VS and inverse images of functions: a new derivation of Theorem FIN−FS

Johan G. F. Belinfante
2004 January 4

```
In[1]:= << goedel53.03c; << tools.m

        :Package Title: goedel53.03c      2004 January 3 at 9:05 p.m.

        It is now:  2004 Jan 9 at 14:17

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 January 3

        weightlimit = 40
```

summary

The main theorem derived in this notebook is that functions with finite domains have finite ranges. This fact was the subject of Theorem **FIN−FS** proved using **Otter** in 2000 from first principles, using only the concept of finiteness, defined in terms of **subvar[PS]**. That proof did not use not the equipollence relation **Q** nor various other constructs featured in this notebook. Although the present derivation is much more roundabout, it does introduce many new fascinating and potentially useful rewrite rules, and therefore is mainly interesting from a methodological viewpoint.

```
In[2]:= "Belinfante, J. G. F., The Unifying Concept of Subvariance, in:
            FTP 2000, Third International Workshop on First-Order Theorem Proving,
            St. Andrews, Scotland, edited by P. Baumgartner and  H. Zhang, pp. 56-67,
         Fachberichte Informatik, Universität  Koblenz-Landau, 2000.";
```

The function **VS** assigns to each set **x** the restriction of **VERTSECT[x]** to the domain of **x**. In this notebook a formula is derived that says that for any function, there is a one−to−one correspondence between the values of the function and the set of arguments that produce that value. In other words, the range of a function is equipollent to a subset of the power set of its domain. It follows from this that if the domain is finite, then the range must also be finite.

review of the functions VERTSECT[x] and VS

The function **VERTSECT[x]** assigns to each set **y** the vertical section of **x** at **y**.

```
In[3]:= lambda[y, image[x, singleton[y]]]

Out[3]= VERTSECT[x]
```

The domain of a class **x** is the set of points at which the vertical section is not empty:

```
In[4]:= composite[id[complement[singleton[0]]], VERTSECT[x]]

Out[4]= composite[VERTSECT[x], id[domain[x]]]
```

If all vertical sections are sets, then the union of all its vertical sections is the range of **x**.

```
In[5]:= implies[thin[x], equal[U[image[VERTSECT[x], domain[x]]], range[x]]]
Out[5]= True
```

Although the function **VERTSECT[x]** itself is a proper class even when **x** is a set, the restriction of it to the domain of **x** is a set. Consequently, one can introduce a function **VS** which takes these functions as values:

```
In[6]:= APPLY[VS, x]
Out[6]= union[complement[image[V, singleton[x]]], composite[VERTSECT[x], id[domain[x]]]]
```

The function **VS** is total, and produces all functions which do not have **0** as a value.

```
In[7]:= domain[VS]
Out[7]= V

In[8]:= range[VS]
Out[8]= intersection[FUNS, P[cart[V, complement[singleton[0]]]]]
```

simplification rules for VS

Some basic simplification rules for **VS** are derived in this section.

```
In[9]:= SubstTest[subclass, x, image[inverse[VS], z], z -> FUNS] // Reverse
Out[9]= subclass[image[VS, x], FUNS] == True

In[10]:= subclass[image[VS, x_], FUNS] := True

In[11]:= Assoc[id[FUNS], id[range[VS]], VS]
Out[11]= composite[id[FUNS], VS] == VS

In[12]:= composite[id[FUNS], VS] := VS

In[13]:= ImageComp[id[FUNS], VS, x] // Reverse
Out[13]= intersection[FUNS, image[VS, x]] == image[VS, x]

In[14]:= intersection[FUNS, image[VS, x_]] := image[VS, x]

In[15]:= Assoc[id[P[cart[V, V]]], id[FUNS], VS]
Out[15]= composite[id[P[cart[V, V]]], VS] == VS

In[16]:= composite[id[P[cart[V, V]]], VS] := VS

In[17]:= ImageComp[id[P[cart[V, V]]], VS, x] // Reverse
Out[17]= intersection[image[VS, x], P[cart[V, V]]] == image[VS, x]
```

```
In[18]:= intersection[image[VS, x_], P[cart[V, V]]] := image[VS, x]
```

new formulas for VS

The following properties of **VS** have been established earlier:

```
In[19]:= composite[BIGCUP, IMAGE[SECOND], VS]

Out[19]= IMAGE[SECOND]

In[20]:= composite[IMAGE[FIRST], VS]

Out[20]= IMAGE[FIRST]
```

From these one can deduce a useful property:

```
In[21]:= Map[composite[cross[Id, BIGCUP], #] &,
           Assoc[cross[IMAGE[FIRST], IMAGE[SECOND]], cross[VS, VS], DUP]]

Out[21]= composite[cross[Id, BIGCUP], DORA, VS] == DORA

In[22]:= composite[cross[Id, BIGCUP], DORA, VS] := DORA
```

The function **DORA** yields the domain and range:

```
In[23]:= SubstTest[A, image[z, singleton[y]], {y -> composite[Id, x], z -> DORA}] // Reverse

Out[23]= APPLY[DORA, composite[Id, x]] == PAIR[domain[x], range[x]]

In[24]:= APPLY[DORA, composite[Id, x_]] := PAIR[domain[x], range[x]]
```

inverse images of functions

The fact that producing inverse images of function values is one−to−one can be formally deduced as follows:

```
In[25]:= (composite[id[range[z]], inverse[VERTSECT[inverse[z]]],
             VERTSECT[inverse[z]], id[range[z]]] // RelnRenormality) /. z -> funpart[x]

Out[25]= composite[id[range[funpart[x]]], inverse[VERTSECT[inverse[funpart[x]]]],
           VERTSECT[inverse[funpart[x]]], id[range[funpart[x]]]] ==
         id[intersection[domain[VERTSECT[inverse[funpart[x]]]], range[funpart[x]]]]
```

This is added as a temporary rewrite rule:

```
In[26]:= composite[id[range[funpart[x_]]], inverse[VERTSECT[inverse[funpart[x_]]]],
           VERTSECT[inverse[funpart[x_]]], id[range[funpart[x_]]]] :=
         id[intersection[domain[VERTSECT[inverse[funpart[x]]]], range[funpart[x]]]]
```

A simpler reformulation of this fact is:

```
In[27]:= SubstTest[subclass, composite[z, inverse[z]], Id, z -> composite[
           id[range[funpart[x]]], inverse[VERTSECT[inverse[funpart[x]]]]]] // Reverse

Out[27]= FUNCTION[composite[id[range[funpart[x]]], inverse[VERTSECT[inverse[funpart[x]]]]] ==
          True
```

The following rewrite rule is the basis for our further work:

```
In[28]:= FUNCTION[
           composite[id[range[funpart[x_]]], inverse[VERTSECT[inverse[funpart[x_]]]]] := True
```

Corollary: This is actually a one−to−one correspondence:

```
In[29]:= ONEONE[composite[id[range[funpart[x]]], inverse[VERTSECT[inverse[funpart[x]]]]]

Out[29]= True
```

a variable−free reformulation

The rule derived in the preceding section can be reformulated in a variable−free form:

```
In[30]:= symdif[composite[FUNPART, INVERSE, VS, INVERSE, FUNPART],
           composite[INVERSE, VS, INVERSE, FUNPART]] // VSNormality

Out[30]= union[composite[intersection[complement[INVERSE], composite[FUNPART, INVERSE]],
           VS, INVERSE, FUNPART],
          composite[id[complement[FUNS]], INVERSE, VS, INVERSE, FUNPART]] == 0

In[31]:= % /. Equal -> SetDelayed
```

This can be rewritten as an equation:

```
In[32]:= SubstTest[equal, 0, symdif[u, v],
           {u -> composite[FUNPART, INVERSE, VS, INVERSE, FUNPART],
            v -> composite[INVERSE, VS, INVERSE, FUNPART]}]

Out[32]= True == equal[composite[INVERSE, VS, INVERSE, FUNPART],
           composite[FUNPART, INVERSE, VS, INVERSE, FUNPART]]

In[33]:= composite[FUNPART, INVERSE, VS, INVERSE, FUNPART] :=
          composite[INVERSE, VS, INVERSE, FUNPART]
```

Corollary:

```
In[34]:= Map[subclass[#, FUNS] &,
          ImageComp[composite[FUNPART, INVERSE, VS, INVERSE], FUNPART, V]]

Out[34]= subclass[image[INVERSE, image[VS, image[INVERSE, FUNS]]], FUNS] == True

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

This can be simplified further by imaging with **INVERSE**:

```
In[36]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
           {u -> image[INVERSE, image[VS, image[INVERSE, FUNS]]], v -> FUNS, w -> INVERSE}]

Out[36]= subclass[image[VS, image[INVERSE, FUNS]], image[INVERSE, FUNS]] == True
```

```
In[37]:= % /. Equal -> SetDelayed
```

Since **VERTSECT** is always a function, a sharper result can be stated:

```
In[38]:= SubstTest[subclass, image[VS, image[INVERSE, FUNS]],
          intersection[u, v], {u -> FUNS, v -> image[INVERSE, FUNS]}]

Out[38]= subclass[image[VS, image[INVERSE, FUNS]], BIJ] == True

In[39]:= subclass[image[VS, image[INVERSE, FUNS]], BIJ] := True
```

connection with Q

In this section the inclusion derived in the preceding section is used to derive an inclusion involving the equipollence relation **Q**. The resulting formula no longer involves the function **VS** at all. The first step is to derive a simplification rule for the function **DORA** which will be used for the derivation.

```
In[40]:= ImageComp[DORA, IMAGE[SWAP], FUNS] // Reverse

Out[40]= image[DORA, image[INVERSE, FUNS]] == inverse[image[DORA, FUNS]]

In[41]:= image[DORA, image[INVERSE, FUNS]] := inverse[image[DORA, FUNS]]
```

The next step is to use the **DORA** formula to obtain a temporary lemma:

```
In[42]:= Map[composite[BIGCUP, image[#, image[INVERSE, FUNS]]] &,
          Assoc[cross[IMAGE[FIRST], IMAGE[SECOND]], cross[VS, VS], DUP]]

Out[42]= composite[BIGCUP, image[DORA, image[VS, image[INVERSE, FUNS]]]] ==
          inverse[image[DORA, FUNS]]

In[43]:= % /. Equal -> SetDelayed

In[44]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
            {u -> image[VS, image[INVERSE, FUNS]],
           v -> BIJ, w -> composite[cross[Id, BIGCUP], DORA]}]

Out[44]= subclass[inverse[image[DORA, FUNS]], composite[BIGCUP, Q]] == True

In[45]:= subclass[inverse[image[DORA, FUNS]], composite[BIGCUP, Q]] := True
```

a variable−free version of Theorem FIN−FS

Lemma.

```
In[46]:= SubstTest[implies, subclass[u, v],
          subclass[image[inverse[u], w], image[inverse[v], w]],
             {u -> inverse[image[DORA, FUNS]], v -> composite[BIGCUP, Q], w -> FINITE}]

Out[46]= subclass[image[image[DORA, FUNS], FINITE],
           image[Q, intersection[FINITE, P[FINITE]]]] == True

In[47]:= % /. Equal -> SetDelayed
```

Lemma.

```
In[48]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> intersection[FINITE, P[FINITE]], v -> FINITE, w -> Q}]

Out[48]= subclass[image[Q, intersection[FINITE, P[FINITE]]], FINITE] == True

In[49]:= % /. Equal -> SetDelayed
```

The following result amounts to a variable−free version of Theorem **FIN−FS**.

```
In[50]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
             {u -> image[image[DORA, FUNS], FINITE],
           v -> image[Q, intersection[FINITE, P[FINITE]]], w -> FINITE}]

Out[50]= subclass[image[image[DORA, FUNS], FINITE], FINITE] == True

In[51]:= subclass[image[image[DORA, FUNS], FINITE], FINITE] := True
```

reintroducing variables

To better understand the meaning of the variable−free formula derived in the preceding section, it helps to reintroduce variables. Abstracting on the doubled image, one finds:

```
In[52]:= abstract[x, image[image[DORA, x], FINITE]]

Out[52]= composite[SECOND, id[cart[FINITE, V]], DORA]
```

This permits one to write the double image as a single image:

```
In[53]:= image[composite[SECOND, id[cart[FINITE, V]], DORA], x]

Out[53]= image[image[DORA, x], FINITE]
```

The monotonicity of images yields:

```
In[54]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
             {u -> singleton[x], v -> FUNS, w -> composite[SECOND, id[cart[FINITE, V]], DORA]}]

Out[54]= or[member[range[x], image[image[DORA, FUNS], FINITE]], not[FUNCTION[x]],
           not[member[x, V]], not[member[domain[x], FINITE]], not[member[range[x], V]]] == True

In[55]:= (% /. {x -> x_}) /. Equal -> SetDelayed
```

This can be cleaned up.

simplification rules

From the axiom of replacement, one deduces:

```
In[56]:= Map[implies[member[domain[x], y], #] &, SubstTest[implies,
           and[FUNCTION[x], member[u, V]], member[image[x, u], V], u -> domain[x]]]

Out[56]= or[member[range[x], V], not[FUNCTION[x]], not[member[domain[x], y]]] == True

In[57]:= or[member[range[x_], V], not[FUNCTION[x_]], not[member[domain[x_], y_]]] := True
```

The following result can be improved upon:

```
In[58]:= Map[implies[and[member[domain[x], u], member[range[x], v]], #] &,
          SubstTest[implies, and[equal[x, w], member[w, V]], member[x, V], w -> composite[Id, x]]]

Out[58]= or[member[x, V], not[member[domain[x], u]],
           not[member[range[x], v]], not[subclass[x, cart[V, V]]]] == True

In[59]:= (% /. {u -> u_, v -> v_, x -> x_}) /. Equal -> SetDelayed

In[60]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p2, p3], p4], not[implies[and[p1, p3], p4]],
             {p1 -> subclass[x, cart[y, z]], p2 -> subclass[x, cart[V, V]],
              p3 -> and[member[domain[x], u], member[range[x], v]], p4 -> member[x, V]}]]

Out[60]= or[member[x, V], not[member[domain[x], u]],
           not[member[range[x], v]], not[subclass[x, cart[y, z]]]] == True

In[61]:= or[member[x_, V], not[member[domain[x_], u_]],
           not[member[range[x_], v_]], not[subclass[x_, cart[y_, z_]]]] := True
```

Corollary:

```
In[62]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[and[p1, p2], p4], implies[and[p2, p3, p4], p5],
           not[implies[and[p1, p2], p5]], {p1 -> FUNCTION[x], p2 -> member[domain[x], y],
            p3 -> subclass[x, cart[V, V]], p4 -> member[range[x], V], p5 -> member[x, V]}]]

Out[62]= or[member[x, V], not[FUNCTION[x]], not[member[domain[x], y]]] == True

In[63]:= or[member[x_, V], not[FUNCTION[x_]], not[member[domain[x_], y_]]] := True
```

The result that emerges is analogous to Theorem **FIN−FS** that was proved using **Otter**.

```
In[64]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[and[p1, p2], p4],
            implies[and[p1, p2, p3, p4], p5], implies[p5, p6], not[implies[and[p1, p2], p6]],
              {p1 -> FUNCTION[x], p2 -> member[domain[x], FINITE],
             p3 -> member[range[x], V], p4 -> member[x, V],
              p5 -> member[range[x], image[image[DORA, FUNS], FINITE]],
             p6 -> member[range[x], FINITE]}]]

Out[64]= or[member[range[x], FINITE], not[FUNCTION[x]], not[member[domain[x], FINITE]]] == True

In[65]:= or[member[range[x_], FINITE],
           not[FUNCTION[x_]], not[member[domain[x_], FINITE]]] := True
```