power sets with 2 elements

Johan G. F. Belinfante
2005 January 9

```
In[1]:= SetDirectory["i:"]; << goedel65.08b; << tools.m

:Package Title: goedel65.08b            2005 January 8 at 9:50 p.m.

It is now:  2005 Jan 9 at 11:30

Loading Simplification Rules

TOOLS.M                        Revised 2005 January 7

weightlimit = 40
```

summary

The only power sets with exactly 2 elements are the power sets of singletons.

a needed lemma and a better one

Among other things, the following fact will be needed:

```
In[2]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]],
         subclass[u, w], {u → image[PAIRSET, x], v → range[PAIRSET], w → FINITE}]

Out[2]= subclass[image[PAIRSET, x], FINITE] == True
```

Rather than adding this special rule, a more general rule could be added, justified by the following fact:

```
In[3]:= implies[subclass[range[x], z], subclass[image[x, y], z]]

Out[3]= True
```

This can be made into a conditional rewrite rule.

```
In[4]:= subclass[image[x_, y_], z_] := True /; subclass[range[x], z]
```

Now the special rule is no longer needed:

In[5]:=
```
subclass[image[PAIRSET, x], FINITE]
```

Out[5]=
```
True
```

power sets of singletons have exactly two elements

This is half of the theorem: if **x** is singleton, then **P[x]** has two elements.

In[6]:=
```
Map[implies[member[x, range[SINGLETON]], member[x, image[SINGLETON, #]]] &,
  image[inverse[SINGLETON],
    image[inverse[POWER], image[PAIRSET, Di]]] // Normality]
```

Out[6]=
```
or[member[P[x], image[PAIRSET, Di]], not[member[x, range[SINGLETON]]]] == True
```

In[7]:=
```
or[member[P[x_], image[PAIRSET, Di]],
   not[member[x_, range[SINGLETON]]]] := True
```

To establish the converse, the idea is to capitalize on this rewrite rule that is already available:

In[8]:=
```
equal[P[x], pairset[0, x]]
```

Out[8]=
```
or[equal[0, x], member[x, range[SINGLETON]]]
```

The following is a start toward proving the converse:

In[9]:=
```
SubstTest[implies, and[member[z, v], equal[w, dif[v, singleton[z]]]],
  equal[v, union[singleton[z], w]], {v → P[x], w → singleton[x], z → 0}]
```

Out[9]=
```
or[equal[0, x], member[x, range[SINGLETON]], not[
    equal[intersection[complement[singleton[0]], P[x]], singleton[x]]]] == True
```

In[10]:=
```
(% /. x → x_) /. Equal → SetDelayed
```

covering lemma

The number of elements in a finite set is one more than that of the set obtained by removing one of its elements. Applying this idea to the case of a set with two elements yields:

```
In[11]:= SubstTest[implies, and[equal[t, card[v]],
           member[pair[u, v], K], member[v, FINITE]], equal[t, succ[card[u]]],
          {t → succ[singleton[0]], u → dif[y, singleton[x]], v → y}]

Out[11]= or[member[intersection[y, complement[singleton[x]]], range[SINGLETON]],
           not[member[x, y]], not[member[y, FINITE]],
           not[member[y, image[PAIRSET, Di]]]] == True

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

The finiteness hypothesis can be removed, since any set with 2 elements is finite.

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[and[p1, p2, p3], p4], not[implies[and[p1, p3], p4]],
           {p1 → member[y, image[PAIRSET, Di]], p2 → member[y, FINITE],
            p3 → member[x, y], p4 → member[dif[y, singleton[x]], range[SINGLETON]]}]]

Out[13]= or[member[intersection[y, complement[singleton[x]]], range[SINGLETON]],
           not[member[x, y]], not[member[y, image[PAIRSET, Di]]]] == True

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

---

the main theorem

The following lemma rewrites the condition that **P[x]** has exactly one element besides the empty set.

```
In[15]:= SubstTest[equal, u, singleton[U[u]], u → dif[P[x], singleton[0]]] // Reverse

Out[15]= member[intersection[complement[singleton[0]], P[x]], range[SINGLETON]] ==
          equal[intersection[complement[singleton[0]], P[x]], singleton[x]]

In[16]:= member[intersection[complement[singleton[0]], P[x_]], range[SINGLETON]] :=
          equal[intersection[complement[singleton[0]], P[x]], singleton[x]]
```

If a power set **P[x]** has exactly 2 elements, then **dif[P[x], singleton[0]] = singleton[x]**.

```
In[17]:= SubstTest[implies, and[member[u, v], member[v, image[PAIRSET, Di]]],
          member[dif[v, singleton[u]], range[SINGLETON]], {u → 0, v → P[x]}]

Out[17]= or[equal[intersection[complement[singleton[0]], P[x]], singleton[x]],
           not[member[P[x], image[PAIRSET, Di]]]] == True

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

It follows that if **P[x]** has two elements, then **x** is a singleton.

```
In[19]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p1, not[p3]], implies[p2, or[p3, p4]], not[implies[p1, p4]],
           {p1 -> member[P[x], image[PAIRSET, Di]],
            p2 -> equal[intersection[complement[singleton[0]], P[x]], singleton[x]],
            p3 → equal[0, x], p4 -> member[x, range[SINGLETON]]}]]

Out[19]= or[member[x, range[SINGLETON]], not[member[P[x], image[PAIRSET, Di]]]] == True

In[20]:= (% /. x → x_) /. Equal → SetDelayed
```

This is a logical equivalence, which is made into a rewrite rule.

```
In[21]:= equiv[member[P[x], image[PAIRSET, Di]], member[x, range[SINGLETON]]]

Out[21]= True

In[22]:= member[P[x_], image[PAIRSET, Di]] := member[x, range[SINGLETON]]
```

Corollary

```
In[23]:= SubstTest[member, P[x], union[u, v],
          {u → range[SINGLETON], v → image[PAIRSET, Di]}]

Out[23]= member[P[x], range[PAIRSET]] == or[equal[0, x], member[x, range[SINGLETON]]]

In[24]:= member[P[x_], range[PAIRSET]] := or[equal[0, x], member[x, range[SINGLETON]]]
```

---

variable free formulations

In this section, variable-free formulations are derived.

```
In[25]:= image[inverse[POWER], image[PAIRSET, Di]] // Normality

Out[25]= image[inverse[POWER], image[PAIRSET, Di]] == range[SINGLETON]

In[26]:= image[inverse[POWER], image[PAIRSET, Di]] := range[SINGLETON]

In[27]:= ImageComp[POWER, inverse[POWER], image[PAIRSET, Di]]

Out[27]= intersection[image[PAIRSET, Di], range[POWER]] ==
          image[POWER, range[SINGLETON]]

In[28]:= intersection[image[PAIRSET, Di], range[POWER]] :=
          image[POWER, range[SINGLETON]]

In[29]:= SubstTest[image, inverse[POWER], union[x, y],
          {x → range[SINGLETON], y → image[PAIRSET, Di]}]

Out[29]= image[inverse[POWER], range[PAIRSET]] == union[range[SINGLETON], singleton[0]]
```

In[30]:= **image[inverse[POWER], range[PAIRSET]] := union[range[SINGLETON], singleton[0]]**

In[31]:= **ImageComp[POWER, inverse[POWER], range[PAIRSET]]**

Out[31]= intersection[range[PAIRSET], range[POWER]] ==
union[image[POWER, range[SINGLETON]], singleton[singleton[0]]]

In[32]:= **intersection[range[PAIRSET], range[POWER]] :=**
union[image[POWER, range[SINGLETON]], singleton[singleton[0]]]

---

an explicit formula for image[POWER, range[SINGLETON]]

The idea in this section is to obtain a variable-free version of this rewrite rule for power sets of singletons.

In[33]:= **P[singleton[x]]**

Out[33]= pairset[0, singleton[x]]

Using **reify** did not work directly, but the following result was found by carefully examining the **reify** formulas:

In[34]:= **SubstTest[VERTSECT, inverse[LB[x]], x → Id] // Reverse**

Out[34]= composite[POWER, SINGLETON] ==
composite[ADJOIN[singleton[0]], SINGLETON, SINGLETON]

In[35]:= **composite[POWER, SINGLETON] :=**
composite[ADJOIN[singleton[0]], SINGLETON, SINGLETON]

Corollary.

In[36]:= **ImageComp[POWER, SINGLETON, V]**

Out[36]= image[ADJOIN[singleton[0]], image[SINGLETON, range[SINGLETON]]] ==
image[POWER, range[SINGLETON]]

This formula characterizes the power sets of singletons as those sets that can be written as the union of **singleton[0]** and the singleton of a singleton.