powers of monoid elements

Johan G. F. Belinfante
2012 February 5

```
In[1]:= SetDirectory["l:"]; << goedel.12feb03a

        :Package Title: goedel.12feb03a               2012 February 4 at 4:50 p.m.

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

        It is now:  2012 Feb 5 at 8:54

        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 5 at 9:7
```

summary

If **x** is a monoid and **y ∈ range[x]**, then for every natural number **n ∈ ω** there is an element $\mathbf{y^n} \in$ **range[x]** called the **n**-th power of **y**. One can define the **n**-th power of **y** either by starting with the neutral element **e[x]** and repeatedly multiplying on the left with **y**, or by repeatedly multiplying on the right with **y**. Because monoids are associative, it makes no difference whether left or right multiplication is used.

a consequence of the associative law

The wrapper **monoid[x]** is either a monoid, or the empty set. In either case, it is an associative function. As a consequence, left and right multiplications for **monoid[x]** commute.

Theorem. Left and right multiplications for **monoid[x]** commute.

```
In[2]:= SubstTest[commute, composite[assoc[t], LEFT[y]],
          composite[assoc[t], RIGHT[z]], t → monoid[x]] // Reverse

Out[2]= equal[composite[monoid[x], LEFT[y], monoid[x], RIGHT[z]],
          composite[monoid[x], RIGHT[z], monoid[x], LEFT[y]]] == True

In[3]:= equal[composite[monoid[x_], LEFT[y_], monoid[x_], RIGHT[z_]],
          composite[monoid[x_], RIGHT[z_], monoid[x_], LEFT[y_]]] := True
```

the sequence of powers

If **x** is a monoid, and if **y** and **z** are members of **range[x]**, then the class **iterate[x ∘ LEFT[y], {z}]** is the infinite sequence of elements starting with **z** obtained by repeatedly multiplying on the left with **y**. The **n**-th member of this sequence is usually denoted by $\mathbf{y}^{\mathbf{n}}\ \mathbf{z}$. The following theorem allows one to rewrite this sequence in terms of the special case **z = e[x]**.

Theorem.

```
In[7]:= Map[implies[and[member[y, range[monoid[x]]], 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[7]= 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[y, range[monoid[x]]]], not[member[z, range[monoid[x]]]]] == True
```

```
In[9]:= 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[y_, range[monoid[x_]]]], not[member[z_, range[monoid[x_]]]]] := True
```

Similarly, the class **iterate[x ∘ RIGHT[y], {z}]** is the infinite sequence of elements starting with **z** obtained by repeatedly multiplying on the right with **y**. The **n**-th member of the latter sequence is usually denoted by $\mathbf{z}\ \mathbf{y}^{\mathbf{n}}$. When **z** is the identity element **e[x]** these sequences are equal because the associative law for **x** implies that left multiplication commutes with right multiplication. These facts have already been established earlier, but at that time the monoid wrapper **monoid[x]** had not yet been defined. It is worthwhile to repeat the derivation using this wrapper to witness the advantages of automatic simplifications that take place, and also to slightly improve on the old results by eliminating a redundant literal.

Lemma 1. An application of the uniqueness theorem for **iterate**.

```
In[10]:= Map[implies[member[y, 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], RIGHT[y]], v -> set[y], w -> composite[monoid[x], RIGHT[y],
               iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]}]] // Reverse
```

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

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

Lemma 2. A second application of the uniqueness theorem for **iterate**.

```
In[12]:= Map[implies[member[y, 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], RIGHT[y]], v -> set[y], w -> composite[monoid[x], LEFT[y],
               iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]}]] // Reverse

Out[12]= or[equal[composite[monoid[x], LEFT[y],
            iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]],
           iterate[composite[monoid[x], RIGHT[y]], set[y]]],
          not[member[y, range[monoid[x]]]] == True

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

Lemma 3. A third application of the uniqueness theorem for **iterate**.

```
In[14]:= 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[e[monoid[x]]],
            w -> iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]}] // Reverse

Out[14]= or[equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]],
           not[equal[composite[monoid[x], LEFT[y],
              iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]], composite[monoid[x],
              RIGHT[y], iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]]] == True

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

Lemma 4. If **y ∈ range[monoid[x]]**, then two expressions for the sequence of powers of **y** are equal.

```
In[18]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], (*implies[and[p2,p3],p4],*)
            implies[p4, p5], not[implies[p1, p5]], {p1 -> member[y, range[monoid[x]]],
             p2 → equal[composite[monoid[x], RIGHT[y], iterate[composite[monoid[x], RIGHT[y]],
                 set[e[monoid[x]]]]], iterate[composite[monoid[x], RIGHT[y]], set[y]]],
             p3 → equal[composite[monoid[x], LEFT[y], iterate[composite[monoid[x], RIGHT[y]],
                 set[e[monoid[x]]]]], iterate[composite[monoid[x], RIGHT[y]], set[y]]],
             p4 → equal[composite[monoid[x], LEFT[y], iterate[composite[monoid[x], RIGHT[y]],
                 set[e[monoid[x]]]]], composite[monoid[x], RIGHT[y],
                iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]],
             p5 -> equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
               iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]}]] // Reverse

Out[18]= or[equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]],
           not[member[y, range[monoid[x]]]] == True

In[19]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The literal **y ∈ range[monoid[x]]** in the preceding lemma is redundant. It will now be removed.

Lemma 5. The case that **y** is not a member of **range[monoid[x]]**.

```
In[21]:= SubstTest[implies, and[empty[u], empty[v]],
           equal[iterate[u, set[e[monoid[x]]]], iterate[v, set[e[monoid[x]]]]],
           {u -> composite[monoid[x], LEFT[y]], v -> composite[monoid[x], RIGHT[y]]}] // Reverse

Out[21]= or[equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]],
           member[y, range[monoid[x]]]] == True

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

Theorem. Two expressions for the sequence of powers are equal.

```
In[23]:= SubstTest[and, implies[p, q], or[p, q],
           {p → and[member[y, range[monoid[x]]], not[empty[monoid[x]]]],
            q -> equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
              iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]]}] // MapNotNot

Out[23]= equal[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
           iterate[composite[monoid[x], RIGHT[y]], set[e[monoid[x]]]]] == True

In[24]:= equal[iterate[composite[monoid[x_], LEFT[y_]], set[e[monoid[x_]]]],
           iterate[composite[monoid[x_], RIGHT[y_]], set[e[monoid[x_]]]]] := True
```

The **monoid** wrapper can be removed.

Corollary. If **x ∈ MONOIDS**, then **iterate[x ∘ LEFT[y], {e[x]}]** = **iterate[x ∘ RIGHT[y], {e[x]}]**.

```
In[25]:= SubstTest[implies, equal[x, monoid[t]],
            equal[iterate[composite[x, LEFT[y]], set[e[x]]],
             iterate[composite[x, RIGHT[y]], set[e[x]]], t → x] // Reverse // MapNotNot

Out[25]= or[equal[iterate[composite[x, LEFT[y]], set[e[x]]],
            iterate[composite[x, RIGHT[y]], set[e[x]]]], not[member[x, MONOIDS]]] == True

In[26]:= or[equal[iterate[composite[x_, LEFT[y_]], set[e[x_]]],
            iterate[composite[x_, RIGHT[y_]], set[e[x_]]]], not[member[x_, MONOIDS]]] := True
```

other basic facts

Theorem. If **x ∈ MONOIDS**, then **iterate[monoid[x] ∘ LEFT[y], {z}]** is a function.

```
In[27]:= SubstTest[implies, equal[x, monoid[t]],
            FUNCTION[iterate[composite[x, LEFT[y]], set[z]]], t → x] // Reverse // MapNotNot

Out[27]= or[FUNCTION[iterate[composite[x, LEFT[y]], set[z]]], not[member[x, MONOIDS]]] == True

In[28]:= or[FUNCTION[iterate[composite[x_, LEFT[y_]], set[z_]]],
           not[member[x_, MONOIDS]]] := True
```

Theorem. An analogous result, but with **LEFT** replaced with **RIGHT**.

```
In[29]:= SubstTest[implies, equal[x, monoid[t]],
           FUNCTION[iterate[composite[x, RIGHT[y]], set[z]]], t → x] // Reverse // MapNotNot

Out[29]= or[FUNCTION[iterate[composite[x, RIGHT[y]], set[z]]], not[member[x, MONOIDS]]] == True

In[30]:= or[FUNCTION[iterate[composite[x_, RIGHT[y_]], set[z_]]],
           not[member[x_, MONOIDS]]] := True
```

A better result will now be derived.

Theorem. Left multiplication is a unary operation.

```
In[33]:= SubstTest[member, composite[binop[t], LEFT[y]], UNOPS, t → monoid[x]] // Reverse

Out[33]= member[composite[monoid[x], LEFT[y]], UNOPS] == True

In[34]:= member[composite[monoid[x_], LEFT[y_]], UNOPS] := True
```

Theorem. Right multiplication is a unary operation.

```
In[35]:= SubstTest[member, composite[binop[t], RIGHT[y]], UNOPS, t → monoid[x]] // Reverse

Out[35]= member[composite[monoid[x], RIGHT[y]], UNOPS] == True

In[36]:= member[composite[monoid[x_], RIGHT[y_]], UNOPS] := True
```

The **monoid** wrapper can be removed.

Corollary. If **x** is a nonoid, then **x ∘ LEFT[y]** is a unary operation.

```
In[37]:= SubstTest[implies, equal[x, monoid[t]],
           member[composite[x, LEFT[y]], UNOPS], t → x] // Reverse // MapNotNot

Out[37]= or[member[composite[x, LEFT[y]], UNOPS], not[member[x, MONOIDS]]] == True

In[38]:= or[member[composite[x_, LEFT[y_]], UNOPS], not[member[x_, MONOIDS]]] := True
```

Corollary. If **x** is a nonoid, then **x ∘ RIGHT[y]** is a unary operation.

```
In[39]:= SubstTest[implies, equal[x, monoid[t]],
           member[composite[x, RIGHT[y]], UNOPS], t → x] // Reverse // MapNotNot

Out[39]= or[member[composite[x, RIGHT[y]], UNOPS], not[member[x, MONOIDS]]] == True

In[40]:= or[member[composite[x_, RIGHT[y_]], UNOPS], not[member[x_, MONOIDS]]] := True
```

domain of the power sequence

Theorem.

```
In[41]:= SubstTest[implies,
           and[FUNCTION[t], subclass[range[t], domain[t]], member[y, domain[t]]],
           equal[domain[iterate[t, set[y]]], omega], t → unop[x]] // Reverse

Out[41]= or[equal[omega, domain[iterate[unop[x], set[y]]]],
           not[member[y, domain[unop[x]]]]] == True

In[42]:= or[equal[omega, domain[iterate[unop[x_], set[y_]]]],
           not[member[y_, domain[unop[x_]]]]] := True
```

Corollary. If **x** is a unary operation, and **y ∈ domain[x]**, then **domain[iterate[x, {y}]** = ω.

```
In[43]:= SubstTest[implies, equal[x, unop[t]], or[equal[omega, domain[iterate[x, set[y]]]],
            not[member[y, domain[x]]]], t → x] // Reverse

Out[43]= or[equal[omega, domain[iterate[x, set[y]]]],
           not[member[x, UNOPS]], not[member[y, domain[x]]]] == True

In[44]:= or[equal[omega, domain[iterate[x_, set[y_]]]],
           not[member[x_, UNOPS]], not[member[y_, domain[x_]]]] := True
```

This result will now be applied to powers of monoid elements.

Lemma.

```
In[45]:= SubstTest[member, pair[u, v], cartsq[t], t → range[monoid[x]]] // Reverse

Out[45]= member[pair[u, v], domain[monoid[x]]] ==
          and[member[u, range[monoid[x]]], member[v, range[monoid[x]]]]

In[46]:= member[pair[u_, v_], domain[monoid[x_]]] :=
          and[member[u, range[monoid[x]]], member[v, range[monoid[x]]]]
```

Theorem. If **y** and **z** are members of **range[monoid[x]]**, then **domain[iterate[monoid[x] ∘ LEFT[y], {z}]** = ω. In other words, the quantity $\mathbf{y^n\ z}$ is defined for every natural number **n**.

```
In[47]:= SubstTest[or, equal[omega, domain[iterate[unop[t], set[z]]]],
           not[member[z, domain[unop[t]]]], t → composite[monoid[x], LEFT[y]]] // Reverse

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

In[48]:= or[equal[omega, domain[iterate[composite[monoid[x_], LEFT[y_]], set[z_]]]],
           not[member[y_, range[monoid[x_]]]], not[member[z_, range[monoid[x_]]]]] := True
```

Theorem. Dual result: $\mathbf{z\ y^n}$ is defined for every natural number **n**.

```
In[49]:= SubstTest[or, equal[omega, domain[iterate[unop[t], set[z]]]],
           not[member[z, domain[unop[t]]]], t → composite[monoid[x], RIGHT[y]]] // Reverse

Out[49]= or[equal[omega, domain[iterate[composite[monoid[x], RIGHT[y]], set[z]]]],
           not[member[y, range[monoid[x]]]], not[member[z, range[monoid[x]]]]] == True
```

```
In[50]:= or[equal[omega, domain[iterate[composite[monoid[x_], RIGHT[y_]], set[z_]]]],
            not[member[y_, range[monoid[x_]]]], not[member[z_, range[monoid[x_]]]]] := True
```

The monoid wrapper can be removed from each of the above statement.

Corollary. If **x ∈ MONOIDS** and **y, z ∈ range[x]**, then **domain[iterate[x ∘ LEFT[y], {z}] = ω**.

```
In[51]:= SubstTest[implies, equal[x, monoid[t]],
             or[equal[omega, domain[iterate[composite[x, LEFT[y]], set[z]]]],
              not[member[y, range[x]]], not[member[z, range[x]]]], t → x] // Reverse // MapNotNot
```

```
Out[51]= or[equal[omega, domain[iterate[composite[x, LEFT[y]], set[z]]]],
            not[member[x, MONOIDS]], not[member[y, range[x]]], not[member[z, range[x]]]] == True
```

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

Corollary. If **x ∈ MONOIDS** and **y, z ∈ range[x]**, then **domain[iterate[x ∘ RIGHT[y], {z}] = ω**.

```
In[53]:= SubstTest[implies, equal[x, monoid[t]],
             or[equal[omega, domain[iterate[composite[x, RIGHT[y]], set[z]]]],
              not[member[y, range[x]]], not[member[z, range[x]]]], t → x] // Reverse // MapNotNot
```

```
Out[53]= or[equal[omega, domain[iterate[composite[x, RIGHT[y]], set[z]]]],
            not[member[x, MONOIDS]], not[member[y, range[x]]], not[member[z, range[x]]]] == True
```

```
In[54]:= or[equal[omega, domain[iterate[composite[x_, RIGHT[y_]], set[z_]]]],
            not[member[x_, MONOIDS]], not[member[y_, range[x_]]],
            not[member[z_, range[x_]]]] := True
```

The special case **z = e[x]** is of greatest interest.

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

```
In[55]:= 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[e[x], range[x]],
              p3 -> equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]}]] // Reverse
```

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

```
In[57]:= or[equal[omega, domain[iterate[composite[x_, LEFT[y_]], set[e[x_]]]]],
            not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```

the 0-th and 1-st powers

Lemma.

```
In[60]:= SubstTest[implies,
           and[FUNCTION[t], equal[image[t, set[u]], set[v]], member[u, domain[t]]],
           equal[APPLY[t, u], v], {t → iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
            u → 0, v → e[monoid[x]]}] // Reverse

Out[60]= or[equal[0, monoid[x]],
           equal[APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], 0],
            e[monoid[x]]]] == True

In[61]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Simplification rule.

```
In[62]:= equiv[and[member[x, MONOIDS], not[equal[0, x]]], member[x, MONOIDS]]

Out[62]= True

In[63]:= and[member[x_, MONOIDS], not[equal[0, x_]]] := member[x, MONOIDS]
```

Theorem. If $\mathbf{x \in MONOIDS}$, then $\mathbf{y^0 = e[x]}$.

```
In[64]:= SubstTest[implies, equal[x, monoid[t]],
            or[equal[0, x], equal[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], 0], e[x]]],
            t → x] // Reverse // MapNotNot

Out[64]= or[equal[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], 0], e[x]],
           not[member[x, MONOIDS]]] == True

In[65]:= or[equal[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], 0], e[x_]],
           not[member[x_, MONOIDS]]] := True
```

Lemma. (Specialize to the case $\mathbf{n = 1}$.)

```
In[68]:= Map[implies[member[y, range[monoid[x]]], #] &, SubstTest[implies,
            and[FUNCTION[t], equal[image[t, set[u]], set[v]], member[u, domain[t]]], equal[
             APPLY[t, u], v], {t → iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]],
             u → set[0], v → y}]] // Reverse

Out[68]= or[equal[y, APPLY[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]], set[0]]],
           not[member[y, range[monoid[x]]]], not[member[set[0],
             domain[iterate[composite[monoid[x], LEFT[y]], set[e[monoid[x]]]]]]]] == True

In[69]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

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

```
In[71]:= SubstTest[implies, equal[x, monoid[t]],
           or[equal[y, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], set[0]]],
            not[member[y, range[x]]],
            not[member[set[0], domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]]],
           t → x] // Reverse // MapNotNot

Out[71]= or[equal[y, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], set[0]]],
          not[member[x, MONOIDS]], not[member[y, range[x]]],
          not[member[set[0], domain[iterate[composite[x, LEFT[y]], set[e[x]]]]]] == True

In[72]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. If $\mathbf{x}$ is a monoid and $\mathbf{y \in range[x]}$, then $\mathbf{y^1 = y}$.

```
In[74]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[and[p1, p3], p4],
           not[implies[p1, p4]], {p1 -> and[member[x, MONOIDS], member[y, range[x]]],
            p2 → equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]],
            p3 -> member[set[0], domain[iterate[composite[x, LEFT[y]], set[e[x]]]]], p4 ->
             equal[y, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], set[0]]]}]] // Reverse

Out[74]= or[equal[y, APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], set[0]]],
          not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[76]:= or[equal[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], set[0]], y_],
          not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```

---

range of the power sequence

Theorem. A hull formula for the range of the sequence $\mathbf{y^n\ z}$.

```
In[79]:= SubstTest[range, iterate[funpart[t], set[z]],
           t -> composite[monoid[x], LEFT[y]]] // Reverse

Out[79]= range[iterate[composite[monoid[x], LEFT[y]], set[z]]] ==
          hull[invar[composite[monoid[x], LEFT[y]]], set[z]]

In[80]:= range[iterate[composite[monoid[x_], LEFT[y_]], set[z_]]] :=
          hull[invar[composite[monoid[x], LEFT[y]]], set[z]]
```

Theorem. A hull formula for the range of the sequence $\mathbf{z\ y^n}$.

```
In[81]:= SubstTest[range, iterate[funpart[t], set[z]],
           t -> composite[monoid[x], RIGHT[y]]] // Reverse

Out[81]= range[iterate[composite[monoid[x], RIGHT[y]], set[z]]] ==
          hull[invar[composite[monoid[x], RIGHT[y]]], set[z]]

In[82]:= range[iterate[composite[monoid[x_], RIGHT[y_]], set[z_]]] :=
          hull[invar[composite[monoid[x], RIGHT[y]]], set[z]]
```

Lemma.

```
In[83]:= SubstTest[subclass, range[iterate[t, z]], union[range[t], z],
           {t → composite[monoid[x], LEFT[y]], z → set[e[monoid[x]]]}] // Reverse

Out[83]= subclass[hull[invar[composite[monoid[x], LEFT[y]]], set[e[monoid[x]]]],
           union[image[monoid[x], cart[set[y], V]], set[e[monoid[x]]]]] == True

In[84]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. An upper bound for the range.

```
In[85]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u -> hull[invar[composite[monoid[x], LEFT[y]]], set[e[monoid[x]]]],
            v -> union[image[monoid[x], cart[set[y], V]], set[e[monoid[x]]]],
            w → range[monoid[x]]}] // Reverse

Out[85]= subclass[hull[invar[composite[monoid[x], LEFT[y]]], set[e[monoid[x]]]],
           range[monoid[x]]] == True

In[86]:= subclass[hull[invar[composite[monoid[x_], LEFT[y_]]], set[e[monoid[x_]]]],
           range[monoid[x_]]] := True
```

Corollary.

```
In[87]:= SubstTest[implies, equal[x, monoid[t]],
            subclass[hull[invar[composite[x, LEFT[y]]], set[e[x]]], range[x]],
            t → x] // Reverse // MapNotNot

Out[87]= or[not[member[x, MONOIDS]],
           subclass[hull[invar[composite[x, LEFT[y]]], set[e[x]]], range[x]]] == True

In[88]:= or[not[member[x_, MONOIDS]],
           subclass[hull[invar[composite[x_, LEFT[y_]]], set[e[x_]]], range[x_]]] := True
```

Corollary. If **x** is a monoid, the set of all powers of **y** is a subset of **range[x]**.

```
In[89]:= SubstTest[implies, equal[x, monoid[t]],
            subclass[range[iterate[composite[x, LEFT[y]], set[e[x]]]], range[x]],
            t → x] // Reverse // MapNotNot

Out[89]= or[not[member[x, MONOIDS]],
           subclass[range[iterate[composite[x, LEFT[y]], set[e[x]]]], range[x]]] == True

In[90]:= or[not[member[x_, MONOIDS]],
           subclass[range[iterate[composite[x_, LEFT[y_]], set[e[x_]]]], range[x_]]] := True
```

Theorem. If **x** is a monoid and **y ∈ range[x]**, then $\mathbf{y^n \in range[x]}$ for any natural number **n = nat[z]**.

```
In[91]:= Map[not, SubstTest[and, (*implies[p1,p2],*) implies[p2, p3], (*implies[p1,p4],*)
           implies[p1, p5], (*implies[and[p3,p5],p6],*) implies[and[p4, p6], p7],
           not[implies[p1, p7]], {p1 -> and[member[x, MONOIDS], member[y, range[x]]],
            p2 -> equal[omega, domain[iterate[composite[x, LEFT[y]], set[e[x]]]]],
            p3 → member[nat[z], domain[iterate[composite[x, LEFT[y]], set[e[x]]]]],
            p4 -> subclass[range[iterate[composite[x, LEFT[y]], set[e[x]]]], range[x]],
            p5 → FUNCTION[iterate[composite[x, LEFT[y]], set[e[x]]]],
            p6 → member[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[z]],
              range[iterate[composite[x, LEFT[y]], set[e[x]]]], p7 -> member[
              APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[z]], range[x]]}]] // Reverse

Out[91]= or[member[APPLY[iterate[composite[x, LEFT[y]], set[e[x]]], nat[z]], range[x]],
           not[member[x, MONOIDS]], not[member[y, range[x]]]] == True

In[92]:= or[member[APPLY[iterate[composite[x_, LEFT[y_]], set[e[x_]]], nat[z_]], range[x_]],
           not[member[x_, MONOIDS]], not[member[y_, range[x_]]]] := True
```