Out[8]= image[modulo[nat[x]], nat[x]] == nat[x]

In[9]:= image[modulo[nat[x_]], nat[x_]] := nat[x]
```

Temporary rule.

```
In[10]:= ImageComp[composite[modulo[nat[x]], NATADD],
           cross[modulo[nat[x]], modulo[nat[x]]], V] // Reverse

Out[10]= union[image[modulo[nat[x]], image[NATADD, cart[nat[x],
             union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]]]],
           intersection[omega, complement[image[V, nat[x]]]]] ==
          union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]

In[11]:= union[image[modulo[nat[x_]], image[NATADD, cart[nat[x_],
             union[intersection[omega, complement[image[V, nat[x_]]]], nat[x_]]]]],
           intersection[omega, complement[image[V, nat[x_]]]]] :=
          union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]
```

A standard theorem about restrictions of associative operations to classes closed under the operation now yields:

```
In[12]:= SubstTest[implies, and[associative[u], subclass[image[u, cart[v, v]], v]],
          associative[composite[u, id[cart[v, v]]]],
          {u → composite[modulo[nat[x]], NATADD], v → range[modulo[nat[x]]]}]

Out[12]= associative[composite[modulo[nat[x]], NATADD,
           id[union[cart[omega, intersection[omega, complement[image[V, nat[x]]]]],
             cart[nat[x], union[
               intersection[omega, complement[image[V, nat[x]]], nat[x]]]]]]] == True

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

This can be simplified:

```
In[14]:= SubstTest[implies, equal[y, range[modulo[nat[x]]]],
          associative[composite[modulo[nat[x]], NATADD, id[cart[y, y]]]], y → nat[x]]

Out[14]= or[associative[composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]]],
           equal[0, nat[x]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

The equation also holds when **nat[x] = 0,** so that redundant literal can be removed.

```
In[16]:= SubstTest[and, implies[p1, p2],
           or[p1, p2], {p1 → equal[0, nat[x]], p2 -> associative[
              composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]]]}] // Reverse

Out[16]= associative[
           composite[modulo[nat[x]], NATADD, id[cart[nat[x], nat[x]]]]] == True

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

The **nat** wrapper can be removed.

```
In[18]:= SubstTest[implies, equal[y, nat[x]],
          associative[composite[modulo[y], NATADD, id[cart[y, y]]]], y → x]

Out[18]= or[associative[composite[modulo[x], NATADD, id[cart[x, x]]]],
           not[member[x, omega]]] == True

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

The numberhood literal is also redundant:

```
In[20]:= SubstTest[implies, equal[0, y], associative[y],
           y -> composite[modulo[x], NATADD, id[cart[x, x]]]] // MapNotNot

Out[20]= or[associative[composite[modulo[x], NATADD, id[cart[x, x]]]],
           member[x, omega]] == True
```

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

Finally one obtains a simple statement about addition modulo **x** restricted to numbers less than **x**.

```
In[22]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 → member[x, omega],
           p2 -> associative[composite[modulo[x], NATADD, id[cart[x, x]]]]}] // Reverse
Out[22]= associative[composite[modulo[x], NATADD, id[cart[x, x]]]] == True

In[23]:= associative[composite[modulo[x_], NATADD, id[cart[x_, x_]]]] := True
```

---

modular multiplication

Lemma.

```
In[24]:= Assoc[composite[modulo[x], NATMUL],
          cross[Id, modulo[x]], composite[cross[Id, NATMUL], ASSOC]]
Out[24]= composite[modulo[x], NATMUL, cross[Id, composite[modulo[x], NATMUL]], ASSOC] ==
          composite[modulo[x], NATMUL, cross[NATMUL, Id]]

In[25]:= composite[modulo[x_], NATMUL, cross[Id, composite[modulo[x_], NATMUL]],
           ASSOC] := composite[modulo[x], NATMUL, cross[NATMUL, Id]]
```

Lemma.

```
In[26]:= Assoc[composite[modulo[x], NATMUL], cross[modulo[x], Id], cross[NATMUL, Id]]
Out[26]= composite[modulo[x], NATMUL, cross[composite[modulo[x], NATMUL], Id]] ==
          composite[modulo[x], NATMUL, cross[NATMUL, Id]]

In[27]:= composite[modulo[x_], NATMUL, cross[composite[modulo[x_], NATMUL], Id]] :=
          composite[modulo[x], NATMUL, cross[NATMUL, Id]]
```

Theorem about the associativity of modular multiplication.

```
In[28]:= SubstTest[implies, and[subclass[y, cart[cart[V, V], V]],
           equal[composite[y, cross[Id, y], ASSOC], composite[y, cross[y, Id]]]],
          associative[y], y -> composite[modulo[x], NATMUL]]
Out[28]= associative[composite[modulo[x], NATMUL]] == True

In[29]:= associative[composite[modulo[x_], NATMUL]] := True
```

The restriction of multiplication modulo **x** to numbers less than **x** is also assocative. The derivation is quite similar to that for addition.

```
In[30]:= ImageComp[composite[modulo[nat[x]], NATMUL],
           cross[modulo[nat[x]], modulo[nat[x]]], V] // Reverse

Out[30]= union[image[modulo[nat[x]], image[NATMUL, cart[nat[x],
             union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]]]],
           intersection[omega, complement[image[V, nat[x]]]]] ==
         union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]

In[31]:= union[image[modulo[nat[x_]], image[NATMUL, cart[nat[x_],
             union[intersection[omega, complement[image[V, nat[x_]]]], nat[x_]]]]],
           intersection[omega, complement[image[V, nat[x_]]]]] :=
         union[intersection[omega, complement[image[V, nat[x]]]], nat[x]]

In[32]:= SubstTest[implies, and[associative[u], subclass[image[u, cart[v, v]], v]],
          associative[composite[u, id[cart[v, v]]]],
          {u → composite[modulo[nat[x]], NATMUL], v → range[modulo[nat[x]]]}]

Out[32]= associative[composite[modulo[nat[x]], NATMUL,
            id[union[cart[omega, intersection[omega, complement[image[V, nat[x]]]]],
              cart[nat[x], union[
                intersection[omega, complement[image[V, nat[x]]]], nat[x]]]]]]] == True

In[33]:= (% /. x → x_) /. Equal → SetDelayed

In[34]:= SubstTest[implies, equal[y, range[modulo[nat[x]]]],
          associative[composite[modulo[nat[x]], NATMUL, id[cart[y, y]]]], y → nat[x]]

Out[34]= or[associative[composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]]],
           equal[0, nat[x]]] == True

In[35]:= (% /. x → x_) /. Equal → SetDelayed

In[36]:= SubstTest[and, implies[p1, p2],
           or[p1, p2], {p1 → equal[0, nat[x]], p2 -> associative[
              composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]]]}] // Reverse

Out[36]= associative[
           composite[modulo[nat[x]], NATMUL, id[cart[nat[x], nat[x]]]]] == True

In[37]:= (% /. x → x_) /. Equal → SetDelayed

In[38]:= SubstTest[implies, equal[y, nat[x]],
          associative[composite[modulo[y], NATMUL, id[cart[y, y]]]], y → x]

Out[38]= or[associative[composite[modulo[x], NATMUL, id[cart[x, x]]]],
           not[member[x, omega]]] == True

In[39]:= (% /. x → x_) /. Equal → SetDelayed
```

```
In[40]:= Map[or[member[x, omega], #] &, SubstTest[implies, equal[0, y],
           associative[y], y -> composite[modulo[x], NATMUL, id[cart[x, x]]]]]

Out[40]= or[associative[composite[modulo[x], NATMUL, id[cart[x, x]]]],
           member[x, omega]] == True

In[41]:= (% /. x → x_) /. Equal → SetDelayed
```

This is the final result.

```
In[42]:= SubstTest[and, implies[p1, p2], or[p1, p2], {p1 → member[x, omega],
            p2 -> associative[composite[modulo[x], NATMUL, id[cart[x, x]]]]}] // Reverse

Out[42]= associative[composite[modulo[x], NATMUL, id[cart[x, x]]]] == True

In[43]:= associative[composite[modulo[x_], NATMUL, id[cart[x_, x_]]]] := True
```