ONEONE[inttimes[x]]

Johan G. F. Belinfante
2009 June 25

```
In[1]:= SetDirectory["l:"]; << goedel.09jun24a;<< tools.m

:Package Title: goedel.09jun24a              2009 June 24 at 6:25 a.m.

It is now:  2009 Jun 25 at 8:33

Loading Simplification Rules

TOOLS.M                                     Revised 2009 June 1

weightlimit = 40
```

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.

```
In[2]:= image[inverse[INTTIMES], BIJ]

Out[2]= intersection[Z, complement[set[id[omega]]]]
```

Before the **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.

```
In[3]:= SubstTest[member, int[x], image[inverse[INTTIMES], z], z → BIJ]

Out[3]= FUNCTION[inverse[inttimes[int[x]]]] == not[equal[id[omega], int[x]]]
```

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

derivation

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

```
In[4]:= equal[intmul[int[x], int[y]], id[omega]]

Out[4]= or[equal[id[omega], int[x]], equal[id[omega], int[y]]]
```

The idea behind the proof is standard: if **x y = x z**, then **x (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 (– y)** can be rewritten as **– (x y)**.

```
In[5]:= Map[A, ImageComp[INTMUL, cross[Id, INVERSE], set[PAIR[int[x], int[y]]]]] // Reverse

Out[5]= intmul[int[x], inverse[int[y]]] == inverse[intmul[int[x], int[y]]]

In[6]:= intmul[int[x_], inverse[int[y_]]] := inverse[intmul[int[x], int[y]]]
```

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

```
In[7]:= Attributes[intmul]

Out[7]= {Flat, OneIdentity, Orderless}
```

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

```
In[8]:= SubstTest[equal, id[omega], intmul[int[x], int[t]],
          t → intadd[int[y], inverse[int[z]]]] // Reverse

Out[8]= equal[intmul[int[x], int[y]], intmul[int[x], int[z]]] ==
         or[equal[id[omega], int[x]], equal[int[y], int[z]]]

In[9]:= equal[intmul[int[x_], int[y_]], intmul[int[x_], int[z_]]] :=
         or[equal[id[omega], int[x]], equal[int[y], int[z]]]
```

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.)

```
In[10]:= SubstTest[implies,
           and[not[equal[int[x], id[omega]]], equal[y, int[u]], equal[z, int[v]],
            equal[intmul[int[x], y], intmul[int[x], z]]], equal[y, z], {u → y, v → z}] // Reverse

Out[10]= or[equal[y, z], equal[id[omega], int[x]],
           not[equal[intmul[y, int[x]], intmul[z, int[x]]]],
           not[member[y, Z]], not[member[z, Z]]] == True

In[11]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

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 → 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 → composite[inverse[inttimes[int[x]]], inttimes[int[x]]],
            t → int[x], w → id[omega]}]]

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

In[15]:= (% /. x → x_) /. Equal → 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 → 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 → not[equal[int[x], id[omega]]], p2 → not[subclass[omega, fix[int[x]]]],
             p3 -> FUNCTION[inverse[inttimes[int[x]]]]}]] // Reverse

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

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