rotate[INTADD]

Johan G. F. Belinfante
2003 August 12

```
In[1]:= << goedel52.s76; << tools.m

        :Package Title: goedel52.s76       2003 August 11 at 12:15 noon

        It is now:  2003 Aug 13 at 15:57

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

Integer subtraction is reduced to addition via the formula **x − y = x + (−y)**. A variable free version of this fact is derived in this notebook, obtained as a corollary of a version of the formula **x + (−x) = 0** for integer addition. Among the immediate consequences of this formula is the fact that **rotate[INTADD]** is a function. An explicit formula for this function is derived.

When variables are introduced, it is a challenge to avoid complicated expressions due to the need for those variables to be restricted to integers. For any fixed integer **a** one can introduce the translation (right−addition) **f(x) = x + a**, the inverse function **g(x) = x − a**, and the function **h(x) = a − x**, which is its own inverse. It proves convenient to express the first two of these functions in terms of the third.

lemmas

The immediate goal is to show that **intadd[x, inverse[x]] = id[omega]]** when **x** is an integer. Without further work one obtains a weaker result than what is desired.

```
In[2]:= SubstTest[member, w, image[z, cart[singleton[x], singleton[inverse[x]]]],
            {w -> id[omega], z -> INTADD}]

Out[2]= equal[id[omega], intadd[x, inverse[x]]] == and[FUNCTION[composite[Id, x]],
          member[x, Z], member[inverse[x], Z], subclass[range[x], omega]]
```

Some lemmas are neded to simplify this result.

```
In[3]:= SubstTest[implies, and [subclass[u, v], subclass[v, w]], subclass[u, w],
            {u -> Z, v -> BIJ, w -> FUNS}]

Out[3]= subclass[Z, FUNS] == True

In[4]:= subclass[Z, FUNS] := True
```

```
In[5]:= SubstTest[implies, and[member[x, z], subclass[z, w]], member[x, w],
            {z -> Z, w -> FUNS}] // MapNotNot

Out[5]= or[FUNCTION[x], not[member[x, Z]]] == True

In[6]:= or[FUNCTION[x_], not[member[x_, Z]]] := True
```

The following related result will not be needed, but it is worth noting anyhow:

```
In[7]:= SubstTest[implies, and[member[x, z], subclass[z, w]], member[x, w],
            {z -> Z, w -> BIJ}] // MapNotNot

Out[7]= or[FUNCTION[inverse[x]], not[member[x, Z]]] == True

In[8]:= or[FUNCTION[inverse[x_]], not[member[x_, Z]]] := True
```

The following weaker fact is actually needed.

```
In[9]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                {p1 -> member[x, Z], p2 -> FUNCTION[x], p3 -> FUNCTION[composite[Id, x]]}]]

Out[9]= or[FUNCTION[composite[Id, x]], not[member[x, Z]]] == True

In[10]:= or[FUNCTION[composite[Id, x_]], not[member[x_, Z]]] := True
```

A second needed fact:

```
In[11]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
             {y -> Z, z -> image[inverse[IMAGE[SECOND]], P[omega]]}] // MapNotNot

Out[11]= or[not[member[x, Z]], subclass[range[x], omega]] == True

In[12]:= or[not[member[x_, Z]], subclass[range[x_], omega]] := True
```

The following related fact is not needed, but it is interesting in its own right:

```
In[13]:= SubstTest[implies, and[member[x, y], subclass[y, z]], member[x, z],
             {y -> Z, z -> image[inverse[IMAGE[FIRST]], P[omega]]}] // MapNotNot

Out[13]= or[not[member[x, Z]], subclass[domain[x], omega]] == True

In[14]:= or[not[member[x_, Z]], subclass[domain[x_], omega]] := True
```

Putting these facts together, one finds

```
In[15]:= not[not[equiv[and[FUNCTION[composite[Id, x]], member[x, Z],
             member[inverse[x], Z], subclass[range[x], omega]], member[x, Z]]]

Out[15]= True
```

This justifies adding a corresponding temporary rewrite rule:

```
In[16]:= and[FUNCTION[composite[Id, x_]], member[x_, Z],
           member[inverse[x_], Z], subclass[range[x_], omega]] := member[x, Z]
```

derivation

Returning to the topic of interest, the equation **x + (−x) = 0**, one now obtains a better result::

```
In[17]:= SubstTest[member, w, image[z, cart[singleton[x], singleton[inverse[x]]]],
             {w -> id[omega], z -> INTADD}]

Out[17]= equal[id[omega], intadd[x, inverse[x]]] == member[x, Z]
```

This result will immediately be replaced with a stronger result, but for now it can be added as a temporary rule:

```
In[18]:= equal[id[omega], intadd[x_, inverse[x_]]] := member[x, Z]
```

The following reformulation suggests a better rule:

```
In[19]:= equal[intadd[x, inverse[x]],
          union[complement[image[V, intersection[Z, singleton[x]]]], id[omega]]]

Out[19]= True
```

This stronger result is now added as a rewrite rule:

```
In[20]:= intadd[x_, inverse[x_]] :=
          union[complement[image[V, intersection[Z, singleton[x]]]], id[omega]]
```

From it, one can derive a variable−free version:

```
In[21]:= composite[INTADD, id[INVERSE], inverse[FIRST]] // VSNormality

Out[21]= composite[INTADD, id[INVERSE], inverse[FIRST]] == cart[Z, singleton[id[omega]]]

In[22]:= composite[INTADD, id[INVERSE], inverse[FIRST]] := cart[Z, singleton[id[omega]]]
```

comment

To deal with the expression **intadd[x,inverse[x]]** when it occurs inside another **intadd**, one needs the following simplification rule:

```
In[23]:= SubstTest[APPLY, INTADD, PAIR[x, w], w -> union[y, complement[image[V, z]]]] // Reverse

Out[23]= intadd[x, union[y, complement[image[V, z]]]] ==
          union[complement[image[V, z]], intadd[x, y]]

In[24]:= intadd[x_, union[y_, complement[image[V, z_]]]] :=
          union[complement[image[V, z]], intadd[x, y]]
```

For example:

```
In[25]:= equal[y, intadd[x, inverse[x], y]]

Out[25]= or[and[member[x, Z], member[y, Z]], equal[V, y]]
```

corollary: rotate[INTADD] is a function

The following corollary of the associative law is generally useful, but will shortly be replaced by a different rule.

```
In[26]:= Assoc[INTADD, composite[cross[Id, INTADD], ASSOC],
           composite[RIGHT[x], RIGHT[y]]] // Reverse

Out[26]= composite[INTADD, RIGHT[x], INTADD, RIGHT[y]] == composite[INTADD, RIGHT[intadd[x, y]]]
```

```
In[27]:= composite[INTADD, RIGHT[x_], INTADD, RIGHT[y_]] :=
          composite[INTADD, RIGHT[intadd[x, y]]]
```

```
In[28]:= Map[FUNCTION, Assoc[composite[INTADD, RIGHT[inverse[x]]],
            composite[INTADD, RIGHT[x]], inverse[composite[INTADD, RIGHT[x]]]]] // Reverse

Out[28]= FUNCTION[composite[inverse[RIGHT[x]], inverse[INTADD]]] == True
```

The following fact is made into a temporary rewrite rule which will later be rendered superfluous.

```
In[29]:= FUNCTION[composite[inverse[RIGHT[x_]], inverse[INTADD]]] := True
```

```
In[30]:= SubstTest[assert,
           forall[x, FUNCTION[composite[inverse[RIGHT[x]], inverse[z]]]], z -> INTADD] // Reverse

Out[30]= FUNCTION[rotate[INTADD]] == True
```

This fact need not be made ino a permanent rule because an explicit formula for **rotate[INTADD]** will be derived shortly from which this fact follows directly.

```
In[31]:= FUNCTION[rotate[INTADD]] := True
```

The fact that **rotate[INTADD]** is a function has an important corollary that is relevant to the topic of this notebook. To derive it, one needs a simplification rule:

```
In[32]:= IminComp[INTADD, id[cart[V, V]], x] // Reverse

Out[32]= composite[Id, image[inverse[INTADD], x]] == image[inverse[INTADD], x]
```

```
In[33]:= composite[Id, image[inverse[INTADD], x_]] := image[inverse[INTADD], x]
```

The following function will be used to eliminate right−addition and left−addition and their inverses:

```
In[34]:= SubstTest[FUNCTION, composite[funpart[y], LEFT[x]], y -> rotate[INTADD]]

Out[34]= FUNCTION[image[inverse[INTADD], singleton[x]]] == True
```

This important fact deserves to be made into a permanent rewrite rule:

```
In[35]:= FUNCTION[image[inverse[INTADD], singleton[x_]]] := True
```

the function that takes x to −x.

A special formula for the involution will now be derived for **x = id[omega]**. The following simplification rules will be needed:

```
In[36]:= Assoc[INTADD, id[cart[Z, Z]], id[cart[Z, V]]] // Reverse

Out[36]= composite[INTADD, id[cart[Z, V]]] == INTADD

In[37]:= composite[INTADD, id[cart[Z, V]]] := INTADD

In[38]:= IminComp[INTADD, id[cart[Z, V]], x] // Reverse

Out[38]= composite[image[inverse[INTADD], x], id[Z]] == image[inverse[INTADD], x]

In[39]:= composite[image[inverse[INTADD], x_], id[Z]] := image[inverse[INTADD], x]
```

The special case is derived as follows:

```
In[40]:= Map[subclass[Z, #] &,
           SubstTest[class, x, and[member[x, Z], member[pair[x, inverse[x]], y]],
             y -> image[inverse[INTADD], singleton[id[omega]]]] // InvertFix] // Reverse

Out[40]= subclass[Z, fix[composite[inverse[IMAGE[SWAP]],
             image[inverse[INTADD], singleton[id[omega]]]]]] == True

In[41]:= subclass[Z, fix[composite[inverse[IMAGE[SWAP]],
             image[inverse[INTADD], singleton[id[omega]]]]]] := True
```

This result can be solved for the inverse image.

```
In[42]:= SubstTest[implies, subclass[u, v], subclass[composite[w, u], composite[w, v]],
             {u -> id[Z],
           v -> composite[inverse[IMAGE[SWAP]], image[inverse[INTADD], singleton[id[omega]]]],
                w -> IMAGE[SWAP]}]

Out[42]= subclass[composite[id[Z], INVERSE],
           image[inverse[INTADD], singleton[id[omega]]]] == True

In[43]:= subclass[composite[id[Z], INVERSE],
           image[inverse[INTADD], singleton[id[omega]]]] := True
```

The inclusion can be strengthened to an equation:

```
In[44]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
             equal[u, composite[v, id[domain[u]]]],
             {u -> composite[id[Z], INVERSE],
           v -> image[inverse[INTADD], singleton[id[omega]]]}]

Out[44]= equal[composite[id[Z], INVERSE], image[inverse[INTADD], singleton[id[omega]]]] == True
```

This formula says that two integers add up to zero if and only if one is the negative of the other.

```
In[45]:= image[inverse[INTADD], singleton[id[omega]]] := composite[id[Z], INVERSE]
```

a formula for rotate[INTADD]

The final goal is to derive a variable−free formula corresponding to **x − y = x + (−y)**. The following simplication rule is needed:

```
In[46]:= Assoc[INTADD, id[cart[Z, Z]], cross[Id, INVERSE]]

Out[46]= composite[INTADD, cross[id[Z], composite[id[Z], INVERSE]]] ==
          composite[INTADD, cross[Id, INVERSE]]

In[47]:= composite[INTADD, cross[id[Z], composite[id[Z], INVERSE]]] :=
          composite[INTADD, cross[Id, INVERSE]]

In[48]:= Map[composite[id[Z], INVERSE, #, cross[Id, INVERSE], id[cart[Z, Z]]] &, IminComp[
            INTADD, composite[cross[Id, INTADD], ASSOC], singleton[id[omega]]]] // Reverse

Out[48]= rotate[INTADD] == composite[INTADD, cross[Id, INVERSE]]

In[49]:= rotate[INTADD] := composite[INTADD, cross[Id, INVERSE]]
```

eliminating right and left addition, and their inverses

The fact that integer subtraction can be reduced to addition implies that the three functions **composite[INTADD,-RIGHT[x]]**, and its inverse can both be expressed in terms of the involution **image[inverse[INTADD],singleton[x]]**. More generally, the class of all pairs of integers whose sum belongs to a class **x** is a symmetric relation:

```
In[50]:= IminComp[INTADD, SWAP, x] // Reverse

Out[50]= inverse[image[inverse[INTADD], x]] == image[inverse[INTADD], x]

In[51]:= inverse[image[inverse[INTADD], x_]] := image[inverse[INTADD], x]

In[52]:= Assoc[composite[INTADD, RIGHT[x]],
           id[domain[composite[INTADD, RIGHT[x]]]], id[Z]] // Reverse

Out[52]= composite[INTADD, RIGHT[x], id[Z]] == composite[INTADD, RIGHT[x]]

In[53]:= composite[INTADD, RIGHT[x_], id[Z]] := composite[INTADD, RIGHT[x]]

In[54]:= Map[composite[#, id[Z], INVERSE] &,
          SubstTest[composite, rotate[z], LEFT[x], z -> INTADD]]

Out[54]= composite[INTADD, RIGHT[x]] == composite[image[inverse[INTADD], singleton[x]], INVERSE]

In[55]:= composite[INTADD, RIGHT[x_]] :=
          composite[image[inverse[INTADD], singleton[x]], INVERSE]

In[56]:= composite[inverse[RIGHT[x]], inverse[INTADD]] // DoubleInverse

Out[56]= composite[inverse[RIGHT[x]], inverse[INTADD]] ==
          composite[INVERSE, image[inverse[INTADD], singleton[x]]]

In[57]:= composite[inverse[RIGHT[x_]], inverse[INTADD]] :=
          composite[INVERSE, image[inverse[INTADD], singleton[x]]]
```

The old rule for left−addition needs to be changed on account of this:

```
In[58]:= composite[INTADD, LEFT[x_]] =.
```

Replacement rules are derived:

```
In[59]:= Assoc[INTADD, SWAP, RIGHT[x]]

Out[59]= composite[INTADD, LEFT[x]] == composite[image[inverse[INTADD], singleton[x]], INVERSE]

In[60]:= composite[INTADD, LEFT[x_]] := composite[image[inverse[INTADD], singleton[x]], INVERSE]

In[61]:= composite[inverse[LEFT[x]], inverse[INTADD]] // DoubleInverse

Out[61]= composite[inverse[LEFT[x]], inverse[INTADD]] ==
          composite[INVERSE, image[inverse[INTADD], singleton[x]]]

In[62]:= composite[inverse[LEFT[x]], inverse[INTADD]] :=
          composite[INVERSE, image[inverse[INTADD], singleton[x]]]
```

a new rule for composition of right−additions

The rule for composing right−additions also needs to be modified. Some simplification rules are needed to do this.

```
In[63]:= Assoc[id[image[V, singleton[x]]],
          composite[id[image[V, singleton[x]]], y, INTADD], composite[RIGHT[x], id[Z], INVERSE]]

Out[63]= composite[id[image[V, singleton[x]]], y, image[inverse[INTADD], singleton[x]]] ==
          composite[y, image[inverse[INTADD], singleton[x]]]

In[64]:= composite[id[image[V, singleton[x_]]], y___, image[inverse[INTADD], singleton[x_]]] :=
          composite[y, image[inverse[INTADD], singleton[x]]]

In[65]:= composite[image[inverse[INTADD], singleton[x]],
           y, id[image[V, singleton[x]]]] // DoubleInverse

Out[65]= composite[image[inverse[INTADD], singleton[x]], y, id[image[V, singleton[x]]]] ==
          composite[image[inverse[INTADD], singleton[x]], y]

In[66]:= composite[image[inverse[INTADD], singleton[x_]], y___, id[image[V, singleton[x_]]]] :=
          composite[image[inverse[INTADD], singleton[x]], y]

In[67]:= Assoc[image[inverse[INTADD], x], id[Z], id[y]]

Out[67]= composite[image[inverse[INTADD], x], id[intersection[y, Z]]] ==
          composite[image[inverse[INTADD], x], id[y]]

In[68]:= composite[image[inverse[INTADD], x_], id[intersection[y_, Z]]] :=
          composite[image[inverse[INTADD], x], id[y]]
```

The replacement for the rule about composites of right−additions is this:

```
In[69]:= Assoc[INTADD, composite[cross[Id, INTADD], ASSOC],
          composite[cross[RIGHT[y], composite[id[Z], INVERSE]], LEFT[x]]]

Out[69]= composite[image[inverse[INTADD], singleton[x]],
           INVERSE, image[inverse[INTADD], singleton[y]]] ==
          image[inverse[INTADD], singleton[intadd[x, y]]]
```

```
In[70]:= composite[image[inverse[INTADD], singleton[x_]],
           INVERSE, image[inverse[INTADD], singleton[y_]]] :=
          image[inverse[INTADD], singleton[intadd[x, y]]]
```