a recursion relation for the prime sequence

Johan G. F. Belinfante
2010 September 22

```
In[1]:= SetDirectory["l:"]; << goedel.10sep21a
        :Package Title: goedel.10sep21a            2010 September 21 at 4:25 p.m.
        It is now:  2010 Sep 22 at 7:11
        Loading Simplification Rules
        TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3
        weightlimit = 40
```

summary

The function **PRIMESEQ** that lists all primes in increasing order satisfies the following recursion relation:

HULL[PRIMES] ∘ **IMAGE[PRIMESEQ]** ∘ **id[**ω**]** = **PRIMESEQ**.

introduction

The prime sequence **PRIMESEQ** was defined by an iterative construction:

```
In[2]:= iterate[composite[HULL[PRIMES], SUCC], set[succ[set[0]]]]
Out[2]= PRIMESEQ
```

This sequence is a mapping from the set ω of natural numbers to the set **PRIMES** $\subset \omega$ of all primes satisfying two basic iteration rules. The first says that **2 = succ[set[0]]** is the the **0**-th prime, and the second says that at any point in the iteration, the next prime after a given one is the least prime that contains its successor. (One could replace successor with singleton here, and in fact the latter will be used later.)

```
In[3]:= hull[PRIMES, succ[APPLY[PRIMESEQ, nat[x]]]]
Out[3]= APPLY[PRIMESEQ, succ[nat[x]]]
```

The same information can also be expressed in a variable-free fashion:

```
In[4]:= composite[PRIMESEQ, SUCC]
Out[4]= composite[HULL[PRIMES], SUCC, PRIMESEQ]
```

In this notebook, a recursion relation is derived that instead expresses each prime **APPLY[PRIMESEQ, nat[x]]** as the least prime that contains the set **image[PRIMESEQ, nat[x]]** of all preceding primes.

derivation

The set **image[PRIMESEQ, nat[x]]** of all preceding primes is a finite set of natural numbers, and is therefore either empty or has a greatest member. (The set of all finite subsets of ω is **image[inverse[S], ω]**.)

Lemma.

```
In[5]:= SubstTest[implies, member[t, image[inverse[S], omega]],
          or[empty[t], member[U[t], t]], t → image[PRIMESEQ, nat[x]]] // Reverse

Out[5]= or[equal[0, nat[x]], member[U[image[PRIMESEQ, nat[x]]], image[PRIMESEQ, nat[x]]]] == True

In[6]:= (% /. x → x_) /. Equal → SetDelayed
```

A simpler rewrite rule is possible:

Theorem. Once one gets started, the set of primes listed up to any point has a greatest member.

```
In[7]:= equiv[member[U[image[PRIMESEQ, nat[x]]], image[PRIMESEQ, nat[x]]],
         not[equal[0, nat[x]]]]

Out[7]= True

In[8]:= member[U[image[PRIMESEQ, nat[x_]]], image[PRIMESEQ, nat[x_]]] := not[equal[0, nat[x]]]
```

Because the function **PRIMESEQ** is monotone increasing, and each natural number is the set of all lesser numbers, the set of primes listed previously is a subset of the prime being listed at any point.

Lemma.

```
In[9]:= SubstTest[implies, and[member[u, v], subclass[v, w]],
          member[u, w], {u -> U[image[PRIMESEQ, nat[x]]],
           v -> image[PRIMESEQ, nat[x]], w → APPLY[PRIMESEQ, nat[x]]}] // Reverse

Out[9]= or[equal[0, nat[x]], member[U[image[PRIMESEQ, nat[x]]], APPLY[PRIMESEQ, nat[x]]] == True

In[10]:= (% /. x → x_) /. Equal → SetDelayed
```

The literal **equal[0, nat[x]]** in the lemma is redundant, and can be removed.

Theorem. The greatest prime among those previously listed is less than the prime being listed at any point.

```
In[11]:= SubstTest[and, implies[p, q], or[p, q], {p -> equal[0, nat[x]],
           q -> member[U[image[PRIMESEQ, nat[x]]], APPLY[PRIMESEQ, nat[x]]]}]

Out[11]= member[U[image[PRIMESEQ, nat[x]]], APPLY[PRIMESEQ, nat[x]]] == True
```

```
In[12]:= member[U[image[PRIMESEQ, nat[x_]]], APPLY[PRIMESEQ, nat[x_]]] := True
```

A slightly weaker result will actually suffice.

Lemma.

```
In[13]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
           {u -> U[image[PRIMESEQ, nat[x]]], v -> image[PRIMESEQ, nat[x]], w → omega}] // Reverse

Out[13]= or[equal[0, nat[x]], member[U[image[PRIMESEQ, nat[x]]], omega]] == True

In[14]:= (% /. x → x_) /. Equal→ SetDelayed
```

Again, the literal **equal[0, nat[x]]** in the lemma is redundant, and can be removed.

Theorem. The greatest prime previously listed is of course a natural number.

```
In[15]:= SubstTest[and, implies[p, q], or[p, q],
          {p -> equal[0, nat[x]], q -> member[U[image[PRIMESEQ, nat[x]]], omega]}]

Out[15]= member[U[image[PRIMESEQ, nat[x]]], omega] == True

In[16]:= member[U[image[PRIMESEQ, nat[x_]]], omega] := True
```

Membership implies inclusion for natural numbers.

Lermma. The greatest prime listed previously is a subset of the current prime.

```
In[17]:= SubstTest[implies, and[member[u, v], member[v, omega]], subclass[u, v],
           {u -> U[image[PRIMESEQ, nat[x]]], v → APPLY[PRIMESEQ, nat[x]]}] // Reverse

Out[17]= subclass[U[image[PRIMESEQ, nat[x]]], APPLY[PRIMESEQ, nat[x]]] == True

In[18]:= subclass[U[image[PRIMESEQ, nat[x_]]], APPLY[PRIMESEQ, nat[x_]]] := True
```

Lemma. A simplification rule.

```
In[19]:= SubstTest[implies, equal[u, v], equal[hull[PRIMES, succ[u]], hull[PRIMES, succ[v]]],
           {u -> U[image[PRIMESEQ, succ[nat[x]]]], v → APPLY[PRIMESEQ, nat[x]]}] // Reverse

Out[19]= equal[APPLY[PRIMESEQ, succ[nat[x]]], hull[PRIMES,
            succ[union[APPLY[PRIMESEQ, nat[x]], U[image[PRIMESEQ, nat[x]]]]]]] == True

In[20]:= hull[PRIMES, succ[union[APPLY[PRIMESEQ, nat[x_]], U[image[PRIMESEQ, nat[x_]]]]]] :=
          APPLY[PRIMESEQ, succ[nat[x]]]
```

The transitive closure of a finite nonempty set of natural numbers is the successor of its sum class. The next lemma is an application of this idea to the set of previously listed primes.

Lemma.

```
In[21]:= SubstTest[or, equal[0, t], equal[succ[U[t]], tc[t]], not[member[t, FINITE]],
           not[subclass[t, omega]], t → image[PRIMESEQ, nat[x]]] // Reverse

Out[21]= or[equal[0, nat[x]],
           equal[succ[U[image[PRIMESEQ, nat[x]]]], tc[image[PRIMESEQ, nat[x]]]] == True

In[22]:= (% /. x → x_) /. Equal→ SetDelayed
```

A simpler formulation is possible:

Theorem. The transitive closure of the set of previously listed primes at any point is the successor of the largest prime previously listed.

```
In[23]:= equiv[equal[succ[U[image[PRIMESEQ, nat[x]]]], tc[image[PRIMESEQ, nat[x]]]],
          not[equal[0, nat[x]]]]

Out[23]= True

In[24]:= equal[succ[U[image[PRIMESEQ, nat[x_]]]], tc[image[PRIMESEQ, nat[x_]]]] :=
          not[equal[0, nat[x]]]
```

The least prime containing a set also contains its transitive closure: **hull[PRIMES, tc[x]] = hull[PRIMES, x]**.

Theorem. A simplification rule.

```
In[25]:= (SubstTest[implies, equal[tc[t], succ[U[t]]], equal[hull[PRIMES, succ[U[t]]],
              hull[PRIMES, t]], t → image[PRIMESEQ, nat[s]]] // Reverse) /. s → succ[nat[x]]

Out[25]= equal[APPLY[PRIMESEQ, succ[nat[x]]],
           hull[PRIMES, union[image[PRIMESEQ, nat[x]], set[APPLY[PRIMESEQ, nat[x]]]]]] == True

In[26]:= hull[PRIMES, union[image[PRIMESEQ, nat[x_]], set[APPLY[PRIMESEQ, nat[x_]]]]] :=
          APPLY[PRIMESEQ, succ[nat[x]]]
```

Any nonzero natural number is the successor of its predecessor.

Lemma.

```
In[27]:= SubstTest[implies, equal[x, succ[nat[t]]],
           equal[hull[PRIMES, image[PRIMESEQ, x]], APPLY[PRIMESEQ, x]], t → U[nat[x]]] // Reverse

Out[27]= or[equal[0, x], equal[APPLY[PRIMESEQ, x], hull[PRIMES, image[PRIMESEQ, x]]],
           not[member[x, omega]]] == True

In[28]:= (% /. x → x_) /. Equal→ SetDelayed
```

The literal **equal[0, x]** here is redundant, and can be removed.

Theorem. A recursion relation for **PRIMESEQ**.

```
In[29]:= SubstTest[and, implies[p, q], or[p, q],
          {p → empty[x], q → or[equal[APPLY[PRIMESEQ, x], hull[PRIMES, image[PRIMESEQ, x]]],
             not[member[x, omega]]]}]

Out[29]= or[equal[APPLY[PRIMESEQ, x], hull[PRIMES, image[PRIMESEQ, x]]],
           not[member[x, omega]]] == True

In[30]:= or[equal[APPLY[PRIMESEQ, x_], hull[PRIMES, image[PRIMESEQ, x_]]],
           not[member[x_, omega]]] := True
```

To prepare for removing the variables, the following more compact formulation of this is helpful.

Theorem. Restatement of the recursion relation for **PRIMESEQ** using **nat** wrappers.

```
In[31]:= SubstTest[implies, member[t, omega],
           equal[APPLY[PRIMESEQ, t], hull[PRIMES, image[PRIMESEQ, t]]], t → nat[x]] // Reverse

Out[31]= equal[APPLY[PRIMESEQ, nat[x]], hull[PRIMES, image[PRIMESEQ, nat[x]]]] == True

In[32]:= hull[PRIMES, image[PRIMESEQ, nat[x_]]] := APPLY[PRIMESEQ, nat[x]]
```

variable-free formulation

In this section a variable-free formulation of the recursion relation for **PRIMESEQ** is derived using **reify**.

Theorem. At each point of the iteration, the set of previously listed primes is a finite set of natural numbers.

```
In[33]:= Map[empty, SubstTest[reify, x,
           dif[set[image[PRIMESEQ, nat[x]]], t], t -> image[inverse[S], omega]]]

Out[33]= subclass[image[IMAGE[PRIMESEQ], omega], image[inverse[S], omega]] == True

In[34]:= subclass[image[IMAGE[PRIMESEQ], omega], image[inverse[S], omega]] := True
```

Corollary. A simplification rule.

```
In[35]:= equal[intersection[omega,
           complement[image[inverse[IMAGE[PRIMESEQ]], image[inverse[S], omega]]]], 0]

Out[35]= True

In[36]:= intersection[omega,
           complement[image[inverse[IMAGE[PRIMESEQ]], image[inverse[S], omega]]]] := 0
```

Theorem. Variable-free form of the recursion relation.

```
In[37]:= Map[composite[VERTSECT[composite[#, id[omega]]], id[omega]] &,
          SubstTest[reify, x, hull[u, image[v, nat[x]]], {u → PRIMES, v → PRIMESEQ}]]

Out[37]= composite[HULL[PRIMES], IMAGE[PRIMESEQ], id[omega]] == PRIMESEQ
```

In[38]:= **composite[HULL[PRIMES], IMAGE[PRIMESEQ], id[omega]] := PRIMESEQ**