powers of commuting elements of a monoid

Johan G. F. Belinfante
2013 June 11

```
In[1]:= SetDirectory["l:"]; << goedel.13jun09a

        :Package Title: goedel.13jun09a                2013 June 9 at 8:05 a.m.

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

        It is now:  2013 Jun 11 at 14:4

        Loading Simplification Rules

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

        weightlimit = 40

        Loading completed.

        It is now:  2013 Jun 11 at 14:21
```

summary

If two elements of a monoid commute, then all powers of the one commute with all powers of the other.

derivation

First it will be shown that if two elements of a monoid commute, then each one commutes with all powers of the other.

Lemma. (Application of an intertwine rule for **iterate**.)

```
In[2]:= Map[implies[and[member[y, range[monoid[x]]], member[z, range[monoid[x]]]], #] &,
          SubstTest[implies, commute[u, v], equal[iterate[u, image[v, w]],
            composite[v, iterate[u, w]]], {u → composite[monoid[x], LEFT[y]],
            v → composite[monoid[x], LEFT[z]], w → set[e[monoid[x]]]}]] // Reverse

Out[2]= or[equal[composite[monoid[x], LEFT[z],
            iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
           iterate[composite[monoid[x], LEFT[y]], intersection[range[monoid[x]], set[z]]]],
          not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
          not[member[y, range[monoid[x]]]], not[member[z, range[monoid[x]]]]] == True

In[3]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The above result can be cleaned up. The first step is to eliminate an unnecessary intersection.

Lemma. (Eliminate an intersection.)

```
In[4]:= SubstTest[implies, equal[t, intersection[range[monoid[x]], set[z]]],
          or[equal[composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
              set[e[monoid[x]]]]]], iterate[composite[monoid[x], LEFT[y]], t]],
           not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
           not[member[y, range[monoid[x]]]],
           not[member[z, range[monoid[x]]]]], t → set[z]] // Reverse

Out[4]= or[equal[composite[monoid[x], LEFT[z],
            iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]],
           iterate[composite[monoid[x], LEFT[y]], set[z]]],
          not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
          not[member[y, range[monoid[x]]]], not[member[z, range[monoid[x]]]]] == True

In[5]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

The next step is to rewrite the expression **iterate[monoid[x] ∘ LEFT[y], {z}]** in terms of the power list **iterate[monoid[x] ∘ LEFT[y], {e[monoid[x]]}]**.

Lemma.

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
           {p1 -> and[member[y, range[monoid[x]]], member[z, range[monoid[x]]],
              equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
            p2 -> equal[composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
                set[e[monoid[x]]]]], iterate[composite[monoid[x], LEFT[y]], set[z]]],
            p3 -> equal[composite[monoid[x], RIGHT[z], iterate[composite[monoid[x], LEFT[y]],
                set[e[monoid[x]]]]], iterate[composite[monoid[x], LEFT[y]], set[z]]],
            p4 -> equal[composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
                set[e[monoid[x]]]]], composite[monoid[x], RIGHT[z],
               iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]}]] // Reverse

Out[8]= or[equal[composite[monoid[x], LEFT[z],
            iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
            RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
          not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
          not[member[y, range[monoid[x]]]], not[member[z, range[monoid[x]]]]] == True

        (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

There are two redundant literals here.

Lemma. The membership literal for **z** is redundant.

```
In[13]:= Map[or[#, member[z, range[monoid[x]]]] &,
          SubstTest[implies, and[empty[u], empty[v]], equal[u, v],
           {u -> composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
               set[e[monoid[x]]]]], v -> composite[monoid[x], RIGHT[z],
              iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]}]] // Reverse

Out[13]= or[equal[composite[monoid[x], LEFT[z],
             iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
             RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
           member[z, range[monoid[x]]]] == True

In[14]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma. The membership literal for **y** is redundant.

```
In[16]:= SubstTest[implies, empty[t],
          equal[composite[monoid[x], LEFT[z], iterate[t, set[e[monoid[x]]]]],
           composite[monoid[x], RIGHT[z], iterate[t, set[e[monoid[x]]]]]],
          t → composite[monoid[x], LEFT[y]]] // Reverse

Out[16]= or[equal[composite[monoid[x], LEFT[z],
             iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
             RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
           member[y, range[monoid[x]]]] == True

In[17]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. If elements **y** and **z** of a monoid commute, then every power of **y** commutes with **z**.

```
In[19]:= SubstTest[and, or[p, q], implies[p, q],
          {p -> and[member[y, range[monoid[x]]], member[z, range[monoid[x]]]],
           q -> or[equal[composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
                set[e[monoid[x]]]]], composite[monoid[x], RIGHT[z],
               iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]], not[
              equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]]]}] // MapNotNot

Out[19]= or[equal[composite[monoid[x], LEFT[z],
             iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
             RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
           not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]]] == True

In[21]:= or[equal[composite[monoid[x_], LEFT[z_], iterate[
              composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]]], composite[monoid[x_],
             RIGHT[z_], iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]]]],
           not[equal[APPLY[monoid[x_], PAIR[y_, z_]], APPLY[monoid[x_], PAIR[z_, y_]]]]] := True
```

introducing additional variables

The result of the preceding section can be made more explicit by introducing an additional variable.

Lemma. Introducing a new variable **w**.

```
In[22]:= SubstTest[implies, equal[u, v], equal[image[u, set[w]], image[v, set[w]]],
           {u -> composite[monoid[x], LEFT[z], iterate[composite[monoid[x], LEFT[y]],
               set[e[monoid[x]]]]], v -> composite[monoid[x], RIGHT[z],
              iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]}] // Reverse

Out[22]= or[equal[APPLY[monoid[x],
             PAIR[z, APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], w]]],
            APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], w], z]]],
           not[equal[composite[monoid[x], LEFT[z],
              iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
              RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]]] == True

In[23]:= (% /. {w → w_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem.

```
In[24]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 -> equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]],
             p2 -> equal[composite[monoid[x], LEFT[z], iterate[
                 composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], composite[monoid[x],
               RIGHT[z], iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]],
             p3 -> equal[APPLY[monoid[x], PAIR[z, APPLY[iterate[composite[monoid[x], LEFT[y]],
                   set[e[monoid[x]]]], u]]], APPLY[monoid[x], PAIR[APPLY[iterate[
                   composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u], z]]]}]] // Reverse

Out[24]= or[equal[APPLY[monoid[x],
             PAIR[z, APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u]]],
            APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u], z]]],
           not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]]] == True

In[35]:= or[equal[APPLY[monoid[x_],
             PAIR[APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], u_],
              z_]], APPLY[monoid[x_], PAIR[z_,
              APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], u_]]]],
           not[equal[APPLY[monoid[x_], PAIR[y_, z_]], APPLY[monoid[x_], PAIR[z_, y_]]]]] := True
```

Corollary. If **y** commute with **z**, then any power of **y** commutes with any power of **z**.

```
In[37]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
          {p1 -> equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]],
           p2 -> equal[APPLY[monoid[x],
              PAIR[z, APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u]]],
             APPLY[monoid[x], PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]],
                 set[e[monoid[x]]]], u], z]]], p3 -> equal[APPLY[monoid[x],
              PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
               APPLY[iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]], v]]],
             APPLY[monoid[x], PAIR[APPLY[iterate[composite[monoid[x], LEFT[z]],
                 set[e[monoid[x]]]], v], APPLY[
                iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u]]]]}]] // Reverse

Out[37]= or[equal[APPLY[monoid[x],
           PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
            APPLY[iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]], v]]], APPLY[
           monoid[x], PAIR[APPLY[iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]],
             v], APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u]]]],
         not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]]] == True

In[39]:= or[equal[APPLY[monoid[x_],
           PAIR[APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], u_],
            APPLY[iterate[composite[monoid[x_], LEFT[z_]], set[e[monoid[x_]]]],
             v_]]], APPLY[monoid[x_],
           PAIR[APPLY[iterate[composite[monoid[x_], LEFT[z_]], set[e[monoid[x_]]]], v_],
            APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], u_]]]],
         not[equal[APPLY[monoid[x_], PAIR[y_, z_]], APPLY[monoid[x_], PAIR[z_, y_]]]]] := True
```

eliminating the variables u and v

The following special rule helps with eliminating the two variables **u** and **v** introduced in the preceding section.

Lemma. A simplification rule.

```
In[41]:= composite[cross[x, y], id[composite[Id, z]]] // TripleRotate

Out[41]= composite[cross[x, y], id[composite[Id, z]]] == composite[cross[x, y], id[z]]

In[42]:= composite[cross[x_, y_], id[composite[Id, z_]]] := composite[cross[x, y], id[z]]
```

Lemma.

```
In[43]:= intersection[case[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
           dif[composite[monoid[x],
             cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
              iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]], composite[monoid[x],
             SWAP, cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
              iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]]] // VSTriNormality

Out[43]= composite[intersection[composite[complement[monoid[x]], SWAP], monoid[x]],
           cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]],
           id[intersection[complement[image[V, intersection[APPLY[monoid[x], PAIR[y, z]],
                complement[APPLY[monoid[x], PAIR[z, y]]]]]], complement[image[V, intersection[
                APPLY[monoid[x], PAIR[z, y]], complement[APPLY[monoid[x], PAIR[y, z]]]]]]]]] == 0

In[44]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[46]:= SubstTest[empty, intersection[case[p], dif[u, v]],
          {p -> equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]],
           u -> composite[monoid[x], cross[iterate[composite[monoid[x], LEFT[y]],
               set[e[monoid[x]]]], iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]],
           v -> composite[monoid[x], SWAP, cross[iterate[composite[monoid[x], LEFT[y]], set[
                e[monoid[x]]]], iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]}]

Out[46]= or[not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
           subclass[composite[monoid[x],
             cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
              iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]], composite[monoid[x],
             SWAP, cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
              iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]]] == True

In[47]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[48]:= SubstTest[implies, subclass[u, v],
           subclass[composite[u, w], composite[v, w]], {u → composite[x, cross[y, z]],
            v → composite[x, SWAP, cross[y, z]], w → SWAP}] // Reverse

Out[48]= or[not[subclass[composite[x, cross[y, z]], composite[x, SWAP, cross[y, z]]]],
           subclass[composite[x, SWAP, cross[z, y]], composite[x, cross[z, y]]]] == True

In[49]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[50]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]],
            p2 -> subclass[composite[monoid[x],
               cross[iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]], iterate[
                 composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]], composite[monoid[x],
               SWAP, cross[iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]],
                iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]],
            p3 -> subclass[composite[monoid[x], SWAP, cross[
                iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
                iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]], composite[
               monoid[x], cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
                iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]]}]] // Reverse

Out[50]= or[not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]],
          subclass[composite[monoid[x], SWAP,
            cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]], composite[
            monoid[x], cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]]] == True

In[51]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. If **y** and **z** are commuting elements of a monoid, then all powers of **y** commute with all powers of **z**.

```
In[52]:= SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
          {p -> equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]],
           u -> composite[monoid[x], SWAP, cross[iterate[composite[monoid[x], LEFT[y]],
               set[e[monoid[x]]]], iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]],
           v -> composite[monoid[x], cross[iterate[composite[monoid[x], LEFT[y]], set[
                e[monoid[x]]]], iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]}]

Out[52]= or[equal[composite[monoid[x],
            cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]], composite[monoid[x],
            SWAP, cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             iterate[composite[monoid[x], LEFT[z]], set[e[monoid[x]]]]]]],
          not[equal[APPLY[monoid[x], PAIR[y, z]], APPLY[monoid[x], PAIR[z, y]]]]] == True

In[53]:= or[equal[composite[monoid[x_],
            cross[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], iterate[
              composite[monoid[x_], LEFT[z_]], set[e[monoid[x_]]]]]], composite[monoid[x_],
            SWAP, cross[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]],
             iterate[composite[monoid[x_], LEFT[z_]], set[e[monoid[x_]]]]]]],
          not[equal[APPLY[monoid[x_], PAIR[y_, z_]], APPLY[monoid[x_], PAIR[z_, y_]]]]] := True
```

a special case

A special application of interest is the case that **y** and **z** are inverse elements of a group.

Corollary. Any power of an element of a group commutes with any power of its inverse.

```
In[55]:= SubstTest[implies,
           equal[APPLY[monoid[t], PAIR[y, z]], APPLY[monoid[t], PAIR[z, y]]], equal[composite[
             monoid[t], cross[iterate[composite[monoid[t], LEFT[y]], set[e[monoid[t]]]],
              iterate[composite[monoid[t], LEFT[z]], set[e[monoid[t]]]]]], composite[monoid[t],
             SWAP, cross[iterate[composite[monoid[t], LEFT[y]], set[e[monoid[t]]]],
              iterate[composite[monoid[t], LEFT[z]], set[e[monoid[t]]]]]]]],
           {t → gp[x], z → APPLY[inv[gp[x]], y]}] // Reverse

Out[55]= equal[composite[gp[x], cross[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]]],
           composite[gp[x], SWAP, cross[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]]]] == True

In[57]:= composite[gp[x_], SWAP, cross[iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]],
            composite[inv[gp[x_]], iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]]]]] :=
          composite[gp[x], cross[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
            composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]]]
```

Corollary. (An analogous result with **y** replaced by its inverse.)

```
In[59]:= SubstTest[composite, gp[x], SWAP,
           cross[iterate[composite[gp[x], LEFT[t]], set[e[gp[x]]]],
            composite[inv[gp[x]], iterate[composite[gp[x], LEFT[t]], set[e[gp[x]]]]]],
           t → APPLY[inv[gp[x]], y]] // Reverse

Out[59]= composite[gp[x], SWAP,
           cross[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]],
            iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]] == composite[gp[x],
           cross[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]],
            iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]]

In[60]:= composite[gp[x_], SWAP,
           cross[composite[inv[gp[x_]], iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]]],
            iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]]]] := composite[gp[x],
           cross[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]],
            iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]]]
```