integer powers and homomorphisms

Johan G. F. Belinfante
2013 July 26

```
In[1]:= SetDirectory["l:"]; << goedel.13jul24a

        :Package Title: goedel.13jul24a                    2013 July 24 at 4:20 p.m.

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

        It is now:  2013 Jul 26 at 20:58

        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 Jul 26 at 21:15
```

summary

The focus in this third of a series of notebooks about integer powers of group elements is on rewrite rules involving group homomorphisms.

sethood rule

Theorem. The integer power function **intpow[gp[x], y]** is a set.

```
In[2]:= (SubstTest[implies, and[subclass[u, v], member[v, V]], member[u, V],
            {u → domain[funpart[t]], v → Z}] /. t → intpow[gp[x], y]) // Reverse

Out[2]= member[intpow[gp[x], y], V] == True

In[3]:= member[intpow[gp[x_], y_], V] := True
```

a replacement rule

Any binary homomorphism **t** from **INTADD** to a group **gp[x]** is determined by the element **APPLY[t, id[ω] ∘ SUCC] ∈ range[gp[x]]**. A rewrite rule obtained 2013 February 21 in the notebook **bhzaddgp.nb** is rederived here using the integer power function **intpow[gp[x], y]** that was introduced in the meantime.

Lemma.

```
In[4]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[p1, p4]], {p1 -> member[t, binhom[INTADD, x]], p2 → FUNCTION[t],
           p3 → equal[domain[t], Z], p4 -> subclass[t, cart[Z, V]]}]] // Reverse

Out[4]= or[not[member[t, binhom[INTADD, x]]], subclass[t, cart[Z, V]]] == True

In[5]:= (% /. {t → t_, x → x_}) /. Equal → SetDelayed
```

The following is a replacement for a rewrite rule derived 2013 February 21.

Main Theorem. Any binary homomorphism **t** from the group **INTADD** to a group **gp[x]** is the integer power function for the group element **APPLY[t, id[ω] ∘ SUCC] ∈ range[gp[x]]**.

```
In[6]:= (Map[not, SubstTest[and, (*implies[and[p1,p2,p3],p4],
             implies[and[p1,p2],p5],implies[and[p1,p3],p6],*)
            implies[and[p4, p5, p6], p7], not[implies[and[p1, p2, p3], p7]],
            {p1 → member[t, binhom[INTADD, gp[x]]], p2 → equal[u, composite[t, PLUS]],
             p3 → equal[v, composite[t, INVERSE, PLUS]], p4 → equal[t,
               union[composite[u, inverse[PLUS]], composite[v, inverse[PLUS], INVERSE]]], p5 →
              equal[u, iterate[composite[gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]],
                set[e[gp[x]]]]], p6 → equal[v, composite[inv[gp[x]], iterate[composite[
                  gp[x], LEFT[APPLY[t, composite[id[omega], SUCC]]]], set[e[gp[x]]]]]],
             p7 → equal[t, intpow[gp[x], APPLY[t, composite[id[omega], SUCC]]]]}]] // Reverse) /.
         {u -> composite[t, PLUS], v -> composite[t, INVERSE, PLUS]}

Out[6]= or[equal[t, intpow[gp[x], APPLY[t, composite[id[omega], SUCC]]]],
          not[member[t, binhom[INTADD, gp[x]]]]] == True

In[7]:= or[equal[t_, intpow[gp[x_], APPLY[t_, composite[id[omega], SUCC]]]],
          not[member[t_, binhom[INTADD, gp[x_]]]]] := True
```

---

intpow[gp[x], y] as a binary homomorphism

The law of exponents for integer powers implies that the integer power function for any element **y ∈ range[gp[x]]** is a binary homomorphism from **INTADD** to **gp[x]**.

Lemma.

```
In[8]:= Map[implies[member[y, range[gp[x]]], #] &,
          (member[t, binhom[u, v]] // AssertTest) /. {t → intpow[gp[x], y], u → INTADD, v → gp[x]}]

Out[8]= or[member[intpow[gp[x], y], binhom[INTADD, gp[x]]], not[member[y, range[gp[x]]]]] == True

In[9]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Converse.)

```
In[10]:= SubstTest[implies, member[t, binhom[u, v]], equal[domain[t], fix[domain[u]]],
           {t → intpow[gp[x], y], u → INTADD, v → gp[x]}] // Reverse
Out[10]= or[member[y, range[gp[x]]], not[member[intpow[gp[x], y], binhom[INTADD, gp[x]]]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[12]:= equiv[member[intpow[gp[x], y], binhom[INTADD, gp[x]]], member[y, range[gp[x]]]]
Out[12]= True

In[13]:= member[intpow[gp[x_], y_], binhom[INTADD, gp[x_]]] := member[y, range[gp[x]]]
```

Comment. The following corollary is automatic.

```
In[14]:= functor[intpow[gp[x], y], INTADD, gp[x]]
Out[14]= member[y, range[gp[x]]]
```

emptiness rule

Lemma.

```
In[15]:= SubstTest[implies, member[u, v], not[empty[v]],
           {u → intpow[gp[x], y], v → binhom[INTADD, gp[x]]}] // Reverse
Out[15]= or[not[equal[0, binhom[INTADD, gp[x]]]], not[member[y, range[gp[x]]]]] == True

In[16]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[17]:= Map[equal[V, #] &, SubstTest[class, y, or[not[empty[u]], not[member[y, v]]],
           {u -> binhom[INTADD, gp[x]], v → range[gp[x]]}]]
Out[17]= or[equal[0, gp[x]], not[equal[0, binhom[INTADD, gp[x]]]]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. A better rewrite rule.

```
In[19]:= equiv[equal[0, binhom[INTADD, gp[x]]], equal[0, gp[x]]]
Out[19]= True

In[20]:= equal[0, binhom[INTADD, gp[x_]]] := equal[0, gp[x]]
```

Comment. The following application was the inspiration for the results in this section.

```
In[21]:= subclass[binhom[INTADD, gp[x]], domain[eval[y]]]

Out[21]= or[equal[0, gp[x]], member[y, Z]]
```

transforms of integer powers under binary homomorphisms

Lemma.

```
In[22]:= SubstTest[implies,
           and[member[t, binhom[gp[x], gp[z]]], member[u, binhom[INTADD, gp[x]]]],
           member[composite[t, u], binhom[INTADD, gp[z]]], u → intpow[gp[x], y]] // Reverse

Out[22]= or[member[composite[t, intpow[gp[x], y]], binhom[INTADD, gp[z]]],
           not[member[t, binhom[gp[x], gp[z]]]], not[member[y, range[gp[x]]]]] == True

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

Lemma.

```
In[24]:= SubstTest[or, equal[u, intpow[gp[z], APPLY[u, composite[id[omega], SUCC]]]],
           not[member[u, binhom[INTADD, gp[z]]]], u → composite[t, intpow[gp[x], y]]] // Reverse

Out[24]= or[equal[composite[t, intpow[gp[x], y]], intpow[gp[z],
             union[APPLY[t, y], complement[image[V, intersection[range[gp[x]], set[y]]]]]]],
           not[member[composite[t, intpow[gp[x], y]], binhom[INTADD, gp[z]]]]] == True

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

Lemma. Temporary rewrite rule.

```
In[26]:= SubstTest[implies, and[equal[u, intpow[gp[z], union[APPLY[t, y], v]]], empty[v]],
           equal[u, intpow[gp[z], APPLY[t, y]]],
           v -> complement[image[V, intersection[range[gp[x]], set[y]]]]] // Reverse

Out[26]= or[equal[u, intpow[gp[z], APPLY[t, y]]], not[equal[u, intpow[gp[z],
              union[APPLY[t, y], complement[image[V, intersection[range[gp[x]], set[y]]]]]]]],
           not[member[y, range[gp[x]]]]] == True

In[27]:= (% /. {t → t_, u → u_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. Effect of a group homomorphism on powers of an element of a group.

```
In[28]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[p3, p4], implies[and[p1, p4], p5], not[implies[and[p1, p2], p5]],
            {p1 → member[y, range[gp[x]]], p2 → member[t, binhom[gp[x], gp[z]]],
             p3 -> member[composite[t, intpow[gp[x], y]], binhom[INTADD, gp[z]]],
             p4 -> equal[composite[t, intpow[gp[x], y]], intpow[gp[z],
                union[APPLY[t, y], complement[image[V, intersection[range[gp[x]], set[y]]]]]]],
             p5 -> equal[composite[t, intpow[gp[x], y]], intpow[gp[z], APPLY[t, y]]]}] // Reverse

Out[28]= or[equal[composite[t, intpow[gp[x], y]], intpow[gp[z], APPLY[t, y]]],
           not[member[t, binhom[gp[x], gp[z]]]], not[member[y, range[gp[x]]]]] == True

In[29]:= or[equal[composite[t_, intpow[gp[x_], y_]], intpow[gp[z_], APPLY[t_, y_]]],
           not[member[t_, binhom[gp[x_], gp[z_]]]], not[member[y_, range[gp[x_]]]]] := True
```

---

multiplicative law of exponents

Lemma.

```
In[30]:= SubstTest[implies,
           and[member[t, binhom[INTADD, gp[x]]], member[u, binhom[INTADD, INTADD]]],
           member[composite[t, u], binhom[INTADD, gp[x]]],
           {t → intpow[gp[x], y], u → inttimes[z]}] // Reverse

Out[30]= or[member[composite[intpow[gp[x], y], inttimes[z]], binhom[INTADD, gp[x]]],
           not[member[y, range[gp[x]]]], not[member[z, Z]]] == True

In[31]:= (% /. {x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Lemma.

```
In[32]:= SubstTest[implies, member[t, binhom[INTADD, gp[x]]],
           equal[t, intpow[gp[x], APPLY[t, composite[id[omega], SUCC]]]],
           t -> composite[intpow[gp[x], y], inttimes[z]]] // Reverse

Out[32]= or[equal[composite[intpow[gp[x], y], inttimes[z]],
             intpow[gp[x], APPLY[intpow[gp[x], y], z]]],
           not[member[composite[intpow[gp[x], y], inttimes[z]], binhom[INTADD, gp[x]]]]] == True

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

Theorem.

```
In[34]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p3, p4],
            not[implies[and[p1, p2], p4]], {p1 → member[y, range[gp[x]]], p2 → member[z, Z],
             p3 -> member[composite[intpow[gp[x], y], inttimes[z]], binhom[INTADD, gp[x]]],
             p4 -> equal[composite[intpow[gp[x], y], inttimes[z]],
               intpow[gp[x], APPLY[intpow[gp[x], y], z]]]}] // Reverse

Out[34]= or[equal[composite[intpow[gp[x], y], inttimes[z]],
             intpow[gp[x], APPLY[intpow[gp[x], y], z]]],
           not[member[y, range[gp[x]]]], not[member[z, Z]]] == True
```

```
In[35]:= or[equal[composite[intpow[gp[x_], y_], inttimes[z_]],
            intpow[gp[x_], APPLY[intpow[gp[x_], y_], z_]]],
           not[member[y_, range[gp[x_]]]], not[member[z_, Z]]] := True
```

Corollary.

```
In[36]:= SubstTest[or, equal[composite[intpow[gp[x], y], inttimes[t]],
            intpow[gp[x], APPLY[intpow[gp[x], y], t]]],
           not[member[y, range[gp[x]]]], not[member[t, Z]], t → int[z]] // Reverse
```

```
Out[36]= or[equal[composite[intpow[gp[x], y], inttimes[int[z]]],
            intpow[gp[x], APPLY[intpow[gp[x], y], int[z]]], not[member[y, range[gp[x]]]]] == True
```

```
In[37]:= or[equal[composite[intpow[gp[x_], y_], inttimes[int[z_]]],
            intpow[gp[x_], APPLY[intpow[gp[x_], y_], int[z_]]],
           not[member[y_, range[gp[x_]]]]] := True
```

Theorem. Multiplicative law of exponents for integer powers of a group element.

```
In[40]:= Map[not, SubstTest[and, implies[p1, p2],
            implies[p2, p3], not[implies[p1, p3]], {p1 → member[y, range[gp[x]]],
             p2 → equal[composite[intpow[gp[x], y], inttimes[int[u]]],
               intpow[gp[x], APPLY[intpow[gp[x], y], int[u]]]],
             p3 → equal[APPLY[intpow[gp[x], APPLY[intpow[gp[x], y], int[u]]], int[v]],
               APPLY[intpow[gp[x], y], intmul[int[u], int[v]]]]}]] // Reverse
```

```
Out[40]= or[equal[APPLY[intpow[gp[x], y], intmul[int[u], int[v]]],
            APPLY[intpow[gp[x], APPLY[intpow[gp[x], y], int[u]]], int[v]]],
           not[member[y, range[gp[x]]]]] == True
```

```
In[42]:= or[equal[APPLY[intpow[gp[x_], APPLY[intpow[gp[x_], y_], int[u_]]], int[v_]],
            APPLY[intpow[gp[x_], y_], intmul[int[u_], int[v_]]]],
           not[member[y_, range[gp[x_]]]]] := True
```