natmod[x,y] and NATMOD

Johan G. F. Belinfante
2005 May 17

```
In[1]:= SetDirectory["i:"]; << goedel69.17a; << tools.m
```

```
:Package Title: goedel69.17a        2005 May 17 at 3:05 p.m.

It is now:  2005 May 17 at 16:26

Loading Simplification Rules

TOOLS.M                       Revised 2005 April 16

weightlimit = 40
```

summary

The class **natmod[x,y]** is the remainder when a natural number **x** is divided by a natural number **y**, or equal to **V** when **x** or **y** is not a natural number. It is defined by a wrapped membership rule:

```
In[2]:= Begin["Goedel`Private`"];
```

```
In[3]:= FirstMatch[class[u_, member[v_, HoldPattern[natmod[x_, y_]]]]]
```

```
Out[3]= class[u_, member[v_, natmod[x_, y_]]] := Module[{w = Unique[]},
          class[u, and[member[v, V], forall[w, implies[member[w, image[image[
          inverse[NATADD], set[x]], image[DIV, set[y]]]], member[v, w]]]]]]
```

The corresponding function **NATMOD** takes a pair of natural numbers **pair[x,y]** to **natmod[x,y]**.

normalization

Lemma.

```
In[4]:= simplify = False;
```

```
In[5]:= symdif[A[image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]],
          natmod[x, y]] // Normality

Out[5]= union[
          intersection[A[image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]],
           complement[natmod[x, y]]], intersection[
           complement[A[image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]]],
           natmod[x, y]]] == 0

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

In[7]:= simplify = True;
```

Normalization rule for **natmod**.

```
In[8]:= SubstTest[equal, 0, symdif[u, v],
         {u -> A[image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]],
          v -> natmod[x, y]}]

Out[8]= True == equal[
          A[image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]], natmod[x, y]]

In[9]:= A[image[image[inverse[NATADD], set[x_]], image[DIV, set[y_]]]] := natmod[x, y]
```

---

definition of NATMOD and basic properties

The function **NATMOD** is defined by the following equation, which will later be subsumed by a better rewrite rule, removing **VERTSECT** and the factor **SWAP**.

```
In[10]:= composite[VERTSECT[complement[composite[complement[inverse[E]],
              rotate[composite[inverse[DIV], rotate[NATADD]]]]]], SWAP] := NATMOD
```

Before rewriting this definition, some basic properties of **NATMOD** will be derived. That it is a function is immediately evident from the definition:

```
In[11]:= SubstTest[FUNCTION, flip[VERTSECT[x]],
          x -> complement[composite[complement[inverse[E]],
             rotate[composite[inverse[DIV], rotate[NATADD]]]]]]

Out[11]= FUNCTION[NATMOD] == True

In[12]:= FUNCTION[NATMOD] := True
```

A formula for the domain is easy to derive.

```
In[13]:= IminComp[VERTSECT[complement[composite[complement[inverse[E]],
             rotate[composite[inverse[DIV], rotate[NATADD]]]]]], SWAP, V]

Out[13]= domain[NATMOD] == cart[omega, omega]

In[14]:= domain[NATMOD] := cart[omega, omega]
```

Corollary.

```
In[15]:= Assoc[VERTSECT[complement[composite[complement[inverse[E]],
              rotate[composite[inverse[DIV], rotate[NATADD]]]]]],
           SWAP, id[cart[V, V]]] // Reverse

Out[15]= composite[NATMOD, id[cart[V, V]]] == NATMOD

In[16]:= composite[NATMOD, id[cart[V, V]]] := NATMOD
```

---

sethood rule for natmod

```
In[17]:= SubstTest[equal, V, A[z],
          z -> image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]]

Out[17]= equal[V, natmod[x, y]] == or[not[member[x, omega]], not[member[y, omega]]]

In[18]:= equal[V, natmod[x_, y_]] := or[not[member[x, omega]], not[member[y, omega]]]

In[19]:= SubstTest[member, A[z], V,
          z -> image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]]

Out[19]= member[natmod[x, y], V] == and[member[x, omega], member[y, omega]]

In[20]:= member[natmod[x_, y_], V] := and[member[x, omega], member[y, omega]]
```

Lemma.

```
In[21]:= SubstTest[A, image[image[inverse[NATADD], set[w]], image[DIV, set[z]]],
           w → union[x, complement[image[V, y]]]] // Reverse

Out[21]= natmod[union[x, complement[image[V, y]]], z] ==
          union[complement[image[V, y]], natmod[x, z]]

In[22]:= natmod[union[x_, complement[image[V, y_]]], z_] :=
          union[complement[image[V, y]], natmod[x, z]]
```

Lemma.

```
In[23]:= equiv[and[equal[z, natmod[x, y]], member[z, V]],
           and[equal[z, natmod[x, y]], member[x, omega], member[y, omega]]]

Out[23]= True

In[24]:= and[equal[z_, natmod[x_, y_]], member[z_, V]] :=
           and[equal[z, natmod[x, y]], member[x, omega], member[y, omega]]
```

---

membership rule for NATMOD

A wrapped form of membership rule.

```
In[25]:= Map[equal[V, #] &, SubstTest[image, V, intersection[
             composite[VERTSECT[complement[composite[complement[inverse[E]],
                 rotate[composite[inverse[v], rotate[NATADD]]]]]], SWAP],
             set[w]], {v → DIV, w → pair[pair[x, y], z]}]]

Out[25]= member[pair[pair[x, y], z], NATMOD] ==
           and[equal[z, natmod[x, y]], member[x, omega], member[y, omega]]

In[26]:= member[pair[pair[x_, y_], z_], NATMOD] :=
           and[equal[z, natmod[x, y]], member[x, omega], member[y, omega]]
```

---

rewriting the definition

The first step toward replacing the definition with a stronger rewrite rule is to remove the factor **SWAP**.

```
In[27]:= Assoc[VERTSECT[complement[composite[complement[inverse[E]],
             rotate[composite[inverse[DIV], rotate[NATADD]]]]]], SWAP, SWAP]

Out[27]= composite[VERTSECT[complement[composite[complement[inverse[E]],
             rotate[composite[inverse[DIV], rotate[NATADD]]]]]],
           id[cart[V, V]]] == composite[NATMOD, SWAP]

In[28]:= % /. Equal → SetDelayed

In[29]:= Assoc[VERTSECT[complement[composite[complement[inverse[E]],
             rotate[composite[inverse[DIV], rotate[NATADD]]]]]],
           id[cart[omega, omega]], id[cart[V, V]]]

Out[29]= VERTSECT[complement[composite[complement[inverse[E]], rotate[
             composite[inverse[DIV], rotate[NATADD]]]]]] == composite[NATMOD, SWAP]

In[30]:= % /. Equal → SetDelayed
```

Next, one can dig inside the **VERTSECT**.

```
In[31]:= SubstTest[composite, complement[inverse[E]],
           VERTSECT[x], x -> complement[composite[complement[inverse[E]],
              rotate[composite[inverse[DIV], rotate[NATADD]]]]]] // Reverse

Out[31]= composite[complement[inverse[E]],
           rotate[composite[inverse[DIV], rotate[NATADD]]]] ==
         composite[complement[inverse[E]], NATMOD, SWAP]

In[32]:= composite[complement[inverse[E]],
           rotate[composite[inverse[DIV], rotate[NATADD]]]] :=
         composite[complement[inverse[E]], NATMOD, SWAP]

In[33]:= Map[complement, ImageComp[complement[inverse[E]],
           rotate[composite[inverse[DIV], rotate[NATADD]]], cart[set[y], set[x]]]]

Out[33]= APPLY[NATMOD, PAIR[x, y]] == natmod[x, y]

In[34]:= APPLY[NATMOD, PAIR[x_, y_]] := natmod[x, y]

In[35]:= SubstTest[image, funpart[z], set[w], {z → NATMOD, w → PAIR[x, y]}]

Out[35]= image[NATMOD, cart[set[x], set[y]]] == set[natmod[x, y]]

In[36]:= image[NATMOD, cart[set[x_], set[y_]]] := set[natmod[x, y]]
```

---

range

```
In[37]:= SubstTest[implies, subclass[w, omega], or[equal[0, w], member[A[w], omega]],
          w -> image[image[inverse[NATADD], set[x]], image[DIV, set[y]]]]

Out[37]= or[member[natmod[x, y], omega],
           not[member[x, omega]], not[member[y, omega]]] == True

In[38]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

In the other direction, one has:

```
In[39]:= SubstTest[implies, member[w, omega], member[w, V], w → natmod[x, y]]

Out[39]= or[and[member[x, omega], member[y, omega]],
           not[member[natmod[x, y], omega]]] == True

In[40]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Putting these two fact together yields a simple rewrite rule:

```
In[41]:= equiv[member[natmod[x, y], omega], and[member[x, omega], member[y, omega]]]

Out[41]= True

In[42]:= member[natmod[x_, y_], omega] := and[member[x, omega], member[y, omega]]

In[43]:= Map[equal[0, composite[Id, complement[#]]] &, SubstTest[class, pair[x, y],
           subclass[image[w, cart[set[x], set[y]]], omega], w → NATMOD]] // Reverse

Out[43]= subclass[cart[omega, omega], image[inverse[NATMOD], omega]] == True

In[44]:= % /. Equal → SetDelayed
```

Corollary.

```
In[45]:= SubstTest[implies,
          and[FUNCTION[x], subclass[domain[x], image[inverse[x], y]]],
          subclass[range[x], y], {x → NATMOD, y → omega}]

Out[45]= subclass[range[NATMOD], omega] == True

In[46]:= % /. Equal → SetDelayed

In[47]:= (composite[VERTSECT[complement[composite[
                complement[inverse[E]], rotate[composite[inverse[v], rotate[u]]]]]],
             LEFT[0]] // RelnNormality) /. {u → NATADD, v → DIV}

Out[47]= composite[NATMOD, RIGHT[0]] == id[omega]

In[48]:= composite[NATMOD, RIGHT[0]] := id[omega]

In[49]:= Map[subclass[#, range[NATMOD]] &, ImageComp[NATMOD, RIGHT[0], V]]

Out[49]= subclass[omega, range[NATMOD]] == True

In[50]:= % /. Equal → SetDelayed

In[51]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → omega, v → range[NATMOD]}]

Out[51]= True == equal[omega, range[NATMOD]]

In[52]:= range[NATMOD] := omega
```

---

x mod 0 = x

```
In[53]:= natmod[x, 0] // Normality

Out[53]= natmod[x, 0] == union[x, complement[image[V, intersection[omega, set[x]]]]]

In[54]:= natmod[x_, 0] := union[x, complement[image[V, intersection[omega, set[x]]]]]
```

0 mod x = 0

```
In[55]:= natmod[0, x] // Normality
Out[55]= natmod[0, x] == complement[image[V, intersection[omega, set[x]]]]

In[56]:= natmod[0, x_] := complement[image[V, intersection[omega, set[x]]]]

In[57]:= composite[NATMOD, LEFT[0]] // RelnNormality
Out[57]= composite[NATMOD, LEFT[0]] == cart[omega, set[0]]

In[58]:= composite[NATMOD, LEFT[0]] := cart[omega, set[0]]
```

DIV and NATMOD

```
In[59]:= SubstTest[implies, and[subclass[w, omega], equal[0, A[w]]], member[0, w],
          w -> image[image[inverse[NATADD], set[y]], image[DIV, set[x]]]]
Out[59]= or[member[pair[x, y], DIV], not[equal[0, natmod[y, x]]]] == True

In[60]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

In the opposite direction, one has:

```
In[61]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → set[0], v → image[image[inverse[NATADD], set[x]], image[DIV, set[y]]],
           w → complement[inverse[E]]}]
Out[61]= or[equal[0, natmod[x, y]], not[member[pair[y, x], DIV]]] == True

In[62]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The two implications can be combined to obtain a rewrite rule:

```
In[63]:= equiv[equal[0, natmod[x, y]], member[pair[y, x], DIV]]
Out[63]= True

In[64]:= equal[0, natmod[x_, y_]] := member[pair[y, x], DIV]
```

Lemma.

```
In[65]:= IminComp[NATMOD, id[cart[V, V]], x] // Reverse
Out[65]= composite[Id, image[inverse[NATMOD], x]] == image[inverse[NATMOD], x]

In[66]:= composite[Id, image[inverse[NATMOD], x_]] := image[inverse[NATMOD], x]
```

```
In[67]:= image[inverse[NATMOD], set[0]] // VSNormality
Out[67]= image[inverse[NATMOD], set[0]] == inverse[DIV]
In[68]:= image[inverse[NATMOD], set[0]] := inverse[DIV]
```

---

x mod 1 = 0

```
In[69]:= equal[natmod[x, set[0]], complement[image[V, intersection[omega, set[x]]]]]
Out[69]= True
In[70]:= natmod[x_, set[0]] := complement[image[V, intersection[omega, set[x]]]]
```

Variable-free:

```
In[71]:= composite[NATMOD, RIGHT[set[0]]] // RelnNormality
Out[71]= composite[NATMOD, RIGHT[set[0]]] == cart[omega, set[0]]
In[72]:= composite[NATMOD, RIGHT[set[0]]] := cart[omega, set[0]]
```

---

1 mod x = 1 except when x = 1

```
In[73]:= Map[equal[set[0], #] &, natmod[set[0], nat[x]] // Normality]
Out[73]= equal[natmod[set[0], nat[x]], set[0]] == not[equal[nat[x], set[0]]]
In[74]:= equal[natmod[set[0], nat[x_]], set[0]] := not[equal[nat[x], set[0]]]
```

---

lambda[pair[x,y], natmod[x,y]] = NATMOD

```
In[75]:= Map[flip, composite[complement[inverse[E]],
            rotate[NATADD], SWAP, cross[DIV, Id]] // TripleRotate]
Out[75]= composite[complement[inverse[E]], rotate[NATADD], cross[Id, DIV]] ==
          composite[complement[inverse[E]], NATMOD]
In[76]:= composite[complement[inverse[E]], rotate[NATADD], cross[Id, DIV]] :=
          composite[complement[inverse[E]], NATMOD]
```

Lemma.

```
In[77]:= composite[SECOND, intersection[composite[inverse[NATADD], FIRST],
           composite[inverse[FIRST], x, SECOND]]] // TripleRotate

Out[77]= composite[SECOND, intersection[
           composite[inverse[NATADD], FIRST], composite[inverse[FIRST], x, SECOND]]] ==
         composite[rotate[NATADD], cross[Id, x]]

In[78]:= composite[SECOND, intersection[composite[inverse[NATADD], FIRST],
           composite[inverse[FIRST], x_, SECOND]]] :=
         composite[rotate[NATADD], cross[Id, x]]
```

The **lambda** rule is derivable as follows.

```
In[79]:= A[image[image[inverse[u], set[x]], image[v, set[y]]]] /. {u → NATADD, v → DIV}

Out[79]= natmod[x, y]

In[80]:= lambda[pair[x, y], A[image[image[inverse[u], set[x]], image[v, set[y]]]]] /.
          {u → NATADD, v → DIV}

Out[80]= NATMOD
```

---

reify rule

```
In[81]:= SubstTest[reify, x,
          A[image[image[inverse[u], set[f[x]]], image[v, set[g[x]]]],
          {u → NATADD, v → DIV}]

Out[81]= reify[x, natmod[f[x], g[x]]] ==
          composite[Id, complement[composite[complement[inverse[E]], SECOND,
             intersection[composite[inverse[NATADD], VERTSECT[reify[x, f[x]]]],
              composite[inverse[FIRST], DIV, VERTSECT[reify[x, g[x]]]]]]]]

In[82]:= reify[x_, natmod[y_, z_]] :=
          composite[Id, complement[composite[complement[inverse[E]], SECOND,
             intersection[composite[inverse[NATADD], VERTSECT[reify[x, y]]],
              composite[inverse[FIRST], DIV, VERTSECT[reify[x, z]]]]]]]
```