invertible elements are cancellable

Johan G. F. Belinfante
2011 May 14

```
In[1]:= SetDirectory["l:"]; << goedel.11may13a

        :Package Title: goedel.11may13a              2011 May 13 at 4:00 p.m.

        Loading takes about ten minutes, half that time due to builtin pauses.

        It is now:  2011 May 14 at 5:8

        Loading Simplification Rules

        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

        weightlimit = 40

        Loading completed.

        It is now:  2011 May 14 at 5:19
```

summary

The statement **x ∈ MONOIDS** means that **x** is a multiplication law for a monoid. The members of **range[x]** are called the **elements** of the monoid. An element **u ∈ range[x]** is said to be **invertible** if **u ∈ domain[inv[x]]**. An element **u** is said to be **left cancellable** if the left multiplication function **x ∘ LEFT[u]** is one-to-one. Similarly, an element **u** is right cancellable if **x ∘ RIGHT[u]** is one-to-one. It is shown in this notebook that invertibible elements of a monoid are both left and right cancellable. The variable **u** can be eliminated.

some general results

If **x** is a monoid, its domain is the cartesian square of its range. It follows from this that the domain and range of its domain are equal to its range.

Theorem.

```
In[5]:= Map[not, SubstTest[and, implies[p1, p2], not[implies[p1, p3]],
           {p1 → member[x, MONOIDS], p2 → equal[domain[x], cart[range[x], range[x]]],
            p3 -> equal[range[domain[x]], range[x]]}]] // Reverse

Out[5]= or[equal[range[x], range[domain[x]]], not[member[x, MONOIDS]]] == True

In[6]:= or[equal[range[x_], range[domain[x_]]], not[member[x_, MONOIDS]]] := True
```

The other statement is already available.

```
In[3]:= implies[member[x, MONOIDS], equal[domain[domain[x]], range[x]]]

Out[3]= True
```

The following result about semigroups will be needed later.

Theorem. The composite of two left multiplications is the left multiplication by the product.

```
In[29]:= SubstTest[implies, equal[x, semigp[t]],
           implies[member[x, SEMIGPS], equal[composite[x, LEFT[u], x, LEFT[v]],
             composite[x, LEFT[APPLY[x, PAIR[u, v]]]]], t → x] // Reverse

Out[29]= or[equal[composite[x, LEFT[APPLY[x, PAIR[u, v]]]], composite[x, LEFT[u], x, LEFT[v]]],
           not[member[x, SEMIGPS]]] == True

In[30]:= or[equal[composite[x_, LEFT[APPLY[x_, PAIR[u_, v_]]]],
            composite[x_, LEFT[u_], x_, LEFT[v_]]], not[member[x_, SEMIGPS]]] := True
```

Comment. One can achieve a speedup by a factor of 13 by omitting the three proof steps indicated by the comment notation **(* ⋯ *)** in the next theorem.

Theorem.

```
In[31]:= Map[not, SubstTest[and, (*implies[and[p1,p2],p3],implies[p1,p4],implies[p1,p5],*)
            implies[and[p3, p4, p5], p6], not[implies[and[p1, p2], p6]],
            {p1 → member[x, MONOIDS], p2 → member[u, domain[inv[x]]],
             p3 -> equal[APPLY[x, PAIR[u, APPLY[inv[x], u]]], e[x]], p4 → member[x, SEMIGPS],
             p5 -> equal[composite[x, LEFT[e[x]]], id[range[x]]], p6 ->
              equal[id[range[x]], composite[x, LEFT[u], x, LEFT[APPLY[inv[x], u]]]]}]] // Reverse

Out[31]= or[equal[composite[x, LEFT[u], x, LEFT[APPLY[inv[x], u]]], id[range[x]]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[32]:= or[equal[composite[x_, LEFT[u_], x_, LEFT[APPLY[inv[x_], u_]]], id[range[x_]]],
           not[member[u_, domain[inv[x_]]]], not[member[x_, MONOIDS]]] := True
```

Theorem. A similar result, with the factors **u** and **APPLY[inv[x],u]** interchanged.

```
In[39]:= Map[not, SubstTest[and, (*implies[and[p1,p2],p3],implies[p1,p4],implies[p1,p5],*)
            implies[and[p3, p4, p5], p6], not[implies[and[p1, p2], p6]],
            {p1 → member[x, MONOIDS], p2 → member[u, domain[inv[x]]],
             p3 -> equal[APPLY[x, PAIR[APPLY[inv[x], u], u]], e[x]], p4 → member[x, SEMIGPS],
             p5 -> equal[composite[x, LEFT[e[x]]], id[range[x]]], p6 ->
              equal[id[range[x]], composite[x, LEFT[APPLY[inv[x], u]], x, LEFT[u]]]}]] // Reverse

Out[39]= or[equal[composite[x, LEFT[APPLY[inv[x], u]], x, LEFT[u]], id[range[x]]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[41]:= or[equal[composite[x_, LEFT[APPLY[inv[x_], u_]], x_, LEFT[u_]], id[range[x_]]],
           not[member[u_, domain[inv[x_]]]], not[member[x_, MONOIDS]]] := True
```

results from category theory

One can sometimes take advantage of the **cat** wrapper to derive facts about monoids by first proving that the same (or a similar) result holds for categories in general.

Lemma. If **u** is an invertible morphism for a category **cat[x]**, then its inverse is a morphism.

```
In[64]:= SubstTest[implies, and[member[w, y], subclass[y, z]], member[w, z],
           {w -> APPLY[inv[cat[x]], u], y -> domain[inv[cat[x]]], z → range[cat[x]]}] // Reverse

Out[64]= or[member[APPLY[inv[cat[x]], u], range[cat[x]]],
           not[member[u, domain[inv[cat[x]]]]]] == True

In[65]:= or[member[APPLY[inv[cat[x_]], u_], range[cat[x_]]],
           not[member[u_, domain[inv[cat[x_]]]]]] := True
```

Theorem. (Eliminating the **cat** wrapper.)

```
In[66]:= SubstTest[implies, equal[x, cat[t]], or[member[APPLY[inv[x], u], range[x]],
            not[member[u, domain[inv[x]]]]], t → x] // Reverse

Out[66]= or[member[APPLY[inv[x], u], range[x]],
           not[category[x]], not[member[u, domain[inv[x]]]]] == True

In[68]:= or[member[APPLY[inv[x_], u_], range[x_]],
           not[category[x_]], not[member[u_, domain[inv[x_]]]]] := True
```

Since monoids are a special kind of category, the same holds for monoids.

Corollary.

```
In[69]:= Map[not, SubstTest[and, implies[p2, p3],
            not[implies[p1, p3]], {p1 → member[x, MONOIDS], p2 → category[x], p3 -> or[
               member[APPLY[inv[x], u], range[x]], not[member[u, domain[inv[x]]]]]}]] // Reverse

Out[69]= or[member[APPLY[inv[x], u], range[x]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[70]:= or[member[APPLY[inv[x_], u_], range[x_]],
           not[member[u_, domain[inv[x_]]]], not[member[x_, MONOIDS]]] := True
```

x ∘ y = id[z]

When the composite of two relations **x** and **y** is an identity relation, certain expressions involving **x** and **y** are functions.

Theorem.

```
In[34]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]],
           {p1 → equal[composite[x, y], id[z]], p2 -> subclass[composite[x, y], Id],
            p3 -> FUNCTION[composite[x, id[range[y]]]]}]] // Reverse

Out[34]= or[FUNCTION[composite[x, id[range[y]]]], not[equal[composite[x, y], id[z]]] == True

In[35]:= or[FUNCTION[composite[x_, id[range[y_]]]],
           not[equal[composite[x_, y_], id[z_]]] := True
```

Theorem.

```
In[37]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]],
           {p1 → equal[composite[x, y], id[z]], p2 -> subclass[composite[x, y], Id],
            p3 -> FUNCTION[composite[inverse[y], id[domain[x]]]]}]] // Reverse

Out[37]= or[FUNCTION[composite[inverse[y], id[domain[x]]]],
           not[equal[composite[x, y], id[z]]] == True

In[38]:= or[FUNCTION[composite[inverse[y_], id[domain[x_]]]],
           not[equal[composite[x_, y_], id[z_]]] := True
```

---

the main result

Lemma.

```
In[45]:= SubstTest[implies, equal[composite[u, v], id[w]],
           FUNCTION[composite[inverse[v], id[domain[u]]]],
           {u → composite[x, LEFT[APPLY[inv[x], u]]], v -> composite[x, LEFT[u]]}] // Reverse

Out[45]= or[FUNCTION[composite[inverse[LEFT[u]],
             inverse[x], id[image[domain[x], set[APPLY[inv[x], u]]]]],
           not[equal[composite[x, LEFT[APPLY[inv[x], u]], x, LEFT[u]], id[w]]] == True

In[46]:= (% /. {x → x_, u → u_, w → w_}) /. Equal→ SetDelayed
```

Theorem.

```
In[48]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p3, p4], not[implies[and[p1, p2], p4]],
           {p1 → member[x, MONOIDS], p2 → member[u, domain[inv[x]]],
            p3 -> equal[composite[x, LEFT[APPLY[inv[x], u]], x, LEFT[u]], id[range[x]]],
            p4 -> FUNCTION[composite[inverse[LEFT[u]], inverse[x],
               id[image[domain[x], set[APPLY[inv[x], u]]]]]]}]] // Reverse

Out[48]= or[FUNCTION[composite[inverse[LEFT[u]],
             inverse[x], id[image[domain[x], set[APPLY[inv[x], u]]]]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[49]:= (% /. {x → x_, u → u_}) /. Equal→ SetDelayed
```

The purpose of the next lemma is to eliminate **image[domain[x], –]** in the preceding result, making use of the fact that for a semigroup, the domain is the cartesian square of the range.

Lemma.

```
In[51]:= SubstTest[or, FUNCTION[composite[inverse[LEFT[u]],
             inverse[t], id[image[domain[t], set[APPLY[inv[t], u]]]]]],
           not[member[u, domain[inv[t]]]], not[member[t, MONOIDS]], t → semigp[x]] // Reverse

Out[51]= or[FUNCTION[composite[inverse[LEFT[u]],
             inverse[semigp[x]], id[intersection[fix[domain[semigp[x]]], image[V,
                intersection[fix[domain[semigp[x]]], set[APPLY[inv[semigp[x]], u]]]]]]]],
           not[member[u, domain[inv[semigp[x]]]]], not[member[e[semigp[x]], V]]] == True

In[52]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Lemma. (Eliminating the **semigp** wrapper.)

```
In[54]:= SubstTest[implies, equal[x, semigp[t]],
            or[FUNCTION[composite[inverse[LEFT[u]], inverse[x], id[intersection[fix[domain[x]],
                 image[V, intersection[fix[domain[x]], set[APPLY[inv[x], u]]]]]]]],
             not[member[u, domain[inv[x]]]], not[member[e[x], V]]], t → x] // Reverse // MapNotNot

Out[54]= or[FUNCTION[composite[inverse[LEFT[u]], inverse[x], id[intersection[fix[domain[x]],
              image[V, intersection[fix[domain[x]], set[APPLY[inv[x], u]]]]]]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[55]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Lemma. A simplification rule.

```
In[57]:= equiv[and[member[x, MONOIDS], equal[range[x], fix[domain[x]]]], member[x, MONOIDS]]

Out[57]= True

In[58]:= and[equal[fix[domain[x_]], range[x_]], member[x_, MONOIDS]] := member[x, MONOIDS]
```

Lemma.

```
In[59]:= SubstTest[implies, equal[t, fix[domain[x]]],
            or[FUNCTION[composite[inverse[LEFT[u]], inverse[x],
               id[intersection[t, image[V, intersection[t, set[APPLY[inv[x], u]]]]]]]],
             not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]],
            t → range[x]] // Reverse // MapNotNot

Out[59]= or[FUNCTION[composite[inverse[LEFT[u]], inverse[x]]],
           not[member[x, MONOIDS]], not[member[APPLY[inv[x], u], range[x]]]] == True

In[60]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Theorem. If **u** is an invertible element of a monoid **x**, then **u** is left cancellable.

```
In[71]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p3], p4], not[implies[and[p1, p2], p4]], {p1 → member[x, MONOIDS],
            p2 → member[u, domain[inv[x]]], p3 -> member[APPLY[inv[x], u], range[x]],
            p4 -> FUNCTION[composite[inverse[LEFT[u]], inverse[x]]]}]] // Reverse

Out[71]= or[FUNCTION[composite[inverse[LEFT[u]], inverse[x]]],
           not[member[u, domain[inv[x]]]], not[member[x, MONOIDS]]] == True

In[72]:= or[FUNCTION[composite[inverse[LEFT[u_]], inverse[x_]]],
           not[member[u_, domain[inv[x_]]]], not[member[x_, MONOIDS]]] := True
```

---

eliminating the variable u

Lemma.

```
In[73]:= Map[fix[composite[rotate[composite[x, SWAP]], SWAP, inverse[#]]] &,
          rotate[composite[Di, rotate[x]]] // FastReifTriNormality]

Out[73]= fix[composite[rotate[composite[x, SWAP]],
             SWAP, inverse[rotate[composite[Di, rotate[x]]]]]] ==
          domain[fix[composite[inverse[x], x, cross[Id, Di]]]]

In[74]:= fix[composite[rotate[composite[x_, SWAP]],
             SWAP, inverse[rotate[composite[Di, rotate[x_]]]]]] :=
          domain[fix[composite[inverse[x], x, cross[Id, Di]]]]
```

Theorem.

```
In[75]:= FUNCTION[composite[rotate[x], id[cart[V, y]]]] // AssertTest // InvertFix // Reverse

Out[75]= equal[0, intersection[y, domain[fix[composite[inverse[x], x, cross[Id, Di]]]]]] ==
          FUNCTION[composite[rotate[x], id[cart[V, y]]]]

In[76]:= equal[0, intersection[y_, domain[fix[composite[inverse[x_], x_, cross[Id, Di]]]]]] :=
          FUNCTION[composite[rotate[x], id[cart[V, y]]]]
```

The following theorem is a restatement of the fact that invertible elements are left cancellable, without a variable for the element. (This takes quite a while.)

Theorem. If **x** is a monoid, then a certain restriction of **rotate[x]** is a function.

```
In[77]:= Map[equal[V, #] &, SubstTest[class, u,
            implies[and[member[x, v], member[u, w]], FUNCTION[composite[y, RIGHT[u]]]],
            {v → MONOIDS, w→ domain[inv[x]], y → rotate[x]}]] // MapNotNot

Out[77]= or[FUNCTION[composite[rotate[x], id[cart[V, domain[inv[x]]]]]],
           not[member[x, MONOIDS]]] == True

In[78]:= or[FUNCTION[composite[rotate[x_], id[cart[V, domain[inv[x_]]]]]],
           not[member[x_, MONOIDS]]] := True
```

A dual result will now be derived.

Lemma.

```
In[79]:= SubstTest[or, FUNCTION[composite[rotate[t], id[cart[V, domain[inv[t]]]]]],
           not[member[t, MONOIDS]], t → flip[semigp[x]]] // Reverse

Out[79]= or[FUNCTION[composite[rotate[composite[semigp[x], SWAP]],
            id[cart[V, domain[inv[semigp[x]]]]]]], not[member[e[semigp[x]], V]]] == True

In[80]:= (% /. x → x_) /. Equal → SetDelayed
```

Dual Theorem. Invertible elements are right cancellable.

```
In[81]:= SubstTest[implies, equal[x, semigp[t]],
            or[FUNCTION[composite[rotate[composite[x, SWAP]], id[cart[V, domain[inv[x]]]]]],
             not[member[e[x], V]]], t → x] // MapNotNot // Reverse

Out[81]= or[FUNCTION[composite[rotate[composite[x, SWAP]], id[cart[V, domain[inv[x]]]]],
           not[member[x, MONOIDS]]] == True

In[82]:= or[FUNCTION[composite[rotate[composite[x_, SWAP]], id[cart[V, domain[inv[x_]]]]],
           not[member[x_, MONOIDS]]] := True
```