Quaife's lemma M10

Johan G. F. Belinfante
2003 May 1

```
In[1]:= << goedel52.r56; << tools.m

        :Package Title: goedel52.r56         2003 April 30 at 6:25 a.m.

        It is now:  2003 May 2 at 8:42

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 April 1

        weightlimit = 40
```

■ introduction

Quaife develops number theory directly, without reference to set theory. All the variables in his theorems are implicitly assumed to refer to natural numbers. When number theory is developed within set theory, one can no longer simply assume that all variables are numbers. Simply adding a literal of the form **member[x,omega]** for every variable **x** is not a satisfactory solution. Doing so would produce unwieldy clauses with large numbers of unnecessary literals, slowing down the theorem prover. In this notebook a particular instance is investigated with a large number of variables, Quaife's lemma **M10**. The **GOEDEL** program automatically deduces that only one of these variables needs to be explicitly declared to be a number. In the **GOEDEL** program, the sum of natural numbers **x** and **y** is denoted **natadd[x,y]**. If either **x** or **y** is not a natural number then **natadd[x,y]** is equal to the class **V**. The same holds for the difference **natsub[x,y]** and the product **natmul[x,y]**. Many statements involving these constructors do not need to be specialized to natural numbers. For instance, the commutative law for addition can be stated simply as:

```
In[2]:= equal[natadd[x, y], natadd[y, x]]

Out[2]= True
```

Quaife states that his lemma **M10** is needed for his theorem **LD19** about linear differences. This lemma involves seven variables, but the **GOEDEL** program reveals that only one of these seven variables, namely the variable that Quaife calls **x2**, needs to be restricted to natural numbers. The large number of variables in this lemma does raise the spectre that too many possibilities for unifications will occur; an automated theorem prover could be significantly slowed down by the mere presence of such lemmas on the usable list. It would therefore be extremely gratifying if one could show that existing rules in the **GOEDEL** program suffice without requiring the addition of this lemma. Anticipating that this may well be the case, the results derived in this notebook will not be added as new permanent rewrite rules for the time being.

```
Reference: Art Quaife, "Automated Development of Fundamental Mathematical Theories,"
 Kluwer Academic Publishers, 1992.
Lemma M10 is on page 187.
```

■ Quaife's Lemma M10

The statement of Quaife's lemma **M10** without any extra literals about membership in the set **omega** of natural numbers is:

```
In[3]:= implies[equal[natsub[natmul[v2, y2], natmul[u, x2]], singleton[0]],
            equal[natsub[natmul[u, x1], natmul[v1, y1]],
                natsub[natmul[u, natadd[x1, natmul[v1, x2, y1]]], natmul[v1, v2, y1, y2]]]]

Out[3]= or[equal[natsub[natadd[natmul[u, x1], natmul[u, v1, x2, y1]], natmul[v1, v2, y1, y2]],
           natsub[natmul[u, x1], natmul[v1, y1]]],
          not[equal[natmul[v2, y2], succ[natmul[u, x2]]]],
          not[member[v2, omega]], not[member[y2, omega]]]
```

Quaife assumes all the variables to be implicitly restricted to natural numbers, but this is not necessary. The discovery about which membership literals are actually needed was originally made by formally eliminating the equality hypothesis in the lemma via the following replacement (this takes quite a while):

```
In[4]:= % /. natmul[v2, y2] -> succ[natmul[u, x2]]

Out[4]= or[member[x2, omega], not[member[u, omega]], not[member[v1, omega]],
          not[member[v2, omega]], not[member[x1, omega]], not[member[y1, omega]],
          not[member[y2, omega]], not[subclass[natmul[v1, y1], natmul[u, x1]]]]
```

Note that this simplified statement becomes true when one adds one membership hypothesis:

```
In[5]:= % /. member[x2, omega] -> True

Out[5]= True
```

The above discovery does not constitute a derivation of Quaife's lemma because the **GOEDEL** program does not provide a direct bridge back to Quaife's lemma, restoring the eliminated equality hypothesis. The following version of Quaife's lemma, with one added membership literal is verified in the remainder of this notebook:

```
implies[
 and[member[x2, omega], equal[natsub[natmul[v2, y2], natmul[u, x2]], singleton[0]]],
    equal[natsub[natmul[u, x1], natmul[v1, y1]],
        natsub[natmul[u, natadd[x1, natmul[v1, x2, y1]]], natmul[v1, v2, y1, y2]]]]

or[equal[natsub[natadd[natmul[u, x1], natmul[u, v1, x2, y1]], natmul[v1, v2, y1, y2]],
  natsub[natmul[u, x1], natmul[v1, y1]]],
 not[equal[natmul[v2, y2], succ[natmul[u, x2]]]], not[member[v2, omega]],
 not[member[x2, omega]], not[member[y2, omega]]]
```

One drawback of the current version of the **GOEDEL** program is its rudimentary support for equality substitutions. These substitutions currently need to be carried out in great detail. Moreover, one must constantly dodge rewrite rules in the process. This weakness of the **GOEDEL** program makes the proofs below look more complex than they actually are. **Otter** does not suffer from this defect, so it is to be expected that **Otter** will produce simpler proofs. The primary role of the **GOEDEL** program is just to discover good definitions and and lean statements of theorems.

■ restoring the equality hypothesis for a simpler lemma

Note that the following three−variablestatement generalizes the statement produced upon eliminating the equality hypothesis from Quaife's lemma:

```
In[6]:= implies[member[y, omega],
         equal[natsub[natadd[x, natmul[y, z]], natmul[succ[y], z]], natsub[x, z]]]

Out[6]= True
```

The following four−variablestatement resembles Quaife's lemma with the equality hypothesis:

```
In[7]:= implies[and[member[y, omega], equal[w, succ[y]]],
          equal[natsub[natadd[x, natmul[y, z]], natmul[w, z]], natsub[x, z]]]

Out[7]= or[equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
          not[equal[w, succ[y]]], not[member[w, omega]]]
```

We proceed to derive this simplified lemma, and from it will derive Quaife's lemma itself. The starting point is this observation:

```
In[8]:= Map[implies[equal[w, succ[y]], #] &,
          SubstTest[equal, natmul[x, z], natmul[w, z], x -> succ[y]]]

Out[8]= or[equal[natadd[z, natmul[y, z]], natmul[w, z]], not[equal[w, succ[y]]]] == True

In[9]:= or[equal[natadd[z_, natmul[y_, z_]], natmul[w_, z_]], not[equal[w_, succ[y_]]]] := True
```

The transitivity of equality has this consequence:

```
In[10]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
             {u -> natsub[x, z],
            v -> union[complement[image[V, intersection[omega, singleton[y]]]], natsub[x, z]],
                  w -> natsub[natadd[x, natmul[y, z]], natmul[w, z]]}]

Out[10]= or[and[member[x, omega], member[z, omega], subclass[z, x]],
           and[member[w, omega], member[x, omega], member[y, omega],
            member[z, omega], subclass[natmul[w, z], natadd[x, natmul[y, z]]]],
           equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]] == True

In[11]:= or[and[member[x_, omega], member[z_, omega], subclass[z_, x_]],
           and[member[w_, omega], member[x_, omega], member[y_, omega],
            member[z_, omega], subclass[natmul[w_, z_], natadd[x_, natmul[y_, z_]]]],
           equal[natsub[x_, z_], natsub[natadd[x_, natmul[y_, z_]], natmul[w_, z_]]] := True
```

Equality substitution in **natsub** produces:

```
In[12]:= SubstTest[implies, equal[u, v], equal[natsub[t, u], natsub[t, v]],
             {t -> natadd[x, natmul[y, z]], u -> natmul[succ[y], z], v -> natmul[w, z]}]

Out[12]= or[and[not[member[w, omega]], not[subclass[z, x]]],
           and[not[subclass[z, x]], not[subclass[natmul[w, z], natadd[x, natmul[y, z]]]]],
           equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
           not[equal[natadd[z, natmul[y, z]], natmul[w, z]]], not[member[x, omega]],
           not[member[y, omega]], not[member[z, omega]]] == True

In[13]:= or[and[not[member[w_, omega]], not[subclass[z_, x_]]], and[not[subclass[z_, x_]],
            not[subclass[natmul[w_, z_], natadd[x_, natmul[y_, z_]]]]],
           equal[natsub[x_, z_], natsub[natadd[x_, natmul[y_, z_]], natmul[w_, z_]]],
           not[equal[natadd[z_, natmul[y_, z_]], natmul[w_, z_]]],
           not[member[x_, omega]], not[member[y_, omega]], not[member[z_, omega]]] := True
```

Some reasoning is needed:

```
In[14]:= Map[not, SubstTest[and, implies[p2, p4], implies[and[p4, p5], p6],
           not[implies[and[p2, p5], p6]], {p1 -> equal[w, succ[y]], p2 -> member[y, omega],
                 p4 -> equal[natsub[natadd[x, natmul[y, z]],
                natadd[z, natmul[y, z]]], natsub[x, z]],
                 p5 -> equal[natsub[natadd[x, natmul[y, z]], natadd[z, natmul[y, z]]],
                         natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
                 p6 -> equal[natsub[natadd[x, natmul[y, z]], natmul[w, z]], natsub[x, z]]}]]

Out[14]= or[and[member[x, omega], member[z, omega], subclass[z, x]], and[member[w, omega],
            member[x, omega], member[z, omega], subclass[natmul[w, z], natadd[x, natmul[y, z]]]],
           equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
           not[member[y, omega]]] == True
```

```
In[15]:= or[and[member[x_, omega], member[z_, omega], subclass[z_, x_]],
            and[member[w_, omega], member[x_, omega], member[z_, omega],
             subclass[natmul[w_, z_], natadd[x_, natmul[y_, z_]]]],
            equal[natsub[x_, z_], natsub[natadd[x_, natmul[y_, z_]], natmul[w_, z_]]],
            not[member[y_, omega]]] := True
```

The final step in the proof of the simplified lemma requires some additional reasoning:

```
In[16]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p2, p5], p6],
              implies[p3, p5],
                  not[implies[and[p1, p2], p6]],
              {p1 -> equal[w, succ[y]], p2 -> member[y, omega],
               p3 -> equal[natadd[z, natmul[y, z]], natmul[w, z]],
                  p5 -> equal[natsub[natadd[x, natmul[y, z]], natadd[z, natmul[y, z]]],
                          natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
                  p6 -> equal[natsub[natadd[x, natmul[y, z]], natmul[w, z]], natsub[x, z]]}]]

Out[16]= or[equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
            not[equal[w, succ[y]]], not[member[w, omega]]] == True

In[17]:= or[equal[natsub[x_, z_], natsub[natadd[x_, natmul[y_, z_]], natmul[w_, z_]]],
            not[equal[w_, succ[y_]]], not[member[w_, omega]]] := True
```

■ Derivation of Quaife's lemma M10

To derive Quaife's lemma **M10** from the simplified lemma, one just needs one more step:

```
In[18]:= SubstTest[implies, and[equal[w, succ[y]], member[w, omega]],
              equal[natsub[x, z], natsub[natadd[x, natmul[y, z]], natmul[w, z]]],
              {w -> natmul[v2, y2], y -> natmul[u, x2], x -> natmul[u, x1], z -> natmul[v1, y1]}]

Out[18]= or[equal[natsub[natadd[natmul[u, x1], natmul[u, v1, x2, y1]], natmul[v1, v2, y1, y2]],
             natsub[natmul[u, x1], natmul[v1, y1]]],
            not[equal[natmul[v2, y2], succ[natmul[u, x2]]]],
            not[member[v2, omega]], not[member[y2, omega]]] == True

In[19]:= or[equal[natsub[natadd[natmul[u_, x1_], natmul[u_, v1_, x2_, y1_]],
              natmul[v1_, v2_, y1_, y2_]], natsub[natmul[u_, x1_], natmul[v1_, y1_]]],
            not[equal[natmul[v2_, y2_], succ[natmul[u_, x2_]]]],
            not[member[v2_, omega]], not[member[y2_, omega]]] := True
```

This verifies our interpretation of Quaife's lemma **M10** with one explicit membership literal:

```
In[20]:= implies[
           and[member[x2, omega], equal[natsub[natmul[v2, y2], natmul[u, x2]], singleton[0]]],
              equal[natsub[natmul[u, x1], natmul[v1, y1]],
                  natsub[natmul[u, natadd[x1, natmul[v1, x2, y1]]], natmul[v1, v2, y1, y2]]]]

Out[20]= True
```

■ comments

It is not clear at this juncture whether one actually needs Quaife's lemma **M10** or not for the intended application to linear differences. That will require some additional work. The main point of this notebook is merely to illustrate the role of the **GOEDEL** program in discovering how such lemmas with numerous variables can be formulated in a set−theoreticcontext with the addition of only a minimal number of extra membership hypotheses.