transposing in equations and inequalities

Johan G. F. Belinfante
2005 June 24

```
In[1]:= SetDirectory["i:"]; << goedel70.17a; << tools.m
        :Package Title: goedel70.17a       2005 June 17 at 8:35 p.m.
        It is now:  2005 Jun 24 at 20:14
        Loading Simplification Rules
        TOOLS.M                       Revised 2005 June 17
        weightlimit = 40
```

summary

Rewrite rules using **nat** wrappers are derived to facilitate transposing in equations and inequalities involving addition and subtraction of natural numbers. Special efforts have been made to keep to a minimum the number of variables requiring **nat** wrappers.

transposing in equations

The following rules for transposing are currently automatic consquences of the equality substitution rule. These rules are only available when the **equality** flag has the default value **True**.

```
In[2]:= implies[and[equal[natsub[x, y], z], member[z, omega]], equal[x, natadd[y, z]]]
Out[2]= True
In[3]:= implies[and[equal[x, natadd[y, z]], member[x, omega]], equal[natsub[x, y], z]]
Out[3]= True
```

The membership literals in these rules can be replaced with **nat** wrappers. Only one wrapper is needed in each case.

```
In[4]:= SubstTest[implies, and[equal[natsub[x, y], w], member[w, omega]],
         equal[x, natadd[y, w]], w → nat[z]]

Out[4]= or[equal[x, natadd[y, nat[z]]], not[equal[nat[z], natsub[x, y]]]] == True

In[5]:= or[equal[x_, natadd[y_, nat[z_]]], not[equal[nat[z_], natsub[x_, y_]]]] := True

In[6]:= SubstTest[implies, and[equal[w, natadd[y, z]], member[w, omega]],
         equal[natsub[w, y], z], w → nat[x]]

Out[6]= or[equal[z, natsub[nat[x], y]], not[equal[nat[x], natadd[y, z]]]] == True

In[7]:= or[equal[z_, natsub[nat[x_], y_]], not[equal[nat[x_], natadd[y_, z_]]]] := True
```

Comment 1. Equality substitution also implies similar results in which different variables are wrapped, for example:

```
In[8]:= implies[equal[y, natadd[nat[x], nat[z]]], equal[nat[x], natsub[y, nat[z]]]]

Out[8]= True
```

Comment 2. One actually has the following logical equivalence, which conceivably could be made into a rewrite rule that automatically eliminates subtraction in favor of addition. Doing so could make reasoning about subtraction difficult. Note also that this rule requires **nat** wrappers on two variables.

```
In[9]:= equiv[equal[nat[x], natsub[nat[y], z]], equal[nat[y], natadd[nat[x], z]]]

Out[9]= True
```

If one were to add such a rewrite rule, it should be oriented to eliminate **natsub** in favor of **natadd**, not only because the latter is better behaved, but because addition is commutative, the equation **x + y = z** implies both **x = z - y** and **y = z - x**, and there would be no good way to decide whether to transpose **x** or **y**. The following double-transposition rule with **nat** wrappers on two variables is also valid:

```
In[10]:= Map[not,
          SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> equal[nat[z], natsub[nat[y], x]],
            p2 -> equal[nat[y], natadd[x, nat[z]]],
            p3 -> equal[x, natsub[nat[y], nat[z]]]}]]

Out[10]= or[equal[x, natsub[nat[y], nat[z]]],
           not[equal[nat[z], natsub[nat[y], x]]]] == True
```

transposing in inequalities

The rules in the next two sections govern transposition for inequalities of the form **x - y < z** and **x < y + z**. In principle, one could make the following result into a rewrite rule, but it is unclear how best to orient it. In one direction, one has the advantage of eliminating subtraction in favor of addition, which seems desirable, but in the other direction, one has the advantage of reducing two literals to one, which is also desirable.

```
In[11]:= SubstTest[member, natadd[u, w], natadd[v, w],
          {u → natsub[nat[x], nat[y]], v → nat[z], w → nat[y]}]

Out[11]= and[member[nat[x], natadd[nat[y], nat[z]]], not[member[nat[x], nat[y]]]] ==
          member[natsub[nat[x], nat[y]], nat[z]]
```

A conservative compromise is to use this result to derive clauses that allow one to transpose in either direction. This can be done by mapping the above result. This is done in the following two sections.

transposing from x - y < z to x < y + z

From the cancellation law one finds:

```
In[12]:= Map[implies[member[natsub[nat[x], nat[y]], nat[z]], #] &,
           SubstTest[member, natadd[u, w], natadd[v, w],
            {u → natsub[nat[x], nat[y]], v → nat[z], w → nat[y]}]] // MapNotNot

Out[12]= or[member[nat[x], natadd[nat[y], nat[z]]],
           not[member[natsub[nat[x], nat[y]], nat[z]]]] == True

In[13]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

This result contains two unnecessary **nat** wrappers that can be eliminated. The first step toward doing so is to replace these wrappers with corresponding numberhood literals:

```
In[14]:= SubstTest[implies,
          and[equal[u, nat[x]], equal[v, nat[y]], member[natsub[u, v], nat[z]]],
          member[u, natadd[v, nat[z]]], {u → x, v → y}]

Out[14]= or[member[x, natadd[y, nat[z]]], not[member[x, omega]],
           not[member[y, omega]], not[member[natsub[x, y], nat[z]]]] == True
```

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

The two numberhood literals can now be eliminated by some reasoning, yielding a transposition rule that has only one variable with a **nat** wrapper.

```
In[16]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p1, p2, p3], p4], not[implies[p1, p4]],
           {p1 -> member[natsub[x, y], nat[z]], p2 → member[x, omega],
            p3 → member[y, omega], p4 -> member[x, natadd[y, nat[z]]]}]]

Out[16]= or[member[x, natadd[y, nat[z]]], not[member[natsub[x, y], nat[z]]]] == True

In[17]:= or[member[x_, natadd[nat[z_], y_]],
           not[member[natsub[x_, y_], nat[z_]]]] := True
```

---

transposing from x < y + z to x - y < z

From a cancellation law, one readily obtains:

```
In[18]:= Map[implies[#, member[natsub[nat[x], nat[y]], nat[z]]] &,
          SubstTest[member, natadd[u, w], natadd[v, w],
           {u → natsub[nat[x], nat[y]], v → nat[z], w → nat[y]}]]

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

In[19]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The **nat** wrapper on the variable **x** can be replaced with the numberhood literal **member[x, omega]**.

```
In[20]:= SubstTest[implies, and[equal[w, nat[x]], member[w, natadd[nat[y], nat[z]]]],
          or[member[w, nat[y]], member[natsub[w, nat[y]], nat[z]]], w → x]

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

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

Lemma.

```
In[22]:= SubstTest[implies, and[member[x, w], member[w, omega]],
          member[x, omega], w -> natadd[nat[y], nat[z]]]

Out[22]= or[member[x, omega], not[member[x, natadd[nat[y], nat[z]]]]] == True

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

The numberhood literal can now be eliminated, yielding a transposition rule with **nat** wrappers on only two variables.

```
In[24]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], or[p3, p4]], not[implies[p1, or[p3, p4]]],
           {p1 -> member[x, natadd[nat[y], nat[z]]], p2 → member[x, omega],
            p3 -> member[x, nat[y]], p4 -> member[natsub[x, nat[y]], nat[z]]}]]

Out[24]= or[member[x, nat[y]], member[natsub[x, nat[y]], nat[z]],
           not[member[x, natadd[nat[y], nat[z]]]]] == True

In[25]:= or[member[natsub[x_, nat[y_]], nat[z_]], member[x_, nat[y_]],
           not[member[x_, natadd[nat[y_], nat[z_]]]]] := True
```

---

rules for reverse inequalities

The rules in the following two sections govern transposition for inequalities of the form **x - y > z** and **x > y + z**. These are the reverse of the inequalities considered in the preceding sections. The following lemma is needed:

```
In[26]:= SubstTest[implies, and[member[u, V], equal[V, v]],
          member[u, v], {u → nat[x], v → natsub[nat[y], nat[z]]}]

Out[26]= or[member[nat[x], natsub[nat[y], nat[z]]], not[member[nat[y], nat[z]]]] == True

In[27]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Temporary rule:

```
In[28]:= equiv[or[member[nat[x], natsub[nat[z], nat[y]]], member[nat[z], nat[y]]],
          member[nat[x], natsub[nat[z], nat[y]]]]

Out[28]= True

In[29]:= or[member[nat[x_], natsub[nat[z_], nat[y_]]], member[nat[z_], nat[y_]]] :=
          member[nat[x], natsub[nat[z], nat[y]]]
```

Just as before, one may ponder whether to make the following into a rewrite rule. Again, it is unclear how best to orient such a rule. In one direction, one has the advantage of eliminating subtraction in favor of addition, which seems desirable, but in the other direction, one has the advantage of reducing two literals to one, which is also desirable.

```
In[30]:= SubstTest[member, natadd[u, w], natadd[v, w],
          {u → nat[x], v → natsub[nat[y], nat[z]], w → nat[z]}]

Out[30]= or[member[nat[y], nat[z]], member[natadd[nat[x], nat[z]], nat[y]]] ==
          member[nat[x], natsub[nat[y], nat[z]]]
```

A conservative compromise is again to use this result to derive clauses that allow one to go in either direction by mapping the above result.

---

transposing from x > y + z to x - y > z

```
In[31]:= Map[implies[#, member[nat[y], natsub[nat[x], nat[z]]]] &,
           SubstTest[member, natadd[u, w], natadd[v, w],
            {u → nat[y], v → natsub[nat[x], nat[z]], w → nat[z]}]] // MapNotNot

Out[31]= or[member[nat[y], natsub[nat[x], nat[z]]],
           not[member[natadd[nat[y], nat[z]], nat[x]]]] == True

In[32]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Two **nat** wrappers are replaced with corresponding numberhood literals.

```
In[33]:= SubstTest[implies,
          and[equal[u, nat[y]], equal[v, nat[z]], member[natadd[u, v], nat[x]]],
          member[u, natsub[nat[x], v]], {u → y, v → z}]

Out[33]= or[member[y, natsub[nat[x], z]], not[member[y, omega]],
           not[member[z, omega]], not[member[natadd[y, z], nat[x]]]] == True

In[34]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The numberhood literals can be eliminated:

```
In[35]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p1, p2, p3], p4], not[implies[p1, p4]],
           {p1 -> member[natadd[y, z], nat[x]], p2 → member[y, omega],
            p3 → member[z, omega], p4 -> member[y, natsub[nat[x], z]]}]]

Out[35]= or[member[y, natsub[nat[x], z]], not[member[natadd[y, z], nat[x]]]] == True

In[36]:= or[member[y_, natsub[nat[x_], z_]],
           not[member[natadd[y_, z_], nat[x_]]]] := True
```

transposing from x - y > z to x > y + z

In the other direction, one has:

```
In[37]:= Map[implies[member[nat[x], natsub[nat[y], nat[z]]], #] &,
          SubstTest[member, natadd[u, w], natadd[v, w],
           {u → nat[x], v → natsub[nat[y], nat[z]], w → nat[z]}]]

Out[37]= or[member[nat[y], nat[z]], member[natadd[nat[x], nat[z]], nat[y]],
           not[member[nat[x], natsub[nat[y], nat[z]]]]] == True

In[38]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The **nat** wrapper on **x** is replaced with a numberhood literal.

```
In[39]:= SubstTest[implies, and[equal[w, nat[x]], member[w, natsub[nat[y], nat[z]]]],
          or[member[nat[y], nat[z]], member[natadd[w, nat[z]], nat[y]]], w → x]

Out[39]= or[member[nat[y], nat[z]], member[natadd[x, nat[z]], nat[y]],
           not[member[x, omega]], not[member[x, natsub[nat[y], nat[z]]]]] == True

In[40]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[41]:= SubstTest[implies, and[member[x, w], member[w, omega]],
          member[x, omega], w -> natsub[nat[y], nat[z]]]

Out[41]= or[member[x, omega], member[nat[y], nat[z]],
           not[member[x, natsub[nat[y], nat[z]]]]] == True

In[42]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

This numberhood literal can be eliminated:

```
In[43]:= Map[not, SubstTest[and, implies[p1, or[p2, p3]],
           implies[and[p1, p2], or[p3, p4]], not[implies[p1, or[p3, p4]]],
           {p1 -> member[x, natsub[nat[y], nat[z]]], p2 → member[x, omega],
            p3 → member[nat[y], nat[z]], p4 -> member[natadd[x, nat[z]], nat[y]]}]]

Out[43]= or[member[nat[y], nat[z]], member[natadd[x, nat[z]], nat[y]],
           not[member[x, natsub[nat[y], nat[z]]]]] == True

In[44]:= or[member[nat[y_], nat[z_]], member[natadd[nat[z_], x_], nat[y_]],
           not[member[x_, natsub[nat[y_], nat[z_]]]]] := True
```