Aclosure[Z]

Johan G. F. Belinfante
2003 February 26

```
In[1]:= << goedel52.r23; << tools.m

        :Package Title: goedel52.r23          2003 February 25 at 12:50 noon

        It is now:  2003 Feb 26 at 15:29

        Loading Simplification Rules

        TOOLS.M                       Revised 2002 December 27

        weightlimit = 40
```

■ summary

This notebook contains a derivation of the equation **Aclosure[Z]=union[Z,singleton[0]]**. The intuitive idea is to use the fact that since **Z** is the set of equivalence classes of the equivalence relation **EQUIDIFF**, the integers are pairwise disjoint, and therefore adding to **Z** the intersections of subsets of **Z** only adds the empty set.

```
In[2]:= subclass[cart[Z, Z], union[Id, DISJOINT]]

Out[2]= True
```

This fact can also be written as

```
In[3]:= subclass[P[Z], cliques[union[Id, DISJOINT]]]

Out[3]= True
```

The equation for **Aclosure[Z]** is derived by showing that the right side is contained in the left side, and vice versa.

■ inclusion in one direction

The inclusion in one direction is easy. From the fact that **A[Z] = 0**, one concludes:

```
In[4]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → singleton[Z], v → P[Z], w → BIGCAP}]

Out[4]= member[0, Aclosure[Z]] == True

In[5]:= member[0, Aclosure[Z]] := True
```

This yields an inclusion in one direction:

```
In[6]:= subclass[union[Z, singleton[0]], Aclosure[Z]]

Out[6]= True
```

The inclusion in the other direction is a bit harder, and uses a general result derived in the next section.

■ some general results

The idea is to show that **A[x]=0** whenever **x** contains a couple of disjoint members.

```
In[7]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
            {u -> pair[x, y], v -> cart[Z, Z], w -> union[Id, DISJOINT]}] // MapNotNot

Out[7]= or[equal[0, intersection[x, y]],
          equal[x, y], not[member[x, Z]], not[member[y, Z]]] == True

In[8]:= or[equal[0, intersection[x_, y_]],
          equal[x_, y_], not[member[x_, Z]], not[member[y_, Z]]] := True
```

The connection between **intersection** and **A** requires the introduction of **pairset**.

```
In[9]:= implies[and[member[x, z], member[y, z]], member[pairset[x, y], P[z]]] // AssertTest

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

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

This takes a while:

```
In[11]:= implies[and[member[x, V], member[y, V]],
            equal[A[pairset[x, y]], intersection[x, y]]] // AssertTest

Out[11]= or[equal[A[pairset[x, y]], intersection[x, y]],
            not[member[x, V]], not[member[y, V]]] == True

In[12]:= or[equal[A[pairset[x_, y_]], intersection[x_, y_]],
            not[member[x_, V]], not[member[y_, V]]] := True
```

The next step is just to replace **V** in the above statement by an arbitary class.

```
In[13]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p1, p4],
            implies[p2, p5], implies[and[p4, p5], p6], not[implies[and[p1, p2], p6]],
              {p1 -> member[x, z], p2 -> member[y, z], p3 -> subclass[pairset[x, y], z],
                   p4 -> member[x, V], p5 -> member[y, V],
               p6 -> equal[A[pairset[x, y]], intersection[x, y]],
                   p7 -> disjoint[x, y], p8 -> equal[0, A[pairset[x, y]]],
               p9 -> subclass[A[z], A[pairset[x, y]]], p10 -> equal[0, A[z]]}]]

Out[13]= or[equal[A[pairset[x, y]], intersection[x, y]],
            not[member[x, z]], not[member[y, z]]] == True

In[14]:= or[equal[A[pairset[x_, y_]], intersection[x_, y_]],
            not[member[x_, z_]], not[member[y_, z_]]] := True
```

This is the main result:

```
In[15]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p1, p2], p6], implies[and[p6, p7], p8], implies[p3, p9],
             implies[and[p8, p9], p10], not[implies[and[p1, p2, p7], p10]],
             {p1 -> member[x, z], p2 -> member[y, z], p3 -> subclass[pairset[x, y], z],
             p6 -> equal[A[pairset[x, y]], intersection[x, y]],
                 p7 -> disjoint[x, y], p8 -> equal[0, A[pairset[x, y]]],
            p9 -> subclass[A[z], A[pairset[x, y]]], p10 -> equal[0, A[z]]}]]

Out[15]= or[equal[0, A[z]], not[equal[0, intersection[x, y]]],
           not[member[x, z]], not[member[y, z]]] == True

In[16]:= or[equal[0, A[z_]], not[equal[0, intersection[x_, y_]]],
           not[member[x_, z_]], not[member[y_, z_]]] := True
```

The variables **x** and **y** will be eliminated, but first we need to write the statement in negated form.

```
In[17]:= and[equal[0, intersection[x, y]],
           member[x, z], member[y, z], not[equal[0, A[z]]]] // NotNotTest

Out[17]= and[equal[0, intersection[x, y]],
           member[x, z], member[y, z], not[equal[0, A[z]]]] == False

In[18]:= and[equal[0, intersection[x_, y_]],
           member[x_, z_], member[y_, z_], not[equal[0, A[z_]]]] := False
```

The variables **x** and **y** can now be eliminated:

```
In[19]:= SubstTest[class, pair[x, y], and[equal[0, intersection[x, y]],
           member[x, z], member[y, z], not[equal[0, w]]], w -> A[z]]

Out[19]= 0 == composite[id[intersection[z, image[V, A[z]]]], DISJOINT, id[z]]

In[20]:= Map[equal[0, #] &, %] // Reverse

Out[20]= or[equal[0, A[z]], equal[0, intersection[z, image[DISJOINT, z]]]] == True

In[21]:= or[equal[0, A[z_]], equal[0, intersection[z_, image[DISJOINT, z_]]]] := True
```

The variable **z** can also be eliminated.

```
In[22]:= SubstTest[class, z,
           or[equal[0, A[z]], equal[0, intersection[z, image[w, z]]]], w -> DISJOINT] // Reverse

Out[22]= union[cliques[composite[∈, inverse[∈]]],
           intersection[complement[singleton[0]], image[inverse[BIGCAP], singleton[0]]]] == V

In[23]:= union[cliques[composite[E, inverse[E]]],
           intersection[complement[singleton[0]], image[inverse[BIGCAP], singleton[0]]]] := V
```

This formula can be simplified:

```
In[24]:= union[cliques[composite[E, inverse[E]]],
           image[inverse[BIGCAP], singleton[0]]] // Normality

Out[24]= union[cliques[composite[∈, inverse[∈]]], image[inverse[BIGCAP], singleton[0]]] == V

In[25]:= union[cliques[composite[E, inverse[E]]], image[inverse[BIGCAP], singleton[0]]] := V
```

■ inclusion in the other direction

The first step is to sharpen the formulation of the statement about integers being mutually exclusive.

```
In[26]:= SubstTest[cliques, intersection[u, v], {u -> cart[Z, Z], v -> union[DISJOINT, Id]}]

Out[26]= cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]] ==
          intersection[cliques[union[DISJOINT, Id]], P[Z]]

In[27]:= Map[subclass[P[Z], #] &, %]

Out[27]= subclass[cart[Z, Z], union[composite[id[Z], DISJOINT, id[Z]], id[Z]]] == True

In[28]:= subclass[cart[Z, Z], union[composite[id[Z], DISJOINT, id[Z]], id[Z]]] := True
```

The following further reformulation is useful:

```
In[29]:= subclass[P[Z], cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]]

Out[29]= True
```

The functor **cliques** preserves intersections.

```
In[30]:= SubstTest[cliques, intersection[u, v],
             {u -> complement[DISJOINT], v -> union[id[Z], restrict[DISJOINT, Z, Z]]}] //
           Reverse

Out[30]= intersection[cliques[composite[e, inverse[e]]],
           cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]] ==
          union[image[SINGLETON, Z], singleton[0]]

In[31]:= intersection[cliques[composite[e, inverse[e]]],
           cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]] :=
          union[image[SINGLETON, Z], singleton[0]]

In[32]:= SubstTest[implies, subclass[u, v], subclass[intersection[w, u], intersection[w, v]],
           {u → P[Z], v -> cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]],
            w -> cliques[composite[e, inverse[e]]]}]

Out[32]= subclass[intersection[cliques[composite[e, inverse[e]]], P[Z]],
           cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]] == True

In[33]:= subclass[intersection[cliques[composite[e, inverse[e]]], P[Z]],
           cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]] := True

In[34]:= SubstTest[subclass, u, intersection[u, v],
           {u → intersection[cliques[composite[e, inverse[e]]], P[Z]],
            v -> cliques[union[composite[id[Z], DISJOINT, id[Z]], id[Z]]]}]

Out[34]= subclass[intersection[cliques[composite[e, inverse[e]]], P[Z]],
           union[image[SINGLETON, Z], singleton[0]]] == True

In[35]:= subclass[intersection[cliques[composite[e, inverse[e]]], P[Z]],
           union[image[SINGLETON, Z], singleton[0]]] := True
```

Thw next step uses the transitivity of inclusions.

```
In[36]:= SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
          {u -> dif[P[Z], union[image[SINGLETON, Z], singleton[0]]],
           v -> complement[cliques[composite[E, inverse[E]]]],
           w -> image[inverse[BIGCAP], singleton[0]]}]

Out[36]= subclass[P[Z], union[image[SINGLETON, Z],
            image[inverse[BIGCAP], singleton[0]], singleton[0]]] == True

In[37]:= subclass[P[Z], union[image[SINGLETON, Z],
            image[inverse[BIGCAP], singleton[0]], singleton[0]]] := True

In[38]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> P[Z], v -> union[image[SINGLETON, Z],
             image[inverse[BIGCAP], singleton[0]], singleton[0]], w → BIGCAP}]

Out[38]= subclass[Aclosure[Z], union[Z, singleton[0]]] == True

In[39]:= subclass[Aclosure[Z], union[Z, singleton[0]]] := True
```

■ the final step

The equation is established by combining the two inclusions.

```
In[40]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> Aclosure[Z], v -> union[Z, singleton[0]]}]

Out[40]= True == equal[Aclosure[Z], union[Z, singleton[0]]]

In[41]:= Aclosure[Z] := union[Z, singleton[0]]
```