dichotomy

Johan G. F. Belinfante
2005 April 26

```
In[1]:= SetDirectory["i:"]; << goedel68.23a; << tools.m

        :Package Title: goedel68.23a         2005 April 23 at 1:00 p.m.

        It is now:  2005 Apr 26 at 12:24

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 April 16

        weightlimit = 40
```

summary

The dichotomy of ordering for natural numbers says:

```
In[2]:= implies[and[member[x, omega], member[y, omega]],
           equiv[member[x, y], not[subclass[y, x]]]] // not // not

Out[2]= True
```

A consequence of this dichotomy of ordering for natural numbers, one can rewrite statements about ordering of numbers to one of several canonical forms. Quaife's choice is to use dichotomy to eliminate **subclass** in favor of **member**. Another possibility is to do the reverse. Finally, there is the option to use dichotomy to eliminate negation. For the variable-free versions of most of such theorems, which involve the relations **E** and **S**, in the **GOEDEL** program, dichotomy has been used to eliminate complements of these relations. Eliminating member in favor of **subclass** has some appeal in that one can take advantage of the general order properties of **subclass**, which hold independently of arithmetic, to reduce the number of rewrite rules needed. A study of this option led to the discovery that having a rewrite rule that converts member to subclass causes looping for statements of the form **member[x, nat[y]]**. The problem is that the proposed rule would conflict with the wrapped membership rule for the **nat** wrapper.

```
In[3]:= Begin["Goedel`Private`"];
```

```
In[4]:= InfoMatch[class[w_, member[x_, HoldPattern[nat[y_]]]]]

Out[4]//TableForm=
       class[w_, member[x_, nat[y_]]] := class[w, and[member[x, y], member[y, omega]]
```

One could of course remove this wrapped membership rule and try replacing it with an **image[V,-]** type rule, but this unfortunately it was found that that does not prevent looping. No serious consideration was made for the third alternative, using dichotomy to eliminate **not,** because it was felt that doing so would make implicational reasoning unnatural. For these reasons, Quaife's choice will be followed in the **GOEDEL** program. This does have the advantage of making it easy to compare his versions of theorems about arithmetic proved using **Otter** with statements derived in the **GOEDEL** program. The unsuccessful attempt to eliminate **member** in favor of **subclass** was not a completely useless exercise, however, in that a number of interesting new rewrite rules were discovered along the way. In this notebook, these newly discovered rules are derived.

---

new replacement rules (with and without wrappers)

Rule 1.

```
In[5]:= and[member[x, omega], member[nat[y], x], not[subclass[nat[y], x]]] // NotNotTest

Out[5]= and[member[x, omega], member[nat[y], x], not[subclass[nat[y], x]]] == False

In[6]:= and[member[x_, omega],
          member[nat[y_], x_], not[subclass[nat[y_], x_]]] := False
```

Lemma.

```
In[7]:= SubstTest[implies, member[y, omega],
         equiv[subclass[y, U[y]], equal[0, y]], y → nat[x]]

Out[7]= or[equal[0, nat[x]], not[subclass[nat[x], U[nat[x]]]]] == True

In[8]:= (% /. x → x_) /. Equal → SetDelayed
```

Rule 2.

```
In[9]:= subclass[nat[x_], U[nat[x_]]] := equal[0, nat[x]]
```

Rule 3.

```
In[10]:= SubstTest[implies, and[member[x, y], subclass[y, z]],
          member[x, z], {y → omega, z → P[P[omega]]}]

Out[10]= or[not[member[x, omega]], subclass[U[x], omega]] == True

In[11]:= or[not[member[x_, omega]], subclass[U[x_], omega]] := True
```

Rule 4. (corollary of rule 3)

```
In[12]:= SubstTest[implies, member[y, omega], subclass[U[y], omega], y → nat[x]]

Out[12]= subclass[U[nat[x]], omega] == True

In[13]:= subclass[U[nat[x_]], omega] := True
```

Rule 5.

```
In[14]:= SubstTest[implies, equal[z, nat[y]],
          or[member[x, omega], not[member[x, U[z]]]], z → y]

Out[14]= or[member[x, omega], not[member[x, U[y]]], not[member[y, omega]]] == True

In[15]:= or[member[x_, omega], not[member[x_, U[y_]]], not[member[y_, omega]]] := True
```

---

some rules about subsets of natural numbers

Rule 1.

```
In[16]:= SubstTest[implies, and[subclass[x, z], member[z, omega]],
          member[x, image[inverse[S], omega]], z → nat[y]]

Out[16]= or[member[x, image[inverse[S], omega]], not[subclass[x, nat[y]]]] == True

In[17]:= or[member[x_, image[inverse[S], omega]], not[subclass[x_, nat[y_]]]] := True
```

Rule 2, obtained by introducing a variable into a known variable-free formula.

```
In[18]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
          {y -> image[inverse[S], omega], z -> image[inverse[BIGCUP], omega]}]

Out[18]= or[member[U[x], omega], not[member[x, image[inverse[S], omega]]]] == True

In[19]:= or[member[U[x_], omega], not[member[x_, image[inverse[S], omega]]]] := True
```

Rule 3, obtained by introducing a second variable into Rule 2.

```
In[20]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p3, p4],
           not[implies[and[p1, p2], p4]], {p1 → subclass[x, y], p2 → member[y, omega],
            p3 -> member[x, image[inverse[S], omega]], p4 → member[U[x], omega]}]]

Out[20]= or[member[U[x], omega], not[member[y, omega]], not[subclass[x, y]]] == True

In[21]:= or[member[U[x_], omega], not[member[y_, omega]], not[subclass[x_, y_]]] := True
```

Rule 4, obtained by replacing a literal with a wrapper.

```
In[22]:= SubstTest[implies, and[subclass[x, z], member[z, omega]],
          member[U[x], omega], z → nat[y]]

Out[22]= or[member[U[x], omega], not[subclass[x, nat[y]]]] == True

In[23]:= or[member[U[x_], omega], not[subclass[x_, nat[y_]]]] := True
```

---

rules relating predecessors and successors

Rule 1.

```
In[24]:= SubstTest[implies, subclass[u, v],
          subclass[U[u], U[v]], {u → nat[x], v → succ[nat[y]]}]

Out[24]= or[not[subclass[nat[x], succ[nat[y]]]], subclass[U[nat[x]], nat[y]]] == True

In[25]:= or[not[subclass[nat[x_], succ[nat[y_]]]],
           subclass[U[nat[x_]], nat[y_]]] := True
```

Temporary rule:

```
In[26]:= equal[
          composite[Id, complement[union[cart[union[complement[omega], set[0]], V],
             composite[Id, complement[inverse[S]]],
             composite[inverse[BIGCUP], complement[S]]]]], 0]

Out[26]= True

In[27]:= composite[intersection[composite[inverse[BIGCUP], S], inverse[S]],
           id[intersection[omega, complement[set[0]]]]] := 0
```

Corollary:

```
In[28]:= composite[id[intersection[omega, complement[set[0]]]],
           intersection[S, composite[inverse[S], BIGCUP]]] // DoubleInverse

Out[28]= composite[id[intersection[omega, complement[set[0]]]],
           intersection[S, composite[inverse[S], BIGCUP]]] == 0
```

```
In[29]:= % /. Equal → SetDelayed
```

Introducing variables yields:

```
In[30]:= Map[not, SubstTest[member, pair[x, y],
           composite[id[intersection[w, complement[set[0]]]],
            intersection[S, composite[inverse[S], BIGCUP]]], w → omega] // Reverse]

Out[30]= or[equal[0, y], not[member[x, V]], not[member[y, omega]],
          not[subclass[x, y]], not[subclass[y, U[x]]]] == True

In[31]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The above result has a superfluous literal that can be removed as follows:

```
In[32]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
          implies[and[p1, p2, p3, p4], p5], not[implies[and[p1, p2, p4], p5]],
          {p1 -> member[x, omega], p2 -> subclass[y, x],
           p3 → member[y, V], p4 → subclass[x, U[y]], p5 → equal[0, x]}]]

Out[32]= or[equal[0, x], not[member[x, omega]],
          not[subclass[x, U[y]]], not[subclass[y, x]]] == True

In[33]:= or[equal[0, x_], not[member[x_, omega]],
          not[subclass[x_, U[y_]]], not[subclass[y_, x_]]] := True
```

The number literal can be replaced with a wrapper.

```
In[34]:= SubstTest[implies, and[member[z, omega], subclass[z, U[y]], subclass[y, z]],
          equal[0, z], z → nat[x]]

Out[34]= or[equal[0, nat[x]],
          not[subclass[y, nat[x]]], not[subclass[nat[x], U[y]]]] == True

In[35]:= or[equal[0, nat[x_]], not[subclass[y_, nat[x_]]],
          not[subclass[nat[x_], U[y_]]]] := True
```

---

an induction proof

In this section, a proof by ordinary induction is presented, based on this class:

```
In[36]:= class[x, subclass[image[SUCC, U[x]], x]]

Out[36]= fix[composite[S, IMAGE[SUCC], BIGCUP]]
```

Lemma.

```
In[37]:= implies[and[member[x, omega], member[x, y]], member[succ[x], y]] /.
           y -> fix[composite[S, IMAGE[SUCC], BIGCUP]] // NotNotTest

Out[37]= or[and[subclass[image[SUCC, x], succ[x]],
            subclass[image[SUCC, U[x]], succ[x]]],
           not[member[x, omega]], not[subclass[image[SUCC, U[x]], x]]] == True

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

Lemma.

```
In[39]:= Map[equal[V, #] &, SubstTest[class, x,
            implies[and[member[x, w], member[x, y]], member[succ[x], y]],
            {w → omega, y -> fix[composite[S, IMAGE[SUCC], BIGCUP]]}]] // Reverse

Out[39]= subclass[
           intersection[omega, image[SUCC, fix[composite[S, IMAGE[SUCC], BIGCUP]]]],
           fix[composite[S, IMAGE[SUCC], BIGCUP]]] == True

In[40]:= % /. Equal → SetDelayed
```

Induction is used here.

```
In[41]:= SubstTest[implies, INDUCTIVE[x], subclass[omega, x],
           x -> intersection[omega, fix[composite[S, IMAGE[SUCC], BIGCUP]]]]

Out[41]= subclass[omega, fix[composite[S, IMAGE[SUCC], BIGCUP]]] == True

In[42]:= subclass[omega, fix[composite[S, IMAGE[SUCC], BIGCUP]]] := True
```

Introducing variables, one obtains:

```
In[43]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
           {y → omega, z -> fix[composite[S, IMAGE[SUCC], BIGCUP]]}]

Out[43]= or[not[member[x, omega]], subclass[image[SUCC, U[x]], x]] == True

In[44]:= or[not[member[x_, omega]], subclass[image[SUCC, U[x_]], x_]] := True
```

Replacing a number literal with a wrapper, one finds:

```
In[45]:= SubstTest[implies, member[y, omega],
           subclass[image[SUCC, U[y]], y], y → nat[x]]

Out[45]= subclass[image[SUCC, U[nat[x]]], nat[x]] == True

In[46]:= subclass[image[SUCC, U[nat[x_]]], nat[x_]] := True
```