Quaife's Theorem (Q15)

Johan G. F. Belinfante and Claudia D. Huang
2005 July 21

In[1]:=
```
SetDirectory["i:"]; << goedel71.20b; << tools.m
```

```
:Package Title: goedel71.20b         2005 July 20 at 3:15 p.m.

It is now:  2005 Jul 21 at 8:48

Loading Simplification Rules

TOOLS.M                     Revised 2005 July 18

weightlimit = 40
```

summary

Quaife's Theorem **(Q15)** is derived in this notebook. The derivation of this theorem is guided by analogy with Quaife's Theorem **(Q10)**. Using **nat** wrappers simplifies some steps, but the final result is wrapper-free. Removing variables yield a functional statement of the theorem.

In[2]:=
```
"Art Quaife, Automated Development of Fundamental Mathematical Theories,
   Kluwer Academic Publishers, Dordrecht, the Netherlands, 1992.";
```

Quaife's Theorem (DV7)

Quaife's Theorem **(DV7)** will be used as a lemma to derive Theorem **(Q15)**. This lemma will be derived in this section, using the transitivity of divisibility. Only one variable needs to be wrapped.

In[3]:=
```
SubstTest[implies, and[member[pair[x, y], DIV], member[pair[y, w], DIV]],
  member[pair[x, w], DIV], w → natmul[y, nat[z]]]
```

Out[3]=
```
or[member[pair[x, natmul[y, nat[z]]], DIV], not[member[pair[x, y], DIV]]] == True
```

In[4]:=
```
or[member[pair[x_, natmul[y_, nat[z_]]], DIV],
   not[member[pair[x_, y_], DIV]]] := True
```

derivation of (Q15)

An application of Theorem **(DV7)** yields this lemma.

```
In[5]:= SubstTest[implies, member[pair[v, w], DIV],
         member[pair[v, natmul[w, nat[z]]], DIV],
         {v → nat[y], w → natsub[nat[x], natmod[nat[x], nat[y]]]}]

Out[5]= member[pair[nat[y], natsub[natmul[nat[x], nat[z]],
             natmul[nat[z], natmod[nat[x], nat[y]]]]], DIV] == True

In[6]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem **(Q15)** can now be derived with **nat** wrappers as follows.

```
In[7]:= SubstTest[implies, member[pair[w, natsub[u, v]], DIV],
         equal[natmod[u, w], natmod[v, w]], {u -> natmul[nat[x], nat[y]],
          v -> natmul[nat[y], natmod[nat[x], nat[z]]], w → nat[z]}]

Out[7]= equal[natmod[natmul[nat[x], nat[y]], nat[z]],
          natmod[natmul[nat[y], natmod[nat[x], nat[z]]], nat[z]]] == True

In[8]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The **nat** wrappers can be replaced with numberhood literals.

```
In[9]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]], equal[z, nat[w]]],
         equal[natmod[natmul[x, y], z], natmod[natmul[x, natmod[y, z]], z]],
         {u → x, v → y, w → z}]

Out[9]= or[equal[natmod[natmul[x, y], z], natmod[natmul[x, natmod[y, z]], z]],
          not[member[x, omega]], not[member[y, omega]], not[member[z, omega]]] == True

In[10]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The numberhood literals are not needed because the same equation holds when any variable is not a number. In that case both sides of the equation are equal to the universal class **V**.

```
In[11]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
          {u -> natmod[natmul[x, y], z], v → V, w -> natmod[natmul[x, natmod[y, z]], z]}]

Out[11]= or[and[member[x, omega], member[y, omega], member[z, omega]],
           equal[natmod[natmul[x, y], z], natmod[natmul[x, natmod[y, z]], z]]] == True

In[12]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Combining the two cases yields a wrapper-free formulation of Theorem **(Q15)**.

```
In[13]:= SubstTest[and, implies[p1, p2], or[p1, p2],
          {p1 -> and[member[x, omega], member[y, omega], member[z, omega]],
           p2 -> equal[natmod[natmul[x, y], z], natmod[natmul[x, natmod[y, z]], z]]}]

Out[13]= True == equal[natmod[natmul[x, y], z], natmod[natmul[x, natmod[y, z]], z]]

In[14]:= natmod[natmul[x_, natmod[y_, z_]], z_] := natmod[natmul[x, y], z]
```

---

eliminating some variables

Lemma.

```
In[15]:= Assoc[id[x], id[image[V, intersection[omega, set[y]]]], modulo[y]] // Reverse

Out[15]= composite[id[intersection[x, image[V, intersection[omega, set[y]]]]],
           modulo[y]] == composite[id[x], modulo[y]]

In[16]:= composite[id[intersection[x_, image[V, intersection[omega, set[y_]]]]],
           modulo[y_]] := composite[id[x], modulo[y]]
```

One variable is now removed using **reify**.

```
In[17]:= Map[VERTSECT,
           SubstTest[reify, z, natmod[natmul[y, f[z, x]], x], f → natmod]] // Reverse

Out[17]= composite[modulo[x], times[y], modulo[x]] == composite[modulo[x], times[y]]

In[18]:= composite[modulo[x_], times[y_], modulo[x_]] := composite[modulo[x], times[y]]
```

A second variable is removed, again using **reify**.

```
In[19]:= Map[rotate[inverse[#]] &, SubstTest[reify, y,
            composite[modulo[x], f[y], modulo[x]], f → times]] // Reverse

Out[19]= composite[modulo[x], NATMUL, cross[Id, modulo[x]]] ==
          composite[modulo[x], NATMUL]

In[20]:= composite[modulo[x_], NATMUL, cross[Id, modulo[x_]]] :=
          composite[modulo[x], NATMUL]
```

Since multiplication is commutative, this analogous result also holds:

```
In[21]:= Assoc[composite[modulo[x], NATMUL], cross[Id, modulo[x]], SWAP]

Out[21]= composite[modulo[x], NATMUL, cross[modulo[x], Id]] ==
          composite[modulo[x], NATMUL]
```

```
In[22]:= composite[modulo[x_], NATMUL, cross[modulo[x_], Id]] :=
          composite[modulo[x], NATMUL]
```

One further corollary:

```
In[23]:= Assoc[composite[modulo[x], NATMUL],
          cross[Id, modulo[x]], cross[modulo[x], Id]]

Out[23]= composite[modulo[x], NATMUL, cross[modulo[x], modulo[x]]] ==
          composite[modulo[x], NATMUL]

In[24]:= composite[modulo[x_], NATMUL, cross[modulo[x_], modulo[x_]]] :=
          composite[modulo[x], NATMUL]
```