HULL[Z]

Johan G. F. Belinfante
2003 July 25

```
In[1]:= << goedel52.s61; << tools.m

        :Package Title: goedel52.s61       2003 July 23 at 8:55 p.m.

        It is now:  2003 Aug 15 at 10:8

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 July 25

        weightlimit = 40
```

summary

A formula for **HULL[Z]** is derived that depends on the fact that **Z** is a set of equivalence classes. Because integers are pairwise disjoint, a nonempty set can be a subset of at most one integer. This implies that **composite[id[Z], S, id[complement[singleton[0]]]]** is a function. This function is shown to be a restriction of **HULL[Z]**.

a cautionary note

If one is careless and forgets to require that the subset be nonempty, one finds the following result, which is true, but not what is wanted:

```
In[2]:= FUNCTION[composite[id[x], S]] // AssertTest

Out[2]= FUNCTION[composite[id[x], S]] == or[equal[0, x], member[x, range[SINGLETON]]]

In[3]:= FUNCTION[composite[id[x_], S]] := or[equal[0, x], member[x, range[SINGLETON]]]
```

What is wanted is an analog of the following:

```
In[4]:= FUNCTION[composite[id[x], E]]

Out[4]= subclass[cart[x, x], union[DISJOINT, Id]]
```

One of the lemmas used below is found by applying **AssertTest** to this result:

```
In[5]:= subclass[cart[x, x], union[DISJOINT, Id]] // AssertTest // Reverse

Out[5]= equal[0, intersection[x, fix[composite[Di, id[x], E, inverse[E]]]]] ==
         subclass[cart[x, x], union[DISJOINT, Id]]

In[6]:= equal[0, intersection[x_, fix[composite[Di, id[x_], E, inverse[E]]]]] :=
         subclass[cart[x, x], union[DISJOINT, Id]]
```

derivation

A second lemma:

```
In[7]:= FUNCTION[composite[id[x], S, id[complement[singleton[0]]]]] // assert // AssertTest

Out[7]= equal[0, intersection[x,
          fix[composite[S, id[complement[singleton[0]]], inverse[S], id[x], Di]]]] ==
         subclass[cart[x, x], union[DISJOINT, Id]]

In[8]:= equal[0, intersection[x_,
          fix[composite[S, id[complement[singleton[0]]], inverse[S], id[x_], Di]]]] :=
         subclass[cart[x, x], union[DISJOINT, Id]]
```

This is the desired result:

```
In[9]:= FUNCTION[composite[id[x], S, id[complement[singleton[0]]]]] // AssertTest

Out[9]= FUNCTION[composite[id[x], S, id[complement[singleton[0]]]]] ==
         subclass[cart[x, x], union[DISJOINT, Id]]

In[10]:= FUNCTION[composite[id[x_], S, id[complement[singleton[0]]]]] :=
          subclass[cart[x, x], union[DISJOINT, Id]]
```

In particular:

```
In[11]:= FUNCTION[composite[id[Z], S, id[complement[singleton[0]]]]]

Out[11]= True
```

The obvious question is: what exactly is this function? It will be shown to be related to **HULL[Z]**.

HULL[Z] formula

First, a simplification rule:

```
In[12]:= Assoc[HULL[Z], id[domain[HULL[Z]]], id[x]]

Out[12]= composite[HULL[Z], id[intersection[x, cliques[EQUIDIFF]]]] == composite[HULL[Z], id[x]]

In[13]:= composite[HULL[Z], id[intersection[x_, cliques[EQUIDIFF]]]] :=
          composite[HULL[Z], id[x]]

In[14]:= intersection[complement[cliques[EQUIDIFF]], complement[singleton[0]]] //
          DoubleComplement

Out[14]= intersection[complement[cliques[EQUIDIFF]], complement[singleton[0]]] ==
          complement[cliques[EQUIDIFF]]

In[15]:= intersection[complement[cliques[EQUIDIFF]], complement[singleton[0]]] :=
          complement[cliques[EQUIDIFF]]
```

A lemma is needed to remove **VERTSECT** from the final formula.

```
In[16]:= Map[composite[#, id[complement[singleton[0]]]] &, SubstTest[VERTSECT,
           funpart[x], x -> composite[id[Z], S, id[complement[singleton[0]]]]]]

Out[16]= composite[VERTSECT[composite[id[Z], S]], id[complement[singleton[0]]]] ==
          union[cart[complement[cliques[EQUIDIFF]], singleton[0]],
           composite[SINGLETON, id[Z], S, id[complement[singleton[0]]]]]

In[17]:= composite[VERTSECT[composite[id[Z], S]], id[complement[singleton[0]]]] :=
          union[cart[complement[cliques[EQUIDIFF]], singleton[0]],
           composite[SINGLETON, id[Z], S, id[complement[singleton[0]]]]]

In[18]:= Assoc[composite[id[Z], S], id[domain[composite[id[Z], S]]], id[x]]

Out[18]= composite[id[Z], S, id[intersection[x, cliques[EQUIDIFF]]]] ==
          composite[id[Z], S, id[x]]

In[19]:= composite[id[Z], S, id[intersection[x_, cliques[EQUIDIFF]]]] :=
          composite[id[Z], S, id[x]]
```

The basic idea is to use **DoubleComplement** to get the connection with **HULL[Z]**, and to use **VERTSECT** to cancel a factor of **inverse[E]**.

```
In[20]:= Map[VERTSECT,
          composite[inverse[E], id[Z], S, id[complement[singleton[0]]]] // DoubleComplement]

Out[20]= union[cart[union[complement[cliques[EQUIDIFF]], singleton[0]], singleton[0]],
           composite[id[Z], S, id[complement[singleton[0]]]]] ==
          union[cart[union[complement[cliques[EQUIDIFF]], singleton[0]], singleton[0]],
           composite[HULL[Z], id[complement[singleton[0]]]]]

In[21]:= Map[composite[#, id[complement[singleton[0]]]] &, %]

Out[21]= union[cart[complement[cliques[EQUIDIFF]], singleton[0]],
           composite[id[Z], S, id[complement[singleton[0]]]]] ==
          union[cart[complement[cliques[EQUIDIFF]], singleton[0]],
           composite[HULL[Z], id[complement[singleton[0]]]]]

In[22]:= Map[composite[#, id[cliques[EQUIDIFF]]] &, %]

Out[22]= composite[id[Z], S, id[complement[singleton[0]]]] ==
          composite[HULL[Z], id[complement[singleton[0]]]]

In[23]:= composite[id[Z], S, id[complement[singleton[0]]]] :=
          composite[HULL[Z], id[complement[singleton[0]]]]

In[24]:= SubstTest[union, composite[HULL[Z], id[x]], composite[HULL[Z], id[complement[x]]],
            x -> singleton[0]]

Out[24]= union[cart[singleton[0], singleton[0]],
           composite[HULL[Z], id[complement[singleton[0]]]]] == HULL[Z]

In[25]:= union[cart[singleton[0], singleton[0]],
           composite[HULL[Z], id[complement[singleton[0]]]]] := HULL[Z]
```

This is the complete formula for **HULL[Z]**:

```
In[26]:= union[id[singleton[0]], composite[id[Z], S, id[complement[singleton[0]]]]]

Out[26]= HULL[Z]
```

a funpart formula

The result obtained in this section is mainly a curiosity.

Lemma:

```
In[27]:= subclass[range[SINGLETON],
           union[intersection[complement[image[V, intersection[x, complement[y]]]],
             complement[image[V, intersection[y, complement[x]]]]],
            P[complement[singleton[x]]], P[complement[singleton[y]]]]] // AssertTest

Out[27]= subclass[range[SINGLETON],
           union[intersection[complement[image[V, intersection[x, complement[y]]]],
             complement[image[V, intersection[y, complement[x]]]]],
            P[complement[singleton[x]]], P[complement[singleton[y]]]] == True

In[28]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

Theorem: "singletons have only one member"

```
In[29]:= SubstTest[implies, and[member[u, v], subclass[v, w]], member[u, w],
             {u -> z, v -> range[SINGLETON],
            w -> union[intersection[complement[image[V, intersection[x, complement[y]]]],
               complement[image[V, intersection[y, complement[x]]]]],
              P[complement[singleton[x]]], P[complement[singleton[y]]]]}] // MapNotNot

Out[29]= or[equal[x, y], not[member[x, z]],
           not[member[y, z]], not[member[z, range[SINGLETON]]]] == True

In[30]:= or[equal[x_, y_], not[member[x_, z_]],
           not[member[y_, z_]], not[member[z_, range[SINGLETON]]]] := True
```

From this one can derive a formula for **Z**. More lemmas are needed. The first is:

```
In[31]:= equal[intersection[omega, RUSSELL], omega]

Out[31]= True

In[32]:= intersection[omega, RUSSELL] := omega
```

The integers zero and plus one are not equal.

```
In[33]:= equal[composite[id[omega], SUCC], id[omega]] // AssertTest

Out[33]= equal[composite[id[omega], SUCC], id[omega]] == False

In[34]:= equal[composite[id[omega], SUCC], id[omega]] := False
```

It follows that there is more than one integer.

```
In[35]:= Map[not, SubstTest[or, equal[x, y], not[member[x, z]],
           not[member[y, z]], not[member[z, range[SINGLETON]]],
                 {x -> id[omega], y -> composite[id[omega], SUCC], z -> Z}]]

Out[35]= member[Z, range[SINGLETON]] == False

In[36]:= member[Z, range[SINGLETON]] := False
```

Corollary:

```
In[37]:= equal[image[Di, Z], V]

Out[37]= True

In[38]:= image[Di, Z] := V
```

This **image[Di,Z]** formula is needed to get the **funpart** result:

```
In[39]:= Map[funpart, SubstTest[union, composite[u, id[v]], composite[u, id[complement[v]]],
           {u -> composite[id[Z], S], v -> singleton[0]}]] // Reverse

Out[39]= funpart[composite[id[Z], S]] == composite[HULL[Z], id[complement[singleton[0]]]]

In[40]:= funpart[composite[id[Z], S]] := composite[HULL[Z], id[complement[singleton[0]]]]
```