Art Quaife's theorem (EX8)

Johan G. F. Belinfante and Ming Li
2007 February 10

```
In[1]:= SetDirectory["l:"]; << goedel90.08a; << tools.m

        :Package Title: goedel90.08a        2007 February 8 at 9:50 p.m.

        It is now:  2007 Feb 10 at 5:42

        Loading Simplification Rules

        TOOLS.M                       Revised 2007 January 7

        weightlimit = 40
```

summary

The exponential function with natural number base **x** is **composite[NATEXP, LEFT[x]]**. Art Quaife's theorem **(EX8)** says that this function is one-to-one if **x > 1**. Quaife used McCune's automated reasonig program **Otter** to prove this theorem, starting with a set of axioms for natural number arithmetic.

```
In[2]:= "Art Quaife, Automated Development of Fundamental
           Mathematical Theories, Kluwer Academic Publishers, Dordrecht,
           the Netherlands, 1992.  (Appendix 3.  See page 229.)";
```

In this notebook, this theorem is rederived in a class-theoretic framework, using the **GOEDEL** program. In addition, some corollaries are obtained by eliminating the variables.

derivation of (EX8)

Lemma. (Specialization of an existing theorem.)

```
In[3]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p0, p1],
           not[implies[and[p0, p2], p3]], {p0 -> member[set[0], nat[x]], p1 -> member[0, nat[x]],
            p2 -> member[natexp[nat[x], nat[z]], natexp[nat[x], nat[y]]],
            p3 -> member[nat[z], nat[y]]}]] // Reverse

Out[3]= or[member[nat[z], nat[y]], not[member[natexp[nat[x], nat[z]], natexp[nat[x], nat[y]]]],
          not[member[set[0], nat[x]]]] == True

In[4]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Corollary. (An application of trichotomy to the preceding lemma.)

```
In[5]:= Map[not, SubstTest[and, implies[p2, p3], implies[and[p1, p3], p4],
          not[implies[and[p1, p2], p4]], {p1 → member[set[0], nat[x]],
           p2 -> equal[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]],
           p3 → not[member[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]]],
           p4 → not[member[nat[y], nat[z]]]}]] // Reverse

Out[5]= or[not[equal[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]]],
          not[member[nat[y], nat[z]]], not[member[set[0], nat[x]]]] == True

In[6]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

A second application of trichotomy yields a form of Quaife's Theorem **(EX8)** using **nat** wrappers.

```
In[7]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[p1, p4]], {p1 → and[equal[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]],
             member[set[0], nat[x]]], p2 -> not[member[nat[y], nat[z]]],
           p3 -> not[member[nat[z], nat[y]]], p4 → equal[nat[y], nat[z]]}]] // Reverse

Out[7]= or[equal[nat[y], nat[z]], not[equal[natexp[nat[x], nat[y]], natexp[nat[x], nat[z]]]],
          not[member[set[0], nat[x]]]] == True

In[8]:= or[equal[nat[y_], nat[z_]],
          not[equal[natexp[nat[x_], nat[y_]], natexp[nat[x_], nat[z_]]]],
          not[member[set[0], nat[x_]]]] := True
```

Eliminating the **nat** wrappers yields:

```
In[9]:= SubstTest[implies, and[equal[x, nat[u]], equal[y, nat[v]], equal[z, nat[w]]],
          or[equal[y, z], not[equal[natexp[x, y], natexp[x, z]]], not[member[set[0], x]]],
          {u → x, v → y, w -> z}] // Reverse

Out[9]= or[equal[y, z], not[equal[natexp[x, y], natexp[x, z]]], not[member[x, omega]],
          not[member[y, omega]], not[member[z, omega]], not[member[set[0], x]]] == True

In[10]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

---

eliminating the variables y and z

Eliminating the variables yields the following somewhat cryptic result.

```
In[11]:= Map[empty[composite[Id, complement[#]]] &, SubstTest[class, pair[y, z],
          or[equal[y, z], not[member[pair[y, z], composite[inverse[funpart[t]], funpart[t]]]],
           not[member[x, w]], not[member[y, w]], not[member[z, w]], not[member[set[0], x]]],
          {t → composite[NATEXP, LEFT[x]], w → omega}]]

Out[11]= or[not[member[x, omega]],
          not[member[x, domain[fix[composite[inverse[NATEXP], NATEXP, LEFT[x], Di, SECOND]]]]],
          not[member[set[0], x]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

A better formulation of the above result can be derived using **AssertTest**.

```
In[13]:= or[not[member[x, omega]], not[member[set[0], x]], subclass[
           composite[inverse[LEFT[x]], inverse[NATEXP], NATEXP, LEFT[x]], Id]] // AssertTest

Out[13]= or[not[member[x, omega]], not[member[set[0], x]],
           subclass[composite[inverse[LEFT[x]], inverse[NATEXP], NATEXP, LEFT[x]], Id]] == True

In[14]:= (% /. x → x_) /. Equal → SetDelayed
```

A still better formulation:

```
In[15]:= Map[implies[and[member[x, omega], member[set[0], x]], #] &, SubstTest[subclass,
           composite[inverse[funpart[w]], funpart[w]], Id, w → composite[NATEXP, LEFT[x]]]]

Out[15]= or[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]],
           not[member[x, omega]], not[member[set[0], x]]] == True

In[16]:= (% /. x → x_) /. Equal → SetDelayed
```

A similar result holds when **x** is not a number.

```
In[17]:= SubstTest[implies, equal[0, z], FUNCTION[z],
           z -> composite[inverse[LEFT[x]], inverse[NATEXP]]] // Reverse

Out[17]= or[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]], member[x, omega]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

Eliminating the redundant numberhood literal yields:

```
In[19]:= SubstTest[and, or[p, q], implies[p, q], {p → member[x, omega], q -> or[
             FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]], not[member[set[0], x]]]}]

Out[19]= or[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]],
           not[member[set[0], x]]] == True

In[20]:= (% /. x → x_) /. Equal → SetDelayed
```

A converse of this result is derived in the next section.

converse

Lemma. (The special case **x = 1**.)

```
In[21]:= Map[FUNCTION, composite[inverse[LEFT[set[0]]], inverse[NATEXP]] // DoubleInverse]

Out[21]= FUNCTION[composite[inverse[LEFT[set[0]]], inverse[NATEXP]]] == False

In[22]:= % /. Equal → SetDelayed
```

A similar result holds when **x = 0**. All powers of **0** are **0**, except for the **0**-th power, which is **1**. In any case, there are infinitely many powers of **0** that are the same. This cardinality argument can be formalised as follows:

```
In[23]:= SubstTest[equal, card[t], set[0], t → dif[omega, set[0]]]

Out[23]= member[intersection[omega, complement[set[0]]], range[SINGLETON]] == False

In[24]:= % /. Equal → SetDelayed
```

Corollary. A constant function on **dif[omega,set[0]]** can not be one-to-one.

```
In[25]:= SubstTest[FUNCTION, cart[set[0], t], t → dif[omega, set[0]]] // Reverse

Out[25]= subclass[intersection[omega,
           image[Di, intersection[omega, complement[set[0]]]]], set[0]] == False

In[26]:= % /. Equal → SetDelayed
```

Corollary. If a function is not one-to-one, neither is an extension of it.

```
In[27]:= Map[not, SubstTest[implies, and[subclass[u, v], FUNCTION[v]], FUNCTION[u],
           {u -> cart[set[0], intersection[omega, complement[set[0]]]],
            v -> union[cart[set[0], intersection[omega, complement[set[0]]]],
              cart[set[set[0]], set[0]]]}]] // Reverse

Out[27]= FUNCTION[union[cart[set[0], intersection[omega, complement[set[0]]]],
           cart[set[set[0]], set[0]]]] == False

In[28]:= % /. Equal → SetDelayed
```

Corollary. (The special case **x = 0**.)

```
In[29]:= Map[FUNCTION, composite[inverse[LEFT[0]], inverse[NATEXP]] // DoubleInverse]

Out[29]= FUNCTION[composite[inverse[LEFT[0]], inverse[NATEXP]]] == False

In[30]:= % /. Equal → SetDelayed
```

An application of trichotomy yields:

```
In[31]:= SubstTest[and, implies[p1, p3], implies[p2, p3],
          {p1 → equal[0, nat[x]], p2 → equal[set[0], nat[x]],
           p3 → not[FUNCTION[composite[inverse[LEFT[nat[x]]], inverse[NATEXP]]]]}]

Out[31]= or[member[set[0], nat[x]],
           not[FUNCTION[composite[inverse[LEFT[nat[x]]], inverse[NATEXP]]]]] == True

In[32]:= (% /. x → x_) /. Equal → SetDelayed
```

Eliminating the **nat** wrapper yields:

```
In[33]:= SubstTest[implies, equal[x, nat[t]], or[member[set[0], x],
           not[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]]]], t → x] // Reverse

Out[33]= or[member[set[0], x], not[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]]],
           not[member[x, omega]]] == True

In[34]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[35]:= equiv[FUNCTION[composite[inverse[LEFT[x]], inverse[NATEXP]]],
           or[not[member[x, omega]], member[set[0], x]]] // not // not

Out[35]= True

In[36]:= FUNCTION[composite[inverse[LEFT[x_]], inverse[NATEXP]]] :=
          or[member[set[0], x], not[member[x, omega]]]
```

UB rules

Lemma.

```
In[37]:= image[UB[x], intersection[complement[FUNS], P[cart[V, V]]]] // Renormality

Out[37]= image[UB[x], intersection[complement[FUNS], P[cart[V, V]]]] ==
          fix[composite[x, cross[Id, Di], inverse[x]]]

In[38]:= image[UB[x_], intersection[complement[FUNS], P[cart[V, V]]]] :=
          fix[composite[x, cross[Id, Di], inverse[x]]]
```

Theorem.

```
In[39]:= SubstTest[empty, image[UB[x], y], y -> intersection[complement[FUNS], P[cart[V, V]]]]

Out[39]= subclass[intersection[domain[UB[x]], P[cart[V, V]]], FUNS] == FUNCTION[rotate[x]]

In[40]:= subclass[intersection[domain[UB[x_]], P[cart[V, V]]], FUNS] := FUNCTION[rotate[x]]
```

Lemma.

```
In[41]:= image[UB[x], intersection[complement[image[INVERSE, FUNS]], P[cart[V, V]]]] //
          Renormality

Out[41]= image[UB[x], intersection[complement[image[INVERSE, FUNS]], P[cart[V, V]]]] ==
          fix[composite[x, cross[Di, Id], inverse[x]]]

In[42]:= image[UB[x_], intersection[complement[image[INVERSE, FUNS]], P[cart[V, V]]]] :=
          fix[composite[x, cross[Di, Id], inverse[x]]]
```

Theorem.

```
In[43]:= SubstTest[empty, image[UB[x], y],
          y -> intersection[complement[image[INVERSE, FUNS]], P[cart[V, V]]]]

Out[43]= subclass[intersection[domain[UB[x]], P[cart[V, V]]], image[INVERSE, FUNS]] ==
          FUNCTION[rotate[composite[x, SWAP]]]

In[44]:= subclass[intersection[domain[UB[x_]], P[cart[V, V]]], image[INVERSE, FUNS]] :=
          FUNCTION[rotate[composite[x, SWAP]]]
```

Corollary.

```
In[45]:= Map[equal[V, #] &, SubstTest[class, x,
           subclass[P[composite[t, LEFT[x]]], u], {t → NATEXP, u → image[INVERSE, FUNS]}]]

Out[45]= FUNCTION[rotate[NATEXP]] == False

In[46]:= FUNCTION[rotate[NATEXP]] := False
```

---

variable-free formulation

```
In[47]:= SubstTest[FUNCTION, composite[id[image[V, s]], t],
           {s -> intersection[complement[image[V, intersection[set[x], succ[set[0]]]]],
              image[V, intersection[omega, set[x]]]],
            t → composite[inverse[LEFT[x]], inverse[NATEXP]]}] // Reverse

Out[47]= FUNCTION[
           composite[id[intersection[complement[image[V, intersection[set[x], succ[set[0]]]]],
              image[V, intersection[omega, set[x]]]],
            inverse[LEFT[x]], inverse[NATEXP]]] == True

In[48]:= (% /. x → x_) /. Equal → SetDelayed

In[49]:= Map[complement, SubstTest[class, x,
           subclass[P[composite[t, LEFT[x]]], u], {t → NATEXP, u → image[INVERSE, FUNS]}]]

Out[49]= fix[composite[rotate[composite[NATEXP, SWAP]],
            cross[Id, Di], inverse[rotate[composite[NATEXP, SWAP]]]]] == succ[set[0]]

In[50]:= fix[composite[rotate[composite[NATEXP, SWAP]], cross[Id, Di],
            inverse[rotate[composite[NATEXP, SWAP]]]]] := succ[set[0]]
```

Lemma.

```
In[51]:= SubstTest[FUNCTION, flip[t], t -> composite[rotate[x], id[cart[z, y]]]] // Reverse

Out[51]= FUNCTION[composite[rotate[x], SWAP, id[cart[y, z]]]] ==
          FUNCTION[composite[rotate[x], id[cart[z, y]]]]

In[52]:= FUNCTION[composite[rotate[x_], SWAP, id[cart[y_, z_]]]] :=
          FUNCTION[composite[rotate[x], id[cart[z, y]]]]
```

Theorem. Variable-free restatement of Quaife's Theorem (EX8).

```
In[53]:= SubstTest[empty, fix[composite[x, cross[Id, Di], inverse[x]]],
           x -> composite[id[complement[y]], z]] /.
         {y → succ[set[0]], z -> rotate[composite[NATEXP, SWAP]]}

Out[53]= FUNCTION[composite[rotate[NATEXP], id[cart[V, complement[succ[set[0]]]]]]] == True

In[54]:= FUNCTION[composite[rotate[NATEXP], id[cart[V, complement[succ[set[0]]]]]]] := True
```