NATEXP, part 1

Johan G. F. Belinfante
2006 June 27

```
In[1]:= SetDirectory["l:"]; << goedel82.22a; << tools.m

        :Package Title: goedel82.22a         2006 June 22 at 10:30 p.m.

        It is now:  2006 Jun 27 at 17:3

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 June 21

        weightlimit = 40
```

summary

The exponential function **NATEXP** in the arithmetic of natural numbers is defined, and some of its basic properties are derived.

NATEXP definition

The definition of **NATEXP** is inspired by an analogy with **NATMUL**. Instead of an explicit membership rule, or even one wrapped with **class**, the idea is to define this function by using a formula for **image[V, intersection[set[x], NATEXP]]**. This has the advantage of hiding the definition so that it does not forever expand out, but at the same time making it available via **assert**.

```
In[2]:= image[V, intersection[set[x_], NATEXP]] :=
         intersection[image[V, intersection[omega, set[first[first[x]]]]],
          image[V, intersection[iterate[times[first[first[x]]], set[set[0]]],
            set[pair[second[first[x]], second[x]]]]]]
```

composites with identities

The following rules are needed to simplify various expressions that come up frequently.

```
In[3]:= Map[equal[0, #] &, dif[NATEXP, cart[cart[V, V], V]] // Renormality]

Out[3]= subclass[NATEXP, cart[cart[V, V], V]] == True

In[4]:= subclass[NATEXP, cart[cart[V, V], V]] := True
```

Corollary 1.

```
In[5]:= equal[composite[NATEXP, id[cart[V, V]]], NATEXP]
Out[5]= True

In[6]:= composite[NATEXP, id[cart[V, V]]] := NATEXP
```

Corollary 2.

```
In[7]:= Assoc[NATEXP, id[cart[V, V]], Id] // Reverse
Out[7]= composite[Id, NATEXP] == NATEXP

In[8]:= composite[Id, NATEXP] := NATEXP
```

The following temporary special rules will be needed when the domain is studied. They will later be replaced with more precise rewrite rules.

```
In[9]:= Map[equal[0, #] &, dif[NATEXP, cart[cart[omega, V], V]] // Renormality]
Out[9]= subclass[domain[NATEXP], cart[omega, V]] == True

In[10]:= % /. Equal → SetDelayed
```

Corollary.

```
In[11]:= SubstTest[implies, subclass[u, v],
           subclass[domain[u], domain[v]], {u → domain[NATEXP], v -> cart[omega, V]}]
Out[11]= subclass[domain[domain[NATEXP]], omega] == True

In[12]:= % /. Equal → SetDelayed
```

composite with LEFT[nat[x]]

The following key theorem is exploited heavily in this notebook.

```
In[13]:= Map[class[pair[y, z], #] &,
           member[pair[pair[nat[x], y], z], NATEXP] // AssertTest] // Reverse
Out[13]= iterate[times[nat[x]], set[set[0]]] == composite[NATEXP, LEFT[nat[x]]]

In[14]:= iterate[times[nat[x_]], set[set[0]]] := composite[NATEXP, LEFT[nat[x]]]
```

From this one obtains the following recursion relation.

```
In[15]:= SubstTest[composite, iterate[u, v], SUCC, {u → times[nat[x]], v → set[set[0]]}]
Out[15]= composite[NATEXP, LEFT[nat[x]], SUCC] == composite[times[nat[x]], NATEXP, LEFT[nat[x]]]
```

```
In[16]:= composite[NATEXP, LEFT[nat[x_]], SUCC] :=
          composite[times[nat[x]], NATEXP, LEFT[nat[x]]]
```

powers of 0 and 1

Lemma.

```
In[17]:= (union[cart[set[0], set[set[0]]],
             cart[union[intersection[omega, complement[succ[set[0]]]], set[set[0]]], set[0]]] //
           Normality) /. Equal → SetDelayed
```

Most powers of **0** are equal to **0**. The sole exception is that the **0**th power of **0** is **1** according to the definition given.

```
In[18]:= SubstTest[iterate, times[nat[x]], set[set[0]], x → 0] // Reverse
```

```
Out[18]= composite[NATEXP, LEFT[0]] == union[
           cart[intersection[omega, complement[set[0]]], set[0]], cart[set[0], set[set[0]]]]
```

```
In[19]:= composite[NATEXP, LEFT[0]] := union[
           cart[intersection[omega, complement[set[0]]], set[0]], cart[set[0], set[set[0]]]]
```

Every power of **1** is equal to **1**.

```
In[20]:= SubstTest[iterate, times[nat[x]], set[set[0]], x → set[0]] // Reverse
```

```
Out[20]= composite[NATEXP, LEFT[set[0]]] == cart[omega, set[set[0]]]
```

```
In[21]:= composite[NATEXP, LEFT[set[0]]] := cart[omega, set[set[0]]]
```

an upper bound on the range

The range of **iterate[u,v]** is bounded above by the union of **range[u]** and **v**.

```
In[22]:= SubstTest[subclass, range[iterate[u, v]],
           union[range[u], v], {u → times[nat[x]], v → set[set[0]]}]
```

```
Out[22]= subclass[image[NATEXP, cart[set[nat[x]], V]],
           union[image[DIV, set[nat[x]]], set[set[0]]]] == True
```

```
In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary:

```
In[24]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> image[NATEXP, cart[set[nat[x]], V]],
            v -> union[image[DIV, set[nat[x]]], set[set[0]]], w → omega}]
```

```
Out[24]= subclass[image[NATEXP, cart[set[nat[x]], V]], omega] == True
```

```
In[25]:= (% /. x → x_) /. Equal → SetDelayed
```

Applying **reify** yields:

```
In[26]:= Map[implies[equal[0, #], subclass[range[NATEXP], omega]] &,
           SubstTest[reify, x, dif[image[NATEXP, cart[set[nat[x]], V]], w], w -> omega]]

Out[26]= subclass[range[NATEXP], omega] == True

In[27]:= % /. Equal → SetDelayed
```

This inclusion will be sharpened to an equation in a later section.

---

domain

If **v** is not empty, then **0** belongs to the domain of **iterate[u,v]**,. From this observation one deduces:

```
In[28]:= SubstTest[member, 0, domain[iterate[u, v]], {u → times[nat[x]], v → set[set[0]]}]

Out[28]= member[pair[nat[x], 0], domain[NATEXP]] == True

In[29]:= (% /. x → x_) /. Equal → SetDelayed
```

The variable **x** can be eliminated using **reify**.

```
In[30]:= Map[implies[equal[0, #], subclass[omega, image[inverse[domain[NATEXP]], set[0]]]] &,
           SubstTest[reify, x, dif[set[PAIR[nat[x], 0]], y], y → domain[NATEXP]]]

Out[30]= subclass[omega, image[inverse[domain[NATEXP]], set[0]]] == True

In[31]:= % /. Equal → SetDelayed
```

The reverse inclusion also holds, so one obtains an equation:

```
In[32]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> omega, v -> image[inverse[domain[NATEXP]], set[0]]}]

Out[32]= True == equal[omega, image[inverse[domain[NATEXP]], set[0]]]

In[33]:= image[inverse[domain[NATEXP]], set[0]] := omega
```

The next step of the derivation is a classic induction argument. The base case is:

```
In[34]:= member[0, image[domain[NATEXP], set[nat[x]]]]

Out[34]= True
```

The induction step is:

```
In[35]:= Map[subclass[image[domain[NATEXP], set[nat[x]]], #] &,
           IminComp[composite[NATEXP, LEFT[nat[x]]], SUCC, omega]] // Reverse

Out[35]= subclass[image[SUCC, image[domain[NATEXP], set[nat[x]]]],
           image[domain[NATEXP], set[nat[x]]]] == True

In[36]:= (% /. x → x_) /. Equal → SetDelayed
```

Applying induction yields:

```
In[37]:= SubstTest[implies, INDUCTIVE[w],
           subclass[omega, w], w -> image[domain[NATEXP], set[nat[x]]]]

Out[37]= subclass[omega, image[domain[NATEXP], set[nat[x]]]] == True

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

In general, the domain of **iterate[u,v]** is a subclass of **omega**.

```
In[39]:= SubstTest[subclass, domain[iterate[u, v]], omega, {u → times[nat[x]], v → set[set[0]]}]

Out[39]= subclass[image[domain[NATEXP], set[nat[x]]], omega] == True

In[40]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining these results yields an equation.

```
In[41]:= SubstTest[and, subclass[u, v], subclass[v, u],
           {u -> image[domain[NATEXP], set[nat[x]]], v -> omega}]

Out[41]= True == equal[omega, image[domain[NATEXP], set[nat[x]]]]

In[42]:= image[domain[NATEXP], set[nat[x_]]] := omega
```

An application of **reify** removes the variable **x**.

```
In[43]:= Map[equal[domain[NATEXP], composite[#, id[omega]]] &,
           SubstTest[reify, x, image[y, set[nat[x]]], y → domain[NATEXP]]]

Out[43]= equal[cart[omega, omega], domain[NATEXP]] == True

In[44]:= domain[NATEXP] := cart[omega, omega]
```

FUNCTION rule

Lemma.

```
In[45]:= SubstTest[FUNCTION, iterate[funpart[u], set[v]], {u → times[nat[x]], v → set[0]}]

Out[45]= FUNCTION[composite[NATEXP, LEFT[nat[x]]]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

The use of **reify** eliminates the wrapped variable **x**.

```
In[47]:= Map[equal[0, #] &,
           SubstTest[reify, x, dif[P[composite[NATEXP, LEFT[nat[x]]], y], y → FUNS]] // Reverse

Out[47]= FUNCTION[union[cart[cart[complement[omega], intersection[omega, complement[set[0]]]],
             set[0]], cart[cart[complement[omega], set[0]], set[set[0]]],
            composite[NATEXP, id[cart[omega, V]]]]] == True

In[48]:= % /. Equal → SetDelayed
```

The function **NATEXP** itself is a restriction of the above function.

```
In[49]:= SubstTest[implies, FUNCTION[x], FUNCTION[composite[x, id[cart[omega, omega]]]],
          x -> union[composite[NATEXP, id[cart[omega, V]]],
            composite[NATEXP, LEFT[0], SECOND, id[cart[complement[omega], V]]]]]

Out[49]= FUNCTION[NATEXP] == True

In[50]:= FUNCTION[NATEXP] := True
```

natexp[x,y] and APPLY

The definition of **natexp[x,y]** has been added to the **GOEDEL** program in the form of a membership rule wrapped with **class** in a manner that is completely analogous to the corresponding rule for other arithmetic constructors, such as **natmod[x,y]** and **natmul[x,y]**. From this one finds:

```
In[51]:= natexp[x, y] // Normality // Reverse

Out[51]= APPLY[NATEXP, pair[x, y]] == natexp[x, y]

In[52]:= APPLY[NATEXP, pair[x_, y_]] := natexp[x, y]
```

The constructor **natexp[x,y]** is equal to **V** when **x** or **y** is not a natural number.

```
In[53]:= SubstTest[equal, V, APPLY[z, pair[x, y]], z → NATEXP]

Out[53]= equal[V, natexp[x, y]] == or[not[member[x, omega]], not[member[y, omega]]]

In[54]:= equal[V, natexp[x_, y_]] := or[not[member[x, omega]], not[member[y, omega]]]
```

Otherwise it is a set:

```
In[55]:= SubstTest[member, APPLY[u, v], V, {u → NATEXP, v → pair[x, y]}]

Out[55]= member[natexp[x, y], V] == and[member[x, omega], member[y, omega]]

In[56]:= member[natexp[x_, y_], V] := and[member[x, omega], member[y, omega]]
```

simplification rules for natexp[x,y]

Simplification rule:

```
In[57]:= equal[union[complement[image[V, set[x]]], natexp[x, y]], natexp[x, y]]
Out[57]= True
```

```
In[58]:= union[complement[image[V, set[x_]]], natexp[x_, y_]] := natexp[x, y]
```

Similarly:

```
In[59]:= equal[union[complement[image[V, set[y]]], natexp[x, y]], natexp[x, y]]
Out[59]= True
```

```
In[60]:= union[complement[image[V, set[y_]]], natexp[x_, y_]] := natexp[x, y]
```

Corollary: One can use **pair** or **PAIR** interchangeably. It makes no difference.

```
In[61]:= APPLY[NATEXP, PAIR[x, y]] // Normality
Out[61]= APPLY[NATEXP, PAIR[x, y]] == natexp[x, y]
```

```
In[62]:= APPLY[NATEXP, PAIR[x_, y_]] := natexp[x, y]
```

Vertical section rules:

```
In[63]:= SubstTest[image, funpart[w], cart[set[x], set[y]], w → NATEXP]
Out[63]= image[NATEXP, cart[set[x], set[y]]] == set[natexp[x, y]]
```

```
In[64]:= image[NATEXP, cart[set[x_], set[y_]]] := set[natexp[x, y]]
```

```
In[65]:= SubstTest[image, funpart[w], set[pair[x, y]], w → NATEXP]
Out[65]= image[NATEXP, set[pair[x, y]]] == set[natexp[x, y]]
```

```
In[66]:= image[NATEXP, set[pair[x_, y_]]] := set[natexp[x, y]]
```

reify rule for natexp

The **reify** rule for **natexp** is simplified by using a double complement:

```
In[67]:= Map[complement[complement[#]] &,
          SubstTest[reify, x, A[image[z, cart[set[f[x]], set[g[x]]]]], z → NATEXP]]

Out[67]= reify[x, natexp[f[x], g[x]]] ==
          union[cart[union[complement[image[inverse[VERTSECT[reify[x, f[x]]]], omega]],
             complement[image[inverse[VERTSECT[reify[x, g[x]]]], omega]]], V],
           composite[inverse[E], NATEXP, intersection[
             composite[inverse[FIRST], VERTSECT[reify[x, f[x]]]],
             composite[inverse[SECOND], VERTSECT[reify[x, g[x]]]]]]]

In[68]:= reify[x_, natexp[y_, z_]] :=
          union[cart[union[complement[image[inverse[VERTSECT[reify[x, y]]], omega]],
             complement[image[inverse[VERTSECT[reify[x, z]]], omega]]], V],
           composite[inverse[E], NATEXP, intersection[composite[inverse[FIRST],
              VERTSECT[reify[x, y]]], composite[inverse[SECOND], VERTSECT[reify[x, z]]]]]]
```

powers of 0 and 1, revisited

Powers of **0**:

```
In[69]:= Map[A, ImageComp[NATEXP, LEFT[0], set[nat[x]]] // Reverse

Out[69]= natexp[0, nat[x]] == intersection[complement[image[V, nat[x]]], set[0]]

In[70]:= natexp[0, nat[x_]] := intersection[complement[image[V, nat[x]]], set[0]]
```

Powers of **1**.

```
In[71]:= Map[A, ImageComp[NATEXP, LEFT[set[0]], set[nat[x]]] // Reverse

Out[71]= natexp[set[0], nat[x]] == set[0]

In[72]:= natexp[set[0], nat[x_]] := set[0]
```

0th power

The **0**th power of any number is **1**.

```
In[73]:= Map[member[set[0], class[z, #]] &,
          member[pair[pair[nat[x], 0], z], NATEXP] // AssertTest]

Out[73]= equal[natexp[nat[x], 0], set[0]] == True

In[74]:= natexp[nat[x_], 0] := set[0]
```

The variable **x** can be removed:

```
In[75]:= Map[composite[VERTSECT[#], id[omega]] &,
           SubstTest[reify, x, natexp[f[x], 0], f → nat]] // Reverse
```

```
Out[75]= composite[NATEXP, RIGHT[0]] == cart[omega, set[set[0]]]
```

```
In[76]:= composite[NATEXP, RIGHT[0]] := cart[omega, set[set[0]]]
```

1st power

Lemma.

```
In[77]:= Map[class[x, #] &, member[pair[pair[x, set[0]], x], NATEXP] // AssertTest]
```

```
Out[77]= image[inverse[fix[composite[inverse[NATEXP], FIRST]]], set[set[0]]] == omega
```

```
In[78]:= % /. Equal → SetDelayed
```

Theorem.

```
In[79]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]],
           {u -> id[omega], v -> composite[NATEXP, RIGHT[set[0]]]}]
```

```
Out[79]= equal[composite[NATEXP, RIGHT[set[0]]], id[omega]] == True
```

```
In[80]:= composite[NATEXP, RIGHT[set[0]]] := id[omega]
```

Corollary: The **1**st power of any number is itself.

```
In[81]:= Map[A, ImageComp[NATEXP, RIGHT[set[0]], set[nat[x]]]] // Reverse
```

```
Out[81]= natexp[nat[x], set[0]] == nat[x]
```

```
In[82]:= natexp[nat[x_], set[0]] := nat[x]
```

range

The theorem on **1**st powers yields a lower bound for the range of **NATEXP**.

```
In[83]:= Map[subclass[#, range[NATEXP]] &, ImageComp[NATEXP, RIGHT[set[0]], V]]
```

```
Out[83]= subclass[omega, range[NATEXP]] == True
```

```
In[84]:= % /. Equal → SetDelayed
```

Combining this with the upper bound derived earlier yields:

```
In[85]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> omega, v -> range[NATEXP]}]
```

```
Out[85]= True == equal[omega, range[NATEXP]]
```

```
In[86]:= range[NATEXP] := omega
```

Corollary.

```
In[87]:= SubstTest[member, APPLY[funpart[z], w], range[funpart[z]],
           {w → PAIR[x, y], z → NATEXP}]
Out[87]= member[natexp[x, y], omega] == and[member[x, omega], member[y, omega]]

In[88]:= member[natexp[x_, y_], omega] := and[member[x, omega], member[y, omega]]
```