integer powers of group elements, part 1

Johan G. F. Belinfante
2013 June 19

```
In[1]:= SetDirectory["l:"]; << goedel.13jun11a

        :Package Title: goedel.13jun11a                2013 June 11 at 10:15 p.m.

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

        It is now:  2013 Jun 19 at 11:1

        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 19 at 11:18
```

summary

This is the first of a series of notebooks on integer powers of group elements. In this notebook a class **intpow[x, y]** is introduced, and some of its basic properties are derived. In particular, it is shown that if **x** is a group and **y ∈ range[x]**, then **intpow[x, y]** is a mapping from **Z** to **range[x]**.

integer power function

Defnition. The class **intpow[x, y]** is defined as follows.

```
In[2]:= union[composite[iterate[composite[x_, LEFT[y_]], set[e[x_]]], inverse[PLUS]],
          composite[inv[x_], iterate[composite[x_, LEFT[y_]], set[e[x_]]],
           inverse[PLUS], INVERSE]] := intpow[x, y]
```

It will be shown in this notebook that if **x** is a group and **y ∈ range[x]**, then **intpow[x, y]** is a mapping from the set **Z** of all integers to **range[x]**.

simplification rules

Two basic simplification rules for **intpow[x, y]** are derived in this section.

Theorem. The domain of **intpow[x, y]** is a subset of the set of integers.

```
In[3]:= SubstTest[composite, union[u, v], id[Z],
          {u -> composite[iterate[composite[x, LEFT[y]], set[e[x]]], inverse[PLUS]],
           v -> composite[inv[x], iterate[composite[x, LEFT[y]], set[e[x]]],
             inverse[PLUS], INVERSE]}] // Reverse

Out[3]= composite[intpow[x, y], id[Z]] == intpow[x, y]

In[4]:= composite[intpow[x_, y_], id[Z]] := intpow[x, y]
```

For the other simplification rule, the **gp** wrapper is needed.

Theorem. The range of **intpow[gp[x], y]** is a subset of the range of **gp[x]**.

```
In[5]:= SubstTest[composite, id[range[gp[x]]], union[u, v],
          {u -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
           v -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE]}] // Reverse

Out[5]= composite[id[range[gp[x]]], intpow[gp[x], y]] == intpow[gp[x], y]

In[6]:= composite[id[range[gp[x_]]], intpow[gp[x_], y_]] := intpow[gp[x], y]
```

connection with the power list

For non-negative integers, the following theorem relates the class **intpow[x, y]** to the list **iterate[x ∘ LEFT[y], {e[x]}]** of powers of the element **y**. The wrapper **gp[x]** is needed here.

Theorem. Connection between integer powers and the natural number power list.

```
In[8]:= Map[equal[#, iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]] &,
          SubstTest[composite, union[u, v], PLUS,
           {u -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            v -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
              inverse[PLUS], INVERSE]}]] // Reverse

Out[8]= equal[composite[intpow[gp[x], y], PLUS],
          iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]] == True

In[9]:= composite[intpow[gp[x_], y_], PLUS] := iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]
```

The following intertwine formula relates negatuve powers to inverses of positive powers.

Theorem. Intertwine formula connecting **INVERSE** to **inv[x]**.

```
In[10]:= Map[composite[inv[gp[x]], #] &, SubstTest[composite, inv[gp[x]], union[u, v], INVERSE,
           {u -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            v -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
              inverse[PLUS], INVERSE]}]] // Reverse

Out[10]= composite[intpow[gp[x], y], INVERSE] == composite[inv[gp[x]], intpow[gp[x], y]]

In[11]:= composite[intpow[gp[x_], y_], INVERSE] := composite[inv[gp[x]], intpow[gp[x], y]]
```

---

mapping rule for intpow[gp[x], y]

Lemma.

```
In[12]:= (SubstTest[implies,
            and[member[s, map[v, w]], member[t, map[u, v]]], member[composite[s, t], map[u, w]],
            {u → range[PLUS], v → omega, w → range[gp[x]], t → inverse[PLUS]}] // Reverse) /.
          s → iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]]

Out[12]= or[member[composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            map[range[PLUS], range[gp[x]]]], not[member[y, range[gp[x]]]]] == True

In[13]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Converse.

```
In[14]:= SubstTest[implies, and[member[r, map[s, t]], member[w, s]], member[APPLY[r, w], t],
           {r -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            s -> range[PLUS], t -> range[gp[x]], w → plus[set[0]]}] // Reverse

Out[14]= or[member[y, range[gp[x]]],
           not[member[composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
              inverse[PLUS]], map[range[PLUS], range[gp[x]]]]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The above lemma and its converse can be combined into a single rewrite rule.

Theorem.

```
In[16]:= equiv[
          member[composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
           map[range[PLUS], range[gp[x]]]], member[y, range[gp[x]]]]

Out[16]= True

In[17]:= member[composite[iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]], inverse[PLUS]],
           map[range[PLUS], range[gp[x_]]]] := member[y, range[gp[x]]]
```

Corollary. (Replace the element **y** with its inverse.)

```
In[18]:= SubstTest[member,
           composite[iterate[composite[gp[x], LEFT[t]], set[e[gp[x]]]], inverse[PLUS]],
           map[range[PLUS], range[gp[x]]], t → APPLY[inv[gp[x]], y]] // Reverse

Out[18]= member[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
           inverse[PLUS]], map[range[PLUS], range[gp[x]]]] == member[y, range[gp[x]]]

In[19]:= member[composite[inv[gp[x_]], iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]],
           inverse[PLUS]], map[range[PLUS], range[gp[x_]]]] := member[y, range[gp[x]]]
```

Lemma.

```
In[20]:= member[composite[id[range[PLUS]], INVERSE],
           map[image[INVERSE, range[PLUS]], range[PLUS]]] // AssertTest

Out[20]= member[composite[id[range[PLUS]], INVERSE],
           map[image[INVERSE, range[PLUS]], range[PLUS]]] == True

In[21]:= member[composite[id[range[PLUS]], INVERSE],
           map[image[INVERSE, range[PLUS]], range[PLUS]]] := True
```

Lemma.

```
In[22]:= (SubstTest[implies, and[member[s, map[v, w]], member[t, map[u, v]]], member[
             composite[s, t], map[u, w]], {u → image[INVERSE, range[PLUS]], v → range[PLUS],
             w → range[gp[x]], t -> composite[id[range[PLUS]], INVERSE]}] // Reverse) /.
           s -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS]]

Out[22]= or[member[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE], map[image[INVERSE, range[PLUS]], range[gp[x]]]],
           not[member[y, range[gp[x]]]]] == True

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

Converse.

```
In[24]:= SubstTest[implies, and[member[r, map[s, t]], member[w, s]], member[APPLY[r, w], t],
           {r -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE], s → image[INVERSE, range[PLUS]],
             t -> range[gp[x]], w → inverse[plus[set[0]]]}] // Reverse

Out[24]= or[member[y, range[gp[x]]],
           not[member[composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE], map[image[INVERSE, range[PLUS]], range[gp[x]]]]]] == True

In[25]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The preceding lemma and its converse can be combined into a single rule.

Theorem.

```
In[26]:= equiv[member[composite[inv[gp[x]],
           iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS], INVERSE],
          map[image[INVERSE, range[PLUS]], range[gp[x]]]], member[y, range[gp[x]]]]

Out[26]= True

In[27]:= member[composite[inv[gp[x_]],
           iterate[composite[gp[x_], LEFT[y_]], set[e[gp[x_]]]], inverse[PLUS], INVERSE],
          map[image[INVERSE, range[PLUS]], range[gp[x_]]]] := member[y, range[gp[x]]]
```

Theorem.

```
In[28]:= SubstTest[or, member[union[s, t], map[union[y, z], w]], not[
           equal[composite[s, id[intersection[y, z]]], composite[t, id[intersection[y, z]]]]],
          not[member[s, map[y, w]]], not[member[t, map[z, w]]],
          {s -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
           t -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE], y → range[PLUS],
           z → image[INVERSE, range[PLUS]], w → range[gp[x]]}] // Reverse

Out[28]= or[member[intpow[gp[x], y], map[Z, range[gp[x]]]], not[member[y, range[gp[x]]]]] == True

In[29]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

A converse will now be derived.

Lemma.

```
In[30]:= SubstTest[APPLY, union[u, v], plus[set[0]],
           {u -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            v -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
              inverse[PLUS], INVERSE]}] // Reverse

Out[30]= APPLY[intpow[gp[x], y], composite[id[omega], SUCC]] ==
          union[y, complement[image[V, intersection[range[gp[x]], set[y]]]]]

In[31]:= APPLY[intpow[gp[x_], y_], composite[id[omega], SUCC]] :=
          union[y, complement[image[V, intersection[range[gp[x]], set[y]]]]]
```

Converse.

```
In[32]:= SubstTest[implies, and[member[r, map[s, t]], member[w, s]], member[APPLY[r, w], t],
           {r -> union[composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
               inverse[PLUS]], composite[inv[gp[x]],
               iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS], INVERSE]],
            s → Z, t -> range[gp[x]], w → plus[set[0]]}] // Reverse

Out[32]= or[member[y, range[gp[x]]], not[member[intpow[gp[x], y], map[Z, range[gp[x]]]]]] == True

In[33]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Mapping Theorem. If **y** is an element of a group **gp[x]**, then **intpow[gp[x], y]** is a mapping from the set of integers to the range of **gp[x]**.

```
In[34]:= equiv[member[intpow[gp[x], y], map[Z, range[gp[x]]]], member[y, range[gp[x]]]]
Out[34]= True

In[35]:= member[intpow[gp[x_], y_], map[Z, range[gp[x_]]]] := member[y, range[gp[x]]]
```

FUNCTION rule

Lemma.

```
In[36]:= SubstTest[implies, member[t, map[u, v]], FUNCTION[t],
           {t → intpow[gp[x], y], u → Z, v → range[gp[x]]}] // Reverse
Out[36]= or[FUNCTION[intpow[gp[x], y]], not[member[y, range[gp[x]]]]] == True

In[37]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[38]:= SubstTest[implies, equal[t, 0],
           FUNCTION[union[composite[iterate[t, set[e[gp[x]]]], inverse[PLUS]],
             composite[inv[gp[x]], iterate[t, set[e[gp[x]]], inverse[PLUS], INVERSE]]],
           t → composite[gp[x], LEFT[y]]] // Reverse
Out[38]= or[FUNCTION[intpow[gp[x], y]], member[y, range[gp[x]]]] == True

In[39]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The two lemmas above can be combined into a single rule.

Theorem.

```
In[40]:= SubstTest[and, implies[p, q], or[p, q],
           {p → member[y, range[gp[x]]], q -> FUNCTION[intpow[gp[x], y]]}]
Out[40]= FUNCTION[intpow[gp[x], y]] == True

In[41]:= FUNCTION[intpow[gp[x_], y_]] := True
```

Corollary. Vertical section rule.

```
In[42]:= SubstTest[image, funpart[t], set[z], t → intpow[gp[x], y]] // Reverse
Out[42]= image[intpow[gp[x], y], set[z]] == set[APPLY[intpow[gp[x], y], z]]

In[43]:= image[intpow[gp[x_], y_], set[z_]] := set[APPLY[intpow[gp[x], y], z]]
```

powers of inverse elements

Theorem. Integer powers of the inverse of an element is the inverse of the corresponding powers of the element itself.

```
In[44]:= Map[composite[inv[gp[x]], #, INVERSE] &, SubstTest[union,
           composite[iterate[composite[gp[x], LEFT[t]], set[e[gp[x]]]], inverse[PLUS]],
           composite[inv[gp[x]], iterate[composite[gp[x], LEFT[t]], set[e[gp[x]]]],
            inverse[PLUS], INVERSE], t → APPLY[inv[gp[x]], y]]]

Out[44]= intpow[gp[x], APPLY[inv[gp[x]], y]] == composite[inv[gp[x]], intpow[gp[x], y]]

In[45]:= intpow[gp[x_], APPLY[inv[gp[x_]], y_]] := composite[inv[gp[x]], intpow[gp[x], y]]
```

flip rule

The function **intpow[gp[x], y]** is unchanged if the group **gp[x]** is replaced with its **flip**.

Lemma.

```
In[46]:= SubstTest[implies, equal[u, v],
           equal[image[t, u], image[t, v]], {t → union[Id, cross[INVERSE, inv[gp[x]]]],
            u → composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
            v → composite[iterate[composite[gp[x], RIGHT[y]], set[e[gp[x]]]],
              inverse[PLUS]]}] // Reverse

Out[46]= equal[intpow[gp[x], y],
           union[composite[iterate[composite[gp[x], RIGHT[y]], set[e[gp[x]]]], inverse[PLUS]],
            composite[inv[gp[x]], iterate[composite[gp[x], RIGHT[y]], set[e[gp[x]]]],
             inverse[PLUS], INVERSE]]] == True

In[47]:= union[composite[iterate[composite[gp[x_], RIGHT[y_]], set[e[gp[x_]]]], inverse[PLUS]],
           composite[inv[gp[x_]], iterate[composite[gp[x_], RIGHT[y_]], set[e[gp[x_]]]],
            inverse[PLUS], INVERSE]] := intpow[gp[x], y]
```

Theorem.

```
In[48]:= SubstTest[union,
          composite[iterate[composite[gp[t], LEFT[y]], set[e[gp[t]]]], inverse[PLUS]],
          composite[inv[gp[t]], iterate[composite[gp[t], LEFT[y]], set[e[gp[t]]]],
           inverse[PLUS], INVERSE], t → flip[gp[x]]]

Out[48]= intpow[composite[gp[x], SWAP], y] == intpow[gp[x], y]

In[49]:= intpow[composite[gp[x_], SWAP], y_] := intpow[gp[x], y]
```

examples

Some examples of the function **intpow[x, y]** are considered in this section.

Example 1.

```
In[50]:= SubstTest[union,
          composite[iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]], inverse[PLUS]],
          composite[inv[gp[u]], iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]],
           inverse[PLUS], INVERSE], {u → INTADD, v → int[x]}]

Out[50]= intpow[INTADD, int[x]] == inttimes[int[x]]

In[51]:= intpow[INTADD, int[x_]] := inttimes[int[x]]
```

Example 2.

```
In[52]:= SubstTest[union,
          composite[iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]], inverse[PLUS]],
          composite[inv[gp[u]], iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]],
           inverse[PLUS], INVERSE], {u → RATADD, v → rat[x]}]

Out[52]= intpow[RATADD, rat[x]] == composite[rattimes[rat[x]], INTTIMES]

In[53]:= intpow[RATADD, rat[x_]] := composite[rattimes[rat[x]], INTTIMES]
```

Observations. The integer power function is a binary homomorphism for both of the above examples.

```
In[54]:= member[intpow[gp[u], v], binhom[INTADD, gp[u]]] /. {u → INTADD, v → int[x]}

Out[54]= True

In[55]:= member[intpow[gp[u], v], binhom[INTADD, gp[u]]] /. {u → RATADD, v → rat[x]}

Out[55]= True
```

Example 3. A trivial case.

```
In[56]:= SubstTest[union,
          composite[iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]], inverse[PLUS]],
          composite[inv[gp[u]], iterate[composite[gp[u], LEFT[v]], set[e[gp[u]]]],
           inverse[PLUS], INVERSE], {u → 0, v → x}]

Out[56]= intpow[0, x] == 0

In[57]:= intpow[0, x_] := 0
```

Observation. The function **intpow[x, y]** is not a binary homomorphism when **x = 0**.

```
In[58]:= member[intpow[gp[u], v], binhom[INTADD, gp[u]]] /. {u → 0, v → x}

Out[58]= False
```

inclusions rules for cartesian products

Lemma.

```
In[62]:= Assoc[Id, intpow[x, y], id[Z]]

Out[62]= composite[Id, intpow[x, y]] == intpow[x, y]

In[63]:= composite[Id, intpow[x_, y_]] := intpow[x, y]
```

Theorem.

```
In[64]:= SubstTest[subclass, composite[Id, t], cart[u, v], t → intpow[x, y]] // Reverse

Out[64]= subclass[intpow[x, y], cart[u, v]] ==
          and[subclass[domain[intpow[x, y]], u], subclass[range[intpow[x, y]], v]]

In[65]:= subclass[intpow[x_, y_], cart[u_, v_]] :=
          and[subclass[domain[intpow[x, y]], u], subclass[range[intpow[x, y]], v]]
```

Theorem. Domain rule.

```
In[66]:= SubstTest[equal, composite[t, id[Z]], t, t -> intpow[x, y]]

Out[66]= subclass[domain[intpow[x, y]], Z] == True

In[67]:= subclass[domain[intpow[x_, y_]], Z] := True
```

Theorem. Range rule.

```
In[68]:= Map[subclass[#, range[gp[x]]] &, SubstTest[range, union[u, v],
            {u -> composite[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]], inverse[PLUS]],
             v -> composite[inv[gp[x]], iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]]],
               inverse[PLUS], INVERSE]}]] // Reverse

Out[68]= subclass[range[intpow[gp[x], y]], range[gp[x]]] == True

In[69]:= subclass[range[intpow[gp[x_], y_]], range[gp[x_]]] := True
```

APPLY rules

Theorem. The zero-th power of any element is the identity element.

```
In[70]:= Map[A, ImageComp[intpow[gp[x], y], PLUS, set[0]]] // Reverse

Out[70]= APPLY[intpow[gp[x], y], id[omega]] == e[gp[x]]

In[71]:= APPLY[intpow[gp[x_], y_], id[omega]] := e[gp[x]]
```

Theorem. An **APPLY** rule for non-negative powers.

```
In[72]:= Map[A, ImageComp[intpow[gp[x], y], PLUS, set[nat[z]]]] // Reverse

Out[72]= APPLY[intpow[gp[x], y], plus[nat[z]]] ==
          APPLY[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]], nat[z]]

In[73]:= APPLY[intpow[gp[x_], y_], plus[nat[z_]]] :=
          APPLY[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]], nat[z]]
```

Theorem. Negative powers are inverses of the corresponding positive powers.

```
In[74]:= Map[A, ImageComp[intpow[gp[x], y], composite[INVERSE, PLUS], set[nat[z]]]] // Reverse

Out[74]= APPLY[intpow[gp[x], y], inverse[plus[nat[z]]]] ==
          APPLY[inv[gp[x]], APPLY[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]], nat[z]]]

In[75]:= APPLY[intpow[gp[x_], y_], inverse[plus[nat[z_]]]] :=
          APPLY[inv[gp[x]], APPLY[iterate[composite[gp[x], LEFT[y]], set[e[gp[x]]], nat[z]]]
```