APPLY[u,x] ∈ APPLY[v,y]

Johan G. F. Belinfante
2010 March 15

```
In[1]:= SetDirectory["l:"]; << goedel.10mar13a;<< tools.m

        :Package Title: goedel.10mar13a               2010 March 13 at 10:00 p.m.

        It is now:  2010 Mar 15 at 3:28

        Loading Simplification Rules

        TOOLS.M                                    Revised 2010 February 26

        weightlimit = 40
```

summary

The infinite sequence of primes listed in increasing order is **PRIMESEQ**. This is a function that maps each natural number **n** to the **n**-th prime **APPLY[PRIMESEQ, n]**, starting with **APPLY[PRIMESEQ, 0] = 2**. The function **PRIMESEQ** is strictly monotone: if **m < n**, then **APPLY[PRIMESEQ, m]** < **APPLY[PRIMESEQ, n]**. When one is dealing with natural numbers, or more generally with ordinal numbers, as constructed by von Neumann, the membership statement **x ∈ y** has the interpretation of strict inequality **x < y**.

```
In[2]:= implies[and[member[x, y], member[y, omega]],
          member[APPLY[PRIMESEQ, x], APPLY[PRIMESEQ, y]]]

Out[2]= True
```

This strict monotonicity of the function **PRIMESEQ** is expressed by this variabke-free condition:

```
In[3]:= subclass[PRIMESEQ, compositeS, IMAGE[PRIMESEQ]]]

Out[3]= True
```

This notebook is concerned with interpreting in general the condition **v ⊂ S ∘ IMAGE[u]** when **u** and **v** are both functions, not necessarily the same function. By introducing two extra variables **x** and **y**, one can interpret this condition as a sort of generalized form of strict monotonicity with respect to the membership relation: if **x ∈ y ⋂ domain[u]**, then **APPLY[u, x] ∈ APPLY[v, y]**. These results are stated twice, once using **funpart** wrappers for the functions, and then again using **FUNCTION** predicates. With **funpart** wrappers, no special rewrite rule is needed for the special case **u = v**. When the **FUNCTION** predicate is used, however, the special case **u = v** does require a separate rewrite rule.

derivation

Lemma.

```
In[6]:= SubstTest[implies, and[member[r, s], subclass[s, t]],
          member[r, t], {r → pair[y, APPLY[funpart[v], y]],
           s → funpart[v], t → composite[S, IMAGE[funpart[u]]]}] // Reverse

Out[6]= or[not[member[y, domain[funpart[v]]]],
          not[subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]],
          subclass[image[funpart[u], y], APPLY[funpart[v], y]]] == True

In[7]:= (% /. {u → u_, v → v_, y → y_}) /. Equal → SetDelayed
```

Lemma. When **y** is not in the domain of **funpart[v]**, one has **APPLY[funpart[v], y] = V**.

```
In[8]:= SubstTest[implies, and[member[t, V], equal[V, w]], member[t, w],
          {t → APPLY[funpart[u], x], w → APPLY[funpart[v], y]}] // Reverse

Out[8]= or[member[y, domain[funpart[v]]], member[APPLY[funpart[u], x], APPLY[funpart[v], y]],
          not[member[x, domain[funpart[u]]]]] == True

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

Main Theorem. Interpreting the condition **funpart[v] ⊂ S ∘ IMAGE[funpart[u]]**.

```
In[11]:= Map[not, SubstTest[and, implies[and[p2, p4], p5], implies[and[p1, p3], p6],
           implies[and[p5, p6], p7], implies[and[not[p2], p1], p7],
           not[implies[and[p1, p3, p4], p7]], {p1 -> member[x, domain[funpart[u]]],
            p2 → member[y, domain[funpart[v]]], p3 → member[x, y],
            p4 -> subclass[funpart[v], composite[S, IMAGE[funpart[u]]]],
            p5 -> subclass[image[funpart[u], y], APPLY[funpart[v], y]],
            p6 -> member[APPLY[funpart[u], x], image[funpart[u], y]],
            p7 -> member[APPLY[funpart[u], x], APPLY[funpart[v], y]]}]] // Reverse

Out[11]= or[member[APPLY[funpart[u], x], APPLY[funpart[v], y]],
           not[member[x, y]], not[member[x, domain[funpart[u]]]],
           not[subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]]] == True

In[13]:= or[member[APPLY[funpart[u_], x_], APPLY[funpart[v_], y_]],
           not[member[x_, domain[funpart[u_]]]], not[member[x_, y_]],
           not[subclass[funpart[v_], composite[S, IMAGE[funpart[u_]]]]]] := True
```

Corollary. Restatement of the main theorem with **FUNCTION** predicates in place of **funpart** wrappers.

```
In[14]:= SubstTest[implies, and[equal[u, funpart[s]], equal[v, funpart[t]]],
           or[member[APPLY[u, x], APPLY[v, y]], not[member[x, y]], not[member[x, domain[u]]],
            not[subclass[v, composite[S, IMAGE[u]]]]], {s → u, t → v}] // Reverse

Out[14]= or[member[APPLY[u, x], APPLY[v, y]], not[FUNCTION[u]],
           not[FUNCTION[v]], not[member[x, y]], not[member[x, domain[u]]],
           not[subclass[v, composite[S, IMAGE[u]]]]] == True

In[16]:= or[member[APPLY[u_, x_], APPLY[v_, y_]],
           not[FUNCTION[u_]], not[FUNCTION[v_]], not[member[x_, domain[u_]]],
           not[member[x_, y_]], not[subclass[v_, composite[S, IMAGE[u_]]]]] := True
```

Corollary. When using the **FUNCTION** predicate, a separate rewrite rule is needed for the special case **u = v**.

```
In[18]:= SubstTest[or, member[APPLY[u, x], APPLY[v, y]], not[FUNCTION[u]],
           not[FUNCTION[v]], not[member[x, y]], not[member[x, domain[u]]],
           not[subclass[v, composite[S, IMAGE[u]]]], v → u] // Reverse

Out[18]= or[member[APPLY[u, x], APPLY[u, y]], not[FUNCTION[u]], not[member[x, y]],
           not[member[x, domain[u]]], not[subclass[u, composite[S, IMAGE[u]]]]] == True

In[20]:= or[member[APPLY[u_, x_], APPLY[u_, y_]],
           not[FUNCTION[u_]], not[member[x_, domain[u_]]], not[member[x_, y_]],
           not[subclass[u_, composite[S, IMAGE[u_]]]]] := True
```

an example

The function **ordlist[t]** which lists the first countably many ordinals in the class **t** is an example of a strictly monotone function:

```
In[23]:= subclass[ordlist[t], composite[S, IMAGE[ordlist[t]]]]

Out[23]= True
```

Theorem. Strict monotonicity for the function **ordlist[t]**.

```
In[26]:= SubstTest[or, member[APPLY[u, x], APPLY[u, y]],
           not[FUNCTION[u]], not[member[x, y]], not[member[x, domain[u]]],
           not[subclass[u, composite[S, IMAGE[u]]]], u → ordlist[t]] // Reverse

Out[26]= or[member[APPLY[ordlist[t], x], APPLY[ordlist[t], y]],
           not[member[x, y]], not[member[x, domain[ordlist[t]]]]] == True
```

interpretation using the membership relation E

In this section it is shown that for any function **x**, the conditions **x ⊂ S ∘ IMAGE[x]** and **x ∘ E ∘ inverse[x] ⊂ E** are equivalent. The main idea is to introduce a **funpart** wrapper, which causes the condition **x ∘ E ∘ inverse[x] ⊂ E** to be automatically rewritten.

Lemma.

```
In[31]:= SubstTest[implies, subclass[x, y],
           subclass[composite[w, x], composite[w, y]], {w → funpart[v], x → Id,
            y → composite[inverse[IMAGE[funpart[u]]], IMAGE[funpart[u]]]}] // Reverse

Out[31]= subclass[funpart[v],
           composite[funpart[v], inverse[IMAGE[funpart[u]]], IMAGE[funpart[u]]]] == True

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

Lemma.

```
In[44]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 -> subclass[composite[funpart[v], inverse[IMAGE[funpart[u]]]], S],
             p2 → subclass[composite[funpart[v], inverse[IMAGE[funpart[u]]],
                IMAGE[funpart[u]]], composite[S, IMAGE[funpart[u]]]],
             p3 -> subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]}]] // Reverse

Out[44]= or[not[subclass[composite[funpart[v], inverse[IMAGE[funpart[u]]]], S]],
           subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]] == True

In[45]:= (% /. {u → u_, v → v_}) /. Equal → SetDelayed

In[33]:= implies[subclass[composite[funpart[v], E, inverse[funpart[u]]], E],
            subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]]

Out[33]= or[not[subclass[composite[funpart[v], inverse[IMAGE[funpart[u]]]], S]],
            subclass[funpart[v], composite[S, IMAGE[funpart[u]]]]]
```

Theorem. If **x** is a function, then $\mathbf{x} \circ \mathbf{E} \circ \mathbf{inverse[x]} \subset \mathbf{E}$ implies $\mathbf{x} \subset \mathbf{S} \circ \mathbf{IMAGE[x]}$.

```
In[48]:= (SubstTest[implies, and[equal[x, funpart[u]],
              equal[y, funpart[v]], subclass[composite[y, E, inverse[x]], E]],
             subclass[y, composite[S, IMAGE[x]]], {u → x, v → y}] // Reverse) /. y → x

Out[48]= or[not[FUNCTION[x]], not[subclass[composite[x, E, inverse[x]], E]],
           subclass[x, composite[S, IMAGE[x]]]] == True

In[49]:= or[not[FUNCTION[x_]], not[subclass[composite[x_, E, inverse[x_]], E]],
           subclass[x_, composite[S, IMAGE[x_]]]] := True
```

Lemma. If **x** is a function, then $\mathbf{x} \subset \mathbf{S} \circ \mathbf{IMAGE[x]}$ implies $\mathbf{x} \circ \mathbf{E} \circ \mathbf{inverse[x]} \subset \mathbf{E}$.

```
In[53]:= SubstTest[implies, subclass[u, v],
           subclass[composite[u, w], composite[v, w]], {u → funpart[x],
            v -> composite[S, IMAGE[funpart[x]]], w → inverse[IMAGE[funpart[x]]]}] // Reverse

Out[53]= or[not[subclass[funpart[x], composite[S, IMAGE[funpart[x]]]]],
           subclass[composite[funpart[x], inverse[IMAGE[funpart[x]]], S]] == True

In[54]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[59]:= SubstTest[implies, and[equal[x, funpart[t]], subclass[x, composite[S, IMAGE[x]]]],
           subclass[composite[x, E, inverse[x]], E], t → x] // Reverse

Out[59]= or[not[FUNCTION[x]], not[subclass[x, composite[S, IMAGE[x]]]],
           subclass[composite[x, E, inverse[x]], E]] == True

In[60]:= or[not[FUNCTION[x_]], not[subclass[x_, composite[S, IMAGE[x_]]]],
           subclass[composite[x_, E, inverse[x_]], E]] := True
```

comment

Among the named classes, the function **RANK** is another example of a function that satisfies **x ⊂ S ∘ IMAGE[x]**. For this function, the interpretation of this condition is this:

```
In[69]:= implies[and[member[x, y], member[x, REGULAR]], member[rank[x], rank[y]]]

Out[69]= True
```

Note that **RANK** satisfies the condition considered in the last section.

```
In[72]:= subclass[composite[RANK, E, inverse[RANK]], E]

Out[72]= True
```

Both **PRIMESEQ** and **RANK** actually satisfy the following stronger condition

```
In[79]:= subcommute[x, E]

Out[79]= subclass[composite[x, E], composite[E, x]]
```

This stronger condition holds because in both cases, their domains are full.

Theorem. If **x** subcommutes with the membership relation **E**, then **domain[x]** is full.

```
In[81]:= SubstTest[implies, subclass[u, v], subclass[domain[u], domain[v]],
           {u → composite[x, E], v → composite[E, x]}] // Reverse

Out[81]= or[not[subclass[composite[x, E], composite[E, x]]],
           subclass[U[domain[x]], domain[x]]] == True

In[82]:= or[not[subclass[composite[x_, E], composite[E, x_]]],
           subclass[U[domain[x_]], domain[x_]]] := True
```

Theorem. If a function **x** satisfies **x ∘ E ∘ inverse[x] ⊂ E** and if its domain is full, then it subcommutes with **E**.

In[86]:=
```
SubstTest[implies, equal[x, funpart[t]],
  implies[and[subclass[composite[x, E, inverse[x]], E], full[domain[x]]],
   subcommute[x, E]], t → x] // Reverse
```

Out[86]=
```
or[not[FUNCTION[x]], not[subclass[composite[x, E, inverse[x]], E]],
  not[subclass[U[domain[x]], domain[x]]],
  subclass[composite[x, E], composite[E, x]]] == True
```

In[87]:=
```
or[not[FUNCTION[x_]], not[subclass[composite[x_, E, inverse[x_]], E]],
  not[subclass[U[domain[x_]], domain[x_]]],
  subclass[composite[x_, E], composite[E, x_]]] := True
```