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]]


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 \[Function\] 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:

```plaintext
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.

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

This yields the following implication.

```plaintext
In[7]:= Map[implies[#, equal[V, x]] &,
    SubstTest[and, equal[v, V], subclass[v, x], v \rightarrow natmod[x, y]]]
Out[7]= or[and[member[x, omega], member[y, omega]],
    equal[V, x], not[subclass[natmod[x, y], x]]] \rightarrow True
```

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

```plaintext
In[8]:= (% /.(x \rightarrow x_, y \rightarrow y_)) /. Equal \rightarrow SetDelayed
```

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

```plaintext
In[9]:= 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] :=  
\text{SubstTest}[\text{implies}, \text{and}[\text{equal}[u, \text{nat}[x]], \text{equal}[v, \text{nat}[y]]],
\text{not}[\text{member}[u, \text{natmod}[u, v]]], \{x \mapsto u, y \mapsto v\}]

Out[11] =  
or[\text{not}[, \text{member}[u, \text{omega}]]],
\text{not}[\text{member}[u, \text{natmod}[u, v]]], \text{not}[\text{member}[v, \text{omega}]]] = \text{True}

In[12] :=  
\% / . \{u \mapsto u\_\-, v \mapsto v\_\} / \text{. Equal} \to \text{SetDelayed}

Restatement:

In[13] :=  
\text{implies}[\text{member}[x, \text{natmod}[x, y]],
\text{or}[\text{not}[, \text{member}[x, \text{omega}]], \text{not}[\text{member}[y, \text{omega}]]]]

Out[13] =  \text{True}

In the reverse direction, one has:

In[14] :=  
\text{SubstTest}[\text{implies}, \text{and}[\text{member}[x, v], \text{equal}[v, V]],
\text{member}[x, v], v \mapsto \text{natmod}[x, y]]

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

In[15] :=  
\% / . \{x \mapsto x\_\-, y \mapsto y\_\} / \text{. Equal} \to \text{SetDelayed}

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

In[16] :=  
\text{equiv}[\text{member}[x, \text{natmod}[x, y]],
\text{and}[\text{member}[x, V], \text{or}[\text{not}[, \text{member}[x, \text{omega}]], \text{not}[\text{member}[y, \text{omega}]]]]]

Out[16] =  \text{True}

In[17] :=  
\text{member}[x\_\-, \text{natmod}[x\_\-, y\_\-]] := \text{or}[\text{and}[, \text{member}[x, V], \text{not}[\text{member}[x, \text{omega}]]],
\text{and}[\text{member}[x, V], \text{not}[\text{member}[y, \text{omega}]]]]}