an inequality for natexp

Johan G. F. Belinfante
2006 November 30

```
In[1]:= SetDirectory["l:"]; << goedel87.30a; << tools.m

        :Package Title: goedel87.30a          2006 November 30 at 1:05 p.m.

        It is now:  2006 Nov 30 at 13:52

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 November 22

        weightlimit = 40
```

summary

The binomial theorem says that $(1 + x)^n = 1 + n\,x + \cdots$. If all variables refer to natural numbers, the unwritten terms can only increase the sum on the right, and therefore $n\,x < (1 + x)^n$. This basic inequality for exponentiation of natural numbers is derived in this notebook directly using induction, without reference to the binomial theorem.

simplification rules

Lemma.

```
In[2]:= Assoc[NATEXP, id[cart[omega, omega]], id[cart[V, omega]]] // Reverse
Out[2]= composite[NATEXP, id[cart[V, omega]]] == NATEXP

In[3]:= composite[NATEXP, id[cart[V, omega]]] := NATEXP
```

Similarly:

```
In[4]:= Assoc[NATEXP, id[cart[omega, omega]], id[cart[omega, V]]] // Reverse
Out[4]= composite[NATEXP, id[cart[omega, V]]] == NATEXP

In[5]:= composite[NATEXP, id[cart[omega, V]]] := NATEXP
```

An application needed below is this result:

```
In[6]:= Map[image[#, y] &, SubstTest[fix, composite[v, id[u]],
          {u → cart[V, omega], v → composite[inverse[x], NATEXP]}]] // InvertFix

Out[6]= intersection[omega, image[fix[composite[inverse[NATEXP], x]], y]] ==
         image[fix[composite[inverse[NATEXP], x]], y]

In[7]:= intersection[omega, image[fix[composite[inverse[NATEXP], x_]], y_]] :=
         image[fix[composite[inverse[NATEXP], x]], y]
```

nonself-membership for natexp

```
In[8]:= SubstTest[or, not[member[z, omega]], not[member[z, z]], z → natexp[x, y]] // Reverse

Out[8]= or[not[member[x, omega]], not[member[y, omega]],
          not[member[natexp[x, y], natexp[x, y]]]] == True

In[9]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[10]:= SubstTest[implies, equal[z, V], not[member[z, z]], z → natexp[x, y]] // Reverse

Out[10]= or[and[member[x, omega], member[y, omega]],
           not[member[natexp[x, y], natexp[x, y]]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[12]:= Map[not, SubstTest[and, implies[p, q],
           or[p, q], {p -> and[member[x, omega], member[y, omega]],
            q -> not[member[natexp[x, y], natexp[x, y]]]}]]

Out[12]= member[natexp[x, y], natexp[x, y]] == False

In[13]:= member[natexp[x_, y_], natexp[x_, y_]] := False
```

a conditional rule for removing nat wrappers

```
In[14]:= implies[member[x, omega], equal[nat[U[x]], U[x]]]

Out[14]= True

In[15]:= nat[U[x_]] := U[x] /; member[x, omega]
```

an inequality for natexp: n x < $(1 + x)^n$

In the **GOEDEL** program, induction must be applied to an inductive class, not to a statement:

```
In[16]:= implies[INDUCTIVE[x], subclass[omega, x]]

Out[16]= True
```

The idea is to fix the base and use induction on the exponent. The class to which induction must be applied is:

```
In[17]:= class[y, member[natmul[z, y], natexp[succ[z], y]]] /. z → nat[x]

Out[17]= image[
          fix[composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]]], E, SECOND]],
          set[succ[nat[x]]]]
```

The following lemma suffices for the base case:

```
In[18]:= member[pair[succ[nat[x]], 0], fix[composite[inverse[NATEXP],
             inverse[IMAGE[inverse[times[nat[x]]]]], E, SECOND]]] // AssertTest

Out[18]= member[pair[succ[nat[x]], 0], fix[composite[inverse[NATEXP],
             inverse[IMAGE[inverse[times[nat[x]]]]], E, SECOND]]] == True

In[19]:= (% /. x → x_) /. Equal → SetDelayed
```

For the induction step, the following temporary lemma is needed to crack open the fixed point class.

```
In[20]:= member[pair[u, v], fix[composite[inverse[NATEXP],
             inverse[IMAGE[inverse[times[nat[x]]]]], E, SECOND]]] // AssertTest

Out[20]= member[pair[u, v], fix[
            composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]]], E, SECOND]]] ==
          and[member[u, omega], member[v, omega], member[natmul[v, nat[x]], natexp[u, v]]]

In[21]:= member[pair[u_, v_], fix[
            composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x_]]]]], E, SECOND]]] :=
          and[member[u, omega], member[v, omega], member[natmul[v, nat[x]], natexp[u, v]]]
```

Lemma.

```
In[22]:= (SubstTest[implies, and[subclass[nat[z], nat[s]], member[nat[s], nat[w]]],
             member[nat[z], nat[w]], s -> natadd[nat[t], nat[z]]] // Reverse) /.
          {w -> natmul[nat[x], nat[y]], t → natmul[nat[x], nat[z]]}

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

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

Lemma.

```
In[24]:= SubstTest[succ, U[nat[u]], u -> natexp[succ[nat[x]], nat[y]]] // Reverse

Out[24]= succ[U[natexp[succ[nat[x]], nat[y]]]] == natexp[succ[nat[x]], nat[y]]

In[25]:= succ[U[natexp[succ[nat[x_]], nat[y_]]]] := natexp[succ[nat[x]], nat[y]]
```

more

Lemma.

```
In[26]:= SubstTest[or, member[natadd[nat[x], natmul[nat[x], nat[y]]],
           natadd[succ[nat[t]], natmul[nat[x], succ[nat[t]]]]],
          not[member[natmul[nat[x], nat[y]], succ[nat[t]]]],
          t -> natexp[succ[nat[x]], nat[y]]] // Reverse

Out[26]= or[member[natexp[succ[nat[x]], nat[y]], natmul[nat[x], nat[y]]],
          not[member[natadd[natexp[succ[nat[x]], nat[y]],
             natmul[nat[x], natexp[succ[nat[x]], nat[y]]]], natmul[nat[x], nat[y]]]]] == True

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

Lemma.

```
In[28]:= SubstTest[implies, equal[t, succ[nat[s]]],
          or[member[natadd[nat[x], natmul[nat[x], nat[y]]], natadd[t, natmul[t, nat[x]]]],
           not[member[natmul[nat[x], nat[y]], t]]],
          {s → U[natexp[succ[nat[x]], nat[y]]], t -> natexp[succ[nat[x]], nat[y]]}] // Reverse

Out[28]= or[member[natadd[nat[x], natmul[nat[x], nat[y]]], natadd[
            natexp[succ[nat[x]], nat[y]], natmul[nat[x], natexp[succ[nat[x]], nat[y]]]],
          not[member[natmul[nat[x], nat[y]], natexp[succ[nat[x]], nat[y]]]] == True

In[29]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The **nat** wrapper on **y** needs to be removed prior to eliminating this variable.

```
In[30]:= SubstTest[implies, equal[y, nat[t]], or[member[natadd[nat[x], natmul[nat[x], y]],
            natadd[natexp[succ[nat[x]], y], natmul[nat[x], natexp[succ[nat[x]], y]]]],
           not[member[natmul[nat[x], y], natexp[succ[nat[x]], y]]], t → y] // Reverse

Out[30]= or[member[natadd[nat[x], natmul[y, nat[x]]],
           natadd[natexp[succ[nat[x]], y], natmul[nat[x], natexp[succ[nat[x]], y]]],
          not[member[y, omega]], not[member[natmul[y, nat[x]], natexp[succ[nat[x]], y]]] == True

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

The variable **y** can now be removed:

```
In[32]:= Map[equal[V, #] &,
          SubstTest[class, y, implies[and[member[y, omega], member[y, z]], member[succ[y], z]],
           z -> image[fix[composite[inverse[NATEXP],
               inverse[IMAGE[inverse[times[nat[x]]]], E, SECOND]], set[succ[nat[x]]]]]]

Out[32]= subclass[image[SUCC, image[fix[composite[inverse[NATEXP],
              inverse[IMAGE[inverse[times[nat[x]]]], E, SECOND]], set[succ[nat[x]]]], image[
            fix[composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]], E, SECOND]],
            set[succ[nat[x]]]] == True
```

```
In[33]:= (% /. x → x_) /. Equal → SetDelayed
```

Applying induction, one obtains:

```
In[34]:= SubstTest[implies, INDUCTIVE[z], subclass[omega, z],
           z -> image[fix[composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]]],
               E, SECOND]], set[succ[nat[x]]]]] // Reverse
```

```
Out[34]= subclass[omega, image[
           fix[composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]], E, SECOND]],
           set[succ[nat[x]]]]] == True
```

```
In[35]:= (% /. x → x_) /. Equal → SetDelayed
```

Restoring variables yields the result sought after:

```
In[36]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
           {u → nat[y], v → omega,
            w → image[fix[composite[inverse[NATEXP], inverse[IMAGE[inverse[times[nat[x]]]]],
                E, SECOND]], set[succ[nat[x]]]]}] // Reverse
```

```
Out[36]= member[natmul[nat[x], nat[y]], natexp[succ[nat[x]], nat[y]]] == True
```

```
In[37]:= member[natmul[nat[x_], nat[y_]], natexp[succ[nat[x_]], nat[y_]]] := True
```

The **nat** wrappers can be removed:

```
In[38]:= SubstTest[implies, and[equal[x, nat[s]], equal[y, nat[t]]],
           member[natmul[x, y], natexp[succ[x], y]], {s → x, t → y}] // Reverse
```

```
Out[38]= or[member[natmul[x, y], natexp[succ[x], y]],
           not[member[x, omega]], not[member[y, omega]]] == True
```

```
In[39]:= or[member[natmul[x_, y_], natexp[succ[x_], y_]],
           not[member[x_, omega]], not[member[y_, omega]]] := True
```

A variable-free formulation can be derived as follows:

```
In[40]:= fix[composite[inverse[E], IMAGE[inverse[NATMUL]], NATEXP, cross[SUCC, Id]]] //
          RelnNormality
```

```
Out[40]= fix[composite[inverse[E], IMAGE[inverse[NATMUL]], NATEXP, cross[SUCC, Id]]] ==
          cart[omega, omega]
```

```
In[41]:= fix[composite[inverse[E], IMAGE[inverse[NATMUL]], NATEXP, cross[SUCC, Id]]] :=
          cart[omega, omega]
```