definition of NATMUL and a doubling formula

Johan G. F. Belinfante
2002 July 9

There is a simple formula for the function NATADD corresponding to addition of natural numbers:

```
composite[id[omega], rotate[inverse[power[SUCC]]], SWAP]
```

NATADD
An extensive body of facts about \texttt{NATADD} has been derived (using hand-guided proofs) in the \texttt{GOEDEL} program, and this theory is now ready for work using \texttt{Otter}. This is the subject of other notebooks and will not be repeated here.

\section*{proposed formula for NATMUL}

The situation for multiplication is slightly less satisfactory. The best definition found so far is this:

\begin{verbatim}
muldefn :=
  Equal[NATMUL, composite[rotate[composite[complement[composite[composite[composite[
  rotate[composite[inverse[power[SUCC]], SWAP]], RIF, cross[SECOND,
  composite[SWAP, SECOND]], id[composite[id[cart[omega, V]], inverse[FIRST],
  SUCC, FIRST]]], cross[inverse[E], Id]]], id[composite[inverse[E],
  IMAGE[id[composite[cart[singleton[0], Id]]]]], inverse[FIRST]],
  inverse[IMAGE[cross[Id, inverse[LEFT[0]]]]]], E]], id[cart[omega, V]]]]
\end{verbatim}

Despite the apparent complexity of this definition, it is feasible as a starting point. The first step in using this definition is to derive from it a formula for the composite of \texttt{NATMUL} with \texttt{LEFT[x]}.

\begin{verbatim}
Map[composite[#, LEFT[x]] & muldefn]
  composite[id[image[V, intersection[omega, singleton[x]]]],
     iterate[iterate[iterate[SUCC, singleton[x]], singleton[0]]] ==
     composite[iterate[iterate[iterate[SUCC, singleton[x]], singleton[0]]],
     id[image[V, intersection[omega, singleton[x]]]]]]
\end{verbatim}

These are equal because \texttt{id[image[V,z]]} commutes with every relation: the \texttt{GOEDEL} program recognizes the truth of this when asked:

\begin{verbatim}
% /. Equal -> equal
True
\end{verbatim}

From the formula for \texttt{composite[NATMUL,LEFT[x]]}, it is easy to derive the membership rule:

\begin{verbatim}
member[x, NATMUL]
  and[member[first[first[x]], omega], member[pair[second[first[x]], second[x]],
  iterate[iterate[SUCC, singleton[first[first[x]]]], singleton[0]]]]
\end{verbatim}

In the \texttt{GOEDEL} program, this membership rule has been used as the starting point for the theory of multiplication. At this point only a few facts have been derived:

\begin{verbatim}
FUNCTION[NATMUL]
  True

domain[NATMUL]
  cart[omega, omega]

range[NATMUL]
  omega
\end{verbatim}

Multiplication examples:
A formula for doubling.

The formula $2x = x + x$ will be proved as follows. First we derive a duplication formula for addition:

\[
\begin{align*}
\text{Assoc} & : \text{NATADD, cross}[\text{SUCC, Id}], \text{cross}[\text{Id, SUCC}] \\
\text{composite} & : \text{NATADD, cross}[\text{SUCC, SUCC}] = \text{composite}[\text{SUCC, SUCC, NATADD}] \\
\text{composite} & : \text{NATADD, cross}[\text{SUCC, SUCC}] = : \text{composite}[\text{SUCC, SUCC, NATADD}] \\
\text{Assoc} & : \text{NATADD, cross}[\text{SUCC, SUCC}], \text{DUP} \\
\text{composite} & : \text{NATADD, DUP, SUCC} = \text{composite}[\text{SUCC, SUCC, NATADD}, \text{DUP}] \\
\text{composite} & : \text{NATADD, DUP, SUCC} = : \text{composite}[\text{SUCC, SUCC, NATADD}, \text{DUP}] \\
\text{ImageComp} & : \text{NATADD, RIGHT}[0], \text{singleton}[0] // Reverse \\
\text{image} & : \text{NATADD, cart}[\text{singleton}[0], \text{singleton}[0]] = \text{singleton}[0] \\
\text{image} & : \text{NATADD, cart}[\text{singleton}[0], \text{singleton}[0]] = : \text{singleton}[0]
\end{align*}
\]

The uniqueness of \text{iterate} implies:

\[
\begin{align*}
\text{SubstTest} & : \text{implies, and}[\text{equal}[\text{image}[w, \text{singleton}[0]], v], \\
& \quad \text{equal}[\text{composite}[u, w], \text{composite}[w, \text{SUCC}]]], \\
& \quad \text{equal}[\text{composite}[w, \text{id[omega]]}, \text{iterate}[u, v]], \\
& \quad \{u \rightarrow \text{composite}[\text{SUCC, SUCC}], v \rightarrow \text{singleton}[0], \\
& \quad w \rightarrow \text{composite}[\text{NATADD, DUP}])
\end{align*}
\]

\[
\text{equal}[\text{composite}[\text{NATADD, DUP}], \text{iterate}[\text{composite}[\text{SUCC, SUCC}], \text{singleton}[0]]] = \text{True}
\]

This justifies the rule:

\[
\text{iterate}[\text{composite}[\text{SUCC, SUCC}], \text{singleton}[0]] = : \text{composite}[\text{NATADD, DUP}]
\]

On the multiplication side, we derive these lemmas:

\[
\begin{align*}
\text{Assoc} & : \text{id[omega], NATMUL, LEFT}[\text{succ}[\text{singleton}[0]]] \\
\text{composite} & : \text{id[omega], iterate}[\text{composite}[\text{id[omega]}, \text{SUCC, SUCC}], \text{singleton}[0]] = \text{iterate}[\text{composite}[\text{id[omega]}, \text{SUCC, SUCC}], \text{singleton}[0]] \\
\text{composite} & : \text{id[omega], iterate}[\text{composite}[\text{id[omega]}, \text{SUCC, SUCC}], \text{singleton}[0]] = : \text{iterate}[\text{composite}[\text{id[omega]}, \text{SUCC, SUCC}], \text{singleton}[0]]
\end{align*}
\]
Map[subclass[#, omega] &,
  ImageComp[id[omega]], iterate[composite[id[omega]], SUCC, SUCC], singleton[0]], V]]
subclass[range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]], omega] == True
subclass[range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]], omega] := True
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
  {u -> range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]],
   v -> omega, w -> SUCC}]
subclass[image[SUCC, range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]]],
   omega] == True
subclass[image[SUCC, range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]]],
   omega] := True
subclass[image[SUCC, range[iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]]],
   omega] := True

The last step is to use uniqueness of iterate again:

SubstTest[implies, and[equal[image[w, singleton[0]], v]],
   equal[composite[u, w], composite[w, SUCC]]],
   equal[composite[w, id[omega]], iterate[u, v]],
   {u -> composite[SUCC, SUCC], v -> singleton[0],
    w -> iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]}]

equal[composite[NATADD, DUP],
   iterate[composite[id[omega]], SUCC, SUCC], singleton[0]]] == True

iterate[composite[id[omega]], SUCC, SUCC], singleton[0]] := composite[NATADD, DUP]

This says that $2x = x + x$:

composite[NATMUL, LEFT[succ[singleton[0]]]]

composite[NATADD, DUP]