composite numbers, part 2.

Johan G. F. Belinfante
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 12:21

        Loading Simplification Rules

        TOOLS.M                       Revised 2007 January 7

        weightlimit = 40
```

summary

A natural number is **composite** if it is the product of two numbers, both greater than **1**. In this notebook a different characterization is derived for composite numbers that is expected to be useful for proofs involving strong induction: a natural number is composite if and only if it is the product of two smaller ones.

a membership rule

The following is a membership rule for the set of numbers that can be factored as products of smaller numbers:

```
In[2]:= (member[x, fix[t]] // AssertTest) /. t -> composite[inverse[E], IMAGE[NATMUL], CART, DUP]

Out[2]= member[x, fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]] ==
         member[x, image[NATMUL, cart[x, x]]]

In[3]:= member[x_, fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]] :=
         member[x, image[NATMUL, cart[x, x]]]
```

A related fact:

```
In[4]:= SubstTest[subclass, fix[composite[u, v]],
          range[u], {u → NATMUL, v → composite[inverse[E], x]}] // Reverse

Out[4]= subclass[fix[composite[inverse[E], IMAGE[NATMUL], x]], omega] == True

In[5]:= subclass[fix[composite[inverse[E], IMAGE[NATMUL], x_]], omega] := True
```

inclusion in one direction

Lemma. (Comment. This derivation has been accelerated by deliberately omitting some steps.)

```
In[6]:= Map[not,
          SubstTest[and, implies[p1, or[p2, p3]], implies[p1, p4], implies[and[p1, p3], p6],
           implies[and[p1, p2], p7], not[and[p1, p5, p6]], not[and[p1, p5, p7]],
           {p1 → and[equal[x, natmul[u, v]], member[u, x], member[v, x], member[x, PRIMES]],
            p2 → equal[set[0], u], p3 → equal[set[0], v], p4 → member[x, omega],
            p5 → not[member[x, x]], p6 → equal[x, u], p7 → equal[x, v]}]]

Out[6]= and[equal[x, natmul[u, v]], member[u, x], member[v, x], member[x, PRIMES]] == False

In[7]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

All the variables can be eliminated at the same time:

```
In[8]:= Map[empty, SubstTest[class, pair[pair[u, v], x],
           and[member[x, intersection[t, image[NATMUL, set[PAIR[u, v]]]]],
            member[u, x], member[v, x]], t -> PRIMES]]

Out[8]= equal[0, intersection[PRIMES, fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]]] ==
         True

In[9]:= intersection[PRIMES, fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]] := 0
```

Theorem. (An inclusion in one direction.)

```
In[10]:= SubstTest[and, subclass[u, union[v, w]], disjoint[u, v],
           {u -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]], v → PRIMES,
            w -> image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]}]

Out[10]= subclass[fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]],
           image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]] == True

In[11]:= % /. Equal → SetDelayed
```

the reverse inclusion

Lemma.

```
In[12]:= SubstTest[implies, and[equal[u, nat[s]], equal[v, nat[t]], equal[x, natmul[u, v]],
            member[set[0], u], member[set[0], v]], member[u, x], {s → u, t → v, w → x}] // Reverse

Out[12]= or[member[u, x], not[equal[x, natmul[u, v]]], not[member[u, omega]],
           not[member[v, omega]], not[member[set[0], u]], not[member[set[0], v]]] == True

In[13]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[14]:= SubstTest[implies, subclass[w, z], subclass[image[t, w], image[t, z]],
           {t → NATMUL, w → set[PAIR[u, v]], z → cart[x, x]}] // Reverse

Out[14]= or[member[natmul[u, v], image[NATMUL, cart[x, x]]], not[member[u, omega]],
           not[member[u, x]], not[member[v, omega]], not[member[v, x]]] == True

In[15]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[16]:= Map[not, SubstTest[and, implies[and[p0, p1], p2], implies[and[p0, p1], p3],
            implies[and[p0, p1], p4], implies[and[p0, p2, p3, p4], p5],
            not[implies[and[p0, p1], p5]], {p0 → equal[x, natmul[u, v]],
             p1 → and[member[u, omega], member[v, omega], member[set[0], u], member[set[0], v]],
             p2 → member[u, x], p3 → member[v, x], p4 → member[x, omega],
             p5 → member[x, image[NATMUL, cart[x, x]]]}]] // Reverse

Out[16]= or[member[x, image[NATMUL, cart[x, x]]],
           not[equal[x, natmul[u, v]]], not[member[u, omega]], not[member[v, omega]],
           not[member[set[0], u]], not[member[set[0], v]]] == True

In[17]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Eliminating all variables, one finds:

```
In[18]:= Map[empty[composite[complement[#], id[cart[V, V]]]] &,
          SubstTest[class, pair[pair[u, v], x], or[member[x, s],
            not[equal[x, APPLY[w, PAIR[u, v]]]], not[member[u, t]], not[member[v, t]]],
           {s -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]],
            t → dif[omega, succ[set[0]]], w → NATMUL}]]

Out[18]= subclass[image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]],
           fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]] == True

In[19]:= % /. Equal → SetDelayed
```

Theorem. Equality of two expressions for the class of composite numbers. (Comment. In principle, this equation could be made into a rewrite rule, but how best to orient such a rewrite rule is unclear. For now this equation is merely retained as a statement of fact.)

```
In[20]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]],
           v -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]}]

Out[20]= equal[fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]],
           image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]] == True

In[21]:= equal[fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]],
           image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]] := True
```

Corollary. If **x** is the product of two numbers both greater than **1**, then **x** is the product of two numbers less than **x**.

```
In[22]:= SubstTest[implies, and[member[x, y], equal[y, z]], member[x, z],
           {y -> image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]],
            z -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]}] // Reverse

Out[22]= or[member[x, image[NATMUL, cart[x, x]]], not[member[x,
             image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]]] == True

In[23]:= or[member[x_, image[NATMUL, cart[x_, x_]]], not[member[x_,
             image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]]] := True
```

Corollary. Conversely, a number that can be factored into smaller ones is the product of two numbers both greater than **1**.

```
In[24]:= SubstTest[implies, and[member[x, y], equal[y, z]], member[x, z],
           {y -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]], z -> image[NATMUL,
              cart[complement[succ[set[0]]], complement[succ[set[0]]]]]}] // Reverse

Out[24]= or[member[x, image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]],
           not[member[x, image[NATMUL, cart[x, x]]]]] == True

In[25]:= or[
           member[x_, image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]],
           not[member[x_, image[NATMUL, cart[x_, x_]]]]] := True
```

corollaries

Corollary. If $\mathbf{x} > \mathbf{1}$ is not prime, then **x** can be written as the product of two lesser numbers.

```
In[26]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 → and[member[set[0], x], not[member[x, PRIMES]]], p2 -> member[x,
               image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]],
             p3 -> member[x, image[NATMUL, cart[x, x]]]}]] // Reverse

Out[26]= or[member[x, PRIMES], member[x, image[NATMUL, cart[x, x]]],
           not[member[x, omega]], not[member[set[0], x]]] == True

In[27]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If $\mathbf{x} = \mathbf{u}\,\mathbf{v}$ with $\mathbf{u} > \mathbf{1}$ and $\mathbf{v} > \mathbf{1}$, then $\mathbf{x} > \mathbf{1}$.

```
In[28]:= SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4],
           {p1 -> member[x,
              image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]],
            p2 → member[x, omega], p3 → not[equal[0, x]], p4 → not[equal[x, set[0]]]}] // MapNotNot

Out[28]= or[member[set[0], x], not[member[x,
             image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]]] == True

In[29]:= or[member[set[0], x_], not[member[x_,
             image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]]] := True
```

Corollary. A number that can be factored into smaller ones must be greater than **1**.

```
In[30]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
             not[implies[p1, p3]], {p3 → member[set[0], x], p2 -> member[x,
                image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]],
              p1 -> member[x, image[NATMUL, cart[x, x]]]}]] // Reverse // MapNotNot

Out[30]= or[member[set[0], x], not[member[x, image[NATMUL, cart[x, x]]]]] == True

In[31]:= or[member[set[0], x_], not[member[x_, image[NATMUL, cart[x_, x_]]]]] := True
```

Theorem. Numbers greater than **1** are either prime or can be factored as a product of smaller numbers.

```
In[32]:= equiv[or[member[x, PRIMES], member[x, image[NATMUL, cart[x, x]]]],
            and[member[x, omega], member[set[0], x]]] // not // not

Out[32]= True

In[33]:= or[member[x_, PRIMES], member[x_, image[NATMUL, cart[x_, x_]]]] :=
          and[member[x, omega], member[set[0], x]]
```

Corollary. A number can not be both prime and composite.

```
In[34]:= SubstTest[member, x, intersection[y, z], {y → PRIMES,
           z -> image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]}]

Out[34]= and[member[x, PRIMES], member[x,
            image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]] == False

In[35]:= and[member[x_, PRIMES], member[x_,
            image[NATMUL, cart[complement[succ[set[0]]], complement[succ[set[0]]]]]]] := False
```

Corollary. Another formulation of the same fact.

```
In[36]:= SubstTest[member, x, intersection[y, z],
          {y → PRIMES, z -> fix[composite[inverse[E], IMAGE[NATMUL], CART, DUP]]}]

Out[36]= and[member[x, PRIMES], member[x, image[NATMUL, cart[x, x]]]] == False

In[37]:= and[member[x_, PRIMES], member[x_, image[NATMUL, cart[x_, x_]]]] := False
```