subclass[natmod[x,y], x]

Johan G. F. Belinfante and Claudia Huang
2005 May 27

In[1]:=
```
SetDirectory["i:"]; << goedel69.26a; << tools.m
```

```
:Package Title: goedel69.26a       2005 May 26 at 9:05 a.m.

It is now:  2005 May 27 at 9:36

Loading Simplification Rules

TOOLS.M                        Revised 2005 May 17

weightlimit = 40
```

---

summary

For natural numbers, **natmod[x,y]** is a subclass of the first argument **x**. In this notebook, existing rewrite rules for this statement are generalized, removing the condition that **x** and **y** be natural numbers.

---

a variant of an existing rule

For natural numbers, **natmod[x,y]** is less than or equal to the first argument **x**.

In[2]:=
```
subclass[natmod[nat[x], nat[y]], nat[x]]
```

Out[2]= True

For natural numbers, inclusion can be replaced with membership. This yields the following variant:

In[3]:=
```
Map[not, SubstTest[subclass, nat[z], nat[x], z → natmod[nat[x], nat[y]]]] //
 Reverse
```

Out[3]= member[nat[x], natmod[nat[x], nat[y]]] == False

In[4]:=
```
member[nat[x_], natmod[nat[x_], nat[y_]]] := False
```

Both of these rewrite rules will be generalized, removing the **nat** wrappers.

generalizing the subclass rule

For the **subclass** statement, the following wrapper-free statement is known:

```
In[5]:= implies[and[member[x, omega], member[y, omega]], subclass[natmod[x, y], x]]

Out[5]= True
```

A converse implication can be derived using the following rewrite rule.

```
In[6]:= and[equal[v, V], subclass[v, x]]

Out[6]= and[equal[v, V], equal[V, x]]
```

This yields the following implication.

```
In[7]:= Map[implies[#, equal[V, x]] &,
         SubstTest[and, equal[v, V], subclass[v, x], v → natmod[x, y]]]

Out[7]= or[and[member[x, omega], member[y, omega]],
           equal[V, x], not[subclass[natmod[x, y], x]]] == True

In[8]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Combining the two implications yields a logical equivalence, which can be made into a new rewrite rule.

```
In[9]:= equiv[subclass[natmod[x, y], x],
           or[equal[x, V], and[member[x, omega], member[y, omega]]]] // not // not

Out[9]= True
```

This rewrite rule subsumes the existing wrapper-free rule, which can be removed.

```
In[10]:= subclass[natmod[x_, y_], x_] :=
          or[and[member[x, omega], member[y, omega]], equal[V, x]]
```

generalizing the member rule

The **nat** wrappers are removed from the **member** rule as follows:

```
In[11]:= SubstTest[implies, and[equal[u, nat[x]], equal[v, nat[y]]],
          not[member[u, natmod[u, v]]], {x → u, y → v}]

Out[11]= or[not[member[u, omega]],
            not[member[u, natmod[u, v]]], not[member[v, omega]]] == True

In[12]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed
```

Restatement:

```
In[13]:= implies[member[x, natmod[x, y]],
          or[not[member[x, omega]], not[member[y, omega]]]]

Out[13]= True
```

In the reverse direction, one has:

```
In[14]:= SubstTest[implies, and[member[x, V], equal[v, V]],
          member[x, v], v → natmod[x, y]]

Out[14]= or[and[member[x, omega], member[y, omega]],
            member[x, natmod[x, y]], not[member[x, V]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Combining the statements derived above yields a logical equivalence, which can be made into a new rewrite rule.

```
In[16]:= equiv[member[x, natmod[x, y]],
          and[member[x, V], or[not[member[x, omega]], not[member[y, omega]]]]]

Out[16]= True

In[17]:= member[x_, natmod[x_, y_]] := or[and[member[x, V], not[member[x, omega]]],
            and[member[x, V], not[member[y, omega]]]]
```