# NATEXP, part 2

Johan G. F. Belinfante
2006 June 29

```
In[1]:= SetDirectory["l:"]; << goedel82.27a; << tools.m

        :Package Title: goedel82.27a         2006 June 27 at 5:50 p.m.

        It is now:  2006 Jun 29 at 2:24

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 June 27

        weightlimit = 40
```

## summary

In this second notebook of a series about exponentiation for natural number, three laws of exponents for **natexp[x,y]** are derived. In each case, the result is first derived with all three variables wrapped with **nat**. After this is done, the **nat** wrappers are removed by noting that when any variable fails to be a natural number, each law reduces to the equation **V = V**. An important tool used in the derivations of the first and third laws of exponents is the uniqueness theorem for **iterate**. For the first and second law of exponents, theorems about powers are also used. The derivation of the third law makes use of a corollary of the first law of exponents. The first and second laws are here oriented to expand. Each of these laws contains a duplicated variable on one side of the equation. The third law has just one occurrence of each variable on each side of the equation. Its orientation as a rewrite rule is dictated by the fact that multiplication is commutative.

## the first law of exponents: $x^{y+z} = x^y\, x^z$

The uniqueness theorem for **iterate** yields:

```
In[2]:= SubstTest[implies,
          and[equal[composite[u, w], composite[w, SUCC]], equal[image[w, set[0]], v]],
          equal[composite[w, id[omega]], iterate[u, v]],
          {u → times[nat[x]], v → set[nat[y]], w → composite[times[nat[y]], NATEXP, LEFT[nat[x]]]}]

Out[2]= equal[composite[times[nat[y]], NATEXP, LEFT[nat[x]]],
          iterate[times[nat[x]], set[nat[y]]]] == True

In[3]:= iterate[times[nat[x_]], set[nat[y_]]] := composite[times[nat[y]], NATEXP, LEFT[nat[x]]]
```

This rule is an extension of a similar rule for the special case **y = set[0]** which was obtained in part 1 of this series. For the first law of exponents, the following corollary is needed:

```
In[4]:= SubstTest[iterate, times[nat[x]], set[nat[w]], w -> natexp[nat[y], nat[z]]]

Out[4]= iterate[times[nat[x]], set[natexp[nat[y], nat[z]]]] ==
         composite[times[natexp[nat[y], nat[z]]], NATEXP, LEFT[nat[x]]]

In[5]:= iterate[times[nat[x_]], set[natexp[nat[y_], nat[z_]]]] :=
         composite[times[natexp[nat[y], nat[z]]], NATEXP, LEFT[nat[x]]]
```

The first law of exponents now follows for the special case that all variables are wrapped with **nat**.

```
In[6]:= Map[A, ImageComp[image[power[times[nat[x]]], set[nat[y]]],
          image[power[times[nat[x]]], set[nat[z]]], set[set[0]]]]

Out[6]= natexp[nat[x], natadd[nat[y], nat[z]]] ==
         natmul[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]]

In[7]:= natexp[nat[x_], natadd[nat[y_], nat[z_]]] :=
         natmul[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]]
```

The next step is to replace the **nat** wrappers with numberhood literals:

```
In[8]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]], equal[z, nat[w]]],
         equal[natexp[x, natadd[y, z]], natmul[natexp[x, y], natexp[x, z]]], {u → x, v → y, w → z}]

Out[8]= or[equal[natexp[x, natadd[y, z]], natmul[natexp[x, y], natexp[x, z]]],
          not[member[x, omega]], not[member[y, omega]], not[member[z, omega]]] == True

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

When any variable is not a natural number, the equation reduces to **V** = **V**.

```
In[10]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
          {u -> natexp[x, natadd[y, z]], v → V, w -> natmul[natexp[x, y], natexp[x, z]]}]

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

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

Combining the two cases yields the first law of exponents without wrappers:

```
In[12]:= SubstTest[and, implies[p1, p2], or[p1, p2],
          {p1 → and[member[x, omega], member[y, omega], member[z, omega]],
           p2 -> equal[natexp[x, natadd[y, z]], natmul[natexp[x, y], natexp[x, z]]]}]

Out[12]= True == equal[natexp[x, natadd[y, z]], natmul[natexp[x, y], natexp[x, z]]]

In[13]:= natexp[x_, natadd[y_, z_]] := natmul[natexp[x, y], natexp[x, z]]
```

This rewrite rule is here oriented to expand. The **Orderless** attributes of **natadd** and **natmul** make both sides of the equation symmetric under interchange of the variables **y** and **z**.

## a corollary of the first law of exponents

Lemma.

```
In[14]:= equal[composite[NATEXP, LEFT[x], id[omega]], composite[NATEXP, LEFT[x]]]

Out[14]= True

In[15]:= composite[NATEXP, LEFT[x_], id[omega]] := composite[NATEXP, LEFT[x]]
```

Theorem

```
In[16]:= Map[equal[composite[NATEXP, LEFT[x], plus[y]],
           composite[VERTSECT[composite[#, id[omega]]], id[omega]]] &,
         SubstTest[reify, z, natexp[x, f[y, z]], f → natadd]]

Out[16]= equal[composite[NATEXP, LEFT[x], plus[y]],
          composite[times[natexp[x, y]], NATEXP, LEFT[x]]] == True

In[17]:= composite[times[natexp[x_, y_]], NATEXP, LEFT[x_]] :=
         composite[NATEXP, LEFT[x], plus[y]]
```

This corollary of the first law of exponents is used in the derivation of the third law below.

## the second law of exponents: $(x\,y)^z = x^z\,y^z$

Lemma.

```
In[18]:= SubstTest[iterate, times[nat[z]], set[set[0]], z → natmul[nat[x], nat[y]]]

Out[18]= iterate[times[natmul[nat[x], nat[y]]], set[set[0]]] ==
          composite[NATEXP, LEFT[natmul[nat[x], nat[y]]]]

In[19]:= iterate[times[natmul[nat[x_], nat[y_]]], set[set[0]]] :=
         composite[NATEXP, LEFT[natmul[nat[x], nat[y]]]]
```

Applying **reify** to the **iterate** formula derived in the previous section yields a formula involving **power[times[nat[x]]]**.

```
In[20]:= Map[image[#, set[nat[y]]] &,
          Map[inverse[flip[rotate[rotate[composite[id[omega], inverse[#]]]]]] &,
           SubstTest[reify, y, iterate[z, set[nat[y]]], z → times[nat[x]]] // Reverse]

Out[20]= composite[image[power[times[nat[x]]], set[nat[y]]], id[omega]] ==
          times[natexp[nat[x], nat[y]]]

In[21]:= composite[image[power[times[nat[x_]]], set[nat[y_]]], id[omega]] :=
         times[natexp[nat[x], nat[y]]]
```

Since **times[x]** and **times[y]** commute, any power of their composite is the composite of their powers:

```
In[22]:= SubstTest[implies, commute[u, v], equal[image[power[composite[u, v]], set[w]],
           composite[image[power[u], set[w]], image[power[v], set[w]]]],
          {u → times[x], v → times[y], w → z}]

Out[22]= equal[composite[image[power[times[x]], set[z]], image[power[times[y]], set[z]]],
           image[power[times[natmul[x, y]]], set[z]]] == True

In[23]:= composite[image[power[times[x_]], set[z_]], image[power[times[y_]], set[z_]]] :=
          image[power[times[natmul[x, y]]], set[z]]
```

The second law of exponents, with wrapped variables, is an immediate consequence:

```
In[24]:= Map[A, ImageComp[image[power[times[nat[x]]], set[nat[z]]],
           image[power[times[nat[y]]], set[nat[z]]], set[set[0]]]]

Out[24]= natexp[natmul[nat[x], nat[y]], nat[z]] ==
          natmul[natexp[nat[x], nat[z]], natexp[nat[y], nat[z]]]

In[25]:= natexp[natmul[nat[x_], nat[y_]], nat[z_]] :=
          natmul[natexp[nat[x], nat[z]], natexp[nat[y], nat[z]]]
```

The next step is to replace the **nat** wrappers with numberhood literals:

```
In[26]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]], equal[z, nat[w]]],
          equal[natexp[natmul[x, y], z], natmul[natexp[x, z], natexp[y, z]]], {u → x, v → y, w → z}]

Out[26]= or[equal[natexp[natmul[x, y], z], natmul[natexp[x, z], natexp[y, z]]],
           not[member[x, omega]], not[member[y, omega]], not[member[z, omega]]] == True

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

Again, if any variable is not a natural number, the equation just reduces to $\mathbf{V} = \mathbf{V}$.

```
In[28]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
          {u -> natexp[natmul[x, y], z], v → V, w -> natmul[natexp[x, z], natexp[y, z]]}]

Out[28]= or[and[member[x, omega], member[y, omega], member[z, omega]],
           equal[natexp[natmul[x, y], z], natmul[natexp[x, z], natexp[y, z]]]] == True

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

Combining the cases yields a wrapper-free version of the second law of exponents.

```
In[30]:= SubstTest[and, implies[p1, p2], or[p1, p2],
          {p1 → and[member[x, omega], member[y, omega], member[z, omega]],
           p2 -> equal[natexp[natmul[x, y], z], natmul[natexp[x, z], natexp[y, z]]]}]

Out[30]= True == equal[natexp[natmul[x, y], z], natmul[natexp[x, z], natexp[y, z]]]

In[31]:= natexp[natmul[x_, y_], z_] := natmul[natexp[x, z], natexp[y, z]]
```

This law of exponents is oriented to expand. The **Orderless** attribute of **natmul** makes both sides symmetric under interchange of **y** and **z**.

## third law of exponents: $(x^y)^z = x^{(y\,z)}$

From the basic **iterate** formula derived in the first part of this series, one obtains this lemma:

```
In[32]:= SubstTest[iterate, times[nat[z]], set[set[0]], z -> natexp[nat[x], nat[y]]]

Out[32]= iterate[times[natexp[nat[x], nat[y]]], set[set[0]]] ==
          composite[NATEXP, LEFT[natexp[nat[x], nat[y]]]]

In[33]:= iterate[times[natexp[nat[x_], nat[y_]]], set[set[0]]] :=
          composite[NATEXP, LEFT[natexp[nat[x], nat[y]]]]
```

The corollary of the first law of exponents is needed for the following application of the uniqueness theorem for **iterate**:

```
In[34]:= SubstTest[implies,
          and[equal[composite[u, w], composite[w, SUCC]], equal[image[w, set[0]], v]],
          equal[composite[w, id[omega]], iterate[u, v]], {u → times[natexp[nat[x], nat[y]]],
           v → set[set[0]], w → composite[NATEXP, LEFT[nat[x]], times[nat[y]]]}]

Out[34]= equal[composite[NATEXP, LEFT[natexp[nat[x], nat[y]]]],
           composite[NATEXP, LEFT[nat[x]], times[nat[y]]]] == True

In[35]:= composite[NATEXP, LEFT[natexp[nat[x_], nat[y_]]]] :=
          composite[NATEXP, LEFT[nat[x]], times[nat[y]]]
```

The third law of exponents, with wrapped variables, is an immediate consequence:

```
In[36]:= Map[A, ImageComp[NATEXP, LEFT[natexp[nat[x], nat[y]]], set[nat[z]]]] // Reverse

Out[36]= natexp[natexp[nat[x], nat[y]], nat[z]] == natexp[nat[x], natmul[nat[y], nat[z]]]

In[37]:= natexp[natexp[nat[x_], nat[y_]], nat[z_]] := natexp[nat[x], natmul[nat[y], nat[z]]]
```

As for the other laws, the **nat** wrappers are now replaced with numberhood literals:

```
In[38]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]], equal[z, nat[w]]],
          equal[natexp[natexp[x, y], z], natexp[x, natmul[y, z]]], {u → x, v → y, w → z}]

Out[38]= or[equal[natexp[x, natmul[y, z]], natexp[natexp[x, y], z]],
           not[member[x, omega]], not[member[y, omega]], not[member[z, omega]]] == True

In[39]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The case of non-numbers is handled as in the other two cases:

```
In[40]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
          {u -> natexp[x, natmul[y, z]], v → V, w -> natexp[natexp[x, y], z]}]

Out[40]= or[and[member[x, omega], member[y, omega], member[z, omega]],
           equal[natexp[x, natmul[y, z]], natexp[natexp[x, y], z]]] == True

In[41]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The cases of numbers and non-numbers are combined to obtain the general case of the third law of exponents:

```
In[42]:= SubstTest[and, implies[p1, p2], or[p1, p2],
          {p1 → and[member[x, omega], member[y, omega], member[z, omega]],
           p2 -> equal[natexp[x, natmul[y, z]], natexp[natexp[x, y], z]]}]

Out[42]= True == equal[natexp[x, natmul[y, z]], natexp[natexp[x, y], z]]

In[43]:= natexp[natexp[x_, y_], z_] := natexp[x, natmul[y, z]]
```

The orientation of this rewrite rule was chosen as indicated because the **Orderless** attribute of **natmul** makes the right side symmetric with respect to interchanging the variables **y** and **z**. Had the opposite orientation been used, the result of rewriting would depend on the names of these variables.