additive law of exponents for powers in monoids

Johan G. F. Belinfante
2012 February 7

```
In[1]:= SetDirectory["l:"]; << goedel.12feb05a

        :Package Title: goedel.12feb05a               2012 February 5 at 3:45 p.m.

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

        It is now:  2012 Feb 7 at 14:41

        Loading Simplification Rules

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

        weightlimit = 40

        Loading completed.

        It is now:  2012 Feb 7 at 14:55
```

summary

The **n**-th power of an element **y ∈ range[x]** of a monoid **x** is $\mathbf{y^n}$ = **APPLY[iterate[x ∘ LEFT[y], {e[x]}], n]**. In this notebook, the additive law of exponents for such powers is derived: $\mathbf{y^m \cdot y^n = y^{m+n}}$. It follows that the sequence of powers of **y** is a functor from **NATADD** to **x**.

a general law of exponents

Lemma.

```
In[2]:= Map[A, ImageComp[image[power[funpart[x]], set[v]],
           image[power[funpart[x]], set[u]], set[z]]] // Reverse

Out[2]= APPLY[iterate[funpart[x], set[APPLY[iterate[funpart[x], set[z]], u]]], v] ==
         APPLY[iterate[funpart[x], set[z]], natadd[u, v]]

In[3]:= APPLY[iterate[funpart[x_], set[APPLY[iterate[funpart[x_], set[z_]], u_]]], v_] :=
         APPLY[iterate[funpart[x], set[z]], natadd[u, v]]
```

Theorem. (Remove the **funpart** wrapper.)

```
In[4]:= SubstTest[implies, equal[x, funpart[t]], equal[APPLY[iterate[x, set[z]], natadd[u, v]],
          APPLY[iterate[x, set[APPLY[iterate[x, set[z]], u]]], v]], t → x] // Reverse

Out[4]= or[equal[APPLY[iterate[x, set[z]], natadd[u, v]],
          APPLY[iterate[x, set[APPLY[iterate[x, set[z]], u]]], v]], not[FUNCTION[x]]] == True

In[5]:= or[equal[APPLY[iterate[x_, set[APPLY[iterate[x_, set[z_]], u_]]], v_],
          APPLY[iterate[x_, set[z_]], natadd[u_, v_]]], not[FUNCTION[x_]]] := True
```

Theorem. (Apply the general law of exponents to the case of left multiplication for a monoid.)

```
In[6]:= SubstTest[or, equal[APPLY[iterate[t, set[z]], natadd[u, v]],
          APPLY[iterate[t, set[APPLY[iterate[t, set[z]], u]]], v]],
         not[FUNCTION[t]], t → composite[monoid[x], LEFT[y]]] // Reverse

Out[6]= equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], natadd[u, v]],
         APPLY[iterate[composite[monoid[x], LEFT[y]],
           set[APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], u]]], v]] == True

In[7]:= APPLY[iterate[composite[monoid[x_], LEFT[y_]],
          set[APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[z_]], u_]]], v_] :=
        APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], natadd[u, v]]
```

Corollary. (Remove the **monoid** wrapper.)

```
In[8]:= SubstTest[implies, equal[x, monoid[t]],
          equal[APPLY[iterate[composite[x, LEFT[y]], set[z]], natadd[u, v]],
           APPLY[iterate[composite[x, LEFT[y]], set[APPLY[
               iterate[composite[x, LEFT[y]], set[z]], u]]], v]], t → x] // Reverse // MapNotNot

Out[8]= or[equal[APPLY[iterate[composite[x, LEFT[y]], set[z]], natadd[u, v]], APPLY[iterate[
             composite[x, LEFT[y]], set[APPLY[iterate[composite[x, LEFT[y]], set[z]], u]]], v]],
         not[member[x, MONOIDS]]] == True

In[9]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal→ SetDelayed
```

---

the additive law of exponents

If **x** is a monoid and **y ∈ range[x]**, then **iterate[x ∘ LEFT[y], {e[x]}]** is the infinite sequence of powers $\mathbf{y^n}$ where **n ∈ ω**. More generally, if **z ∈ range[x]**, then **iterate[x ∘ LEFT[y], {z}]** is the sequence of products $\mathbf{y^n\ z}$ where **n ∈ ω**. A **monoid** wrapper has been used in the following theorem in place of the literal **x ∈ MONOIDS**, and the redundant literal **y ∈ range[x]** is left out.

Theorem. For **z ∈ range[monoid[x]]**, the sequence **iterate[monoid[x] ∘ LEFT[y], {z}]** can be expressed in terms of the sequence of powers of **y**.

```
In[10]:= Map[implies[member[z, range[monoid[x]]], #] &,
          SubstTest[implies, and[equal[composite[w, SUCC], composite[u, w]],
             equal[image[w, set[0]], v]], equal[composite[w, id[omega]], iterate[u, v]],
            {u → composite[monoid[x], LEFT[y]], v → set[z], w -> composite[monoid[x], RIGHT[z],
               iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]}] // Reverse]

Out[10]= or[equal[composite[monoid[x], RIGHT[z],
             iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]],
            iterate[composite[monoid[x], LEFT[y]], set[z]]],
           not[member[z, range[monoid[x]]]] == True

In[11]:= or[equal[composite[monoid[x_], RIGHT[z_],
             iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]]],
            iterate[composite[monoid[x_], LEFT[y_]], set[z_]]],
           not[member[z_, range[monoid[x_]]]] := True
```

To obtain the traditional form for the law of exponents, one needs to consider the individual terms of the sequence of powers. This is done formally by applying the list of powers, considered as a function, to natural numbers. The variables **u** and **v** will be used for natural numbers in the following, but most of the formulas do not require an explicit statement that they belong to the set ω of natural numbers.

Lemma. Simplification rule for a function application.

```
In[14]:= SubstTest[image, funpart[t], set[u],
           t → iterate[composite[monoid[x], LEFT[y]], set[z]]] // Reverse

Out[14]= image[iterate[composite[monoid[x], LEFT[y]], set[z]], set[u]] ==
          set[APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], u]]

In[15]:= image[iterate[composite[monoid[x_], LEFT[y_]], set[z_]], set[u_]] :=
          set[APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], u]]
```

Corollary. The **u**-th member of the list **iterate[x ∘ LEFT[y], {z}]** is $y^u z$.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
             not[implies[p1, p3]], {p1 -> member[z, range[monoid[x]]],
              p2 -> equal[composite[monoid[x], RIGHT[z], iterate[composite[monoid[x], LEFT[y]],
                  set[e[monoid[x]]]]], iterate[composite[monoid[x], LEFT[y]], set[z]]],
              p3 → equal[APPLY[composite[monoid[x], RIGHT[z],
                  iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]], u],
                APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], u]}]] // Reverse

Out[17]= or[equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[z]], u], APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u], z]]],
           not[member[z, range[monoid[x]]]] == True

In[18]:= or[equal[APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[z_]], u_],
             APPLY[monoid[x_], PAIR[
               APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], u_], z_]]],
           not[member[z_, range[monoid[x_]]]] := True
```

Theorem. (Remove the **monoid** wrapper.)

```
In[19]:= SubstTest[implies, equal[x, monoid[u]],
           or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], t], z]],
             APPLY[iterate[composite[x, LEFT[y]], set[z]], t]],
            not[member[z, range[x]]]], u → x] // Reverse // MapNotNot

Out[19]= or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], t], z]],
           APPLY[iterate[composite[x, LEFT[y]], set[z]], t]],
         not[member[x, MONOIDS]], not[member[z, range[x]]]] == True

In[20]:= or[equal[APPLY[iterate[composite[x_, LEFT[y_]], set[z_]], t_],
           APPLY[x_, PAIR[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], t_], z_]]],
         not[member[x_, MONOIDS]], not[member[z_, range[x_]]]] := True
```

The next step will be to use this result to simply the general result derived in the preceding section for the special case $\mathbf{z = y^v}$. The variable **v** is here wrapped with **nat**, but this wrapper will later be eliminated. The hypothesis $\mathbf{y \in range[x]}$ will also later be eliminated.

Lemma. A tentative version for the additive law of exponents.

```
In[21]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2], p3], implies[p1, p4], implies[and[p3, p4], p5],
           not[implies[p1, p5]], {p1 → and[member[x, MONOIDS], member[y, range[x]]],
            p2 → member[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]], range[x]],
            p3 → equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
                APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]]]],
              APPLY[iterate[composite[x, LEFT[y]],
                set[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]]]], u]],
            p4 → equal[APPLY[iterate[composite[x, LEFT[y]],
                set[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]]]], u],
              APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, nat[v]]]],
            p5 → equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
                APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]]]], APPLY[
               iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, nat[v]]]]}]] // Reverse

Out[21]= or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
             APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[v]]]],
           APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, nat[v]]]],
         not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[22]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary. Replace the **nat** wrapper with the (redundant) literal $\mathbf{v \in \omega}$.

```
In[23]:= SubstTest[implies, equal[v, nat[t]],
           or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
               APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]]],
             APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]],
            not[member[x, MONOIDS]], not[member[y, range[x]]]], t → v] // Reverse
```

```
Out[23]= or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
             APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]]],
           APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]],
          not[member[v, omega]], not[member[x, MONOIDS]], not[member[y, range[x]]]] == True
```

```
In[24]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (The case that **v** is not a natural number.)

```
In[25]:= (Map[not, SubstTest[and, implies[and[p0, p1], p2], implies[and[p0, p1], p3],
             implies[and[p2, p3], p4], not[implies[and[p0, p1], p4]],
             {p0 → and[equal[s, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]],
                equal[t, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]]],
              p1 → not[member[v, omega]], p2 → equal[V, s], p3 → equal[V, t], p4 -> equal[
                APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u], s]], t]}]] //
            Reverse) /. {s -> APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v],
           t -> APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]}
```

```
Out[25]= or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
             APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]]],
           APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]],
          member[v, omega]] == True
```

```
In[26]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (Eliminate the redundant numberhood literal, and reintroduce the **monoid** wrapper.)

```
In[27]:= SubstTest[and, implies[p, q], or[p, q], {p → member[v, omega],
            q → or[equal[APPLY[t, PAIR[APPLY[iterate[composite[t, LEFT[y]], set[e[t]]], u],
                APPLY[iterate[composite[t, LEFT[y]], set[e[t]]], v]]],
              APPLY[iterate[composite[t, LEFT[y]], set[e[t]]], natadd[u, v]]],
             not[member[t, MONOIDS]], not[member[y, range[t]]]]}] /. t → monoid[x]
```

```
Out[27]= or[equal[0, monoid[x]],
           equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             natadd[u, v]], APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
              APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]]],
           not[member[y, range[monoid[x]]]]] == True
```

```
In[28]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. The literal **y ∈ range[monoid[x]]** is redundant.

```
In[29]:= SubstTest[implies, equal[t, 0],
           equal[APPLY[monoid[x], PAIR[APPLY[iterate[t, set[e[monoid[x]]]], u],
              APPLY[iterate[t, set[e[monoid[x]]]], v]]],
            APPLY[iterate[t, set[e[monoid[x]]]], natadd[u, v]]],
           t → composite[monoid[x], LEFT[y]]] // Reverse

Out[29]= or[equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             natadd[u, v]], APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
              APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]]],
           member[y, range[monoid[x]]]] == True

In[30]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma. Eliminate the redundant literal.

```
In[31]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> member[y, range[monoid[x]]], q -> or[equal[0, monoid[x]],
             equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
               natadd[u, v]], APPLY[monoid[x],
               PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
                APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]]]}]

Out[31]= or[equal[0, monoid[x]],
           equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             natadd[u, v]], APPLY[monoid[x],
             PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
              APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]]] == True

In[32]:= (% /. {u → u_, v → v_, x → x_, y → y_}) /. Equal→ SetDelayed
```

The hypothesis that **monoid[x]** be nonempty can also be removed.

Theorem. Wrapped version for the additive law of powers for monoids.

```
In[33]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> equal[0, monoid[x]], q -> or[equal[APPLY[iterate[composite[monoid[x], LEFT[y]],
                set[e[monoid[x]]]], natadd[u, v]], APPLY[monoid[x],
               PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
                APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]]]}]

Out[33]= equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], natadd[u, v]],
           APPLY[monoid[x],
            PAIR[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], u],
             APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], v]]] == True

In[34]:= APPLY[monoid[x_],
           PAIR[APPLY[iterate[composite[monoid[x_], LEFT[y]], set[e[monoid[x_]]]], u_],
            APPLY[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], v_]]] :=
          APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], natadd[u, v]]
```

Corollary. Unwrapped version of the additive law of exponents for powers in a monoid.

```
In[35]:= SubstTest[implies, equal[x, monoid[t]],
           equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
              APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]]],
            APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]],
           t → x] // Reverse // MapNotNot

Out[35]= or[equal[APPLY[x, PAIR[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], u],
             APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], v]]],
           APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], natadd[u, v]]],
          not[member[x, MONOIDS]]] == True

In[36]:= or[equal[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], natadd[u_, v_]],
           APPLY[x_, PAIR[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], u_],
             APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], v_]]]],
          not[member[x_, MONOIDS]]] := True
```

the power sequence is a functor

The variables **u** and **v** can be eliminated from the additive law of exponents using **reify**. One can remove both of these variables in a single step by first replacing **u** with **first[t]** and **v** with **second[t]**.

Theorem.

```
In[37]:= Map[VERTSECT, SubstTest[reify, t,
           APPLY[w, PAIR[APPLY[iterate[composite[w, LEFT[y]], set[e[w]]], first[t]],
             APPLY[iterate[composite[w, LEFT[y]], set[e[w]]],
              second[t]]]], w → monoid[x]]] // Reverse

Out[37]= composite[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], NATADD] ==
          composite[monoid[x], cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]

In[38]:= composite[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], NATADD] :=
          composite[monoid[x], cross[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]
```

A functorial interpretation of this rewrite rule can be derived.

Lemma.

```
In[39]:= Map[implies[equal[omega,
             domain[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]], #] &,
           functor[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            NATADD, monoid[x]] // AssertTest]

Out[39]= or[functor[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            NATADD, monoid[x]], not[equal[omega,
             domain[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]]] == True

In[40]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Corollary. (Remove the **monoid** wrapper.)

```
In[41]:= SubstTest[implies, equal[x, monoid[t]],
           or[functor[iterate[composite[x, LEFT[y]], set[e[x]]], NATADD, x],
            not[equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]]],
           t → x] // Reverse // MapNotNot

Out[41]= or[functor[iterate[composite[x, LEFT[y]], set[e[x]]], NATADD, x],
           not[equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]],
           not[member[x, MONOIDS]]] == True

In[42]:= (% /. {x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. If **x** is a monoid and **y ∈ range[x]**, then the sequence of powers of **y** is a functor from **NATADD** to **x**.

```
In[43]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
           not[implies[p1, p3]], {p1 → and[member[x, MONOIDS], member[y, range[x]]],
            p2 -> equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]],
            p3 → functor[iterate[composite[x, LEFT[y]], set[e[x]]], NATADD, x]}]] // Reverse

Out[43]= or[functor[iterate[composite[x, LEFT[y]], set[e[x]]], NATADD, x],
           not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[44]:= or[functor[iterate[composite[x_, LEFT[y_]], set[e[x_]]], NATADD, x_],
           not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```

---

related results

Theorem. A sethood result for the list of powers in a monoid.

```
In[45]:= member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], V] // AssertTest

Out[45]= member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], V] == True

In[46]:= member[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]], V] := True
```

A mapping statement will next be derived.

Lemma.

```
In[47]:= Map[implies[equal[omega,
             domain[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]], #] &,
           member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            map[omega, range[monoid[x]]]] // AssertTest]

Out[47]= or[member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            map[omega, range[monoid[x]]]], not[equal[omega,
             domain[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]]] == True

In[48]:= (% /. {x -> x_, y → y_}) /. Equal→ SetDelayed
```

Corollary. (Eliminate the **monoid** wrapper.)

```
In[49]:= SubstTest[implies, equal[x, monoid[t]],
           or[member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]],
            not[equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]],
           t → x] // Reverse // MapNotNot

Out[49]= or[member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]],
           not[equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]],
           not[member[x, MONOIDS]]] == True

In[50]:= (% /. {x -> x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. If **x ∈ MONOIDS** and **y ∈ range[x]**, then **iterate[x ∘ LEFT[y],{e[x]}] ∈ map[*ω*, range[x]]**.

```
In[51]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
           not[implies[p1, p3]], {p1 → and[member[x, MONOIDS], member[y, range[x]]],
            p2 -> equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]], p3 → member[
              iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]]}] // Reverse

Out[51]= or[member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]],
           not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[52]:= or[member[iterate[composite[x_, LEFT[y_]], set[e[x_]]], map[omega, range[x_]]],
           not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```

The final task will be to show that the list of powers is a binary homomorphism.

Lemma. (An application of **AssertTest**.)

```
In[53]:= Map[implies[member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             map[omega, range[monoid[x]]]], #] &, (member[t, binhom[u, v]] // AssertTest) /.
           {t -> iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            u → NATADD, v→ monoid[x]}]

Out[53]= or[member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            binhom[NATADD, monoid[x]]],
           not[member[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             map[omega, range[monoid[x]]]]]] == True

In[54]:= (% /. {x -> x_, y → y_}) /. Equal→ SetDelayed
```

Lemma. (Eliminate the **monoid** wrapper.)

```
In[55]:= SubstTest[implies, equal[x, monoid[t]],
           or[member[iterate[composite[x, LEFT[y]], set[e[x]]], binhom[NATADD, x]],
            not[member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]]]],
           t → x] // Reverse // MapNotNot

Out[55]= or[member[iterate[composite[x, LEFT[y]], set[e[x]]], binhom[NATADD, x]],
           not[member[x, MONOIDS]],
           not[member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]]]] == True
```

```
In[56]:= (% /. {x -> x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. If **x** is a monoid and **y ∈ range[x]**, then the sequence of powers of **y** is a binary homomorphism from **NAT-ADD** to **x**.

```
In[57]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
           not[implies[p1, p3]], {p1 → and[member[x, MONOIDS], member[y, range[x]]],
            p2 -> member[iterate[composite[x, LEFT[y]], set[e[x]]], map[omega, range[x]]], p3 ->
             member[iterate[composite[x, LEFT[y]], set[e[x]]], binhom[NATADD, x]]}]] // Reverse

Out[57]= or[member[iterate[composite[x, LEFT[y]], set[e[x]]], binhom[NATADD, x]],
          not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[58]:= or[member[iterate[composite[x_, LEFT[y_]], set[e[x_]]], binhom[NATADD, x_]],
          not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```