Quaife's DF group

Johan G. F. Belinfante and Claudia Huang
2005 June 17

```
In[1]:= SetDirectory["i:"]; << goedel70.08a; << tools.m

        :Package Title: goedel70.08a       2005 June 8 at 9:05 a.m.

        It is now:  2005 Jun 17 at 20:21

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 June 17

        weightlimit = 40
```

---

summary

The purpose of this notebook is to derive new rewrite rules that enable the **GOEDEL** program to recognize the truth of the theorems in Quaife's **DF** group about floored subtraction of natural numbers.

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

In the **GOEDEL** program, the arithmetic of natural numbers is developed within Gödel's class theory, so variables are not automatically assumed to refer to natural numbers. To specialize to natural numbers, all variables will simply be wrapped with **nat**. Doing so eliminates the need for numerous literals for the form **member[x, omega]** and makes it possible to derive rewrite rules closely resembling Quaife's demodulators. Combining the **GOEDEL** program's **natsub** constructor with the **nat** wrapper yields a constructor **monus** that behaves like Quaife's floored subtraction.

```
In[3]:= monus[x_, y_] := nat[natsub[x, y]]
```

This constructor yields **0** when one attempts to subtract a larger number from a smaller one:

```
In[4]:= SubstTest[equal, 0, intersection[y, image[V, intersection[omega, set[z]]]],
          y → z] /. z → natsub[nat[x], nat[y]]

Out[4]= equal[0, nat[natsub[nat[x], nat[y]]]] == not[member[nat[y], nat[x]]]
```

---

a more general rule

One can generalize the rule for **monus** equal to **0** by omitting the **nat** wrappers:

```
In[5]:= SubstTest[equal, 0, intersection[y, image[V, intersection[omega, set[z]]]],
          y → z] /. z → natsub[x, y]

Out[5]= equal[0, nat[natsub[x, y]]] == or[equal[x, y],
          not[member[x, omega]], not[member[y, omega]], not[subclass[y, x]]]

In[6]:= equal[0, nat[natsub[x_, y_]]] := or[equal[x, y],
          not[member[x, omega]], not[member[y, omega]], not[subclass[y, x]]]
```

---

replacing subclass with member

In the arithmetic of natural numbers within Gödel's class theory, the weak inequality **x ≤ y** is translated as **subclass[x,y]**, and the strict inequality **x < y** is translated as **member[x, y]**. Because of the law of dichotomy for natural numbers, Quaife has chosen to always eliminate weak inequalities in favor of negated strict inequalities. A similar rewrite rule is available in the **GOEDEL** program when **nat** wrappers are used:

```
In[7]:= subclass[nat[x], nat[y]]

Out[7]= not[member[nat[y], nat[x]]]
```

When explicit **nat** wrappers are not present, this conversion from a **subclass** literal to a negated **member** literal is still valid when the variables refer to natural numbers, and such conversions could of course be made automatic by adding a conditional rewrite rule. Unfortunately, the presence of such a conditional rule would likely slow down the **GOEDEL** program because membership in the set of natural numbers would then have to be tested twice every time the predicate **subclass** occurs in any statement even for statements having nothing to do with arithmetic. The alternative adopted here is to just add special rules as needed. For example, since the sum of two natural numbers yields a natural number, the following rules can be derived:

```
In[8]:= SubstTest[subclass, nat[w], nat[z], w -> natadd[nat[x], nat[y]]]

Out[8]= subclass[natadd[nat[x], nat[y]], nat[z]] ==
         not[member[nat[z], natadd[nat[x], nat[y]]]]

In[9]:= subclass[natadd[nat[x_], nat[y_]], nat[z_]] :=
         not[member[nat[z], natadd[nat[x], nat[y]]]]

In[10]:= SubstTest[subclass, nat[x], nat[w], w -> natadd[nat[y], nat[z]]]

Out[10]= subclass[nat[x], natadd[nat[y], nat[z]]] ==
          not[member[natadd[nat[y], nat[z]], nat[x]]]

In[11]:= subclass[nat[x_], natadd[nat[y_], nat[z_]]] :=
          not[member[natadd[nat[y], nat[z]], nat[x]]]
```

A similar argument yields a rewrite rule needed for Quaife's clause **(DF5)**, but in this case, oriented in the opposite direction.

```
In[12]:= Map[not, SubstTest[subclass, nat[z], nat[x], z -> natadd[nat[x], nat[y]]]] //
          Reverse

Out[12]= member[nat[x], natadd[nat[x], nat[y]]] == not[equal[0, nat[y]]]

In[13]:= member[nat[x_], natadd[nat[x_], nat[y_]]] := not[equal[0, nat[y]]]
```

---

subclass rules for natsub

In this section, rules for eliminating **subclass** are derived for statements involving subtraction of natural numbers. Unlike addition, subtraction does not always yield a natural number. The following temporary lemma concerns this possibility:

```
In[14]:= SubstTest[implies, equal[v, V], subclass[u, v],
          {u -> nat[x], v -> natsub[nat[y], nat[z]]}]

Out[14]= or[not[member[nat[y], nat[z]]],
           subclass[nat[x], natsub[nat[y], nat[z]]]] == True

In[15]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

A second temporary lemma is used to eliminate a redundant literal:

```
In[16]:= equiv[or[member[nat[y], nat[z]], subclass[nat[x], natsub[nat[y], nat[z]]]],
          subclass[nat[x], natsub[nat[y], nat[z]]]]

Out[16]= True
```

```
In[17]:= or[member[nat[y_], nat[z_]], subclass[nat[x_], natsub[nat[y_], nat[z_]]]] :=
          subclass[nat[x], natsub[nat[y], nat[z]]]
```

A cancellation rule for **natadd** now yields a rewrite rule for eliminating **subclass** when its second argument is **natsub**.

```
In[18]:= SubstTest[subclass, natadd[nat[x], nat[z]],
           natadd[w, nat[z]], w → natsub[nat[y], nat[z]]] // Reverse

Out[18]= subclass[nat[x], natsub[nat[y], nat[z]]] ==
          or[member[nat[y], nat[z]], not[member[nat[y], natadd[nat[x], nat[z]]]]]

In[19]:= subclass[nat[x_], natsub[nat[y_], nat[z_]]] :=
          or[member[nat[y], nat[z]], not[member[nat[y], natadd[nat[x], nat[z]]]]]
```

A similar rule can be derived when **natsub** is the first argument. Since a subclass of a set is a set, one has:

```
In[20]:= SubstTest[implies, and[subclass[u, v], member[v, V]], member[u, V],
          {u → natsub[nat[x], nat[y]], v → nat[z]}]

Out[20]= or[not[member[nat[x], nat[y]]],
           not[subclass[natsub[nat[x], nat[y]], nat[z]]] == True

In[21]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

In the opposite direction, one has:

```
In[22]:= SubstTest[implies,
          and[member[u, omega], member[v, omega], not[subclass[u, v]]],
          member[v, u], {u -> natsub[nat[x], nat[y]], v → nat[z]}]

Out[22]= or[member[nat[x], nat[y]], member[nat[z], natsub[nat[x], nat[y]]],
           subclass[natsub[nat[x], nat[y]], nat[z]]] == True

In[23]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Combining these statements yields:

```
In[24]:= equiv[subclass[natsub[nat[x], nat[y]], nat[z]],
            and[not[member[nat[x], nat[y]]],
             not[member[nat[z], natsub[nat[x], nat[y]]]]]] // not // not

Out[24]= True
```

This is the desired rewrite rule for eliminating **subclass** when **natsub** occurs as its first argument.

```
In[25]:= subclass[natsub[nat[x_], nat[y_]], nat[z_]] :=
          and[not[member[nat[x], nat[y]]], not[member[nat[z], natsub[nat[x], nat[y]]]]]
```

image[V, -] lemmas

For each statement in Gödel's class theory one can derive a logically equivalent equation involving classes of the form **image[V, z]**. In particular, one can derive such an equation corresponding to the fact that **subclass[nat[x], nat[y]]** can be rewritten as **not[member[nat[y], nat[x]]]**, and this equation can be made into a rewrite rule.

```
In[26]:= (image[V, intersection[complement[nat[x]], nat[z]]] // Normality) /. z → nat[y]

Out[26]= image[V, intersection[complement[nat[x]], nat[y]]] ==
          image[V, intersection[nat[y], set[nat[x]]]]

In[27]:= image[V, intersection[complement[nat[x_]], nat[y_]]] :=
          image[V, intersection[nat[y], set[nat[x]]]]
```

Similarly, the following rule corresponds to the statement that **natsub[x,y]** is **0** only when **x** and **y** are equal natural numbers:

```
In[28]:= image[V, natsub[x, y]] // Normality

Out[28]= image[V, natsub[x, y]] ==
          union[complement[image[V, intersection[omega, set[x]]]],
           image[V, intersection[x, complement[y]]],
           image[V, intersection[y, complement[x]]]]

In[29]:= image[V, natsub[x_, y_]] :=
          union[complement[image[V, intersection[omega, set[x]]]],
           image[V, intersection[x, complement[y]]],
           image[V, intersection[y, complement[x]]]]
```

monus[nat[x],nat[y]]

The following formula for **nat[natsub[nat[x], nat[y]]]** holds:

```
In[30]:= equal[intersection[complement[image[V, intersection[nat[y], set[nat[x]]]]],
           natsub[nat[x], nat[y]]], nat[natsub[nat[x], nat[y]]]]

Out[30]= True
```

The corresponding rewrite rule is therefore justified:

```
In[31]:= intersection[complement[image[V, intersection[nat[y_], set[nat[x_]]]]],
           natsub[nat[x_], nat[y_]]] := nat[natsub[nat[x], nat[y]]]
```

A rewrite rule in the opposite direction also holds:

```
In[32]:= equal[union[nat[natsub[nat[x], nat[y]]],
           image[V, intersection[nat[y], set[nat[x]]]]], natsub[nat[x], nat[y]]]

Out[32]= True
```

This is also made into a rewrite rule.

```
In[33]:= union[image[V, intersection[nat[y_], set[nat[x_]]]],
           nat[natsub[nat[x_], nat[y_]]]] := natsub[nat[x], nat[y]]
```

---

rules for the nat wrapper

Here are two rewrite rules that allow intersections with **image[V, y]** or its complement to be pulled outside when **nat** is applied.

```
In[34]:= nat[intersection[x, image[V, y]]] // Normality

Out[34]= nat[intersection[x, image[V, y]]] == intersection[image[V, y], nat[x]]

In[35]:= nat[intersection[x_, image[V, y_]]] := intersection[image[V, y], nat[x]]

In[36]:= nat[intersection[x, complement[image[V, y]]]] // Normality

Out[36]= nat[intersection[x, complement[image[V, y]]]] ==
          intersection[complement[image[V, y]], nat[x]]

In[37]:= nat[intersection[x_, complement[image[V, y_]]]] :=
          intersection[complement[image[V, y]], nat[x]]
```

Similar rules hold for unions with **image[V, y]** or its complement.

```
In[38]:= nat[union[x, image[V, y]]] // Normality

Out[38]= nat[union[x, image[V, y]]] == intersection[complement[image[V, y]], nat[x]]

In[39]:= nat[union[x_, image[V, y_]]] := intersection[complement[image[V, y]], nat[x]]

In[40]:= nat[union[x, complement[image[V, y]]]] // Normality

Out[40]= nat[union[x, complement[image[V, y]]]] == intersection[image[V, y], nat[x]]

In[41]:= nat[union[x_, complement[image[V, y_]]]] := intersection[image[V, y], nat[x]]
```

For the third clause of **(DF5)** one needs this special case:

```
In[42]:= nat[image[V, x]] // Normality

Out[42]= nat[image[V, x]] == 0

In[43]:= nat[image[V, x_]] := 0

In[44]:= nat[complement[image[V, x]]] // Normality

Out[44]= nat[complement[image[V, x]]] == 0

In[45]:= nat[complement[image[V, x_]]] := 0
```

---

application to Quaife's clause (DF6).

Lemma:

```
In[46]:= Map[not, SubstTest[subclass, u,
           intersection[v, w], {u → nat[z], v → natsub[nat[x], nat[y]],
            w → complement[image[V, intersection[nat[y], set[nat[x]]]]]}]]

Out[46]= member[nat[natsub[nat[x], nat[y]]], nat[z]] ==
          or[and[member[nat[x], nat[y]], not[equal[0, nat[z]]]],
           and[member[nat[x], natadd[nat[y], nat[z]]], not[member[nat[x], nat[y]]]]]

In[47]:= member[nat[natsub[nat[x_], nat[y_]]], nat[z_]] :=
          or[and[member[nat[x], nat[y]], not[equal[0, nat[z]]]],
           and[member[nat[x], natadd[nat[y], nat[z]]], not[member[nat[x], nat[y]]]]]
```

Quaife's clause **(DF6)** follows as a corollary.

```
In[48]:= Map[equal[nat[natsub[nat[natsub[nat[x], nat[y]]], nat[z]]], nat[#]] &,
          SubstTest[natsub, union[u, image[V, v]], nat[z],
           {u → nat[natsub[nat[x], nat[y]]], v -> intersection[nat[y], set[nat[x]]]}]]

Out[48]= equal[nat[natsub[nat[x], natadd[nat[y], nat[z]]]],
           nat[natsub[nat[natsub[nat[x], nat[y]]], nat[z]]]] == True
```

This is made into a rewrite rule:

```
In[49]:= nat[natsub[nat[natsub[nat[x_], nat[y_]]], nat[z_]]] :=
          nat[natsub[nat[x], natadd[nat[y], nat[z]]]]
```

monus inside natadd

The following lemma holds because for natural numbers, if **x < y** is true, then **y < x** is false.

```
In[50]:= equal[intersection[complement[image[V, intersection[nat[x], set[nat[y]]]]],
            image[V, intersection[nat[y], set[nat[x]]]],
           image[V, intersection[nat[y], set[nat[x]]]]

Out[50]= True

In[51]:= intersection[complement[image[V, intersection[nat[x_], set[nat[y_]]]]],
            image[V, intersection[nat[y_], set[nat[x_]]]] :=
           image[V, intersection[nat[y], set[nat[x]]]]
```

Lemma.

```
In[52]:= equal[intersection[x, natadd[x, y]], x]

Out[52]= True

In[53]:= intersection[x_, natadd[x_, y_]] := x
```

The following rewrite rule applies when **monus** is substituted into **natadd**.

```
In[54]:= Map[complement, Map[complement[APPLY[composite[NATADD, #], nat[x]]] &,
             RIGHT[intersection[u, complement[image[V, v]]]] // VSNormality]] /.
           {u → natsub[nat[y], nat[z]], v -> intersection[nat[z], set[nat[y]]]}

Out[54]= natadd[nat[x], nat[natsub[nat[y], nat[z]]]] ==
           union[intersection[image[V, intersection[nat[y], set[nat[z]]]],
             natsub[natadd[nat[x], nat[y]], nat[z]]], nat[x]]

In[55]:= natadd[nat[x_], nat[natsub[nat[y_], nat[z_]]]] :=
           union[intersection[image[V, intersection[nat[y], set[nat[z]]]],
             natsub[natadd[nat[x], nat[y]], nat[z]]], nat[x]]
```

a result needed for Quaife's (DF4)

Lemma.

```
In[56]:= SubstTest[natadd, nat[y], monus[nat[x], nat[y]], y → set[0]]

Out[56]= succ[nat[natsub[nat[x], set[0]]]] == union[
             intersection[image[V, intersection[nat[x], set[set[0]]]], nat[x]], set[0]]
```

```
In[57]:= succ[nat[natsub[nat[x_], set[0]]]] := union[
           intersection[image[V, intersection[nat[x], set[set[0]]]], nat[x]], set[0]]
```

Lemma.

```
In[58]:= Map[or[#, member[set[0], nat[x]]] &,
          SubstTest[subclass, nat[x], nat[z], z → set[0]]]

Out[58]= or[member[set[0], nat[x]], subclass[nat[x], set[0]]] == True

In[59]:= or[member[set[0], nat[x_]], subclass[nat[x_], set[0]]] := True
```

Theorem.

```
In[60]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 → not[member[set[0], nat[x]]], p2 -> subclass[nat[x], set[0]],
            p3 -> or[equal[0, nat[x]], equal[nat[x], set[0]]]}]]

Out[60]= or[equal[0, nat[x]], equal[nat[x], set[0]], member[set[0], nat[x]]] == True

In[61]:= or[equal[0, nat[x_]], equal[nat[x_], set[0]], member[set[0], nat[x_]]] := True
```

---

verifying Quaife's clauses

In this final section it is verified that the rewrite rules derived above suffice for all of Quaife's clauses of his **DF** group to be recognized as true by the **GOEDEL** program, using **nat** wrappers on all variables, and interpreting floored subtraction as **monus**. Quaife comments that the first clause of **(DF1)** is subsumed by **(DF2)**.

```
In[62]:= implies[equal[natadd[nat[y], nat[u]], nat[x]],
          equal[natadd[nat[y], monus[nat[x], nat[y]]], nat[x]]]

Out[62]= True
```

The second clause of **(DF1)** is:

```
In[63]:= or[equal[monus[nat[x], nat[y]], 0],
          equal[natadd[nat[y], monus[nat[x], nat[y]]], nat[x]]]

Out[63]= True
```

The first clause of **(DF2)**:

```
In[64]:= implies[equal[natadd[nat[y], nat[z]], nat[x]],
          equal[monus[nat[x], nat[y]], nat[z]]]
```

```
Out[64]= True
```

The second clause of **(DF2)**, interpreting Quaife's **diff1** as **monus**:

```
In[65]:= or[equal[natadd[nat[y], monus[nat[x], nat[y]]], nat[x]],
          equal[monus[nat[x], nat[y]], 0]]
```

```
Out[65]= True
```

Quaife states four corollaries for **(DF2)**. The first corollary of **(DF2)**:

```
In[66]:= equal[monus[natadd[nat[y], nat[x]], nat[y]], nat[x]]
```

```
Out[66]= True
```

The second corollary of **(DF2)**:

```
In[67]:= equal[monus[natadd[nat[x], nat[y]], nat[y]], nat[x]]
```

```
Out[67]= True
```

The third corollary of **(DF2)**:

```
In[68]:= equal[monus[nat[x], nat[x]], 0]
```

```
Out[68]= True
```

The fourth corollary of **(DF2)**:

```
In[69]:= equal[monus[nat[x], 0], nat[x]]
```

```
Out[69]= True
```

First clause of **(DF3)**:

```
In[70]:= implies[equal[monus[nat[x], nat[y]], 0],
          equal[natadd[nat[x], monus[nat[y], nat[x]]], nat[y]]]
```

```
Out[70]= True
```

Second clause of **(DF3)**:

```
In[71]:= implies[equal[natadd[nat[x], monus[nat[y], nat[x]]], nat[y]],
          equal[monus[nat[x], nat[y]], 0]]
```

```
Out[71]= True
```

The clause **(DF4)**:

```
In[72]:= or[equal[nat[x], 0], equal[natadd[set[0], monus[nat[x], set[0]]], nat[x]]]
Out[72]= True
```

The first clause of **(DF5)**:

```
In[73]:= equal[monus[nat[x], natadd[nat[x], nat[y]]], 0]
Out[73]= True
```

Second clause of **(DF5)**:

```
In[74]:= equal[monus[nat[x], natadd[nat[y], nat[x]]], 0]
Out[74]= True
```

Third clause of **(DF5)**:

```
In[75]:= equal[monus[0, nat[x]], 0]
Out[75]= True
```

The clause **(DF6)**:

```
In[76]:= equal[monus[monus[nat[x], nat[y]], nat[z]],
          monus[nat[x], natadd[nat[y], nat[z]]]]
Out[76]= True
```

The clause **(DF7)**:

```
In[77]:= or[equal[natadd[nat[y], monus[nat[x], nat[y]]], nat[x]],
          equal[natadd[nat[x], monus[nat[y], nat[x]]], nat[y]]]
Out[77]= True
```

The first clause of **(DF8)**:

```
In[78]:= implies[and[equal[natadd[nat[y], monus[nat[x], nat[y]]], nat[x]], equal[
            natadd[nat[x], monus[nat[y], nat[x]]], nat[y]]], equal[nat[x], nat[y]]]
Out[78]= True
```

The second clause of **(DF8)**:

In[79]:=
```
implies[and[equal[natadd[nat[y], nat[z]], nat[x]],
  equal[natadd[nat[x], nat[u]], nat[y]]], equal[nat[x], nat[y]]]
```

Out[79]=
```
True
```