lambda[x,HULL[x]]

Johan G. F. Belinfante
2003 May 12

```
In[1]:= << goedel52.r67; << tools.m

        :Package Title: goedel52.r67         2003 May 9 at 11:50 p.m.

        It is now:  2003 May 12 at 12:56

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 May 11

        weightlimit = 40
```

■ summary

The function **LAMBHULL** introduced in this notebook is useful for formulating properties of **HULL[x]** that hold when **x** is a set. In particular, this allows one to write a variable−free equation that express the fact that **Aclosure[x]** is the same as **fix[HULL[x]]** when **x** is a set, as well as various other properties of **HULL[x]**.

■ sethood and other rules for fix[HULL[x]]

In this section a sethood rule for **fix[HULL[x]]** is derived, which will be needed later.

```
In[2]:= SubstTest[implies, member[y, V], member[fix[y], V], y -> HULL[x]]

Out[2]= or[member[fix[HULL[x]], V], not[member[x, V]]] == True

In[3]:= or[member[fix[HULL[x_]], V], not[member[x_, V]]] := True
```

In the other direction:

```
In[4]:= SubstTest[implies, and[subclass[x, y], member[y, V]], member[x, V], y -> fix[HULL[x]]]

Out[4]= or[member[x, V], not[member[fix[HULL[x]], V]]] == True

In[5]:= or[member[x_, V], not[member[fix[HULL[x_]], V]]] := True
```

These can be combined:

```
In[6]:= equiv[member[fix[HULL[x]], V], member[x, V]]

Out[6]= True

In[7]:= member[fix[HULL[x_]], V] := member[x, V]
```

For sets, **fix[HULL[x]]** is equal to **Aclosure[x]**. This implies the following rule which is needed later.

```
In[8]:= equal[singleton[fix[HULL[x]]], singleton[Aclosure[x]]] // AssertTest

Out[8]= equal[singleton[Aclosure[x]], singleton[fix[HULL[x]]]] == True

In[9]:= singleton[fix[HULL[x_]]] := singleton[Aclosure[x]]
```

■ a lemma

The combination of **VERTSECT** with **reify** is essentially equivalent to **lambda**, at least for total functions. In the case of **HULL** one obtains:

```
In[10]:= VERTSECT[reify[w, HULL[w]]]

Out[10]= VERTSECT[composite[SWAP,
           inverse[rotate[composite[VERTSECT[complement[composite[complement[inverse[e]],
                 intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]], SWAP]]]]
```

The inner **VERTSECT** here satisfies this rule:

```
In[11]:= composite[VERTSECT[complement[composite[complement[inverse[E]], intersection[
              composite[S, SECOND], composite[inverse[E], FIRST]]]]], LEFT[x]] // RelnNormality

Out[11]= composite[VERTSECT[complement[composite[complement[inverse[e]],
             intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]],
           LEFT[x]] == composite[id[image[V, singleton[x]]], HULL[x]]

In[12]:= composite[VERTSECT[complement[composite[complement[inverse[E]],
             intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]],
           LEFT[x_]] := composite[id[image[V, singleton[x]]], HULL[x]]
```

With these preliminaries out of the way, one obtains:

```
In[13]:= APPLY[VERTSECT[reify[w, HULL[w]]], x]

Out[13]= union[complement[image[V, singleton[x]]],
          composite[id[image[V, singleton[x]]], HULL[x]]]
```

This formula is the basis for much that follows.

■ The function LAMBHULL (short for lambda−HULL)

The function **LAMBHULL**, short for **lambda HULL**, is defined by this membership rule:

```
In[14]:= member[x_, LAMBHULL] := and[member[first[x], V], equal[second[x], HULL[first[x]]]]
```

Some of the simpler properties follow immediately from this definition, in particular:

```
In[15]:= dif[LAMBHULL, cart[V, V]] // Normality

Out[15]= intersection[LAMBHULL, complement[cart[V, V]]] == 0

In[16]:= intersection[LAMBHULL, complement[cart[V, V]]] := 0
```

```
In[17]:= SubstTest[equal, 0, dif[u, v], {u -> LAMBHULL, v -> cart[V, V]}] // Reverse

Out[17]= subclass[LAMBHULL, cart[V, V]] == True

In[18]:= subclass[LAMBHULL, cart[V, V]] := True

In[19]:= equal[composite[Id, LAMBHULL], LAMBHULL]

Out[19]= True

In[20]:= composite[Id, LAMBHULL] := LAMBHULL
```

■ vertical sections of LAMBHULL

We begin with a technicality needed to derive a vertical section rule for **LAMBHULL**. Without it, one would get a messy result:

```
In[21]:= image[LAMBHULL, singleton[x]] // Normality

Out[21]= image[LAMBHULL, singleton[x]] ==
          singleton[HULL[union[x, complement[image[V, singleton[x]]]]]]
```

The equality substitution rule for **HULL** is needed here:

```
In[22]:= Map[implies[member[x, V], #] &,
          SubstTest[implies, equal[x, y], equal[HULL[x], HULL[y]],
         y -> union[x, complement[image[V, singleton[x]]]]]]

Out[22]= or[equal[HULL[x], HULL[union[x, complement[image[V, singleton[x]]]]],
           not[member[x, V]]] == True

In[23]:= or[equal[HULL[x_], HULL[union[x_, complement[image[V, singleton[x_]]]]],
           not[member[x_, V]]] := True
```

The following consequence holds:

```
In[24]:= AssertTest[equal[singleton[u], singleton[v]]] /.
          {u -> HULL[x], v -> HULL[union[x, complement[image[V, singleton[x]]]]]}

Out[24]= equal[singleton[HULL[x]],
           singleton[HULL[union[x, complement[image[V, singleton[x]]]]]] == True

In[25]:= singleton[HULL[union[x_, complement[image[V, singleton[x_]]]]]] := singleton[HULL[x]]
```

One now obtains an elegant vertical section rule:

```
In[26]:= image[LAMBHULL, singleton[x]] // Normality

Out[26]= image[LAMBHULL, singleton[x]] == singleton[HULL[x]]

In[27]:= image[LAMBHULL, singleton[x_]] := singleton[HULL[x]]
```

■ LAMBHULL is a function

The vertical section rule is used here to establish the equality of **LAMBHULL** and **VERTSECT[reify[w,HULL[w]]]**.

```
In[28]:= symdif[LAMBHULL, VERTSECT[reify[w, HULL[w]]]] // VSNormality

Out[28]= union[
           intersection[LAMBHULL, complement[VERTSECT[composite[SWAP, inverse[rotate[composite[
                  VERTSECT[complement[composite[complement[inverse[e]], intersection[
                      composite[S, SECOND], composite[inverse[e], FIRST]]]]], SWAP]]]]]],
           intersection[complement[LAMBHULL], VERTSECT[composite[SWAP, inverse[rotate[
                composite[VERTSECT[complement[composite[complement[inverse[e]], intersection[
                    composite[S, SECOND], composite[inverse[e], FIRST]]]]], SWAP]]]]]] == 0

In[29]:= union[
           intersection[LAMBHULL, complement[VERTSECT[composite[SWAP, inverse[rotate[composite[
                  VERTSECT[complement[composite[complement[inverse[e]], intersection[
                      composite[S, SECOND], composite[inverse[e], FIRST]]]]], SWAP]]]]]],
           intersection[complement[LAMBHULL], VERTSECT[composite[SWAP, inverse[rotate[
                composite[VERTSECT[complement[composite[complement[inverse[e]], intersection[
                    composite[S, SECOND], composite[inverse[e], FIRST]]]]], SWAP]]]]]] := 0

In[30]:= SubstTest[equal, 0, symdif[u, v], {u -> LAMBHULL, v -> VERTSECT[reify[w, HULL[w]]]}]

Out[30]= True == equal[LAMBHULL,
           VERTSECT[composite[SWAP, inverse[rotate[composite[VERTSECT[complement[
                  composite[complement[inverse[e]], intersection[composite[S, SECOND],
                    composite[inverse[e], FIRST]]]]], SWAP]]]]]]

In[31]:= VERTSECT[composite[SWAP, inverse[rotate[
               composite[VERTSECT[complement[composite[complement[inverse[E]], intersection[
                    composite[S, SECOND], composite[inverse[E], FIRST]]]]], SWAP]]]]] := LAMBHULL
```

This implies, among other things, that **LAMBHULL** is a function:

```
In[32]:= SubstTest[FUNCTION, VERTSECT[x], x -> reify[w, HULL[w]]]

Out[32]= FUNCTION[LAMBHULL] == True

In[33]:= FUNCTION[LAMBHULL] := True
```

■ domain of LAMBHULL

The following lemma is needed here:

```
In[34]:= member[composite[id[image[V, singleton[x]]], HULL[x]], V] // AssertTest

Out[34]= member[composite[id[image[V, singleton[x]]], HULL[x]], V] == True

In[35]:= member[composite[id[image[V, singleton[x_]]], HULL[x_]], V] := True
```

The key to computing the domain is this fact:

```
In[36]:= SubstTest[member, x, domain[VERTSECT[y]], y -> reify[w, HULL[w]]]

Out[36]= member[x, domain[LAMBHULL]] == member[x, V]

In[37]:= member[x_, domain[LAMBHULL]] := member[x, V]
```

The formula for the domain follows immediately:

```
In[38]:= domain[LAMBHULL] // Normality

Out[38]= domain[LAMBHULL] == V

In[39]:= domain[LAMBHULL] := V
```

■ a binary function related to HULL[x]

The binary function under consideration in this section is:

```
In[40]:= lambda[pair[x, y], APPLY[HULL[x], y]]

Out[40]= composite[VERTSECT[
           complement[composite[complement[inverse[e]], intersection[composite[S, SECOND],
             composite[inverse[e], FIRST]]]]], id[cart[V, V]]]
```

The **id[cart[V,V]]** factor is not needed. To establish this, one first applies **VSNormality**:

```
In[41]:= VERTSECT[complement[composite[complement[inverse[E]], intersection[
             composite[S, SECOND], composite[inverse[E], FIRST]]]]] // VSNormality // Reverse

Out[41]= composite[intersection[
            complement[composite[complement[e], complement[composite[complement[inverse[e]],
                intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]],
            complement[composite[complement[inverse[S]], intersection[
               composite[S, SECOND], composite[inverse[e], FIRST]]]]], id[cart[V, V]]] ==
          VERTSECT[complement[composite[complement[inverse[e]],
             intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]

In[42]:= composite[intersection[
            complement[composite[complement[E], complement[composite[complement[inverse[E]],
                intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]]],
            complement[composite[complement[inverse[S]], intersection[
               composite[S, SECOND], composite[inverse[E], FIRST]]]]], id[cart[V, V]]] :=
          VERTSECT[complement[composite[complement[inverse[E]],
             intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]]
```

An application of **VSRenormality** then yields:

```
In[43]:= VERTSECT[complement[composite[complement[inverse[E]], intersection[
             composite[S, SECOND], composite[inverse[E], FIRST]]]]] // VSRenormality // Reverse

Out[43]= composite[VERTSECT[complement[composite[complement[inverse[e]],
             intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]],
           id[cart[V, V]]] == VERTSECT[complement[composite[complement[inverse[e]],
             intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]

In[44]:= composite[VERTSECT[complement[composite[complement[inverse[E]],
             intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]],
           id[cart[V, V]]] := VERTSECT[complement[composite[complement[inverse[E]],
             intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]]
```

The binary function under consideration can be expressed in terms of **LAMBHULL**. The argument needed to establish this connection is a classic combination of **symdif** and **VSNormality**.

```
In[45]:= symdif[VERTSECT[complement[composite[complement[inverse[E]],
              intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]],
            composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]]] // VSNormality

Out[45]= union[intersection[complement[VERTSECT[complement[composite[complement[inverse[e]],
               intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]],
             composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]]],
            intersection[composite[inverse[SINGLETON], Di, IMG, cross[LAMBHULL, SINGLETON]],
             VERTSECT[complement[composite[complement[inverse[e]],
                intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]] == 0

In[46]:= union[intersection[complement[VERTSECT[complement[composite[complement[inverse[e]],
               intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]],
             composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]]],
            intersection[composite[inverse[SINGLETON], Di, IMG, cross[LAMBHULL, SINGLETON]],
             VERTSECT[complement[composite[complement[inverse[e]],
                intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]] := 0

In[47]:= SubstTest[equal, 0, symdif[u, v],
              {v -> composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]],
                 u -> VERTSECT[complement[composite[complement[inverse[E]],
                intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]]}]

Out[47]= True == equal[composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]],
             VERTSECT[complement[composite[complement[inverse[e]],
                intersection[composite[S, SECOND], composite[inverse[e], FIRST]]]]]]

In[48]:= VERTSECT[complement[composite[complement[inverse[E]],
              intersection[composite[S, SECOND], composite[inverse[E], FIRST]]]]] :=
           composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]]
```

The exppression considered at the beginning of this section now becomes:

```
In[49]:= lambda[pair[x, y], APPLY[HULL[x], y]]

Out[49]= composite[inverse[SINGLETON], IMG, cross[LAMBHULL, SINGLETON]]
```

■ a formula involving FUNPART

The **VERTSECT** formula for the function **LAMBHULL** itself now simplifies:

```
In[50]:= VERTSECT[reify[x, HULL[x]]]

Out[50]= composite[FUNPART, LAMBHULL]
```

What is needed here is a variable−free formulation of this fact:

```
In[51]:= funpart[HULL[x]]

Out[51]= HULL[x]
```

This is easily derived:

```
In[52]:= symdif[composite[FUNPART, LAMBHULL], LAMBHULL] // VSNormality

Out[52]= union[composite[id[complement[FUNS]], LAMBHULL],
           intersection[complement[LAMBHULL], composite[FUNPART, LAMBHULL]]] == 0

In[53]:= union[composite[id[complement[FUNS]], LAMBHULL],
           intersection[complement[LAMBHULL], composite[FUNPART, LAMBHULL]]] := 0

In[54]:= SubstTest[equal, 0, symdif[u, v], {u -> composite[FUNPART, LAMBHULL], v -> LAMBHULL}]

Out[54]= True == equal[LAMBHULL, composite[FUNPART, LAMBHULL]]

In[56]:= composite[FUNPART, LAMBHULL] := LAMBHULL
```

■ translating some classics

The various properties of **HULL[x]** yield corresponding formulas for **LAMBHULL**. For example:

```
In[57]:= HULL[Aclosure[x]]

Out[57]= HULL[x]
```

The corresponding formula for **LAMBHULL** is derived as follows:

```
In[58]:= symdif[LAMBHULL, composite[LAMBHULL, ACLOSURE]] // VSNormality

Out[58]= union[intersection[LAMBHULL, composite[complement[LAMBHULL], ACLOSURE]],
           intersection[complement[LAMBHULL], composite[LAMBHULL, ACLOSURE]]] == 0

In[59]:= union[intersection[LAMBHULL, composite[complement[LAMBHULL], ACLOSURE]],
           intersection[complement[LAMBHULL], composite[LAMBHULL, ACLOSURE]]] := 0

In[60]:= SubstTest[equal, 0, symdif[u, v], {u -> LAMBHULL, v -> composite[LAMBHULL, ACLOSURE]}]

Out[60]= True == equal[LAMBHULL, composite[LAMBHULL, ACLOSURE]]

In[61]:= composite[LAMBHULL, ACLOSURE] := LAMBHULL
```

A more interesting connecttion between **LAMBHULL** and **ACLOSURE** is this:

```
In[62]:= symdif[ACLOSURE, composite[FIX, LAMBHULL]] // VSNormality

Out[62]= union[intersection[ACLOSURE, composite[complement[IMAGE[inverse[DUP]]], LAMBHULL]],
           intersection[complement[ACLOSURE], composite[IMAGE[inverse[DUP]], LAMBHULL]]] == 0

In[63]:= union[intersection[ACLOSURE, composite[complement[IMAGE[inverse[DUP]]], LAMBHULL]],
           intersection[complement[ACLOSURE], composite[IMAGE[inverse[DUP]], LAMBHULL]]] := 0

In[64]:= SubstTest[equal, 0, symdif[u, v], {u -> ACLOSURE, v -> composite[FIX, LAMBHULL]}]

Out[64]= True == equal[ACLOSURE, composite[IMAGE[inverse[DUP]], LAMBHULL]]

In[65]:= composite[IMAGE[inverse[DUP]], LAMBHULL] := ACLOSURE
```

This expresses the fact that **fix[HULL[x]] = Aclosure[x]** when **x** is a set. (It is unknown if a similar result holds for proper classes.)

■ domain and range

The formulas for the domain and range of **HULL[x]** imply:

```
In[68]:= composite[IMAGE[FIRST], LAMBHULL] // VSNormality
Out[68]= composite[IMAGE[FIRST], LAMBHULL] == IMAGE[inverse[S]]
```

```
In[69]:= composite[IMAGE[FIRST], LAMBHULL] := IMAGE[inverse[S]]
```

```
In[66]:= composite[IMAGE[SECOND], LAMBHULL] // VSNormality
Out[66]= composite[IMAGE[SECOND], LAMBHULL] == ACLOSURE
```

```
In[67]:= composite[IMAGE[SECOND], LAMBHULL] := ACLOSURE
```

■ some more formulas

The **HULL** formula for power sets is:

```
In[71]:= HULL[P[x]]
Out[71]= id[P[x]]
```

Knowing this, one can derive the counterpart for **LAMBHULL**:

```
In[70]:= composite[LAMBHULL, POWER] // VSNormality
Out[70]= composite[LAMBHULL, POWER] == composite[IMAGE[DUP], POWER]
```

```
In[72]:= composite[LAMBHULL, POWER] := composite[IMAGE[DUP], POWER]
```

A similar result holds for cliques:

```
In[73]:= composite[LAMBHULL, CLIQUES] // VSNormality
Out[73]= composite[LAMBHULL, CLIQUES] == composite[IMAGE[DUP], CLIQUES]
```

```
In[74]:= composite[LAMBHULL, CLIQUES] := composite[IMAGE[DUP], CLIQUES]
```

Here is yet another fact about **HULL[x]**:

```
In[75]:= image[CAP, HULL[x]]
Out[75]= image[inverse[S], x]
```

The variable−free counterpart is:

```
In[76]:= composite[IMAGE[CAP], LAMBHULL] // VSNormality
Out[76]= composite[IMAGE[CAP], LAMBHULL] == IMAGE[inverse[S]]
```

```
In[77]:= composite[IMAGE[CAP], LAMBHULL] := IMAGE[inverse[S]]
```

■ idempotence

The function **HULL[x]** is idempotent:

```
In[78]:= composite[HULL[x], HULL[x]]

Out[78]= HULL[x]
```

Because **HULL[x]** occurs on both sides of the equation, one needs to use symmetric differences to derive the variable−free counterpart:

```
In[80]:= symdif[composite[COMPOSE, DUP, LAMBHULL], LAMBHULL] // VSNormality

Out[80]= union[composite[id[complement[fix[composite[COMPOSE, DUP]]]], LAMBHULL],
           intersection[complement[LAMBHULL], composite[COMPOSE, DUP, LAMBHULL]]] == 0

In[81]:= union[composite[id[complement[fix[composite[COMPOSE, DUP]]]], LAMBHULL],
           intersection[complement[LAMBHULL], composite[COMPOSE, DUP, LAMBHULL]]] := 0

In[82]:= SubstTest[equal, 0, symdif[u, v], {u -> composite[COMPOSE, DUP, LAMBHULL], v -> LAMBHULL}]

Out[82]= True == equal[LAMBHULL, composite[COMPOSE, DUP, LAMBHULL]]
```

This is the **LAMBHULL** counterpart of the idempotence property:

```
In[83]:= composite[COMPOSE, DUP, LAMBHULL] := LAMBHULL
```

■ an inverse image rule

The inverse image rule for **LAMBHULL** can be derived most simply by a little trick:

```
In[87]:= (member[x, image[inverse[funpart[z]], y]] // AssertTest) /. z → LAMBHULL

Out[87]= member[x, image[inverse[LAMBHULL], y]] == member[HULL[x], y]

In[88]:= member[x_, image[inverse[LAMBHULL], y_]] := member[HULL[x], y]
```

Here is a simple application of this:

```
In[91]:= SubstTest[member, x, image[inverse[LAMBHULL], y], y → range[LAMBHULL]] // Reverse

Out[91]= member[HULL[x], range[LAMBHULL]] == member[x, V]

In[92]:= member[HULL[x_], range[LAMBHULL]] := member[x, V]
```

■ final comment: unfinished business

Although our original intent was to study **lambda[x,HULL[x]]**, this expresion still does not simplify to **LAMBHULL** at present. For that one needs one further rule, which is admittedly rather ugly:

```
In[84]:= Map[VERTSECT, composite[inverse[E], LAMBHULL] // VSNormality // Reverse]

Out[84]= VERTSECT[intersection[complement[
             fix[composite[complement[composite[intersection[composite[inverse[FIRST], e],
                  composite[inverse[SECOND], inverse[FIRST], inverse[S]]], complement[e]]],
               complement[inverse[e]], SECOND, SECOND]]],
            composite[inverse[FIRST], inverse[e], IMAGE[inverse[S]]],
            UB[union[composite[inverse[FIRST], complement[inverse[S]]],
              composite[inverse[SECOND], inverse[S]]]]]] == LAMBHULL
```

```
In[85]:= VERTSECT[intersection[complement[
             fix[composite[complement[composite[intersection[composite[inverse[FIRST], e],
                  composite[inverse[SECOND], inverse[FIRST], inverse[S]]], complement[e]]],
               complement[inverse[e]], SECOND, SECOND]]],
            composite[inverse[FIRST], inverse[e], IMAGE[inverse[S]]],
            UB[union[composite[inverse[FIRST], complement[inverse[S]]],
              composite[inverse[SECOND], inverse[S]]]]] := LAMBHULL
```

Although this is arguably not very satisfying, it at least permits one to verify that the function **LAMBHULL** is what it supposed to be:

```
In[86]:= lambda[x, HULL[x]]

Out[86]= LAMBHULL
```