groups are monoids

Johan G. F. Belinfante
2009 January 28

```
In[1]:= SetDirectory["l:"]; << goedel.09jan27a;<< tools.m

        :Package Title: goedel.09jan27a            2009 January 27 at 9:00 p.m.

        It is now:  2009 Jan 28 at 20:57

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 January 20

        weightlimit = 40
```

summary

Following Kurosh, a group is defined in the **GOEDEL** program to be a nonempty associative quasigroup.

```
In[2]:=   "A. G. Kurosh, Lectures on General Algebra, English
           Translation by K. A. Hirsh, Chelsea Publishing Co., New York, 1963.";
```

In this notebook it is shown that groups are monoids, and various corollaries of this fact. A key technique is the use of the double wrapper **quasigp[semigp[x]]**.

the main idea

Technical Lemma. If **v = v · u**, then **v \ v = u**. This lemma is needed due to the repeated occurrence of the variable **v**. The proof is obtained by instantiating a more general result containing an extra variable **t**, and merely setting **t** equal to **v**.

```
In[3]:= SubstTest[implies, and[equal[q, quasigp[x]],
           equal[v, APPLY[q, PAIR[t, u]]], member[t, range[q]], member[u, range[q]]],
          equal[u, APPLY[into[q], PAIR[t, v]]], {t → v, q → quasigp[x]}] // Reverse

Out[3]= or[equal[u, APPLY[rotate[quasigp[x]], PAIR[v, v]]],
          not[equal[v, APPLY[quasigp[x], PAIR[v, u]]]],
          not[member[u, range[quasigp[x]]]], not[member[v, range[quasigp[x]]]] == True

In[4]:= (% /. {u → u_, v → v_, x → x_}) /. Equal → SetDelayed
```

Lemma. If a quasigroup is associative then **u \ u = v \ v** for all elements in its range. This derivation closely mimics the proof given on page 31 in the referenced book by Kurosh. A key idea, borrowed from Kurosh, is to introduce two temporary abbreviations (**t** and **w**) which are eliminated from the final statement of the theorem. The double wrapper **quasigp[semigp[t]]** eliminates the need for an explicit literal for the associativity hypothesis.

```
In[5]:= (Map[not, SubstTest[and, implies[and[p0, p1], p3], implies[p2, p4],
              implies[and[p1, p5], p6], not[implies[and[p0, p1, p2], p7]],
              {p0 → and[equal[t, APPLY[rotate[composite[quasigp[x], SWAP]], PAIR[v, u]]],
                 equal[w, APPLY[rotate[quasigp[x]], PAIR[u, u]]]],
               p1 → and[member[u, range[quasigp[x]]], member[v, range[quasigp[x]]]],
               p2 → member[quasigp[x], SEMIGPS], p3 → equal[v, APPLY[quasigp[x], PAIR[t, u]]],
               p4 → equal[APPLY[quasigp[x], PAIR[t, APPLY[quasigp[x], PAIR[u, w]]]],
                 APPLY[quasigp[x], PAIR[APPLY[quasigp[x], PAIR[t, u]], w]]],
               p5 → equal[v, APPLY[quasigp[x], PAIR[v, w]]], p6 → member[w, range[quasigp[x]]],
               p7 → equal[w, APPLY[rotate[quasigp[x]], PAIR[v, v]]]}] /.
             {t :> APPLY[rotate[composite[quasigp[x], SWAP]], PAIR[v, u]],
              w :> APPLY[rotate[quasigp[x]], PAIR[u, u]]}] // Reverse) /. x → quasigp[semigp[t]]

Out[5]= or[equal[APPLY[rotate[quasigp[semigp[t]]], PAIR[u, u]],
           APPLY[rotate[quasigp[semigp[t]]], PAIR[v, v]]],
          not[member[u, range[quasigp[semigp[t]]]]],
          not[member[v, range[quasigp[semigp[t]]]]]] == True

In[6]:= (% /. {t → t_, u → u_, v → v_}) /. Equal → SetDelayed
```

---

eliminating two variables

Theorem. The function which takes **u** to **u \ u** is **fix[composite[inverse[FIRST], quasigp[semigp[x]]]]**. Eliminating the two variables **u** and **v** yields a statement which implies that this function is constant. The actual form of this statement is that this function contains a cartesian product with the same domain. The double wrapper **quasigp[semigp[x]]** here helps sequester the associativity hypothesis from the **class** rewrite rules that embody Gödel's algorithm for class-formation.

```
In[7]:= Map[empty[composite[Id, complement[#]]] &,
          SubstTest[class, pair[u, v], or[equal[APPLY[funpart[t], u], APPLY[funpart[t], v]],
            not[member[u, r]], not[member[v, r]]],
            {r → range[quasigp[semigp[x]]], t → composite[rotate[quasigp[semigp[x]]], DUP]}]]

Out[7]= subclass[cart[range[quasigp[semigp[x]]],
            range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]],
           fix[composite[inverse[FIRST], quasigp[semigp[x]]]]] == True

In[8]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. The range of a constant function is either empty or a singleton. (Eliminating the double wrapper **quasigp[semigp[x]]** would yield the statement that if **x** is a group, then the range of the function **fix[composite[inverse[FIRST], x]** is a singleton.)

```
In[9]:= SubstTest[implies, and[FUNCTION[w], subclass[cart[u, v], w]],
           or[empty[u], empty[v], member[v, range[SINGLETON]]], {u → range[quasigp[semigp[x]]],
            v :> range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]],
            w :> fix[composite[inverse[FIRST], quasigp[semigp[x]]]]}] // Reverse

Out[9]= or[equal[0, quasigp[semigp[x]]], member[
            range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]], range[SINGLETON]]] == True
```

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

Dual Lemma. One can use **flip** to obtain a similar statement about the function which takes **v** to **v / v**. (Eliminating the double wrapper would produce the statement that if **x** is a group, then the range of the function **inverse[fix[composite[inverse[SECOND], x]]]** is a singleton.)

```
In[11]:= SubstTest[or, equal[0, quasigp[semigp[t]]],
           member[range[fix[composite[inverse[FIRST], quasigp[semigp[t]]]]], range[SINGLETON]],
           t -> flip[quasigp[semigp[x]]]] // Reverse

Out[11]= or[equal[0, quasigp[semigp[x]]],
           member[domain[fix[composite[inverse[SECOND], quasigp[semigp[x]]]]],
            range[SINGLETON]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. An explicit formula for the function **t** that takes **u** to **u / u**. Any constant function **t** can be written as the cartesian product **cart[t, set[A[t]]]**. One does not need to make an exception here for the case of an empty function here because when **t** is empty, one has **A[t] = V**, and hence **set[A[t]] = 0**.

```
In[13]:= SubstTest[implies, and[FUNCTION[w], subclass[cart[domain[w], range[w]], w]],
           equal[w, cart[domain[w], set[A[range[w]]]]],
           {w -> fix[composite[inverse[FIRST], quasigp[semigp[x]]]]}] // Reverse

Out[13]= equal[cart[range[quasigp[semigp[x]]],
            set[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]]],
           fix[composite[inverse[FIRST], quasigp[semigp[x]]]]] == True

In[14]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary. (Obtained by removing the double wrapper.)

```
In[15]:= SubstTest[implies, equal[x, quasigp[semigp[t]]],
            equal[fix[composite[inverse[FIRST], x]], cart[range[x],
              set[A[range[fix[composite[inverse[FIRST], x]]]]]], t → x] // Reverse // MapNotNot

Out[15]= or[equal[cart[range[x], set[A[range[fix[composite[inverse[FIRST], x]]]]]],
            fix[composite[inverse[FIRST], x]]], not[member[x, GROUPS]]] == True

In[16]:= (% /. x → x_) /. Equal → SetDelayed
```

reintroducing a variable

Lemma. Applying the constant function **cart[r, set[v]]** to an element **u ∈ r** yields the value **v**.

```
In[17]:= SubstTest[implies, equal[s, t], equal[APPLY[s, u], APPLY[t, u]],
           {s -> fix[composite[inverse[FIRST], quasigp[semigp[x]]]],
            t -> cart[range[quasigp[semigp[x]]], set[A[range[fix[
                  composite[inverse[FIRST], quasigp[semigp[x]]]]]]]]}] // Reverse // MapNotNot

Out[17]= or[equal[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]],
           APPLY[rotate[quasigp[semigp[x]]], PAIR[u, u]]],
          not[member[u, range[quasigp[semigp[x]]]]] == True

In[18]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Theorem. Reintroducing a variable **u** yields a statement of the form **u = u · t**, where **t** is the single value of the constant function that takes **u** to **u \ u**.

```
In[19]:= Map[not, SubstTest[and, implies[p1, p3], implies[and[p1, p2, p3], p4],
            implies[and[p2, p4], p5], not[implies[and[p1, p2], p5]],
            {p1 → equal[t, A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]],
             p2 → member[u, range[quasigp[semigp[x]]]],
             p3 -> equal[cart[range[quasigp[semigp[x]]], set[t]],
               fix[composite[inverse[FIRST], quasigp[semigp[x]]]]],
             p4 -> equal[t, APPLY[rotate[quasigp[semigp[x]]], PAIR[u, u]]],
             p5 → equal[u, APPLY[quasigp[semigp[x]], PAIR[u, t]]]}] /.
           t -> A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]] // Reverse

Out[19]= or[equal[u, APPLY[quasigp[semigp[x]],
           PAIR[u, A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]]],
          not[member[u, range[quasigp[semigp[x]]]]] == True

In[20]:= (% /. {x → x_, u → u_}) /. Equal → SetDelayed
```

Theorem. Eliminating the variable **u**.

```
In[21]:= Map[equal[V, #] &,
          SubstTest[class, u, or[equal[u, APPLY[funpart[q], u]], not[member[u, r]]],
           {q → composite[quasigp[semigp[x]],
              RIGHT[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]]],
            r → range[quasigp[semigp[x]]]}]]

Out[21]= subclass[range[quasigp[semigp[x]]],
          image[inverse[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]],
           set[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]]] == True

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

Lemma. The single value **v** of the function that takes **u** to **u \ u** is right-neutral; that is, right multiplication by **v** is contained in **Id**.

```
In[23]:= (Map[or[not[subclass[range[quasigp[u]],
                 image[fix[composite[inverse[SECOND], quasigp[u]]], set[v]]]],
               implies[#, subclass[composite[quasigp[u], LEFT[v]], Id]]] &,
             SubstTest[subclass, domain[funpart[t]], fix[funpart[t]],
              t → composite[quasigp[u], LEFT[v]]] // Reverse //
            MapNotNot) /. {u → flip[quasigp[semigp[x]]],
           v -> A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]}

Out[23]= subclass[composite[quasigp[semigp[x]],
           RIGHT[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]]], Id] == True

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

Theorem. Eliminating the double-wrapper **quasigp[semigp[x]]** yields the statement that for any group, there is an element which is right-neutral.

```
In[25]:= Map[implies[member[x, GROUPS], #] &,
           SubstTest[implies, equal[x, quasigp[semigp[t]]], subclass[composite[x,
             RIGHT[A[range[fix[composite[inverse[FIRST], x]]]]], Id], t → x]] // Reverse

Out[25]= or[not[member[x, GROUPS]], subclass[
            composite[x, RIGHT[A[range[fix[composite[inverse[FIRST], x]]]]], Id]] == True

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

Lemma. One only needs to use **flip** rules to establish the existence of a left-neutral element.

```
In[27]:= SubstTest[subclass, composite[quasigp[semigp[t]],
            RIGHT[A[range[fix[composite[inverse[FIRST], quasigp[semigp[t]]]]]]],
           Id, t → flip[quasigp[semigp[x]]]] // Reverse

Out[27]= subclass[composite[quasigp[semigp[x]],
            LEFT[A[domain[fix[composite[inverse[SECOND], quasigp[semigp[x]]]]]], Id] == True

In[28]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. Eliminating the double-wrapper **quasigp[semigp[x]]** this time yields the statement that for any group, there is an element which is left-neutral.

```
In[29]:= Map[implies[member[x, GROUPS], #] &,
           SubstTest[implies, equal[x, quasigp[semigp[t]]], subclass[composite[x,
             LEFT[A[domain[fix[composite[inverse[SECOND], x]]]]], Id], t → x]] // Reverse

Out[29]= or[not[member[x, GROUPS]], subclass[
            composite[x, LEFT[A[domain[fix[composite[inverse[SECOND], x]]]]], Id]] == True

In[30]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma.

```
In[31]:= SubstTest[implies, subclass[u, v], subclass[range[u], range[v]],
           {u -> fix[composite[inverse[FIRST], quasigp[x]]], v → domain[quasigp[x]]}] // Reverse

Out[31]= subclass[range[fix[composite[inverse[FIRST], quasigp[x]]]], range[quasigp[x]]] == True

In[32]:= (% /. x → x_) /. Equal→ SetDelayed
```

Lemma.

```
In[33]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p3, p4],
            not[implies[p1, p4]], {p1 → not[empty[quasigp[semigp[x]]]], p2 -> member[
              range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]], range[SINGLETON]],
             p3 -> member[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]],
              range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]],
             p4 -> member[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]],
              range[quasigp[semigp[x]]]]}]] // Reverse

Out[33]= or[equal[0, quasigp[semigp[x]]],
           member[A[range[fix[composite[inverse[FIRST], quasigp[semigp[x]]]]]],
            range[quasigp[semigp[x]]]]] == True

In[34]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. The right-neutral element belongs to the range.

```
In[35]:= SubstTest[implies, equal[x, quasigp[semigp[t]]],
            or[equal[0, x], member[A[range[fix[composite[inverse[FIRST], x]]]], range[x]]],
            t → x] // MapNotNot // Reverse

Out[35]= or[member[A[range[fix[composite[inverse[FIRST], x]]]], range[x]],
           not[member[x, GROUPS]]] == True

In[36]:= (% /. x → x_) /. Equal→ SetDelayed
```

Similarly, using duality:

```
In[37]:= SubstTest[or, equal[0, quasigp[semigp[t]]],
           member[A[range[fix[composite[inverse[FIRST], quasigp[semigp[t]]]]]],
            range[quasigp[semigp[t]]]], t → flip[quasigp[semigp[x]]]] // Reverse

Out[37]= or[equal[0, quasigp[semigp[x]]],
           member[A[domain[fix[composite[inverse[SECOND], quasigp[semigp[x]]]]]],
            range[quasigp[semigp[x]]]]] == True

In[38]:= (% /. x → x_) /. Equal→ SetDelayed
```

Theorem. The left-neutral element belongs to the range.

```
In[39]:= SubstTest[implies, equal[x, quasigp[semigp[t]]],
            or[equal[0, x], member[A[domain[fix[composite[inverse[SECOND], x]]]], range[x]]],
            t → x] // MapNotNot // Reverse

Out[39]= or[member[A[domain[fix[composite[inverse[SECOND], x]]]], range[x]],
           not[member[x, GROUPS]]] == True
```

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

Theorem. Uniqueness theorem for identities specialized to the case of quasigroups.

```
In[41]:= SubstTest[implies, and[member[u, V], member[v, V],
            member[pair[u, v], domain[t]], subclass[composite[t, LEFT[u]], Id],
            subclass[composite[t, RIGHT[v]], Id]], equal[u, v], t → quasigp[x]] // Reverse
```

```
Out[41]= or[equal[u, v], not[member[u, range[quasigp[x]]]], not[member[v, range[quasigp[x]]]],
           not[subclass[composite[quasigp[x], LEFT[u]], Id]],
           not[subclass[composite[quasigp[x], RIGHT[v]], Id]]] == True
```

```
In[42]:= (% /. {x → x_, u → u_, v → v_}) /. Equal → SetDelayed
```

Corollary. (Uniqueness of identities for groups, obtained by removing the **quasigp** wrapper.) If **x** is a group, and **u ∈ range[x]** is left-neutral and **v ∈ range[x]** is right-neutral, then **u = v**.

```
In[43]:= SubstTest[implies, equal[x, quasigp[semigp[t]]],
            or[equal[u, v], not[member[u, range[x]]],
             not[member[v, range[x]]], not[subclass[composite[x, LEFT[u]], Id]],
             not[subclass[composite[x, RIGHT[v]], Id]]], t → x] // Reverse // MapNotNot
```

```
Out[43]= or[equal[u, v], not[member[u, range[x]]], not[member[v, range[x]]],
           not[member[x, GROUPS]], not[subclass[composite[x, LEFT[u]], Id]],
           not[subclass[composite[x, RIGHT[v]], Id]]] == True
```

```
In[44]:= or[equal[u_, v_], not[member[u_, range[x]]], not[member[v_, range[x]]],
           not[member[x_, GROUPS]], not[subclass[composite[x_, LEFT[u_]], Id]],
           not[subclass[composite[x_, RIGHT[v_]], Id]]] := True
```

Theorem. The left and right neutral elements are the same.

```
In[45]:= (Map[not, SubstTest[and, implies[p1, p2],
             implies[p1, p3], implies[p1, p4], implies[p1, p5], not[implies[p1, p6]],
             {p1 → and[equal[u, A[domain[fix[composite[inverse[SECOND], x]]]]],
                equal[v, A[range[fix[composite[inverse[FIRST], x]]]]], member[x, GROUPS]],
              p2 -> member[u, range[x]], p3 -> member[v, range[x]],
              p4 -> subclass[composite[x, LEFT[u]], Id],
              p5 -> subclass[composite[x, RIGHT[v]], Id], p6 → equal[u, v]}]] // Reverse) /.
          {u -> A[domain[fix[composite[inverse[SECOND], x]]]],
           v -> A[range[fix[composite[inverse[FIRST], x]]]]}
```

```
Out[45]= or[equal[A[domain[fix[composite[inverse[SECOND], x]]]],
             A[range[fix[composite[inverse[FIRST], x]]]]], not[member[x, GROUPS]]] == True
```

```
In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. A condition for membership in the class **ids[x]** for quasigroups.

```
In[47]:= Map[implies[member[t, #], member[t, ids[quasigp[x]]]] &, SubstTest[intersection,
           fix[domain[t]], complement[domain[fix[composite[inverse[SECOND], Di, t]]]],
           complement[range[fix[composite[inverse[FIRST], Di, t]]]], t → quasigp[x]]] // Reverse

Out[47]= or[member[t, ids[quasigp[x]]], not[member[t, range[quasigp[x]]]],
           not[subclass[composite[quasigp[x], LEFT[t]], Id]],
           not[subclass[composite[quasigp[x], RIGHT[t]], Id]]] == True

In[48]:= (% /. {x → x_, t → t_}) /. Equal → SetDelayed
```

Corollary. Obtained by removing the **quasigp** wrapper.

```
In[49]:= SubstTest[implies, equal[x, quasigp[w]], or[member[t, ids[x]],
           not[member[t, range[x]]], not[subclass[composite[x, LEFT[t]], Id]],
           not[subclass[composite[x, RIGHT[t]], Id]]], w → x] // Reverse

Out[49]= or[member[t, ids[x]], not[member[t, range[x]]],
           not[member[x, QUASIGPS]], not[subclass[composite[x, LEFT[t]], Id]],
           not[subclass[composite[x, RIGHT[t]], Id]]] == True

In[50]:= or[member[t_, ids[x_]], not[member[t_, range[x_]]],
           not[member[x_, QUASIGPS]], not[subclass[composite[x_, LEFT[t_]], Id]],
           not[subclass[composite[x_, RIGHT[t_]], Id]]] := True
```

Theorem. (This takes a while.)

```
In[51]:= (Map[not, SubstTest[and, implies[and[p2, p4], p6],
            implies[and[p0, p3, p5, p6], p7], not[implies[p1, p7]], {p0 → member[x, QUASIGPS],
             p1 -> and[equal[u, A[domain[fix[composite[inverse[SECOND], x]]]]],
               equal[v, A[range[fix[composite[inverse[FIRST], x]]]]], member[x, GROUPS]],
             p2 → equal[u, v], p3 -> subclass[composite[x, LEFT[u]], Id],
             p5 → member[u, range[x]], p4 -> subclass[composite[x, RIGHT[v]], Id],
             p5 → member[u, range[x]], p6 -> subclass[composite[x, RIGHT[u]], Id],
             p7 → member[u, ids[x]]}]] // Reverse) /.
         {u -> A[domain[fix[composite[inverse[SECOND], x]]]],
          v -> A[range[fix[composite[inverse[FIRST], x]]]]}

Out[51]= or[member[A[domain[fix[composite[inverse[SECOND], x]]]], ids[x]],
           not[member[x, GROUPS]]] == True

In[52]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[53]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p4], p5],
           not[implies[p1, p5]], {p1 → member[x, GROUPS], p2 → member[x, SEMIGPS],
            p3 -> member[A[domain[fix[composite[inverse[SECOND], x]]]], ids[x]],
            p4 → not[empty[ids[x]]], p5 → member[x, MONOIDS]}]] // Reverse

Out[53]= or[member[x, MONOIDS], not[member[x, GROUPS]]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

Main Theorem. Every group is a monoid.

```
In[55]:= Map[equal[V, #] &, dif[GROUPS, MONOIDS] // complement // Normality]

Out[55]= subclass[GROUPS, MONOIDS] == True

In[56]:= subclass[GROUPS, MONOIDS] := True
```

various corollaries

Theorem. If **x** is a group, then **e[x]** is an identity element.

```
In[57]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, GROUPS],
           p2 -> member[x, MONOIDS], p3 → member[e[x], ids[x]]}]] // Reverse

Out[57]= or[member[e[x], ids[x]], not[member[x, GROUPS]]] == True

In[58]:= or[member[e[x_], ids[x_]], not[member[x_, GROUPS]]] := True
```

Theorem. If **x** is a group, then **e[x] ∈ range[x]**.

```
In[59]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, GROUPS],
           p2 -> member[x, MONOIDS], p3 → member[e[x], range[x]]}]] // Reverse

Out[59]= or[member[e[x], range[x]], not[member[x, GROUPS]]] == True
```

Corollary.

```
In[60]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, GROUPS],
            p2 -> member[e[x], ids[x]], p3 → not[empty[ids[x]]]}]] // Reverse

Out[60]= or[not[equal[0, ids[x]]], not[member[x, GROUPS]]] == True

In[61]:= or[not[equal[0, ids[x_]]], not[member[x_, GROUPS]]] := True
```

Theorem. The class of identities for a group **x** is the singleton of **e[x]**.

```
In[62]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, GROUPS],
           p2 -> member[x, MONOIDS], p3 → equal[set[e[x]], ids[x]]}]] // Reverse

Out[62]= or[equal[ids[x], set[e[x]]], not[member[x, GROUPS]]] == True

In[63]:= or[equal[ids[x_], set[e[x_]]], not[member[x_, GROUPS]]] := True
```

Corollary. The only identity element is **e[x]**.

```
In[64]:= Map[not, SubstTest[and, implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[x, GROUPS], p2 -> member[x, MONOIDS],
            p3 → implies[member[u, ids[x]], equal[u, e[x]]]}]] // Reverse

Out[64]= or[equal[u, e[x]], not[member[u, ids[x]]], not[member[x, GROUPS]]] == True

In[65]:= or[equal[u_, e[x_]], not[member[x_, GROUPS]], not[member[u_, ids[x_]]]] := True
```

Theorem.

```
In[66]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 -> member[x, GROUPS],
            p2 → member[A[domain[fix[composite[inverse[SECOND], x]]]], ids[x]],
            p3 → equal[e[x], A[domain[fix[composite[inverse[SECOND], x]]]]}]] // Reverse

Out[66]= or[equal[A[domain[fix[composite[inverse[SECOND], x]]]], e[x]],
           not[member[x, GROUPS]]] == True

In[67]:= or[equal[A[domain[fix[composite[inverse[SECOND], x_]]]], e[x_]],
           not[member[x_, GROUPS]]] := True
```

Dual result.

```
In[68]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], p3], not[implies[p1, p3]], {p1 -> member[x, GROUPS],
            p2 → equal[e[x], A[domain[fix[composite[inverse[SECOND], x]]]]],
            p3 -> equal[e[x], A[range[fix[composite[inverse[FIRST], x]]]]}]] // Reverse

Out[68]= or[equal[A[range[fix[composite[inverse[FIRST], x]]]], e[x]],
           not[member[x, GROUPS]]] == True

In[69]:= or[equal[A[range[fix[composite[inverse[FIRST], x_]]]], e[x_]],
           not[member[x_, GROUPS]]] := True
```

Theorem. Formula for the function **fix[composite[inverse[FIRST], x]]**.

```
In[70]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], not[implies[p1, p4]],
           {p1 → member[x, GROUPS], p2 → equal[fix[composite[inverse[FIRST], x]],
              cart[range[x], set[A[range[fix[composite[inverse[FIRST], x]]]]]],
            p3 → equal[e[x], A[range[fix[composite[inverse[FIRST], x]]]]], p4 →
             equal[fix[composite[inverse[FIRST], x]], cart[range[x], set[e[x]]]]}]] // Reverse

Out[70]= or[equal[cart[range[x], set[e[x]]], fix[composite[inverse[FIRST], x]]],
           not[member[x, GROUPS]]] == True

In[71]:= or[equal[cart[range[x_], set[e[x_]]], fix[composite[inverse[FIRST], x_]]],
           not[member[x_, GROUPS]]] := True
```

Dual result.

```
In[72]:= SubstTest[implies, member[t, GROUPS], equal[cart[range[t], set[e[t]]],
           fix[composite[inverse[FIRST], t]]], t → flip[x]] // Reverse

Out[72]= or[equal[cart[set[e[x]], image[x, cart[V, V]]], fix[composite[inverse[SECOND], x]]],
           not[member[composite[x, SWAP], GROUPS]]] == True

In[73]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. Range formula needed to clean up dual theorems about groups.

```
In[74]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[x, GROUPS],
             p2 → member[x, QUASIGPS], p3 → equal[range[x], image[x, cart[V, V]]]}]] // Reverse

Out[74]= or[equal[image[x, cart[V, V]], range[x]], not[member[x, GROUPS]]] == True

In[75]:= or[equal[image[x_, cart[V, V]], range[x_]], not[member[x_, GROUPS]]] := True
```

Corollary. The class **fix[composite[inverse[SECOND], x]]** is the inverse of a constant function.

```
In[76]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[and[p2, p3], p4], not[implies[p1, p4]], {p1 → member[x, GROUPS],
            p2 → equal[range[x], image[x, cart[V, V]]], p3 → member[flip[x], GROUPS], p4 →
             equal[cart[set[e[x]], range[x]], fix[composite[inverse[SECOND], x]]]}]] // Reverse

Out[76]= or[equal[cart[set[e[x]], range[x]], fix[composite[inverse[SECOND], x]]],
           not[member[x, GROUPS]]] == True

In[77]:= or[equal[cart[set[e[x_]], range[x_]], fix[composite[inverse[SECOND], x_]]],
           not[member[x_, GROUPS]]] := True
```

APPLY rules for e[x]

In elementary texts on group theory, the axioms for a group **x** are usually formulated using elements of **range[x]**. The laws about the identity element **e[x]** can now be derived as corollaries of similar laws which hold in any monoid.

Theorem. If **x** is a group and **u ∈ range[x]**, then **e[x] · u = u**.

```
In[78]:= Map[not, SubstTest[and, implies[and[p2, p3], p4],
           not[implies[and[p1, p2], p4]], {p1 → member[x, GROUPS], p2 → member[u, range[x]],
            p3 → member[x, MONOIDS], p4 → equal[APPLY[x, PAIR[e[x], u]], u]}]] // Reverse

Out[78]= or[equal[u, APPLY[x, PAIR[e[x], u]]],
           not[member[u, range[x]]], not[member[x, GROUPS]]] == True

In[79]:= or[equal[u_, APPLY[x_, PAIR[e[x_], u_]]],
           not[member[u_, range[x_]]], not[member[x_, GROUPS]]] := True
```

Theorem. Similarly, if **x** is a group and **u ∈ range[x]**, then **u · e[x] = u**.

```
In[80]:= Map[not, SubstTest[and, implies[and[p2, p3], p4],
            not[implies[and[p1, p2], p4]], {p1 → member[x, GROUPS], p2 → member[u, range[x]],
             p3 → member[x, MONOIDS], p4 → equal[APPLY[x, PAIR[u, e[x]]], u]}]] // Reverse

Out[80]= or[equal[u, APPLY[x, PAIR[u, e[x]]]],
           not[member[u, range[x]]], not[member[x, GROUPS]]] == True

In[81]:= or[equal[u_, APPLY[x_, PAIR[u_, e[x_]]]],
           not[member[u_, range[x_]]], not[member[x_, GROUPS]]] := True
```

---

groups are loops

Lemma.

```
In[82]:= Map[not, SubstTest[and, implies[and[p2, p3], p4],
            not[implies[p1, p4]], {p1 → member[x, GROUPS], p2 → member[x, QUASIGPS],
             p3 → not[empty[ids[x]]], p4 → member[x, LOOPS]}]] // Reverse

Out[82]= or[member[x, LOOPS], not[member[x, GROUPS]]] == True

In[83]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. Groups are loops.

```
In[84]:= Map[equal[V, #] &, dif[GROUPS, LOOPS] // complement // Normality]

Out[84]= subclass[GROUPS, LOOPS] == True

In[85]:= subclass[GROUPS, LOOPS] := True
```