inverse elements in groups, part 3.

Johan G. F. Belinfante
2009 February 9

```
In[1]:= SetDirectory["l:"]; << goedel.09feb07a;<< tools.m

        :Package Title: goedel.09feb07a            2009 February 7 at 10:55 p.m.

        It is now:  2009 Feb 9 at 14:52

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 February 4

        weightlimit = 40
```

summary

A group wrapper **gp[x]** is defined to enable the group axioms to be reformulated as rewrite rules.

definition

The following rewrite rule serves to define the **gp[x]** wrapper:

```
In[2]:= image[V, intersection[gp[x_], set[y_]]] :=
         intersection[image[V, intersection[GROUPS, set[x]]], image[V, intersection[x, set[y]]]]
```

normalization

Theorem.

```
In[3]:= Map[fix, SubstTest[reify, y, image[V, intersection[t, set[y]]], t → gp[x]]] // Reverse

Out[3]= intersection[x, image[V, intersection[GROUPS, set[x]]]] == gp[x]

In[4]:= intersection[x_, image[V, intersection[GROUPS, set[x_]]]] := gp[x]
```

wrapper introduction and removal rules

Theorem. (Wrapper removal rule.)

```
In[5]:= SubstTest[equal, x,
          intersection[x, image[V, intersection[t, set[x]]]], t → GROUPS] // Reverse
```

```
Out[5]= equal[x, gp[x]] == or[equal[0, x], member[x, GROUPS]]
```

```
In[6]:= equal[x_, gp[x_]] := or[equal[0, x], member[x, GROUPS]]
```

Theorem. (Automatic removal.)

```
In[7]:= implies[member[x, GROUPS], equal[gp[x], x]]
```

```
Out[7]= True
```

```
In[8]:= gp[x_] := x /; member[x, GROUPS]
```

Lemma.

```
In[9]:= SubstTest[member, intersection[x, image[V, intersection[t, set[x]]]],
          t, t → union[set[0], GROUPS]] // Reverse
```

```
Out[9]= or[equal[0, gp[x]], member[gp[x], GROUPS]] == True
```

```
In[10]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. (Wrapper introduction rule.)

```
In[11]:= equiv[member[gp[x], GROUPS], not[equal[0, gp[x]]]]
```

```
Out[11]= True
```

```
In[12]:= member[gp[x_], GROUPS] := not[equal[0, gp[x]]]
```

Comment. The wrapper introduction rule often produces redundant literals of the form **equal[0, gp[x]]**, but these are generally easy to remove.

FUNCTION rules

Theorem. Groups are binary operations.

```
In[13]:= SubstTest[implies, member[t, GROUPS], member[t, BINOPS], t → gp[x]] // Reverse
```

```
Out[13]= member[gp[x], BINOPS] == True
```

```
In[14]:= member[gp[x_], BINOPS] := True
```

Theorem Groups are functions.

```
In[15]:= SubstTest[FUNCTION, binop[t], t → gp[x]] // Reverse
```

```
Out[15]= FUNCTION[gp[x]] == True
```

```
In[16]:= FUNCTION[gp[x_]] := True
```

Theorem. An **APPLY** rule.

```
In[17]:= SubstTest[image, funpart[t], set[PAIR[u, v]], t → gp[x]] // Reverse

Out[17]= image[gp[x], cart[set[u], set[v]]] == set[APPLY[gp[x], PAIR[u, v]]]

In[18]:= image[gp[x_], cart[set[u_], set[v_]]] := set[APPLY[gp[x], PAIR[u, v]]]
```

associative law

Theorem. Groups are semigroups.

```
In[19]:= SubstTest[implies, member[t, GROUPS], member[t, SEMIGPS], t → gp[x]] // Reverse

Out[19]= member[gp[x], SEMIGPS] == True

In[20]:= member[gp[x_], SEMIGPS] := True
```

Corollary. Groups are associative.

```
In[21]:= SubstTest[associative, semigp[t], t → gp[x]] // Reverse

Out[21]= associative[gp[x]] == True

In[22]:= associative[gp[x_]] := True
```

Theorem. Groups are categories.

```
In[23]:= SubstTest[implies, member[t, GROUPS], category[t], t → gp[x]] // Reverse

Out[23]= category[gp[x]] == True

In[24]:= category[gp[x_]] := True
```

Theorem. The associative law as a rewrite rule.

```
In[25]:= (SubstTest[APPLY, cat[t], PAIR[u, APPLY[cat[t], PAIR[v, w]]], t → gp[x]]) // Reverse

Out[25]= APPLY[gp[x], PAIR[u, APPLY[gp[x], PAIR[v, w]]]] ==
          APPLY[gp[x], PAIR[APPLY[gp[x], PAIR[u, v]], w]]

In[26]:= APPLY[gp[x_], PAIR[u_, APPLY[gp[x_], PAIR[v_, w_]]]] :=
          APPLY[gp[x], PAIR[APPLY[gp[x], PAIR[u, v]], w]]
```

e[x] rules

Lemma. If **x** is a group, then left multiplication by **e[x]** is the identity function on **range[x]**.

```
In[27]:= Map[not, SubstTest[and, implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[x, GROUPS], p2 → member[x, MONOIDS],
            p3 → equal[composite[x, LEFT[e[x]]], id[range[x]]]}]] // Reverse

Out[27]= or[equal[composite[x, LEFT[e[x]]], id[range[x]]], not[member[x, GROUPS]]] == True

In[28]:= or[equal[composite[x_, LEFT[e[x_]]], id[range[x_]]], not[member[x_, GROUPS]]] := True
```

Lemma. If **x** is a group, then right multiplication by **e[x]** is the identity function on **range[x]**.

```
In[29]:= Map[not, SubstTest[and, implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[x, GROUPS], p2 → member[x, MONOIDS],
            p3 → equal[composite[x, RIGHT[e[x]]], id[range[x]]]}]] // Reverse

Out[29]= or[equal[composite[x, RIGHT[e[x]]], id[range[x]]], not[member[x, GROUPS]]] == True

In[30]:= or[equal[composite[x_, RIGHT[e[x_]]], id[range[x_]]], not[member[x_, GROUPS]]] := True
```

Theorem. The element **e[gp[x]]** is left-neutral. Comment. A redundant literal is removed by using **Map**.

```
In[31]:= Map[implies[#, equal[composite[gp[x], LEFT[e[gp[x]]]], id[range[gp[x]]]]] &, SubstTest[
           implies, member[t, GROUPS], equal[composite[t, LEFT[e[t]]], id[range[t]]], t → gp[x]]]

Out[31]= equal[composite[gp[x], LEFT[e[gp[x]]]], id[range[gp[x]]]] == True

In[32]:= composite[gp[x_], LEFT[e[gp[x_]]]] := id[range[gp[x]]]
```

Theorem. The element **e[gp[x]]** is right-neutral.

```
In[33]:= Map[implies[#, equal[composite[gp[x], RIGHT[e[gp[x]]]], id[range[gp[x]]]]] &,
          SubstTest[implies, member[t, GROUPS],
           equal[composite[t, RIGHT[e[t]]], id[range[t]]], t → gp[x]]]

Out[33]= equal[composite[gp[x], RIGHT[e[gp[x]]]], id[range[gp[x]]]] == True

In[34]:= composite[gp[x_], RIGHT[e[gp[x_]]]] := id[range[gp[x]]]
```

Corollary. The group axiom about left-neutrality of the identity element expressed as a rewrite rule.

```
In[35]:= ApComp[gp[x], LEFT[e[gp[x]]], u]

Out[35]= APPLY[gp[x], PAIR[e[gp[x]], u]] ==
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]

In[36]:= APPLY[gp[x_], PAIR[e[gp[x_]], u_]] :=
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]
```

Corollary. The group axiom about right-neutrality of the identity element expressed as a rewrite rule.

```
In[37]:= ApComp[gp[x], RIGHT[e[gp[x]]], u]

Out[37]= APPLY[gp[x], PAIR[u, e[gp[x]]]] ==
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]
```

```
In[38]:= APPLY[gp[x_], PAIR[u_, e[gp[x_]]]] :=
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]
```

In the remainder of this section, some simplification rules involving **e[gp[x]]** are derived.

Lemma.

```
In[39]:= SubstTest[implies, member[t, GROUPS], member[e[t], range[t]], t → gp[x]] // Reverse
Out[39]= or[equal[0, gp[x]], member[e[gp[x]], range[gp[x]]]] == True

In[40]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The element **e[gp[x]]** belongs to **range[gp[x]]** if and only if **gp[x]** is a group.

```
In[41]:= equiv[member[e[gp[x]], range[gp[x]]], not[equal[0, gp[x]]]]
Out[41]= True

In[42]:= member[e[gp[x_]], range[gp[x_]]] := not[equal[0, gp[x]]]
```

Theorem. The set of identities is a singleton if **gp[x]** is a group, and empty otherwise.

```
In[43]:= SubstTest[ids, binop[t], t → gp[x]] // Reverse
Out[43]= ids[gp[x]] == set[e[gp[x]]]

In[44]:= ids[gp[x_]] := set[e[gp[x]]]
```

Theorem. If **gp[x]** is a group, then **e[gp[x]]** is a set.

```
In[45]:= Map[not, SubstTest[equal, ids[cat[t]], 0, t → gp[x]]] // Reverse
Out[45]= member[e[gp[x]], V] == not[equal[0, gp[x]]]

In[46]:= member[e[gp[x_]], V] := not[equal[0, gp[x]]]
```

the inverse pair relation

Theorem. For groups the inverse pair relation is a function.

```
In[47]:= SubstTest[FUNCTION, inv[cat[t]], t → gp[x]] // Reverse
Out[47]= FUNCTION[inv[gp[x]]] == True

In[48]:= FUNCTION[inv[gp[x_]]] := True
```

Theorem. An **APPLY** rule for **inv[gp[x]]**.

```
In[49]:= SubstTest[image, funpart[t], set[u], t → inv[gp[x]]] // Reverse
Out[49]= image[inv[gp[x]], set[u]] == set[APPLY[inv[gp[x]], u]]
```

```
In[50]:= image[inv[gp[x_]], set[u_]] := set[APPLY[inv[gp[x]], u]]
```

Theorem. Every element of a group has an inverse.

```
In[51]:= Map[implies[#, equal[domain[inv[gp[x]]], range[gp[x]]]] &,
          SubstTest[implies, member[t, GROUPS], equal[range[t], domain[inv[t]]], t → gp[x]]]
Out[51]= equal[domain[inv[gp[x]]], range[gp[x]]] == True

In[52]:= domain[inv[gp[x_]]] := range[gp[x]]
```

Theorem. The function **inv[gp[x]]** is an involution.

```
In[53]:= SubstTest[composite, funpart[t], inverse[funpart[t]], t → inv[gp[x]]] // Reverse
Out[53]= composite[inv[gp[x]], inv[gp[x]]] == id[range[gp[x]]]

In[54]:= composite[inv[gp[x_]], inv[gp[x_]]] := id[range[gp[x]]]
```

Theorem. Double inverse as a rewrite rule.

```
In[55]:= ApComp[inv[gp[x]], inv[gp[x]], u]
Out[55]= APPLY[inv[gp[x]], APPLY[inv[gp[x]], u]] ==
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]

In[56]:= APPLY[inv[gp[x_]], APPLY[inv[gp[x_]], u_]] :=
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]]]
```

---

group axioms about inverses

Lemma. Putting wrappers on the right-inverse axiom. This temporary result is not yet a rewrite rule.

```
In[57]:= Map[implies[#, or[equal[APPLY[gp[x], PAIR[u, APPLY[inv[gp[x]], u]]], e[gp[x]]],
              not[member[u, range[gp[x]]]]]] &,
          SubstTest[implies, and[member[t, GROUPS], member[u, range[t]]],
           equal[e[t], APPLY[t, PAIR[u, APPLY[inv[t], u]]]], t → gp[x]]]
Out[57]= or[equal[APPLY[gp[x], PAIR[u, APPLY[inv[gp[x]], u]]], e[gp[x]]],
           not[member[u, range[gp[x]]]]] == True

In[58]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

To obtain a rewrite rule, the variable **u** is first eliminated, and then re-introduced.

Lemma. An inclusion obtained by eliminating the variable **u**.

```
In[59]:= Map[equal[V, #] &,
          SubstTest[class, u, or[equal[APPLY[funpart[t], u], s], not[member[u, r]]],
           {r → range[gp[x]], s → e[gp[x]], t → composite[gp[x], cross[Id, inv[gp[x]]], DUP]}]]

Out[59]= subclass[range[gp[x]],
           fix[composite[inverse[image[inverse[gp[x]], set[e[gp[x]]]]], inv[gp[x]]]]] == True

In[60]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. The inclusion is cleaned up and strengthened to an equation.

```
In[61]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u -> cart[range[gp[x]], set[e[gp[x]]]],
            v -> composite[gp[x], id[inv[gp[x]]], inverse[FIRST]]}] // Reverse

Out[61]= equal[cart[range[gp[x]], set[e[gp[x]]]],
           composite[gp[x], id[inv[gp[x]]], inverse[FIRST]]] == True

In[62]:= composite[gp[x_], id[inv[gp[x_]]], inverse[FIRST]] := cart[range[gp[x]], set[e[gp[x]]]]
```

Lemma. A simplification rule.

```
In[63]:= equal[union[complement[image[V, set[e[gp[x]]]]], e[gp[x]]], e[gp[x]]]

Out[63]= True

In[64]:= union[complement[image[V, set[e[gp[x_]]]]], e[gp[x_]]] := e[gp[x]]
```

Theorem. Axiom about right-inverses as a rewrite rule. (Obtained by reintroducing a variable **u**.)

```
In[65]:= Map[A, ImageComp[composite[gp[x], id[inv[gp[x]]]], inverse[FIRST], set[u]]] // Reverse

Out[65]= APPLY[gp[x], PAIR[u, APPLY[inv[gp[x]], u]]] ==
          union[complement[image[V, intersection[range[gp[x]], set[u]]]], e[gp[x]]]

In[66]:= APPLY[gp[x_], PAIR[u_, APPLY[inv[gp[x_]], u_]]] :=
          union[complement[image[V, intersection[range[gp[x]], set[u]]]], e[gp[x]]]
```

Theorem. A similar result with **SECOND** in place of **FIRST**.

```
In[67]:= Assoc[composite[gp[x], cross[Id, inv[gp[x]]]],
           cross[inv[gp[x]], inv[gp[x]]], DUP] // Reverse

Out[67]= composite[gp[x], id[inv[gp[x]]], inverse[SECOND]] == cart[range[gp[x]], set[e[gp[x]]]]

In[68]:= composite[gp[x_], id[inv[gp[x_]]], inverse[SECOND]] :=
          cart[range[gp[x]], set[e[gp[x]]]]
```

Theorem. Axiom about left-inverses as a rewrite rule. (Obtained by reintroducing a variable **u**.)

```
In[69]:= Map[A, ImageComp[composite[gp[x], id[inv[gp[x]]]], inverse[SECOND], set[u]]] // Reverse

Out[69]= APPLY[gp[x], PAIR[APPLY[inv[gp[x]], u], u]] ==
          union[complement[image[V, intersection[range[gp[x]], set[u]]]], e[gp[x]]]
```

```
In[70]:= APPLY[gp[x_], PAIR[APPLY[inv[gp[x_]], u_], u_]] :=
          union[complement[image[V, intersection[range[gp[x]], set[u]]]], e[gp[x]]]
```

a special case

For the special case of the product of the identity element with itself, an extra rewrite rule is needed to produce a clean result.

Theorem.

```
In[72]:= equal[intersection[range[gp[x]], set[e[gp[x]]]], set[e[gp[x]]]]

Out[72]= True

In[73]:= intersection[range[gp[x_]], set[e[gp[x_]]]] := set[e[gp[x]]]
```

With this rule in place, one has:

```
In[75]:= APPLY[gp[x], PAIR[e[gp[x]], e[gp[x]]]]

Out[75]= e[gp[x]]
```

inverse of the identity element

Theorem. The element **e[gp[x]]** is its own inverse.

```
In[76]:= Map[implies[#, equal[APPLY[inv[gp[x]], e[gp[x]]], e[gp[x]]]] &,
          SubstTest[implies, member[t, GROUPS], equal[APPLY[inv[t], e[t]], e[t]], t → gp[x]]]

Out[76]= equal[APPLY[inv[gp[x]], e[gp[x]]], e[gp[x]]] == True

In[77]:= APPLY[inv[gp[x_]], e[gp[x_]]] := e[gp[x]]
```

a corollary of the associative law

Theorem. A rewrite rule for $(u \cdot v) \cdot v^{-1}$.

```
In[78]:= SubstTest[APPLY, gp[x], PAIR[u, APPLY[gp[x], PAIR[v, w]]], w → APPLY[inv[gp[x]], v]]

Out[78]= APPLY[gp[x], PAIR[APPLY[gp[x], PAIR[u, v]], APPLY[inv[gp[x]], v]]] ==
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]],
           complement[image[V, intersection[range[gp[x]], set[v]]]]]

In[79]:= APPLY[gp[x_], PAIR[APPLY[gp[x_], PAIR[u_, v_]], APPLY[inv[gp[x_]], v_]]] :=
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]],
           complement[image[V, intersection[range[gp[x]], set[v]]]]]
```

Theorem. A rewrite rule for $(u \cdot v^{-1}) \cdot v$.

```
In[80]:= SubstTest[APPLY, gp[x], PAIR[u, APPLY[gp[x], PAIR[w, v]]], w → APPLY[inv[gp[x]], v]]

Out[80]= APPLY[gp[x], PAIR[APPLY[gp[x], PAIR[u, APPLY[inv[gp[x]], v]]], v]] ==
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]],
           complement[image[V, intersection[range[gp[x]], set[v]]]]]

In[81]:= APPLY[gp[x_], PAIR[APPLY[gp[x_], PAIR[u_, APPLY[inv[gp[x_]], v_]]], v_]] :=
          union[u, complement[image[V, intersection[range[gp[x]], set[u]]]],
           complement[image[V, intersection[range[gp[x]], set[v]]]]]
```

---

simplification rules for inverse elements

Theorem. The inverse of an element of **range[gp[x]]** is an element of **range[gp[x]]**.

```
In[82]:= SubstTest[member, APPLY[funpart[t], u], range[funpart[t]], t → inv[gp[x]]] // Reverse

Out[82]= member[APPLY[inv[gp[x]], u], range[gp[x]]] == member[u, range[gp[x]]]

In[83]:= member[APPLY[inv[gp[x_]], u_], range[gp[x_]]] := member[u, range[gp[x]]]
```

Corollary. (Obtained by removing the **gp** wrapper.)

```
In[84]:= SubstTest[implies, and[equal[x, gp[t]], member[APPLY[inv[x], u], range[x]]],
           member[u, range[x]], t → x] // Reverse

Out[84]= or[member[u, range[x]], not[member[x, GROUPS]],
           not[member[APPLY[inv[x], u], range[x]]]] == True

In[85]:= or[member[u_, range[x_]], not[member[x_, GROUPS]],
           not[member[APPLY[inv[x_], u_], range[x_]]]] := True
```

Corollary. A simplification rule.

```
In[86]:= image[V, intersection[range[gp[x]], set[APPLY[inv[gp[x]], u]]]] // Normality

Out[86]= image[V, intersection[range[gp[x]], set[APPLY[inv[gp[x]], u]]]] ==
          image[V, intersection[range[gp[x]], set[u]]]

In[87]:= image[V, intersection[range[gp[x_]], set[APPLY[inv[gp[x_]], u_]]]] :=
          image[V, intersection[range[gp[x]], set[u]]]
```