summary

Multiplication by a non-zero integer is one-to-one. The rewrite rule derived in this notebook is already available in the following variable-free form.

Before the \texttt{int[x]} wrapper was introduced it was hard to do integer arithmetic using variables, and therefore variable-free statements were the preferred formulation of theorems about integers. The following quick derivation of the first major theorem is possible.

This result will be rederived from scratch using the \texttt{int} wrapper because in doing so other useful rewrite rules are discovered. A more general formulation of the above result without the \texttt{int} wrapper is also derived.

derivation

The starting point for the new derivation is the basic fact that the equation \(x \cdot y = 0\) holds if and only if \(x = 0\) or \(y = 0\). Recall that the integer zero is \texttt{plus[0] = id[omega]}. Using \texttt{int} wrappers to denote integers, this fact is available in the following form:

\begin{verbatim}
In[4]:= equal[intmul[int[x], int[y]], id[omega]]
Out[4]= or[equal[id[omega], int[x]], equal[id[omega], int[y]]]
\end{verbatim}
The idea behind the proof is standard: if $x \cdot y = x \cdot z$, then $x \cdot (y - z) = 0$. This implies $x = 0$ or $0 = y - z$, and hence $x = 0$ or $y = z$. The rewrite rules in the GOEDEL program automate the distributive law and the transpositions in this proof. Eliminating the variables $y$ and $z$ yields a rewrite rule for the statement that multiplication by $x$ is one-to-one if and only if $x$ is not zero.

Lemma. The product $x \cdot (-y)$ can be rewritten as $-(x \cdot y)$.

Theorem. (Cancellation law) The equation $x \cdot y = x \cdot z$ holds if and only if $x = 0$ or $y = z$.

Comment. Since intmul has the attribute Orderless, the commutative law is automatic, and so one does not need a separate rule for $(-x) \cdot y$.

Because the GOEDEL program currently lacks a reify rule for the int wrapper, this wrapper must be removed to prepare for the elimination of the variables $y$ and $z$. There is however no need to remove the int wrapper for the variable $x$.

Lemma. (Replacement of two int wrappers by integer membership literals.)

Lemma. (This could also be derived using AssertTest, but that takes a little longer.)
In[12]:  = SubstTest[member, pair[y, z],
    composite[inverse[funpart[t]], funpart[t]], t \rightarrow inttimes[int[x]]] // Reverse

Out[12]= member[pair[y, z], composite[inverse[inttimes[int[x]]], inttimes[int[x]]]] =
    and[equal[intmul[y, int[x]], intmul[z, int[x]]], member[z, Z]]

In[13]:  = member[pair[y_, z_], composite[inverse[inttimes[int[x_]]], inttimes[int[x_]]]] :=
    and[equal[intmul[y, int[x]], intmul[z, int[x]]], member[z, Z]]

Lemma. Elimination of the variables y and z.

In[14]:  = Map[empty[composite[Id, complement[#]]] & \& SubstTest[class, pair[y, z],
    implies[and[not[equal[t, w]], member[y, Z], member[z, Z], member[pair[y, z], s]],
    equal[y, z], {s \rightarrow composite[inverse[inttimes[int[x]]], inttimes[int[x]]],
    t \rightarrow int[x], w \rightarrow id[omega]]}]

Out[14]= or[FUNCTION[inverse[inttimes[int[x]]]], subclass[omega, fix[int[x]]]] = True

In[15]:  = (\% /. x \rightarrow x_) /. Equal \rightarrow SetDelayed

A lemma is needed to recognize the variant of the equation \( x = 0 \) in the above statement.

Lemma.

In[16]:  = SubstTest[implies, subclass[int[t], int[x]],
    equal[int[t], int[x]], t \rightarrow id[omega]] // Reverse

Out[16]= or[equal[id[omega], int[x]], not[subclass[omega, fix[int[x]]]]] = True

In[17]:  = or[equal[id[omega], int[x_]], not[subclass[omega, fix[int[x_]]]]] := True

Comment. The reverse implication also holds, but unfortunately this cannot be made into a rewrite rule because doing so would lead to looping.

In[18]:  = equiv[subclass[omega, fix[int[x]]], equal[id[omega], int[x]]]

Out[18]= True

Lemma. (A more transparent formulation of the preceding lemma.)

In[19]:  = Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
    {p1 \rightarrow not[equal[int[x], id[omega]]], p2 \rightarrow not[subclass[omega, fix[int[x]]]],
    p3 \rightarrow FUNCTION[inverse[inttimes[int[x]]]]}] // Reverse

Out[19]= or[equal[id[omega], int[x]], FUNCTION[inverse[inttimes[int[x]]]]] = True

In[20]:  = (\% /. x \rightarrow x_) /. Equal \rightarrow SetDelayed

The int wrapper can be removed.

Lemma.
In[21]:= SubstTest[implies, and[equal[x, int[t]], not[equal[x, id[omega]]]],
   FUNCTION[inverse[inttimes[x]]], t → x] // Reverse
Out[21]= or[equal[x, id[omega]], FUNCTION[inverse[inttimes[x]]], not[member[x, Z]]] = True

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

The integer membership literal here is redundant. If x is not an integer, inttimes[x] is 0, and hence is also one-to-one.

Lemma.

In[23]:= SubstTest[implies, empty[t], FUNCTION[inverse[t]], t → inttimes[x]] // Reverse
Out[23]= or[FUNCTION[inverse[inttimes[x]]], member[x, Z]] = True

In[24]:= (% /. x → x_) /. Equal → SetDelayed

Lemma. (Elimination of the redundant integerhood literal.)

In[25]:= SubstTest[and, implies[p, q], or[p, q],
   {p → member[x, Z], q → or[equal[x, id[omega]], FUNCTION[inverse[inttimes[x]]]]}]
Out[25]= or[equal[x, id[omega]], FUNCTION[inverse[inttimes[x]]]] = True

In[26]:= (% /. x → x_) /. Equal → SetDelayed

The reverse implication also holds and can be combined with the above into a single rewrite rule.

Theorem.

In[27]:= equiv[FUNCTION[inverse[inttimes[x]]], not[equal[id[omega], x]]]
Out[27]= True

In[28]:= FUNCTION[inverse[inttimes[x_]]] := not[equal[id[omega], x]]