a pigeonhole principle

Johan G. F. Belinfante and Ming Li
2006 February 10

```
In[1]:= SetDirectory["l:"]; << goedel78.10a; << tools.m
```

```
:Package Title: goedel78.10a           2006 February 10 at 5:00 a.m.

It is now:  2006 Feb 10 at 16:57

Loading Simplification Rules

TOOLS.M                      Revised 2006 February 3

weightlimit = 40
```

summary

One version of the pigeonhole principle says that any finite one-to-one mapping from a set to itself must be onto:

```
In[2]:= implies[and[member[x, FINITE], ONEONE[x], subclass[range[x], domain[x]]],
         equal[domain[x], range[x]]]

Out[2]= True
```

In this notebook the theorem of finite choice is used to show that this statement implies its own converse: any mapping of a finite set onto itself is one-to-one.

cross-sections

The theorem of finite choice can be stated succinctly without variables:

```
In[3]:= subclass[FINITE, SELECT]

Out[3]= True
```

Introducing variables produces a more useful version: any finite set admits a cross-section:

```
In[4]:= SubstTest[implies, and[member[x, y], subclass[y, z]],
         member[x, z], {y → FINITE, z → SELECT}]

Out[4]= or[not[equal[0, X[x]]], not[member[x, FINITE]]] == True

In[5]:= or[not[equal[0, X[x_]]], not[member[x_, FINITE]]] := True
```

Replacing **x** with its inverse yields this corollary:

```
In[6]:= SubstTest[implies, member[y, FINITE], not[equal[0, X[y]]], y → inverse[x]]

Out[6]= or[not[equal[0, X[inverse[x]]]],
          not[member[domain[x], FINITE]], not[member[range[x], FINITE]]] == True

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

A simpler version of this corollary says that the inverse of any finite set admits a cross-section

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
          not[implies[p1, p4]], {p1 → member[x, FINITE], p2 → member[domain[x], FINITE],
           p3 → member[range[x], FINITE], p4 → not[empty[X[inverse[x]]]]}]]

Out[8]= or[not[equal[0, X[inverse[x]]]], not[member[x, FINITE]]] == True

In[9]:= or[not[equal[0, X[inverse[x_]]]], not[member[x_, FINITE]]] := True
```

derivation

As a corollary of the fact that any subclass of a function is a restriction one finds:

```
In[10]:= SubstTest[implies, and[subclass[z, x], FUNCTION[x]],
          equal[z, composite[x, id[domain[z]]]], z → inverse[y]]

Out[10]= or[equal[composite[x, id[range[y]]], inverse[y]],
           not[FUNCTION[x]], not[subclass[inverse[y], x]]] == True

In[11]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The known version of the pigeon hole principle is now applied to a cross-section of the inverse of a finite function which maps its domain onto itself.

```
In[12]:= Map[not, SubstTest[and, implies[and[p2, p4], p5], implies[p1, p6],
           implies[and[p4, p6], p7], implies[and[p3, p4], p8], implies[and[p4, p5, p7, p8], p9],
           not[implies[and[p1, p2, p3, p4], p9]], {p1 → member[x, FINITE], p2 → FUNCTION[x],
            p3 → equal[range[x], domain[x]], p4 → member[y, X[inverse[x]]],
            p5 → FUNCTION[inverse[y]], p6 → member[inverse[x], FINITE], p7 → member[y, FINITE],
            p8 → subclass[range[y], domain[y]], p9 → equal[domain[y], range[y]]}]]

Out[12]= or[equal[domain[y], range[y]], not[equal[domain[x], range[x]]],
           not[equal[domain[y], range[x]]], not[FUNCTION[x]], not[FUNCTION[y]],
           not[member[x, FINITE]], not[member[y, V]], not[subclass[y, inverse[x]]]] == True

In[13]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

A statement close to the desired converse is derived from this:

```
In[14]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p4], p5], implies[p4, p6],
           implies[and[p2, p6], p7], implies[and[p3, p4, p5], p8], implies[and[p2, p7, p8], p9],
           implies[and[p4, p9], q], not[implies[and[p1, p2, p3, p4], q]],
           {p1 → member[x, FINITE], p2 → FUNCTION[x], p3 → equal[range[x], domain[x]],
            p4 → member[y, X[inverse[x]]], p5 → equal[domain[y], range[y]],
            p6 → subclass[inverse[y], x], p7 → equal[inverse[y], composite[x, id[range[y]]]], p8 →
             equal[range[y], domain[x]], p9 → equal[x, inverse[y]], q → FUNCTION[inverse[x]]}]]

Out[14]= or[FUNCTION[inverse[x]], not[equal[domain[x], range[x]]],
           not[equal[domain[y], range[x]]], not[FUNCTION[x]], not[FUNCTION[y]],
           not[member[x, FINITE]], not[member[y, V]], not[subclass[y, inverse[x]]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The simplify flag is cleared to speed up the removal of the variable **y** which refers to a cross-section of the inverse of **x**.

```
In[16]:= simplify = False;

In[17]:= Map[equal[V, #] &,
           SubstTest[class, y, implies[and[member[x, u], member[y, v]], member[x, w]],
            {u -> intersection[FINITE, FUNS, image[inverse[DORA], Id]],
             v -> X[inverse[x]], w → BIJ}]] // Reverse

Out[17]= or[equal[0, X[inverse[x]]], FUNCTION[inverse[x]],
           not[equal[domain[x], range[x]]], not[FUNCTION[x]], not[member[x, FINITE]]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

The theorem on finite choice is applied to eliminate all reference to cross-sections, producing the desired converse of the pigeonhole principle.

```
In[19]:= Map[not, SubstTest[and, implies[p1, p2], implies[and[p1, p2], p3], not[implies[p1, p3]],
           {p1 -> and[equal[domain[x], range[x]], FUNCTION[x], member[x, FINITE]],
            p2 → not[equal[0, X[inverse[x]]]], p3 → FUNCTION[inverse[x]]}]]

Out[19]= or[FUNCTION[inverse[x]], not[equal[domain[x], range[x]]],
           not[FUNCTION[x]], not[member[x, FINITE]]] == True

In[20]:= or[FUNCTION[inverse[x_]], not[equal[domain[x_], range[x_]]],
           not[FUNCTION[x_]], not[member[x_, FINITE]]] := True
```

A variable-free version of the theorem is easy to derive:

```
In[21]:= Map[equal[V, #] &,
          dif[intersection[FINITE, FUNS, image[inverse[DORA], Id]], BIJ] // complement //
           Normality]

Out[21]= subclass[intersection[FINITE, FUNS, image[inverse[DORA], Id]], BIJ] == True

In[22]:= subclass[intersection[FINITE, FUNS, image[inverse[DORA], Id]], BIJ] := True
```