Theorem SBV−PS−K

Johan G. F. Belinfante
2004 January 17

```
In[1]:= << goedel53.16a; << tools.m

        :Package Title: goedel53.16a      2004 January 16 at 10:10 p.m.

        It is now:  2004 Jan 21 at 13:21

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 January 3

        weightlimit = 40
```

summary

A proof of length 45 for Theorem **SBV−PS−K** was obtained 2000 October 8 using **Otter**. Reproducing this proof using the **GOEDEL** program sheds little insight into what is going on, but my notes for the work done in October 2000 included a hand−produced proof of a related theorem, a generalization of which is rederived here. Theorem **SBV−PS−K** can be deduced from it in a single step using **reify**.

a comment about the theorem proved by hand in October 2000

The theorem that was proved by hand in 2000 is this:

```
In[2]:= implies[member[union[x, singleton[y]], subvar[PS]], member[x, subvar[PS]]]

Out[2]= or[and[member[y, V], not[member[y, image[PS, x]]]],
          not[member[x, V]], not[subclass[x, union[image[PS, x], intersection[
              complement[singleton[y]], image[S, singleton[y]]]]]], subclass[x, image[PS, x]]]
```

Since the **GOEDEL** program produces an unwieldy expansion of the statement, the hand proof was examined to see what role the **singleton** plays. This suggested a more general theorem without singletons which will be derived now.

a more general theorem

Double negation is all that is needed to derive the first lemma:

```
In[3]:= or[not[disjoint[y, image[PS, y]]], not[subclass[y, union[image[PS, x], image[PS, y]]]],
          subclass[y, image[PS, x]]] // NotNotTest

Out[3]= or[not[equal[0, intersection[y, image[PS, y]]]],
          not[subclass[y, union[image[PS, x], image[PS, y]]], subclass[y, image[PS, x]]] == True

In[4]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The transitive property of the proper subset relation **PS** yields another lemma:

```
In[5]:= SubstTest[implies, subclass[u, v], subclass[image[u, x], image[v, x]],
            {u -> composite[PS, PS], v -> PS}]

Out[5]= subclass[image[PS, image[PS, x]], image[PS, x]] == True

In[6]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The preceding lemma implies another:

```
In[7]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
                 not[implies[p1, p3]],
             {p1 -> subclass[y, image[PS, x]],
                 p2 -> subclass[image[PS, y], image[PS, image[PS, x]]],
                 p3 -> subclass[image[PS, y], image[PS, x]]}]]

Out[7]= or[not[subclass[y, image[PS, x]]], subclass[image[PS, y], image[PS, x]]] == True

In[8]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The following variant of the transitivity of inclusion may be generally useful, and will be made into a permanent rewrite rule:

```
In[9]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
            {u -> x, v -> union[y, z], w -> z}]

Out[9]= or[not[subclass[x, union[y, z]]], not[subclass[y, z]], subclass[x, z]] == True

In[10]:= or[not[subclass[x_, union[y_, z_]]], not[subclass[y_, z_]], subclass[x_, z_]] := True
```

The above results are combined:

```
In[11]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            implies[and[p3, p4], p5], implies[p5, p6], implies[and[p2, p6], p7],
                 not[implies[and[p1, p4], p7]],
              {p1 -> subvariant[PS, union[x, y]],
              p2 -> subclass[x, union[image[PS, x], image[PS, y]]],
                 p3 -> subclass[y, union[image[PS, x], image[PS, y]]],
              p4 -> disjoint[y, image[PS, y]],
              p5 -> subclass[y, image[PS, x]],
              p6 -> subclass[image[PS, y], image[PS, x]],
           p7 -> subvariant[PS, x]}]]

Out[11]= or[not[equal[0, intersection[y, image[PS, y]]]],
            not[subclass[x, union[image[PS, x], image[PS, y]]]],
            not[subclass[y, image[PS, x]]], subclass[x, image[PS, x]]] == True

In[12]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

To help eliminate the variable **x** in this statement, the following reformulation is useful:

```
In[13]:= (implies[and[disjoint[y, image[PS, y]], member[x, u]], member[x, v]] /.
              {u -> image[inverse[ADJOIN[y]], subvar[PS]], v -> subvar[PS]}) // NotNotTest

Out[13]= or[not[equal[0, intersection[y, image[PS, y]]]], not[member[x, V]],
            not[member[y, V]], not[subclass[x, union[image[PS, x], image[PS, y]]]],
            not[subclass[y, union[image[PS, x], image[PS, y]]]], subclass[x, image[PS, x]]] == True

In[14]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The variable **x** is now eliminated:

```
In[15]:= Map[equal[V, #] &, SubstTest[class, x,
           implies[and[disjoint[y, image[PS, y]], member[x, u]], member[x, v]],
           {u -> image[inverse[ADJOIN[y]], subvar[PS]], v -> subvar[PS]}]] // Reverse

Out[15]= or[not[equal[0, intersection[y, image[PS, y]]]],
          subclass[image[inverse[ADJOIN[y]], subvar[PS]], subvar[PS]]] == True

In[16]:= (% /. y -> y_) /. Equal -> SetDelayed
```

derivation of Theorem SBV−PS−K

The condition **disjoint[y, image[PS, y]]** holds for singletons. Specializing to this case produces a version of the theorem that had been deduced by hand on 2000 October 8, but with one fewer variable.

```
In[17]:= SubstTest[implies, disjoint[z, image[PS, z]],
           invariant[inverse[ADJOIN[z]], subvar[PS]], z -> singleton[x]]

Out[17]= subclass[image[inverse[ADJOIN[singleton[x]]], subvar[PS]], subvar[PS]] == True

In[18]:= subclass[image[inverse[ADJOIN[singleton[x_]]], subvar[PS]], subvar[PS]] := True
```

The elimination of the one remaining variable **y** is accomplished most cleanly using **reify** as follows:

```
In[19]:= Map[equal[#, 0] &, SubstTest[reify, x, dif[
             image[inverse[ADJOIN[F[x]]], subvar[PS]], subvar[PS]], F -> singleton]] // Reverse

Out[19]= subclass[image[inverse[K], subvar[PS]], subvar[PS]] == True
```

This yields Theorem **SBV−PS−K**.

```
In[20]:= subclass[image[inverse[K], subvar[PS]], subvar[PS]] := True
```