image[DIV, x] rules

Johan G. F. Belinfante and Ming Li
2007 January 27

```
In[1]:= SetDirectory["l:"]; << goedel89.25a; << tools.m

        :Package Title: goedel89.25a        2007 January 25 at 10:30 a.m.

        It is now:  2007 Jan 27 at 15:3

        Loading Simplification Rules

        TOOLS.M                       Revised 2007 January 7

        weightlimit = 40
```

summary

Rewrite rules are derived for the conditions **equal[omega, image[DIV, x]]** and **member[x, image[DIV, y]]**.

member[x, image[DIV, y]] rules

Observation: The number **x** is a multiple of some member of **y** if and only if some divisor of **x** belongs to **y**.

```
In[2]:= disjoint[image[inverse[DIV], set[x]], y]

Out[2]= not[member[x, image[DIV, y]]]
```

When the divisors of a natural number are known, this rule can be turned around. For example, the set of divisors of **0** is the set of all natural numbers. Accordingly, one has:

```
In[3]:= Map[not, SubstTest[disjoint, image[inverse[DIV], set[w]], x, w → 0]]

Out[3]= member[0, image[DIV, x]] == not[equal[0, intersection[omega, x]]]
```

Comment. An application of **AssertTest** suffices to derive this rewrite rule.

```
In[4]:= member[0, image[DIV, x]] // AssertTest

Out[4]= member[0, image[DIV, x]] == not[equal[0, intersection[omega, x]]]

In[5]:= member[0, image[DIV, x_]] := not[equal[0, intersection[omega, x]]]
```

The only divisor of **1** is itself. Therefore, one has:

```
In[6]:= Map[not, SubstTest[disjoint, image[inverse[DIV], set[w]], x, w → set[0]]]

Out[6]= member[set[0], image[DIV, x]] == member[set[0], x]
```

Here again, one could also derive the same result using **AssertTest**.

```
In[7]:= member[set[0], image[DIV, x]] // AssertTest

Out[7]= member[set[0], image[DIV, x]] == member[set[0], x]

In[8]:= member[set[0], image[DIV, x_]] := member[set[0], x]
```

Since **2** is a prime, its only divisors are **1** and **2**. (Comment. In this case **AssertTest** does not suffice.)

```
In[9]:= SubstTest[member, t, complement[P[complement[image[inverse[DIV], set[y]]]]],
          y → succ[set[0]]] /. t → intersection[omega, x]

Out[9]= member[succ[set[0]], image[DIV, x]] == or[member[set[0], x], member[succ[set[0]], x]]

In[10]:= member[succ[set[0]], image[DIV, x_]] := or[member[set[0], x], member[succ[set[0]], x]]
```

Lemma. Since **3** is a prime, its only divisors are **1** and **3**. This can be made into a rewrite rule.

```
In[11]:= equal[image[inverse[DIV], set[succ[succ[set[0]]]]], set[set[0], succ[succ[set[0]]]]]

Out[11]= True

In[12]:= image[inverse[DIV], set[succ[succ[set[0]]]]] := set[set[0], succ[succ[set[0]]]]
```

Theorem. The number **3** is a multiple of some member of **x** if and only if either **1** or **3** belongs to **x**.

```
In[13]:= SubstTest[member, t, complement[P[complement[image[inverse[DIV], set[y]]]]],
           y → succ[succ[set[0]]]] /. t → intersection[omega, x]

Out[13]= member[succ[succ[set[0]]], image[DIV, x]] ==
          or[member[set[0], x], member[succ[succ[set[0]]], x]]

In[14]:= member[succ[succ[set[0]]], image[DIV, x_]] :=
          or[member[set[0], x], member[succ[succ[set[0]]], x]]
```

More generally: a prime **x** is a multiple of some member of **y** if and only if either **1** or **x** belongs to **y**.

```
In[15]:= SubstTest[implies, and[equal[z, image[inverse[DIV], set[x]]], disjoint[y, z]],
            not[member[x, image[DIV, y]]], z → set[set[0], x]] // Reverse // MapNotNot

Out[15]= or[member[x, y], member[set[0], y],
           not[member[x, PRIMES]], not[member[x, image[DIV, y]]]] == True

In[16]:= or[member[x_, y_], member[set[0], y_],
           not[member[x_, PRIMES]], not[member[x_, image[DIV, y_]]]] := True
```

equal[omega, image[DIV, x]]

Lemma.

```
In[17]:= SubstTest[implies, subclass[u, v],
           subclass[image[w, u], image[w, v]], {u → set[set[0]] , v → x, w → DIV}] // Reverse

Out[17]= or[not[member[set[0], x]], subclass[omega, image[DIV, x]]] == True

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

Theorem.

```
In[19]:= SubstTest[and, implies[p, subclass[u, v]], subclass[v, u],
          {p -> member[set[0], x], u -> omega, v -> image[DIV, x]}]

Out[19]= or[equal[omega, image[DIV, x]], not[member[set[0], x]]] == True

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

Lemma.

```
In[21]:= SubstTest[implies, and[member[u, v], equal[v, w]], member[u, w],
           {u → set[0], v → omega, w → image[DIV, x]}] // Reverse

Out[21]= or[member[set[0], x], not[equal[omega, image[DIV, x]]]] == True

In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[23]:= equiv[equal[omega, image[DIV, x]], member[set[0], x]]

Out[23]= True

In[24]:= equal[omega, image[DIV, x_]] := member[set[0], x]
```

Corollary.

```
In[25]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → omega, v → image[DIV, x]}] // Reverse

Out[25]= subclass[omega, image[DIV, x]] == member[set[0], x]

In[26]:= subclass[omega, image[DIV, x_]] := member[set[0], x]
```

variable-free formulations

Theorem.

In[27]:= **image[inverse[IMAGE[DIV]], set[omega]] // Normality**

Out[27]= image[inverse[IMAGE[DIV]], set[omega]] == complement[P[complement[set[set[0]]]]]

In[28]:= **image[inverse[IMAGE[DIV]], set[omega]] := complement[P[complement[set[set[0]]]]]**

Corollary

In[29]:= **image[inverse[VERTSECT[DIV]], set[omega]] // Normality**

Out[29]= image[inverse[VERTSECT[DIV]], set[omega]] == set[set[0]]

In[30]:= **image[inverse[VERTSECT[DIV]], set[omega]] := set[set[0]]**