PRIMESEQ

Johan G. F. Belinfante
2005 August 17

```
In[1]:= SetDirectory["i:"]; << goedel72.16a; << tools.m

        :Package Title: goedel72.16a              2005 August 16 at 10:15 a.m.

        It is now:  2005 Aug 17 at 12:18

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 August 2

        weightlimit = 40
```

summary

It is shown in this notebook that there are countably infinitely many primes. To do so, one needs a one-to-one correspondence between **omega** and **PRIMES**. The sequence **PRIMESEQ** listing the primes in increasing order is constructed using **iterate**, emulating Quaife's recursive definition of the "n-th prime" **pr[n]**. Induction is used to show that the domain and range of **PRIMESEQ** are **omega** and **PRIMES**, respectively. The one-to-one property is derived from the fact that this sequence is strictly monotone increasing.

```
In[2]:= "Art Quaife, Automated Development of Fundamental Mathematical Theories,
           Kluwer Academic Publishers, Dordrecht, the Netherlands, 1992.";
```

Comment: Art Quaife starts his sequence with **1**, which is not a prime, whereas the sequence **PRIMESEQ** starts with **2** as the **0**-th prime.

definition of PRIMESEQ

The definition of **PRIMESEQ** uses **iterate**.

```
In[3]:= iterate[composite[HULL[PRIMES], SUCC], set[succ[set[0]]]] := PRIMESEQ
```

This is a function:

```
In[4]:= SubstTest[FUNCTION, iterate[funpart[x], set[setpart[y]]],
          {x → composite[HULL[PRIMES], SUCC], y → succ[set[0]]}]

Out[4]= FUNCTION[PRIMESEQ] == True

In[5]:= FUNCTION[PRIMESEQ] := True
```

Vertical sections of functions are singletons.

```
In[6]:= SubstTest[image, funpart[z], set[x], z → PRIMESEQ]

Out[6]= image[PRIMESEQ, set[x]] == set[APPLY[PRIMESEQ, x]]

In[7]:= image[PRIMESEQ, set[x_]] := set[APPLY[PRIMESEQ, x]]
```

The domain of **PRIMESEQ** is contained in the set of natural numbers, because that is a general property of the **iterate** constructor:

```
In[8]:= SubstTest[composite, iterate[u, v], id[omega],
          {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]

Out[8]= composite[PRIMESEQ, id[omega]] == PRIMESEQ

In[9]:= composite[PRIMESEQ, id[omega]] := PRIMESEQ
```

The following temporary rewrite rule will shortly be replaced with a better one.

```
In[10]:= Map[subclass[#, omega] &, IminComp[PRIMESEQ, id[omega], V]]

Out[10]= subclass[domain[PRIMESEQ], omega] == True

In[11]:= % /. Equal → SetDelayed
```

---

the first three primes

The **GOEDEL** program can determine the first few entries in the sequence of primes.

```
In[12]:= Map[A, SubstTest[image, iterate[u, v], set[0],
           {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]]

Out[12]= APPLY[PRIMESEQ, 0] == succ[set[0]]

In[13]:= APPLY[PRIMESEQ, 0] := succ[set[0]]

In[14]:= Map[A, SubstTest[image, iterate[u, v], set[set[0]],
           {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]]

Out[14]= APPLY[PRIMESEQ, set[0]] == succ[succ[set[0]]]
```

```
In[15]:= APPLY[PRIMESEQ, set[0]] := succ[succ[set[0]]]

In[16]:= Map[A, SubstTest[image, iterate[u, v], set[succ[set[0]]],
           {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]]

Out[16]= APPLY[PRIMESEQ, succ[set[0]]] == succ[succ[succ[succ[set[0]]]]]

In[17]:= APPLY[PRIMESEQ, succ[set[0]]] := succ[succ[succ[succ[set[0]]]]]
```

To go much further, one would soon need a sieve algorithm for primes which has not yet been implemented.

---

recursion relation

The recursion relation for the function **PRIMESEQ** can be expressed as follows:

```
In[18]:= Map[A, SubstTest[image, iterate[u, v], set[succ[x]],
            {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]] // Reverse

Out[18]= hull[PRIMES, succ[APPLY[PRIMESEQ, x]]] == APPLY[PRIMESEQ, succ[x]]

In[19]:= hull[PRIMES, succ[APPLY[PRIMESEQ, x_]]] := APPLY[PRIMESEQ, succ[x]]
```

The recursion relation can also be expressed in variable-free form as follows:

```
In[20]:= SubstTest[composite, iterate[u, v], SUCC,
          {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]]}]

Out[20]= composite[PRIMESEQ, SUCC] == composite[HULL[PRIMES], SUCC, PRIMESEQ]

In[21]:= composite[PRIMESEQ, SUCC] := composite[HULL[PRIMES], SUCC, PRIMESEQ]
```

---

range[PRIMESEQ] is contained in PRIMES

In this section it is shown that the range of **PRIMSEQ** is contained in the set **PRIMES** of all primes. Later this inclusion will be sharpened to an equation.

```
In[22]:= SubstTest[implies, and[invariant[u, w], subclass[v, w]],
          subclass[range[iterate[u, v]], w],
          {u -> composite[HULL[PRIMES], SUCC], v -> set[succ[set[0]]], w → PRIMES}]

Out[22]= subclass[range[PRIMESEQ], PRIMES] == True

In[23]:= % /. Equal → SetDelayed
```

Lemma.

```
In[24]:= equal[intersection[PRIMES, range[PRIMESEQ]], range[PRIMESEQ]]

Out[24]= True

In[25]:= intersection[PRIMES, range[PRIMESEQ]] := range[PRIMESEQ]
```

Other useful rewrite rules:

```
In[26]:= Assoc[id[PRIMES], id[range[PRIMESEQ]], PRIMESEQ]

Out[26]= composite[id[PRIMES], PRIMESEQ] == PRIMESEQ

In[27]:= composite[id[PRIMES], PRIMESEQ] := PRIMESEQ

In[28]:= Assoc[id[omega], id[PRIMES], PRIMESEQ]

Out[28]= composite[id[omega], PRIMESEQ] == PRIMESEQ

In[29]:= composite[id[omega], PRIMESEQ] := PRIMESEQ
```

Corollary.

```
In[30]:= Assoc[SUCC, id[omega], PRIMESEQ] // Reverse

Out[30]= composite[id[omega], SUCC, PRIMESEQ] == composite[SUCC, PRIMESEQ]

In[31]:= composite[id[omega], SUCC, PRIMESEQ] := composite[SUCC, PRIMESEQ]
```

Main result of this section:

```
In[32]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
          subclass[u, w], {u → range[PRIMESEQ], v → PRIMES, w → omega}]

Out[32]= subclass[range[PRIMESEQ], omega] == True

In[33]:= % /. Equal → SetDelayed
```

---

domain[PRIMESEQ] = omega

The proof that the domain of **PRIMESEQ** is **omega** uses induction, applied to the domain of **PRIMESEQ**. The base case is:

```
In[34]:= Map[not, SubstTest[equal, 0, image[x, set[0]], x → PRIMESEQ]] // Reverse

Out[34]= member[0, domain[PRIMESEQ]] == True
```

```
In[35]:= % /. Equal → SetDelayed
```

Lemma.

```
In[36]:= IminComp[PRIMESEQ, id[omega], V] // Reverse

Out[36]= intersection[omega, domain[PRIMESEQ]] == domain[PRIMESEQ]

In[37]:= % /. Equal → SetDelayed
```

Lemma.

```
In[38]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> omega,
           v -> image[inverse[SUCC], image[inverse[S], omega]], w → inverse[PRIMESEQ]}]

Out[38]= subclass[domain[PRIMESEQ], image[inverse[PRIMESEQ],
            image[inverse[SUCC], image[inverse[S], omega]]]] == True

In[39]:= % /. Equal → SetDelayed
```

The recursion relation for **PRIMESEQ** yields:

```
In[40]:= Map[subclass[domain[PRIMESEQ], #] &, IminComp[PRIMESEQ, SUCC, V]] // Reverse

Out[40]= subclass[image[SUCC, domain[PRIMESEQ]], domain[PRIMESEQ]] == True

In[41]:= % /. Equal → SetDelayed
```

An application of induction now yields:

```
In[42]:= SubstTest[implies, INDUCTIVE[x], subclass[omega, x], x → domain[PRIMESEQ]]

Out[42]= subclass[omega, domain[PRIMESEQ]] == True

In[43]:= % /. Equal → SetDelayed
```

Theorem.

```
In[44]:= SubstTest[and, subclass[u, v],
          subclass[v, u], {u → omega, v → domain[PRIMESEQ]}]

Out[44]= True == equal[omega, domain[PRIMESEQ]]

In[45]:= domain[PRIMESEQ] := omega
```

everything in the sequence PRIMESEQ is a prime

Everyting listed in the sequence **PRIMESEQ** is a number.

```
In[46]:= (member[x, image[inverse[funpart[z]], omega]] // AssertTest // Reverse) /.
          z → PRIMESEQ

Out[46]= member[APPLY[PRIMESEQ, x], omega] == member[x, omega]

In[47]:= member[APPLY[PRIMESEQ, x_], omega] := member[x, omega]
```

Everyting listed in the sequence **PRIMESEQ** is a prime.

```
In[48]:= (member[x, image[inverse[funpart[z]], PRIMES]] // AssertTest // Reverse) /.
          z → PRIMESEQ

Out[48]= member[APPLY[PRIMESEQ, x], PRIMES] == member[x, omega]

In[49]:= member[APPLY[PRIMESEQ, x_], PRIMES] := member[x, omega]
```

some corollaries of numberhood

Trichotomy:

```
In[50]:= SubstTest[or, member[nat[w], nat[t]], equal[nat[w], nat[t]],
          member[nat[t], nat[w]], t -> APPLY[PRIMESEQ, nat[x]]]

Out[50]= or[equal[APPLY[PRIMESEQ, nat[x]], nat[w]],
           member[APPLY[PRIMESEQ, nat[x]], nat[w]],
           member[nat[w], APPLY[PRIMESEQ, nat[x]]]] == True

In[51]:= or[equal[APPLY[PRIMESEQ, nat[x_]], nat[w_]],
           member[APPLY[PRIMESEQ, nat[x_]], nat[w_]],
           member[nat[w_], APPLY[PRIMESEQ, nat[x_]]]] := True
```

For numbers, one can eliminate **subclass**. Containment can be replaced with non-membership in the reverse direction.

```
In[52]:= SubstTest[subclass, nat[u], nat[v],
          {u -> APPLY[PRIMESEQ, nat[x]], v → APPLY[PRIMESEQ, nat[y]]}]

Out[52]= subclass[APPLY[PRIMESEQ, nat[x]], APPLY[PRIMESEQ, nat[y]]] ==
          not[member[APPLY[PRIMESEQ, nat[y]], APPLY[PRIMESEQ, nat[x]]]]
```

```
In[53]:= subclass[APPLY[PRIMESEQ, nat[x_]], APPLY[PRIMESEQ, nat[y_]]] :=
          not[member[APPLY[PRIMESEQ, nat[y]], APPLY[PRIMESEQ, nat[x]]]]
```

Another such result:

```
In[54]:= SubstTest[subclass, nat[s], nat[w], s -> APPLY[PRIMESEQ, succ[nat[x]]]]

Out[54]= subclass[APPLY[PRIMESEQ, succ[nat[x]]], nat[w]] ==
          not[member[nat[w], APPLY[PRIMESEQ, succ[nat[x]]]]]

In[55]:= subclass[APPLY[PRIMESEQ, succ[nat[x_]]], nat[w_]] :=
          not[member[nat[w], APPLY[PRIMESEQ, succ[nat[x]]]]]
```

---

n is a lower bound on the n-th prime

Lemma.

```
In[56]:= SubstTest[implies, subclass[u, v],
          subclass[composite[u, w], composite[v, w]],
          {u → HULL[PRIMES], v → composite[id[omega], S], w → SUCC}]

Out[56]= subclass[composite[HULL[PRIMES], SUCC], E] == True

In[57]:= subclass[composite[HULL[PRIMES], SUCC], E] := True

In[58]:= SubstTest[implies, and[subclass[u, x], subclass[v, y]],
          subclass[iterate[u, v], iterate[x, y]],
          {u -> composite[HULL[PRIMES], SUCC], v → set[succ[set[0]]],
           x → composite[id[omega], E], y → omega}]

Out[58]= subclass[PRIMESEQ, S] == True

In[59]:= subclass[PRIMESEQ, S] := True

In[60]:= SubstTest[implies, subclass[u, v],
          subclass[APPLY[v, x], APPLY[u, x]], {u → PRIMESEQ, v → S}]

Out[60]= subclass[x, APPLY[PRIMESEQ, x]] == True

In[61]:= subclass[x_, APPLY[PRIMESEQ, x_]] := True
```

---

strict monotonicity

Each entry in the sequence **PRIMESEQ** is strictly less than the next.

```
In[62]:= SubstTest[implies, subclass[u, v], subclass[image[u, w], image[v, w]],
          {u → composite[PRIMESEQ, SUCC], v → composite[E, PRIMESEQ], w → set[nat[x]]}]

Out[62]= member[APPLY[PRIMESEQ, nat[x]], APPLY[PRIMESEQ, succ[nat[x]]]] == True

In[63]:= member[APPLY[PRIMESEQ, nat[x_]], APPLY[PRIMESEQ, succ[nat[x_]]]] := True
```

The sequence **PRIMESEQ** is strictly monotone:

```
In[64]:= SubstTest[implies, subclass[composite[x, y], composite[z, x]],
          subclass[composite[x, trv[y]], composite[trv[z], x]],
          {x → PRIMESEQ, y → composite[id[omega], SUCC], z → composite[id[omega], E]}]

Out[64]= subclass[composite[PRIMESEQ, E], composite[E, PRIMESEQ]] == True

In[65]:= subclass[composite[PRIMESEQ, E], composite[E, PRIMESEQ]] := True
```

---

strict monotone => one-to-one

Lemmas:

```
In[66]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u → Id, v → complement[y], w → inverse[cross[z, z]]}] /.
          {y → composite[id[omega], E], z → PRIMESEQ}

Out[66]= subclass[composite[inverse[PRIMESEQ], PRIMESEQ],
           composite[inverse[PRIMESEQ], id[omega], complement[E], PRIMESEQ]] == True

In[67]:= % /. Equal → SetDelayed
```

Lemmas:

```
In[68]:= Map[assert, SubstTest[implies, subclass[u, v],
            subclass[composite[x, u, y], composite[x, v, y]],
            {u -> composite[PRIMESEQ, E], v -> composite[E, PRIMESEQ],
             x → Id, y → composite[inverse[PRIMESEQ]]}]] // InvertFix

Out[68]= subclass[omega, fix[composite[inverse[PRIMESEQ], S, IMAGE[PRIMESEQ]]]] == True

In[69]:= % /. Equal → SetDelayed

In[70]:= SubstTest[subclass, fix[composite[x, y]], range[x],
          {x -> inverse[PRIMESEQ], y -> composite[S, IMAGE[PRIMESEQ]]}]

Out[70]= subclass[fix[composite[inverse[PRIMESEQ], S, IMAGE[PRIMESEQ]]], omega] == True

In[71]:= % /. Equal → SetDelayed
```

Theorem.

```
In[72]:= SubstTest[and, subclass[u, v], subclass[v, u], {u -> omega,
           v -> fix[composite[inverse[PRIMESEQ], S, IMAGE[PRIMESEQ]]]}] // Reverse

Out[72]= equal[omega, fix[composite[inverse[PRIMESEQ], S, IMAGE[PRIMESEQ]]]] == True

In[73]:= fix[composite[inverse[PRIMESEQ], S, IMAGE[PRIMESEQ]]] := omega
```

Lemma.

```
In[74]:= subclass[composite[PRIMESEQ, inverse[IMAGE[PRIMESEQ]]], S] // AssertTest //
          InvertFix

Out[74]= subclass[composite[PRIMESEQ, inverse[IMAGE[PRIMESEQ]]], S] == True

In[75]:= % /. Equal → SetDelayed
```

Lemma.

```
In[76]:= subclass[composite[id[omega], E],
           composite[inverse[PRIMESEQ], id[omega], E, PRIMESEQ]] // AssertTest

Out[76]= subclass[composite[id[omega], E],
           composite[inverse[PRIMESEQ], id[omega], E, PRIMESEQ]] == True

In[77]:= % /. Equal → SetDelayed

In[78]:= subclass[composite[id[omega], intersection[complement[E],
             complement[inverse[E]]], id[omega]], Id] // AssertTest

Out[78]= subclass[composite[id[omega],
           intersection[complement[E], complement[inverse[E]]], id[omega]], Id] == True

In[79]:= % /. Equal → SetDelayed

In[80]:= SubstTest[implies,
           and[subclass[composite[inverse[funpart[z]], funpart[z]], composite[
              inverse[funpart[z]], complement[y], funpart[z]]], equal[0, intersection[
              x, composite[inverse[funpart[z]], complement[y], funpart[z]]]]],
           subclass[composite[inverse[funpart[z]], funpart[z]], complement[x]],
           {z → PRIMESEQ, x → composite[id[omega], E],
            y → composite[id[omega], E]}] // InvertFix // InvertFix

Out[80]= equal[0, intersection[omega,
           fix[composite[inverse[IMAGE[PRIMESEQ]], E, PRIMESEQ]]]] == True

In[81]:= intersection[omega,
           fix[composite[inverse[IMAGE[PRIMESEQ]], E, PRIMESEQ]]] := 0
```

A better result holds:

```
In[82]:= (or[equal[0, fix[composite[inverse[funpart[z]], funpart[z], x]]], not[equal[0,
              intersection[x, composite[inverse[funpart[z]], funpart[z]]]]]] //
           AssertTest) /. {z → PRIMESEQ, x → composite[id[omega], E],
          y → composite[id[omega], E]}

Out[82]= equal[0, fix[composite[inverse[PRIMESEQ], PRIMESEQ, E]]] == True

In[83]:= fix[composite[inverse[PRIMESEQ], PRIMESEQ, E]] := 0
```

It follows that **PRIMESEQ** is one-to-one.

```
In[84]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
           {u → composite[inverse[z], z], v → intersection[cart[domain[z], domain[z]],
              complement[x], complement[inverse[x]]], w → Id}] /.
         {z → PRIMESEQ, x → composite[id[omega], E], y → composite[id[omega], E]}

Out[84]= FUNCTION[inverse[PRIMESEQ]] == True

In[85]:= FUNCTION[inverse[PRIMESEQ]] := True
```

---

an induction proof

To prove that every prime is listed in the sequence **PRIMESEQ**, a lemma is needed that says in effect that for any listed prime, all lesser primes have been listed. The following is a membership rule for a class needed in the proof: (The base case for the proof follows as an automatic consequence of this rewrite rule.)

```
In[86]:= member[x, fix[composite[inverse[IMAGE[PRIMESEQ]],
             S, IMAGE[id[PRIMES]], PRIMESEQ]]] // AssertTest

Out[86]= member[x,
           fix[composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]] ==
         and[member[x, omega],
          subclass[intersection[PRIMES, APPLY[PRIMESEQ, x]], image[PRIMESEQ, x]]]

In[87]:= member[x_,
           fix[composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]] :=
         and[member[x, omega], subclass[
           intersection[PRIMES, APPLY[PRIMESEQ, x]], image[PRIMESEQ, x]]]
```

Lemma.

```
In[88]:= image[inverse[SUCC], image[inverse[S], omega]] // Normality

Out[88]= image[inverse[SUCC], image[inverse[S], omega]] == omega

In[89]:= image[inverse[SUCC], image[inverse[S], omega]] := omega
```

Lemma. Consequence of the iteration condition for the function **PRIMESEQ**. Every prime less than a listed prime is also less than the next listed prime.

```
In[90]:= SubstTest[implies, member[t, u], subclass[A[u], t], {t → nat[w],
           u → intersection[PRIMES, image[S, set[succ[APPLY[PRIMESEQ, nat[x]]]]]]}]

Out[90]= or[not[member[APPLY[PRIMESEQ, nat[x]], nat[w]]], not[member[nat[w], PRIMES]],
           not[member[nat[w], APPLY[PRIMESEQ, succ[nat[x]]]]]] == True

In[91]:= or[not[member[APPLY[PRIMESEQ, nat[x_]], nat[w_]]],
           not[member[nat[w_], PRIMES]],
           not[member[nat[w_], APPLY[PRIMESEQ, succ[nat[x_]]]]]] := True
```

Lemma. Needed to use the inductive hypothesis.

```
In[92]:= SubstTest[implies, and[member[t, u], subclass[u, v]], member[t, v],
          {t → nat[w], u → intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
           v -> image[PRIMESEQ, nat[x]]}]

Out[92]= or[member[nat[w], image[PRIMESEQ, nat[x]]],
           not[member[nat[w], PRIMES]], not[member[nat[w], APPLY[PRIMESEQ, nat[x]]]],
           not[subclass[intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
             image[PRIMESEQ, nat[x]]]]] == True

In[93]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Lemma.

```
In[94]:= image[PRIMESEQ, succ[x]] // Normality

Out[94]= image[PRIMESEQ, succ[x]] == union[image[PRIMESEQ, x], set[APPLY[PRIMESEQ, x]]]

In[95]:= image[PRIMESEQ, succ[x_]] := union[image[PRIMESEQ, x], set[APPLY[PRIMESEQ, x]]]
```

Theorem: Induction step.

```
In[96]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[p3, or[p4, p5]], implies[and[p1, p4, p6], p7],
           implies[and[p1, p5, p6], p7], not[implies[and[p1, p2, p6], p7]],
           {p1 → member[nat[w], PRIMES],
            p2 → member[nat[w], APPLY[PRIMESEQ, succ[nat[x]]]],
            p3 → not[member[APPLY[PRIMESEQ, nat[x]], nat[w]]],
            p4 → equal[nat[w], APPLY[PRIMESEQ, nat[x]]],
            p5 → member[nat[w], APPLY[PRIMESEQ, nat[x]]],
            p6 → subclass[intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
              image[PRIMESEQ, nat[x]]],
            p7 → member[nat[w], image[PRIMESEQ, succ[nat[x]]]]}]]

Out[96]= or[equal[APPLY[PRIMESEQ, nat[x]], nat[w]],
           member[nat[w], image[PRIMESEQ, nat[x]]], not[member[nat[w], PRIMES]],
           not[member[nat[w], APPLY[PRIMESEQ, succ[nat[x]]]]],
           not[subclass[intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
             image[PRIMESEQ, nat[x]]]] == True

In[97]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Remove the nat wrapper on **w** and then eliminate that variable.

```
In[98]:= Map[equal[V, class[w, #]] &,
          SubstTest[implies, and[equal[w, nat[y]], member[w, PRIMES],
            member[w, APPLY[PRIMESEQ, succ[nat[x]]]], subclass[
             intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]], image[PRIMESEQ, nat[x]]]],
           member[w, image[PRIMESEQ, succ[nat[x]]]], y -> w]]

Out[98]= or[not[subclass[
             intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]], image[PRIMESEQ, nat[x]]]],
           subclass[intersection[PRIMES, APPLY[PRIMESEQ, succ[nat[x]]]],
            union[image[PRIMESEQ, nat[x]], set[APPLY[PRIMESEQ, nat[x]]]]]] == True

In[99]:= (% /. x → x_) /. Equal → SetDelayed
```

Next, eliminate the variable **x**.

```
In[100]:=
        Map[equal[0, complement[fix[#]]] &, SubstTest[class, pair[x, y],
           implies[and[equal[x, nat[y]], member[x, z]], member[succ[x], z]],
           z -> fix[composite[inverse[IMAGE[PRIMESEQ]],
              S, IMAGE[id[PRIMES]], PRIMESEQ]]]] // Reverse

Out[100]=
        subclass[intersection[omega, image[SUCC, fix[
             composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]]],
          fix[composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]] ==
         True
```

In[101]:=
```
% /. Equal → SetDelayed
```

Apply induction.

In[102]:=
```
SubstTest[implies, INDUCTIVE[x], subclass[omega, x], x → intersection[omega,
   fix[composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]]]
```

Out[102]=
```
subclass[omega, fix[
   composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]] == True
```

In[103]:=
```
% /. Equal → SetDelayed
```

Reintroduce variables.

In[104]:=
```
SubstTest[implies, and[member[u, v], subclass[v, w]],
 member[u, w], {u → nat[x], v → omega, w ->
   fix[composite[inverse[IMAGE[PRIMESEQ]], S, IMAGE[id[PRIMES]], PRIMESEQ]]}]
```

Out[104]=
```
subclass[intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
  image[PRIMESEQ, nat[x]]] == True
```

In[105]:=
```
(% /. x → x_) /. Equal → SetDelayed
```

In the next section this result is strengthened to an equation.

---

an equation

The basic idea in this section is to introduce variables to interpret the following inclusion:.

In[106]:=
```
subclass[PRIMESEQ, composite[S, IMAGE[PRIMESEQ]]] // AssertTest
```

Out[106]=
```
subclass[PRIMESEQ, composite[S, IMAGE[PRIMESEQ]]] == True
```

In[107]:=
```
subclass[PRIMESEQ, composite[S, IMAGE[PRIMESEQ]]] := True
```

Lemma:

```
In[108]:=
member[pair[nat[x], APPLY[PRIMESEQ, nat[x]]], PRIMESEQ] // AssertTest
```

```
Out[108]=
member[pair[nat[x], APPLY[PRIMESEQ, nat[x]]], PRIMESEQ] == True
```

```
In[109]:=
(% /. x → x_) /. Equal → SetDelayed
```

Temporary lemma.

```
In[110]:=
member[pair[x, y], composite[S, IMAGE[PRIMESEQ]]] // AssertTest
```

```
Out[110]=
member[pair[x, y], composite[S, IMAGE[PRIMESEQ]]] ==
 and[member[x, V], member[y, V], subclass[image[PRIMESEQ, x], y]]
```

```
In[111]:=
member[pair[x_, y_], composite[S, IMAGE[PRIMESEQ]]] :=
 and[member[x, V], member[y, V], subclass[image[PRIMESEQ, x], y]]
```

Theorem.

```
In[112]:=
SubstTest[implies, and[member[u, v], subclass[v, w]],
 member[u, w], {u → pair[nat[x], APPLY[PRIMESEQ, nat[x]]],
  v → PRIMESEQ, w -> composite[S, IMAGE[PRIMESEQ]]}]
```

```
Out[112]=
subclass[image[PRIMESEQ, nat[x]], APPLY[PRIMESEQ, nat[x]]] == True
```

```
In[113]:=
(% /. x → x_) /. Equal → SetDelayed
```

The inclusion derived in the preceding section can now be strengthened to an equation and made into a rewrite rule.

```
In[114]:=
SubstTest[and, subclass[u, v], subclass[v, u],
 {u -> intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]],
  v -> image[PRIMESEQ, nat[x]]}]
```

```
Out[114]=
True ==
 equal[image[PRIMESEQ, nat[x]], intersection[PRIMES, APPLY[PRIMESEQ, nat[x]]]]
```

```
In[115]:=
intersection[PRIMES, APPLY[PRIMESEQ, nat[x_]]] := image[PRIMESEQ, nat[x]]
```

Applying **reify** yields this variable-free version:

In[116]:=
```
Map[composite[VERTSECT[#], id[omega]] &, SubstTest[reify, x,
   intersection[PRIMES, APPLY[PRIMESEQ, f[x]]], f → nat]] // Reverse
```

Out[116]=
```
composite[IMAGE[id[PRIMES]], PRIMESEQ] == composite[IMAGE[PRIMESEQ], id[omega]]
```

In[117]:=
```
composite[IMAGE[id[PRIMES]], PRIMESEQ] := composite[IMAGE[PRIMESEQ], id[omega]]
```

---

corollary: range[PRIMESEQ] = PRIMES

Lemma.

In[118]:=
```
SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
 {u → nat[x], v → APPLY[PRIMESEQ, nat[x]], w → id[PRIMES]}]
```

Out[118]=
```
subclass[intersection[PRIMES, nat[x]], image[PRIMESEQ, nat[x]]] == True
```

In[119]:=
```
(% /. {x → x_}) /. Equal → SetDelayed
```

Lemma.

In[120]:=
```
SubstTest[implies, and[subclass[t, u], subclass[u, v]],
 subclass[t, v], {t → intersection[PRIMES, nat[x]],
  u -> image[PRIMESEQ, nat[x]], v → range[PRIMESEQ]}]
```

Out[120]=
```
subclass[intersection[PRIMES, nat[x]], range[PRIMESEQ]] == True
```

In[121]:=
```
(% /. x → x_) /. Equal → SetDelayed
```

Remove the variable **x**.

In[122]:=
```
Map[equal[V, #] &, SubstTest[class, x,
   subclass[intersection[PRIMES, nat[x]], y], y -> range[PRIMESEQ]]] // Reverse
```

Out[122]=
```
subclass[PRIMES, range[PRIMESEQ]] == True
```

In[123]:=
```
% /. Equal → SetDelayed
```

This can be strengthened to an equation for the range of **PRIMESEQ**.

```
In[124]:=
SubstTest[and, subclass[u, v], subclass[v, u], {u → range[PRIMESEQ], v → PRIMES}]

Out[124]=
True == equal[PRIMES, range[PRIMESEQ]]

In[125]:=
range[PRIMESEQ] := PRIMES
```

---

corollary: there are countably infinitely many primes

Sethood lemma.

```
In[126]:=
member[PRIMESEQ, V] // AssertTest

Out[126]=
member[PRIMESEQ, V] == True

In[127]:=
member[PRIMESEQ, V] := True
```

Lemma.

```
In[128]:=
SubstTest[implies, member[x, BIJ],
 member[pair[domain[x], range[x]], Q], x → PRIMESEQ]

Out[128]=
member[pair[omega, PRIMES], Q] == True

In[129]:=
% /. Equal → SetDelayed
```

Theorem. The set of primes is countably infinite.

```
In[130]:=
SubstTest[implies, member[pair[x, y], Q],
 equal[card[x], card[y]], {x → omega, y → PRIMES}]

Out[130]=
equal[omega, card[PRIMES]] == True

In[131]:=
card[PRIMES] := omega
```