there is no maximal function

Johan G. F. Belinfante
2007 November 6

```
In[1]:= SetDirectory["l:"]; << goedel99.04a; << tools.m

        :Package Title: goedel99.04a         2007 November 4 at 10:30 a.m.

        It is now:  2007 Nov 6 at 13:11

        Loading Simplification Rules

        TOOLS.M                                        Revised 2007 September 19

        weightlimit = 40
```

summary

Every function can be extended by adding another point to its domain, and therefore the class **FUNS** of all functions is subvariant under the inverse cover relation **inverse[K]**. It follows, in particular, that there can be no maximal function. These results are hardly surprising, but the techniques used to derive them are of interest because very little explicit reasoning needs to be done. Most of the work is accomplished automatically, relying on existing rewrite rules to simplify various expressions that arise.

derivation

The starting point for the derivation of the theorems below is this basic observation: If one adjoins to a function **x** a new point **y** whose first coordinate is not in the domain of the function, one obtains a bigger function. It is convenient here to eliminate **FUNCTION** literals by using **funpart** wrappers and to avoid explicit ordered pairs by using **first**. The use of **AssertTest** suffices to establish this basic observation:

```
In[2]:= or[FUNCTION[union[funpart[x], set[y]]],
          member[first[y], domain[funpart[x]]], not[member[first[y], V]]] // AssertTest

Out[2]= or[FUNCTION[union[funpart[x], set[y]]],
          member[first[y], domain[funpart[x]]], not[member[first[y], V]]] == True

In[3]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Both of the set variables **x** and **y** in the lemma can now be eliminated, yielding this variable-free reformulation:

```
In[4]:= Map[empty[composite[Id, complement[#]]] &,
         SubstTest[class, pair[x, y], or[subclass[P[union[funpart[x], set[y]]], z],
           member[first[y], domain[funpart[x]]], not[member[first[y], V]]], z → FUNS]]

Out[4]= subclass[image[CUP, composite[id[FUNS], inverse[IMAGE[FIRST]],
            complement[E], FIRST, inverse[SINGLETON]]], FUNS] == True

In[5]:= % /. Equal → SetDelayed
```

One can view this as the statement that the class **FUNS** is invariant under the following relation:

```
In[6]:= abstract[x, image[CUP,
          composite[id[x], inverse[IMAGE[FIRST]], complement[E], FIRST, inverse[SINGLETON]]]]

Out[6]= composite[CUP,
         id[composite[SINGLETON, inverse[FIRST], complement[inverse[E]], IMAGE[FIRST]]],
         inverse[FIRST]]
```

Lemma. The above relation is contained in the cover relation **K**.

```
In[7]:= Map[empty,
         dif[composite[CUP, id[composite[SINGLETON, inverse[FIRST], complement[inverse[E]],
              IMAGE[FIRST]]], inverse[FIRST]], K] // VSRenormality]

Out[7]= subclass[composite[CUP,
           id[composite[SINGLETON, inverse[FIRST], complement[inverse[E]], IMAGE[FIRST]]],
           inverse[FIRST]], K] == True

In[8]:= % /. Equal → SetDelayed
```

It will be shown that the class **FUNS** is also subvariant under **inverse[K]**. To do this, some lemmas are needed. The first of these lemmas is a technical one, needed to simplify an expression that comes up when **AssertTest** is applied in the next lemma. The needed simplification rule is obtained automatically by the use of **Normality**:

```
In[9]:= Map[domain,
         image[inverse[SINGLETON], image[inverse[ADJOIN[funpart[x]]], FUNS]] // Normality]

Out[9]= domain[image[inverse[SINGLETON], image[inverse[ADJOIN[funpart[x]]], FUNS]]] ==
         image[V, set[domain[funpart[x]]]]

In[10]:= domain[image[inverse[SINGLETON], image[inverse[ADJOIN[funpart[x_]]], FUNS]]] :=
          image[V, set[domain[funpart[x]]]]
```

The next lemma also simplifies an expression that arises in the statement of the subvariance result that is to be derived. It is also settled automatically, using **AssertTest** and a **funpart** wrapper:

```
In[11]:= (member[t, fix[y]] // AssertTest) /.
          {t → funpart[x], y :> composite[inverse[IMAGE[FIRST]],
             complement[E], FIRST, inverse[SINGLETON], image[inverse[CUP], FUNS]]}

Out[11]= member[funpart[x], fix[composite[inverse[IMAGE[FIRST]], complement[E], FIRST,
            inverse[SINGLETON], image[inverse[CUP], FUNS]]]] == member[funpart[x], V]
```

```
In[12]:= member[funpart[x_], fix[composite[inverse[IMAGE[FIRST]], complement[E], FIRST,
           inverse[SINGLETON], image[inverse[CUP], FUNS]]]] := member[funpart[x], V]
```

The **funpart** wrapper is now removed in the standard way:

```
In[13]:= SubstTest[implies, and[equal[x, funpart[t]], member[x, V]],
           member[x, fix[composite[inverse[IMAGE[FIRST]], complement[E], FIRST,
             inverse[SINGLETON], image[inverse[CUP], FUNS]]], t → x] // Reverse

Out[13]= or[member[x, fix[composite[inverse[IMAGE[FIRST]],
             complement[E], FIRST, inverse[SINGLETON], image[inverse[CUP], FUNS]]]],
           not[FUNCTION[x]], not[member[x, V]]] == True

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

At this point the set variable **x** can be eliminated to obtain a variable-free reformulation of this result.

```
In[15]:= Map[equal[V, #] &, SubstTest[class, x, implies[member[x, u], member[x, v]],
           {u → FUNS, v -> fix[composite[inverse[IMAGE[FIRST]], complement[E],
              FIRST, inverse[SINGLETON], image[inverse[CUP], FUNS]]]}]]

Out[15]= subclass[FUNS, fix[composite[inverse[IMAGE[FIRST]], complement[E],
             FIRST, inverse[SINGLETON], image[inverse[CUP], FUNS]]]] == True

In[16]:= % /. Equal → SetDelayed
```

Restatement. The class **FUNS** is subvariant under the inverse of the relation of extending its domain by adding a single new point.

```
In[18]:= subvariant[inverse[t], FUNS] /.
           t -> composite[CUP, id[composite[SINGLETON, inverse[FIRST],
             complement[inverse[E]], IMAGE[FIRST]]], inverse[FIRST]]

Out[18]= True
```

Corollary. The class **FUNS** is subvariant under **inverse[K]**.

```
In[19]:= SubstTest[implies, and[subclass[u, v], subvariant[u, w]], subvariant[v, w],
           {u → inverse[composite[CUP,
              id[composite[SINGLETON, inverse[FIRST], complement[inverse[E]], IMAGE[FIRST]]],
              inverse[FIRST]]], v → inverse[K], w → FUNS}] // Reverse

Out[19]= subclass[FUNS, image[inverse[K], FUNS]] == True

In[20]:= % /. Equal → SetDelayed
```

Corollary. The class **FUNS** is subvariant under **inverse[PS]**. That is, this class has no maximal elements.

```
In[21]:= SubstTest[implies, and[subclass[u, v], subvariant[u, w]], subvariant[v, w],
           {u → inverse[K], v → inverse[PS], w → FUNS}] // Reverse

Out[21]= subclass[FUNS, image[inverse[PS], FUNS]] == True
```

```
In[22]:= % /. Equal → SetDelayed
```

The above subvariance statements can be conveniently sharpened to equations for inverse images, which can then be added as new rewrite rules. For the inverse image under the proper subset relation, this is accomplished as follows:

```
In[23]:= SubstTest[equal, image[inverse[PS], t], image[inverse[S], t], t → FUNS]
Out[23]= True == equal[FUNS, image[inverse[PS], FUNS]]

In[24]:= image[inverse[PS], FUNS] := FUNS
```

Corollary. Every function can be covered by a larger one.

```
In[25]:= IminComp[S, K, FUNS] // Reverse
Out[25]= image[inverse[K], FUNS] == FUNS

In[26]:= image[inverse[K], FUNS] := FUNS
```