Formisano and Omodeo's Theorem FU-F-O

Johan G. F. Belinfante and Tiffany D. Goble
2004 June 14

```
In[1]:= SetDirectory["p:"]; << goedel58.12g; << tools.m
```

```
:Package Title: goedel58.12g      2004 June 12 at 10:50 p.m.

It is now:  2004 Jun 14 at 16:38

Loading Simplification Rules

TOOLS.M                       Revised 2004 May 14

weightlimit = 40
```

summary

Theorem **FU-F-O** derives its name from "Formisano and Omodeo," because I learned this theorem from a paper they wrote with Temperini. The theorem was proved using **Otter** without any real help, nor even understanding, on my part. The proof that **Otter** found made little use of theorems about functions, aside from the definitions, and provides little insight into what is going on.. The application that Formisano, Omodeo and Temperini had in mind was the construction of "conjugated quasi-projections" in the context of a relational calculus axiomatized by Tarski and Givant. Roughly speaking, one constructs functions that correspond to **FIRST** and **SECOND**, which of course will depend on how the ordered pair is constructed. For certain constructions of the ordered pair, it may be easier to construct the union of these functions than the individual functions. Their idea is that if one can construct **FIRST** and **union[FIRST, SECOND]**, then there is a standard way to obtain a construction of **SECOND** in terms of these. In this notebook it is shown that if **x** is a function, and **y** is any relation, then **intersection[y, complement[composite[Di, dif[y, x]]]]** is a function. In the special case that **x** is **FIRST** and **y** is **union[FIRST, SECOND**], then this function is **SECOND**, and vice versa.

```
In[2]:= intersection[y, complement[composite[Di, dif[y, x]]]] /.
         {x -> FIRST, y -> union[FIRST, SECOND]}
```

```
Out[2]= SECOND
```

```
In[3]:= intersection[y, complement[composite[Di, dif[y, x]]]] /.
         {x -> SECOND, y -> union[FIRST, SECOND]}
```

```
Out[3]= FIRST
```

Reference:

A. Formisano, E. G. Omodeo and M. Temperini, Layered map reasoning: An experimental approach put to trial on sets. In: A. Dovier, M.-C. Meo and A. Omicini, Eds., Declarative Programming-Selected Papers from AGP 2000. Electronic Notes in Theoretical Computer Science 48:1-28, Elsevier Science B.V., 2001.

---

prerequisites: funpart

The function part of a relation **x** is its restriction to those arguments where there is a single value:

```
In[4]:= class[pair[u, v], equal[image[x, singleton[u]], singleton[v]]]
Out[4]= funpart[x]
```

This function can be written as an intersection of two relations:

```
In[5]:= class[pair[u, v], subclass[singleton[v], image[x, singleton[u]]]]
Out[5]= composite[Id, x]
```

```
In[6]:= class[pair[u, v], subclass[image[x, singleton[u]], singleton[v]]]
Out[6]= composite[Id, complement[composite[Di, x]]]
```

This yields the formula

```
In[7]:= dif[composite[Id, x], composite[Di, x]]
Out[7]= funpart[x]
```

Another way to write **funpart[x]** is as the restriction of **x** to the class where is is singlevalued:

```
In[8]:= class[u, exists[v, equal[image[x, singleton[u]], singleton[v]]]]
Out[8]= domain[funpart[x]]
```

```
In[9]:= composite[x, id[domain[funpart[x]]]]
Out[9]= funpart[x]
```

---

non-function part

A similar pair of formulas apply to the non-functional part of **x**.

```
In[10]:= dif[composite[Id, x], funpart[x]] // RelnRenormality

Out[10]= composite[Id, intersection[x, complement[funpart[x]]]] ==
          intersection[x, composite[Di, x]]

In[11]:= composite[Id, intersection[x_, complement[funpart[x_]]]] :=
          intersection[x, composite[Di, x]]
```

The domain of the non-function part is:

```
In[12]:= SubstTest[domain, dif[u, v], {u → composite[Id, x], v → funpart[x]}] // Reverse

Out[12]= fix[composite[inverse[x], complement[funpart[x]]]] ==
          fix[composite[inverse[x], Di, x]]
```

This formula can also be derived using **Normality**:

```
In[13]:= fix[composite[inverse[x], complement[funpart[x]]]] // Normality

Out[13]= fix[composite[inverse[x], complement[funpart[x]]]] ==
          fix[composite[inverse[x], Di, x]]

In[14]:= fix[composite[inverse[x_], complement[funpart[x_]]]] :=
          fix[composite[inverse[x], Di, x]]
```

The domain of the non-functional part is the relative complement of the domain of the functional part:

```
In[15]:= domain[dif[x, funpart[x]]] // Renormality // Reverse

Out[15]= intersection[complement[domain[funpart[x]]], domain[x]] ==
          fix[composite[inverse[x], Di, x]]

In[16]:= intersection[complement[domain[funpart[x_]]], domain[x_]] :=
          fix[composite[inverse[x], Di, x]]
```

The non-functional part is a restriction:

```
In[17]:= equal[intersection[x, composite[Di, x]],
           composite[x, id[fix[composite[inverse[x], Di, x]]]]] // AssertTest

Out[17]= equal[composite[x, id[fix[composite[inverse[x], Di, x]]]],
           intersection[x, composite[Di, x]]] == True

In[18]:= composite[x_, id[fix[composite[inverse[x_], Di, x_]]]] :=
          intersection[x, composite[Di, x]]
```

The restriction of **funpart[x]** to the domain of the non-functional part is empty:

```
In[19]:= composite[funpart[x], id[fix[composite[inverse[x], Di, x]]]] // VSNormality

Out[19]= composite[funpart[x], id[fix[composite[inverse[x], Di, x]]]] == 0
```

```
In[20]:= composite[funpart[x_], id[fix[composite[inverse[x_], Di, x_]]]] := 0
```

The functional part of the non-functional part is empty:

```
In[21]:= SubstTest[funpart, composite[x, id[y]], y → fix[composite[inverse[x], Di, x]]]

Out[21]= funpart[intersection[x, composite[Di, x]]] == 0

In[22]:= funpart[intersection[x_, composite[Di, x_]]] := 0
```

---

application

The class considered by Formisano, Omodeo and Temporini is:

```
In[23]:= F[x_, y_] :=
          intersection[composite[Id, y], complement[composite[Di, dif[y, funpart[x]]]]]
```

The goal in this section is to show that this is always a function.

```
In[24]:= Map[subclass[#,
            union[funpart[x], funpart[intersection[y, complement[funpart[x]]]]]] &,
           SubstTest[union, intersection[z, w], dif[z, w],
            {z → F[x, y], w → funpart[x]}]] // Reverse

Out[24]= subclass[composite[Id, intersection[y,
             complement[composite[Di, intersection[y, complement[funpart[x]]]]]],
           union[funpart[x], funpart[intersection[y, complement[funpart[x]]]]]] == True

In[25]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Restatement: **F[x, y]** is contained in the union of two functions:

```
In[26]:= subclass[F[x, y], union[funpart[x], funpart[dif[y, funpart[x]]]]]

Out[26]= True
```

The idea now is to show that **F[x,y]** is a function by showing that its non-function part is a function. The general idea is this:

```
In[27]:= FUNCTION[intersection[x, composite[Di, x]]] // AssertTest

Out[27]= FUNCTION[intersection[x, composite[Di, x]]] == FUNCTION[composite[Id, x]]

In[28]:= FUNCTION[intersection[x_, composite[Di, x_]]] := FUNCTION[composite[Id, x]]
```

```
In[29]:= Map[equal[0, #] &, intersection[composite[Di, F[x, y]],
           funpart[dif[y, funpart[x]]]] // RelnNormality]

Out[29]= equal[0, intersection[composite[Di, intersection[y,
             complement[composite[Di, intersection[y, complement[funpart[x]]]]]]],
           funpart[intersection[y, complement[funpart[x]]]]] == True
```

This is made into a temporary rewrite rule:

```
In[30]:= intersection[composite[Di, intersection[y_,
            complement[composite[Di, intersection[y_, complement[funpart[x_]]]]]]],
          funpart[intersection[y_, complement[funpart[x_]]]]] := 0
```

Restatement:

```
In[31]:= subclass[composite[Di, F[x, y]], complement[funpart[dif[y, funpart[x]]]]]

Out[31]= True
```

Lemma.

```
In[32]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u → funpart[intersection[x, y]], v → intersection[x, y], w → x}]

Out[32]= subclass[funpart[intersection[x, y]], x] == True

In[33]:= subclass[funpart[intersection[x_, y_]], x_] := True
```

Lemma.

```
In[37]:= SubstTest[implies, and[disjoint[u, x], subclass[v, z]],
           subclass[intersection[u, v], intersection[z, complement[x]]],
           z → union[x, y]] // MapNotNot

Out[37]= or[not[equal[0, intersection[u, x]]],
           not[subclass[v, union[x, y]]], subclass[intersection[u, v], y]] == True

In[38]:= or[not[equal[0, intersection[u_, x_]]],
           not[subclass[v_, union[x_, y_]]], subclass[intersection[u_, v_], y_]] := True
```

The non-function part of **F[x,y]** is a subclass of a function:

```
In[40]:= SubstTest[implies, and[disjoint[t, v], subclass[s, union[u, v]]],
          subclass[intersection[s, t], u], {s → F[x, y], t → composite[Di, F[x, y]],
           u → funpart[x], v -> funpart[dif[y, funpart[x]]]}]

Out[40]= subclass[intersection[y, composite[Di, intersection[y,
             complement[composite[Di, intersection[y, complement[funpart[x]]]]]]]],
           union[composite[Di, intersection[y, complement[funpart[x]]]],
            funpart[x]]] == True
```

```
In[41]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Restatement:

```
In[42]:= subclass[intersection[F[x, y], composite[Di, F[x, y]]], funpart[x]]

Out[42]= True
```

A subclass of a function is a function:

```
In[43]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]], FUNCTION[u],
          {u → intersection[F[x, y], composite[Di, F[x, y]]], v → funpart[x]}]

Out[43]= FUNCTION[composite[Id, intersection[y, complement[
              composite[Di, intersection[y, complement[funpart[x]]]]]]]] == True

In[44]:= FUNCTION[composite[Id, intersection[y_, complement[
              composite[Di, intersection[y_, complement[funpart[x_]]]]]]]] := True
```

Restatement:

```
In[45]:= FUNCTION[F[x, y]]

Out[45]= True
```

The final step is to remove the **funpart** wrappers, using equality substitution:

```
In[46]:= SubstTest[implies,
          and[equal[x, z], FUNCTION[composite[Id, dif[y, composite[Di, dif[y, z]]]]],
          FUNCTION[composite[Id, dif[y, composite[Di, dif[y, x]]]], z → funpart[x]]

Out[46]= or[FUNCTION[composite[Id, intersection[y,
              complement[composite[Di, intersection[y, complement[x]]]]]],
           not[FUNCTION[x]]] == True

In[47]:= or[FUNCTION[composite[Id, intersection[y_, complement[composite[Di,
                intersection[y_, complement[x_]]]]]], not[FUNCTION[x_]]] := True
```