U[binhom[x,y]]

Johan G. F. Belinfante
2006 November 25

```
In[1]:= SetDirectory["l:"]; << goedel87.25a; << tools.m

        :Package Title: goedel87.25a          2006 November 25 at 7:45 a.m.

        It is now:  2006 Nov 25 at 14:8

        Loading Simplification Rules

        TOOLS.M                       Revised 2006 November 22

        weightlimit = 40
```

summary

This notebook is concerned with the relation **U[binhom[x, y]]**, which can be regarded as a generalization of divisibility. A particular case is:

```
In[2]:= U[binhom[NATADD, NATADD]]

Out[2]= DIV
```

In general, **x** and **y** need not be equal. For example, when **x = NATADD** and **y = INTADD**, one is dealing with a mixed divisibility relation connecting natural numbers and integers. It makes perfectly good sense to say that the integer **+4** is divisible by the natural number **2**, for example: it means that there is a mapping **h** from natural numbers to integers that preserves addition and satisfies **APPLY[h, 2] = +4**.

an upper bound

An upper bound for this generalized divisibility relation is:

```
In[3]:= Map[implies[#, subclass[U[binhom[x, y]], cart[fix[domain[x]], fix[domain[y]]]]] &,
         SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
          {u -> binhom[x, y], v -> map[fix[domain[x]], fix[domain[y]]]}]]

Out[3]= subclass[U[binhom[x, y]], cart[fix[domain[x]], fix[domain[y]]]] == True

In[4]:= subclass[U[binhom[x_, y_]], cart[fix[domain[x_]], fix[domain[y_]]]] := True
```

Corollary.

```
In[5]:= SubstTest[implies, subclass[w, cart[u, v]], subclass[domain[w], u],
          {u -> fix[domain[x]], v -> fix[domain[y]], w -> U[binhom[x, y]]}] // Reverse

Out[5]= subclass[domain[U[binhom[x, y]]], fix[domain[x]]] == True

In[6]:= subclass[domain[U[binhom[x_, y_]]], fix[domain[x_]]] := True
```

Corollary.

```
In[7]:= SubstTest[implies, subclass[w, cart[u, v]], subclass[range[w], v],
          {u -> fix[domain[x]], v -> fix[domain[y]], w -> U[binhom[x, y]]}] // Reverse

Out[7]= subclass[range[U[binhom[x, y]]], fix[domain[y]]] == True

In[8]:= subclass[range[U[binhom[x_, y_]]], fix[domain[y_]]] := True
```

Corollary.

```
In[9]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> U[binhom[x, y]], v -> cart[fix[domain[x]], fix[domain[y]]], w → cart[V, V]}] //
         Reverse

Out[9]= subclass[U[binhom[x, y]], cart[V, V]] == True

In[10]:= subclass[U[binhom[x_, y_]], cart[V, V]] := True
```

Corollary.

```
In[11]:= equal[composite[Id, U[binhom[x, y]]], U[binhom[x, y]]]

Out[11]= True

In[12]:= composite[Id, U[binhom[x_, y_]]] := U[binhom[x, y]]
```

---

a generalized transitive property

The composite of a binary homomorphism from **y** to **z** and a binary homomorphism from **x** to **y** is a binary homomorphism from **x** to **z**. This implies that the following generalization of the transitive property of divisibility holds:

```
In[13]:= SubstTest[implies, subclass[u, v], subclass[U[u], U[v]],
           {u -> image[COMPOSE, cart[binhom[y, z], binhom[x, y]]], v -> binhom[x, z]}] // Reverse

Out[13]= subclass[composite[U[binhom[y, z]], U[binhom[x, y]]], U[binhom[x, z]]] == True

In[14]:= subclass[composite[U[binhom[y_, z_]], U[binhom[x_, y_]]], U[binhom[x_, z_]]] := True
```

Corollary.

```
In[15]:= SubstTest[subclass, composite[U[binhom[y, z]], U[binhom[x, y]]],
           U[binhom[x, z]], {y → x, z → x}] // Reverse

Out[15]= TRANSITIVE[U[binhom[x, x]]] == True

In[16]:= TRANSITIVE[U[binhom[x_, x_]]] := True
```

domain

Lemma.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[p3, p4], implies[and[p2, p4], p5], not[implies[p1, p5]],
            {p1 → member[w, binhom[x, y]], p2 → equal[domain[w], fix[domain[x]]], p3 →
              subclass[w, U[binhom[x, y]]], p4 → subclass[domain[w], domain[U[binhom[x, y]]]],
             p5 → subclass[fix[domain[x]], domain[U[binhom[x, y]]]]}]] // Reverse

Out[17]= or[not[member[w, binhom[x, y]]],
           subclass[fix[domain[x]], domain[U[binhom[x, y]]]]] == True

In[18]:= (% /. {w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

The variable **w** can be eliminated:

```
In[19]:= Map[equal[V, #] &, SubstTest[class, w, or[subclass[u, v], not[member[w, z]]],
           {u -> fix[domain[x]], v -> domain[U[binhom[x, y]]], z -> binhom[x, y]}]]

Out[19]= or[equal[0, binhom[x, y]], subclass[fix[domain[x]], domain[U[binhom[x, y]]]]] == True

In[20]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The conclusion can be sharpened to an equation.

```
In[21]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
          {p → not[empty[binhom[x, y]]], u -> fix[domain[x]], v -> domain[U[binhom[x, y]]]}]

Out[21]= or[equal[0, binhom[x, y]], equal[domain[U[binhom[x, y]]], fix[domain[x]]]] == True

In[22]:= or[equal[0, binhom[x_, y_]], equal[domain[U[binhom[x_, y_]]], fix[domain[x_]]]] := True
```

Lemma.

```
In[23]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> empty[binhom[x, y]], q -> or[equal[0, domain[U[binhom[x, y]]]],
             equal[domain[U[binhom[x, y]]], fix[domain[x]]]]}]

Out[23]= or[equal[0, domain[U[binhom[x, y]]]],
           equal[domain[U[binhom[x, y]]], fix[domain[x]]]] == True

In[24]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

the case x = y

Lemma.

```
In[25]:= SubstTest[implies, member[u, v], subclass[u, U[v]],
           {u → id[fix[domain[x]]], v → binhom[x, x]}] // Reverse

Out[25]= or[not[member[id[fix[domain[x]]], binhom[x, x]]],
           subclass[fix[domain[x]], fix[U[binhom[x, x]]]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed

In[27]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 -> member[x, BINOPS], p2 -> member[id[fix[domain[x]]], binhom[x, x]],
              p3 → not[empty[binhom[x, x]]]}]] // Reverse

Out[27]= or[not[equal[0, binhom[x, x]]], not[member[x, BINOPS]]] == True

In[28]:= or[not[equal[0, binhom[x_, x_]]], not[member[x_, BINOPS]]] := True

In[29]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 → member[x, BINOPS], p2 → not[empty[binhom[x, x]]],
              p3 -> equal[domain[U[binhom[x, x]]], fix[domain[x]]]}]] // Reverse

Out[29]= or[equal[domain[U[binhom[x, x]]], fix[domain[x]]], not[member[x, BINOPS]]] == True

In[30]:= or[equal[domain[U[binhom[x_, x_]]], fix[domain[x_]]], not[member[x_, BINOPS]]] := True
```

Theorem.

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 -> member[x, BINOPS], p2 -> member[id[fix[domain[x]]], binhom[x, x]],
              p3 -> subclass[fix[domain[x]], fix[U[binhom[x, x]]]]}]] // Reverse

Out[31]= or[not[member[x, BINOPS]], subclass[fix[domain[x]], fix[U[binhom[x, x]]]]] == True

In[32]:= or[not[member[x_, BINOPS]], subclass[fix[domain[x_]], fix[U[binhom[x_, x_]]]]] := True
```

Corollary.

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
             {p1 → member[x, BINOPS], p2 -> subclass[fix[domain[x]], fix[U[binhom[x, x]]]],
              p3 -> subclass[fix[domain[x]], range[U[binhom[x, x]]]]}]] // Reverse

Out[33]= or[not[member[x, BINOPS]], subclass[fix[domain[x]], range[U[binhom[x, x]]]]] == True

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

Theorem.

```
In[35]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
          {p → member[x, BINOPS], u -> fix[domain[x]], v -> range[U[binhom[x, x]]]}]

Out[35]= or[equal[fix[domain[x]], range[U[binhom[x, x]]]], not[member[x, BINOPS]]] == True

In[36]:= or[equal[fix[domain[x_]], range[U[binhom[x_, x_]]]], not[member[x_, BINOPS]]] := True
```

Theorem.

```
In[37]:= Map[not, SubstTest[and, implies[and[p2, p4], p5],
            implies[and[p3, p4], p6], not[implies[p1, p7]],
            {p1 → member[x, BINOPS], p2 -> equal[fix[domain[x]], domain[U[binhom[x, x]]]],
             p3 -> equal[fix[domain[x]], range[U[binhom[x, x]]]],
             p4 -> subclass[fix[domain[x]], fix[U[binhom[x, x]]]],
             p5 → subclass[domain[U[binhom[x, x]]], fix[U[binhom[x, x]]]],
             p6 → subclass[range[U[binhom[x, x]]], fix[U[binhom[x, x]]]],
             p7 → REFLEXIVE[U[binhom[x, x]]]}]] // Reverse

Out[37]= or[not[member[x, BINOPS]], REFLEXIVE[U[binhom[x, x]]]] == True

In[38]:= or[not[member[x_, BINOPS]], REFLEXIVE[U[binhom[x_, x_]]]] := True
```

Lemma.

```
In[39]:= SubstTest[implies, and[REFLEXIVE[w], TRANSITIVE[w]],
           idempotent[w], w -> U[binhom[x, x]]] // Reverse

Out[39]= or[equal[composite[U[binhom[x, x]], U[binhom[x, x]]], U[binhom[x, x]]],
           not[REFLEXIVE[U[binhom[x, x]]]]] == True

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

Corollary.

```
In[41]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
            {p1 -> member[x, BINOPS], p2 -> REFLEXIVE[U[binhom[x, x]]],
             p3 -> idempotent[U[binhom[x, x]]]}]] // Reverse

Out[41]= or[equal[composite[U[binhom[x, x]], U[binhom[x, x]]], U[binhom[x, x]]],
           not[member[x, BINOPS]]] == True

In[42]:= or[equal[composite[U[binhom[x_, x_]], U[binhom[x_, x_]]], U[binhom[x_, x_]]],
           not[member[x_, BINOPS]]] := True
```