PRIMES

Johan G. F. Belinfante
2005 February 22

```
In[1]:= SetDirectory["i:"]; << goedel66.20a; << tools.m

        :Package Title: goedel66.20a          2005 February 20 at 9:15 p.m.

        It is now:  2005 Feb 22 at 10:40

        Loading Simplification Rules

        TOOLS.M                       Revised 2005 February 13

        weightlimit = 40
```

summary

The class **PRIMES** of prime numbers is defined in this notebook by the condition that its members have exactly two divisors. This definition at first appears to complicate matters by making counting a prerequisite, albeit that one only needs to count up to two. The main advantage of this definition is its conciseness. Various other conditions commonly imposed can be proved as theorems. For example, sets which are not natural numbers have no divisors, and therefore primes must be natural numbers. Since every number divides zero, the number of its divisors is infinite, and thus zero is not a prime. The number one has only one divisior, namely itself, and therefore one is not a prime.

A novelty in this notebook is that an equational definition is given for the class of primes, and membership rules are deduced from this equation instead of the other way around. The membership rule is turned around to eliminate the definition, but a rewrite rule for **image[V, intersection[PRIMES, set[x]]]** is derived, which permits one to access the membership rule indirectly via **assert**. The connection of such a rule with membership is evident from the following general observation:

```
In[2]:= class[w, member[x, y]]

Out[2]= image[V, intersection[y, set[x]]]
```

the definition of PRIMES

The equational definition of the class **PRIMES** is given in the form of a rewrite rule:

```
In[3]:= image[inverse[VERTSECT[inverse[DIV]]], image[PAIRSET, Di]] := PRIMES
```

primes are numbers

The fact that non-numbers have no divisors implies this (temporary) lemma which will later be replaced with a more definitive rewrite rule.

```
In[4]:= SubstTest[implies, equal[w, 0],
         not[member[w, image[PAIRSET, Di]]], w → image[inverse[DIV], set[x]]]

Out[4]= or[member[x, omega],
          not[member[image[inverse[DIV], set[x]], image[PAIRSET, Di]]]] == True

In[5]:= (% /. x → x_) /. Equal → SetDelayed
```

It follows immediately that primes are natural numbers:

```
In[6]:= Map[implies[#, member[x, omega]] &, SubstTest[member, x,
           image[inverse[VERTSECT[inverse[DIV]]], y], y -> image[PAIRSET, Di]]]

Out[6]= or[member[x, omega], not[member[x, PRIMES]]] == True

In[7]:= (% /. x → x_) /. Equal → SetDelayed
```

A variable-free form of this can be derived which makes the above rule superfluous.

```
In[8]:= Map[equal[0, #] &, dif[PRIMES, omega] // Renormality]

Out[8]= subclass[PRIMES, omega] == True

In[9]:= subclass[PRIMES, omega] := True
```

Corollary. The class of primes is a set.

```
In[10]:= SubstTest[implies, and[subclass[u, v], member[v, V]],
          member[u, V], {u → PRIMES, v → omega}]

Out[10]= member[PRIMES, V] == True

In[11]:= member[PRIMES, V] := True
```

a sethood lemma

In this section a temporary lemma is derived that follows as an immediate corollary of the theorem that primes are numbers.

```
In[12]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> member[image[inverse[DIV], set[x]], image[PAIRSET, Di]],
            p2 → member[x, omega], p3 → member[x, V]}]]

Out[12]= or[member[x, V],
           not[member[image[inverse[DIV], set[x]], image[PAIRSET, Di]]]] == True

In[13]:= (% /. x → x_) /. Equal → SetDelayed
```

The following observation is made into a temporary rewrite rule:

```
In[14]:= equiv[
          and[member[x, V], member[image[inverse[DIV], set[x]], image[PAIRSET, Di]]],
          member[image[inverse[DIV], set[x]], image[PAIRSET, Di]]]

Out[14]= True

In[15]:= and[member[x_, V],
           member[image[inverse[DIV], set[x_]], image[PAIRSET, Di]]] :=
          member[image[inverse[DIV], set[x]], image[PAIRSET, Di]]
```

This yields a membership rule for **PRIMES** which is here oriented to eliminate rather than introduce the definition. This makes it easier to make statements about primes.

```
In[16]:= SubstTest[member, x, image[inverse[VERTSECT[setpart[z]]], y],
           {y -> image[PAIRSET, Di], z → inverse[DIV]}] // Reverse

Out[16]= member[image[inverse[DIV], set[x]], image[PAIRSET, Di]] == member[x, PRIMES]

In[17]:= member[image[inverse[DIV], set[x_]], image[PAIRSET, Di]] := member[x, PRIMES]
```

indirect access to the membership rule

Since the membership rule for **PRIMES** eliminates the definition, one needs another mechanism to access the membership rule which introduces the definition on demand. This is provided by the following rule, which provides indirect accesss.

```
In[18]:= image[V, intersection[image[PAIRSET, Di],
             set[image[thinpart[inverse[DIV]], set[x]]]]] // Renormality // Reverse

Out[18]= image[V, intersection[PRIMES, set[x]]] ==
          image[V, intersection[image[PAIRSET, Di], set[image[inverse[DIV], set[x]]]]]

In[19]:= image[V, intersection[PRIMES, set[x_]]] :=
          image[V, intersection[image[PAIRSET, Di], set[image[inverse[DIV], set[x]]]]]
```

As an example of such indirect access, the facts that zero and one are not primes are derived using **AssertTest**. Zero is not a prme because it has infinitely many divisors.

```
In[20]:= member[0, PRIMES] // AssertTest

Out[20]= member[0, PRIMES] == False

In[21]:= member[0, PRIMES] := False
```

One is not a prime because it only has one divisor.

```
In[22]:= member[set[0], PRIMES] // AssertTest

Out[22]= member[set[0], PRIMES] == False

In[23]:= member[set[0], PRIMES] := False
```

On the other hand, the number two is a prime because it is known to have exactly two divisors.

```
In[24]:= member[succ[set[0]], PRIMES] // AssertTest

Out[24]= member[succ[set[0]], PRIMES] == True

In[25]:= member[succ[set[0]], PRIMES] := True
```

Corollary.

```
In[26]:= intersection[succ[set[0]], PRIMES] // Normality

Out[26]= intersection[PRIMES, succ[set[0]]] == 0

In[27]:= intersection[PRIMES, succ[set[0]]] := 0
```

---

two is the least prime

Lemma: Zero is less than the least prime.

```
In[28]:= SubstTest[implies, subclass[u, v],
          subclass[A[v], A[u]], {u -> PRIMES, v -> dif[omega, set[0]]}]

Out[28]= member[0, A[PRIMES]] == True

In[29]:= % /. Equal → SetDelayed
```

Lemma. One is less than the least prime.

```
In[30]:= SubstTest[implies, subclass[u, v], subclass[A[v], A[u]],
          {u -> PRIMES, v -> dif[omega, succ[set[0]]]}]

Out[30]= member[set[0], A[PRIMES]] == True

In[31]:= % /. Equal → SetDelayed
```

Lemma. The least prime is no less than two.

```
In[32]:= SubstTest[implies, member[u, v],
          subclass[A[v], u], {u → succ[set[0]], v → PRIMES}]

Out[32]= subclass[A[PRIMES], succ[set[0]]] == True

In[33]:= % /. Equal → SetDelayed
```

Theorem. Two is the least prime.

```
In[34]:= SubstTest[and, subclass[u, v],
          subclass[v, u], {u -> A[PRIMES], v → succ[set[0]]}]

Out[34]= True == equal[A[PRIMES], succ[set[0]]]

In[35]:= A[PRIMES] := succ[set[0]]
```

The indicated interpretation of this rewrite rule depends on the following fact which requires no new rewrite rule.

```
In[36]:= member[A[PRIMES], PRIMES]

Out[36]= True
```

---

another characterization of primes

Temporary sethood lemma.

```
In[37]:= SubstTest[implies, equal[y, 0],
          not[equal[image[inverse[DIV], y], union[y, set[set[0]]]]], y → set[x]]

Out[37]= or[member[x, V],
           not[equal[image[inverse[DIV], set[x]], set[x, set[0]]]] == True

In[38]:= (% /. x → x_) /. Equal → SetDelayed
```

The following corollary is made a temporary rewrite rule:

```
In[39]:= equiv[or[not[equal[image[inverse[DIV], set[x]], set[x, set[0]]]],
           not[member[x, V]]], not[equal[image[inverse[DIV], set[x]], set[x, set[0]]]]]

Out[39]= True

In[40]:= or[not[equal[image[inverse[DIV], set[x_]], set[x_, set[0]]]],
           not[member[x_, V]]] :=
          not[equal[image[inverse[DIV], set[x]], set[x, set[0]]]]
```

Corollary. If the only divisors of **x** are **1** and **x**, then **x** is **1** or a prime.

```
In[41]:= SubstTest[implies, and[equal[u, v], member[v, w]],
           member[u, w], {u -> image[inverse[DIV], set[x]],
            v -> set[set[0], x], w → image[PAIRSET, Di]}] // MapNotNot

Out[41]= or[equal[x, set[0]], member[x, PRIMES],
           not[equal[image[inverse[DIV], set[x]], set[x, set[0]]]] == True

In[42]:= (% /. x → x_) /. Equal → SetDelayed
```

Since **1** and **x** are always divisors of any number **x**, then assuming **x** is not equal to **1**, this set of two divisors must exhaust the set of divisors when **x** is a prime.

```
In[43]:= SubstTest[implies, and[subclass[u, v],
           equal[card[u], succ[set[0]]], equal[card[v], succ[set[0]]]],
          equal[u, v], {u → set[x, set[0]], v → image[inverse[DIV], set[x]]}]

Out[43]= or[equal[x, set[0]], equal[image[inverse[DIV], set[x]], set[x, set[0]]],
           not[member[x, PRIMES]]] == True

In[44]:= (% /. x → x_) /. Equal → SetDelayed
```

One can improve upon this since **1** is not a prime.

```
In[45]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3],
           not[implies[p1, p3]], {p1 → member[x, PRIMES], p2 → not[equal[x, set[0]]],
            p3 -> equal[image[inverse[DIV], set[x]], set[x, set[0]]]}]]

Out[45]= or[equal[image[inverse[DIV], set[x]], set[x, set[0]]],
           not[member[x, PRIMES]]] == True

In[46]:= (% /. x → x_) /. Equal → SetDelayed
```

Combining these results, one obtains a logical equivalence that can be made into a rewrite rule.

```
In[47]:= equiv[equal[image[inverse[DIV], set[x]], set[x, set[0]]],
            or[equal[x, set[0]], member[x, PRIMES]]] // not // not

Out[47]= True

In[48]:= equal[image[inverse[DIV], set[x_]], set[x_, set[0]]] :=
          or[equal[x, set[0]], member[x, PRIMES]]
```

---

a formula for the set of composite numbers

The following rule is needed to avoid having to add a separate rewrite rules for variants involving the fixed point set of the inverse relation.

```
In[49]:= fix[composite[Di, id[x], inverse[DIV]]] // InvertFixTest

Out[49]= fix[composite[Di, id[x], inverse[DIV]]] == fix[composite[DIV, id[x], Di]]

In[50]:= fix[composite[Di, id[x_], inverse[DIV]]] := fix[composite[DIV, id[x], Di]]
```

The following variable-free formulation of the result derived in the preceding section characterizes a prime **x** as a natural number not equal to **1** and having no divisors other than **1** and **x**.

```
In[51]:= SubstTest[class, x, equal[image[u, set[x]], set[x, v]],
           {u → inverse[DIV], v → set[0]}] // Reverse

Out[51]= intersection[omega,
           complement[fix[composite[DIV, id[complement[set[set[0]]]], Di]]]] ==
         union[PRIMES, set[set[0]]]

In[52]:= % /. Equal → SetDelayed
```

Corollary.

```
In[53]:= SubstTest[equal, 0, dif[u, v], {u -> omega,
           v -> union[PRIMES, fix[composite[DIV, id[complement[set[set[0]]]], Di]],
             set[set[0]]]}] // Reverse

Out[53]= subclass[omega, union[PRIMES,
           fix[composite[DIV, id[complement[set[set[0]]]], Di]], set[set[0]]]] == True

In[54]:= % /. Equal → SetDelayed
```

Lemma.

```
In[55]:= member[set[0], fix[composite[DIV, id[complement[set[set[0]]]], Di]]] //
          AssertTest

Out[55]= member[set[0], fix[composite[DIV, id[complement[set[set[0]]]], Di]]] == False

In[56]:= % /. Equal → SetDelayed
```

Corollary.

```
In[57]:= SubstTest[subclass, intersection[u, v], v, {u -> omega,
           v -> complement[fix[composite[DIV, id[complement[set[set[0]]]], Di]]]}]

Out[57]= equal[0, intersection[PRIMES,
           fix[composite[DIV, id[complement[set[set[0]]]], Di]]]] == True

In[58]:= % /. Equal → SetDelayed
```

Putting all these results together yields a permanent rewrite rule for the set of composite numbers.

```
In[59]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> fix[composite[DIV, id[complement[set[set[0]]]], Di]],
           v -> intersection[omega, complement[PRIMES], complement[set[set[0]]]]}]

Out[59]= True == equal[fix[composite[DIV, id[complement[set[set[0]]]], Di]],
           intersection[omega, complement[PRIMES], complement[set[set[0]]]]]

In[60]:= fix[composite[DIV, id[complement[set[set[0]]]], Di]] :=
          intersection[omega, complement[PRIMES], complement[set[set[0]]]]
```