equality substitution rules for CUT, CORE and HULL

Johan G. F. Belinfante
2003 May 9

```
In[1]:= << goedel52.r66; << tools.m

        :Package Title: goedel52.r66         2003 May 8 at 11:45 a.m.

        It is now:  2003 May 12 at 10:29

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 May 11

        weightlimit = 40
```

■ summary

In this notebook, equality substitution rules are derived for **CUT**, **CORE** and **HULL**. In the first two cases, it is possible to derive two−wayimplications, but unfortunately this appears not to be the case for the function **HULL**. Some remarks are made at the end of this notebook concerning the open problems that currently prevent deriving a corresponding two−way rule for **HULL**. The derivation would go through if certain facts known for sets could be extended to the case of proper classes.

■ CUT

The function **CUT[x] = IMAGE[id[x]]** is defined by

```
In[2]:= lambda[y, intersection[x, y]]

Out[2]= IMAGE[id[x]]
```

From the equality substitution rule for **IMAGE** one obtains:

```
In[3]:= SubstTest[implies, equal[u, v], equal[IMAGE[u], IMAGE[v]], {u -> id[x], v -> id[y]}]

Out[3]= or[equal[IMAGE[id[x]], IMAGE[id[y]]], not[equal[x, y]]] == True

In[4]:= or[equal[IMAGE[id[x_]], IMAGE[id[y_]]], not[equal[x_, y_]]] := True
```

The reverse implication also holds:

```
In[5]:= SubstTest[implies, equal[u, v], equal[range[u], range[v]],
            {u -> IMAGE[id[x]], v -> IMAGE[id[y]]}]

Out[5]= or[equal[x, y], not[equal[IMAGE[id[x]], IMAGE[id[y]]]]] == True

In[6]:= or[equal[x_, y_], not[equal[IMAGE[id[x_]], IMAGE[id[y_]]]]] := True
```

Consquently, one has a logical equivalence:

```
In[7]:= equiv[equal[IMAGE[id[x]], IMAGE[id[y]]], equal[x, y]]

Out[7]= True
```

This justifies adding the following rewrite rule:

```
In[8]:= equal[IMAGE[id[x_]], IMAGE[id[y_]]] := equal[x, y]
```

■ CORE

The function **CORE[x]** is defined as follows: (if **x** were a topology, this would be the function that maps a set to its interior)

```
In[9]:= lambda[y, U[intersection[x, P[y]]]]

Out[9]= CORE[x]
```

This function is related to cutting by:

```
In[10]:= composite[BIGCUP, IMAGE[id[x]], POWER]

Out[10]= CORE[x]
```

This yields:

```
In[11]:= SubstTest[implies, equal[u, v], equal[composite[t, u, w], composite[t, v, w]],
             {u -> IMAGE[id[x]], v -> IMAGE[id[y]], t -> BIGCUP, w -> POWER}]

Out[11]= or[equal[CORE[x], CORE[y]], not[equal[x, y]]] == True

In[12]:= or[equal[CORE[x_], CORE[y_]], not[equal[x_, y_]]] := True
```

To obtain a two−way rule, one needs to introduce **Uclosure**.

```
In[13]:= SubstTest[implies, equal[u, v], equal[CORE[u], CORE[v]],
             {u -> Uclosure[x], v -> Uclosure[y]}]

Out[13]= or[equal[CORE[x], CORE[y]], not[equal[Uclosure[x], Uclosure[y]]]] == True

In[14]:= or[equal[CORE[x_], CORE[y_]], not[equal[Uclosure[x_], Uclosure[y_]]]] := True
```

The reverse implication holds:

```
In[15]:= SubstTest[implies, equal[u, v], equal[fix[u], fix[v]],
             {u -> CORE[x], v -> CORE[y]}]

Out[15]= or[equal[Uclosure[x], Uclosure[y]], not[equal[CORE[x], CORE[y]]]] == True

In[16]:= or[equal[Uclosure[x_], Uclosure[y_]], not[equal[CORE[x_], CORE[y_]]]] := True

In[17]:= equiv[equal[CORE[x], CORE[y]], equal[Uclosure[x], Uclosure[y]]]

Out[17]= True
```

This justifies the formula:

```
In[18]:= equal[CORE[x_], CORE[y_]] := equal[Uclosure[x], Uclosure[y]]
```

■ HULL

The meaning of **HULL[x]** is: (if **x** were the set of closed sets of a topological space, this would be the closure operation)

```
In[19]:= lambda[y, A[intersection[x, image[S, singleton[y]]]]]

Out[19]= HULL[x]
```

This function can be defined as:

```
In[20]:= VERTSECT[complement[composite[complement[inverse[E]], id[x], S]]]

Out[20]= HULL[x]
```

The key formula for the derivation of the substitution rule is:

```
In[21]:= composite[complement[inverse[E]], id[x], S]

Out[21]= composite[complement[inverse[∈]], HULL[x]]
```

The first step resembles that used for **CORE**.

```
In[22]:= SubstTest[implies, equal[u, v], equal[composite[t, u, w], composite[t, v, w]],
             {u -> id[x], v -> id[y], t -> complement[inverse[E]], w -> S}]

Out[22]= or[equal[composite[complement[inverse[∈]], HULL[x]],
            composite[complement[inverse[∈]], HULL[y]]], not[equal[x, y]]] == True

In[23]:= or[equal[composite[complement[inverse[E]], HULL[x_]],
            composite[complement[inverse[E]], HULL[y_]]], not[equal[x_, y_]]] := True
```

The following technical step is needed to cope with complements:

```
In[24]:= SubstTest[equal, complement[u], complement[v],
              {u -> composite[complement[inverse[E]], id[x], S],
            v -> composite[complement[inverse[E]], id[y], S]}]

Out[24]= equal[
           union[complement[cart[image[inverse[S], x], V]], composite[inverse[∈], HULL[x]]],
           union[complement[cart[image[inverse[S], y], V]], composite[inverse[∈], HULL[y]]]] ==
          equal[composite[complement[inverse[∈]], HULL[x]],
           composite[complement[inverse[∈]], HULL[y]]]

In[25]:= equal[union[complement[cart[image[inverse[S], x_], V]],
            composite[inverse[E], HULL[x_]]], union[
            complement[cart[image[inverse[S], y_], V]], composite[inverse[E], HULL[y_]]]] :=
          equal[composite[complement[inverse[E]], HULL[x]],
           composite[complement[inverse[E]], HULL[y]]]
```

The next step uses equality substitution for **VERTSECT**.

```
In[26]:= SubstTest[implies, equal[u, v], equal[VERTSECT[u], VERTSECT[v]],
             {u -> complement[composite[complement[inverse[E]], id[x], S]],
           v -> complement[composite[complement[inverse[E]], id[y], S]]}]

Out[26]= or[equal[HULL[x], HULL[y]], not[equal[composite[complement[inverse[e]], HULL[x]],
             composite[complement[inverse[e]], HULL[y]]]]] == True

In[27]:= or[equal[HULL[x_], HULL[y_]], not[equal[composite[complement[inverse[E]], HULL[x_]],
             composite[complement[inverse[E]], HULL[y_]]]]] := True
```

The final step is to put all this together:

```
In[28]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                 {p1 -> equal[x, y], p2 -> equal[composite[complement[inverse[E]], HULL[x]],
               composite[complement[inverse[E]], HULL[y]]],
                     p3 -> equal[HULL[x], HULL[y]]}]]

Out[28]= or[equal[HULL[x], HULL[y]], not[equal[x, y]]] == True

In[29]:= or[equal[HULL[x_], HULL[y_]], not[equal[x_, y_]]] := True
```

■ an unsolved problem

The question arises whether one could find a converse rule, similar to the equality rule for **CORE**. The first step would be to use:

```
In[30]:= SubstTest[implies, equal[u, v], equal[HULL[u], HULL[v]],
             {u -> Aclosure[x], v -> Aclosure[y]}]

Out[30]= or[equal[HULL[x], HULL[y]], not[equal[Aclosure[x], Aclosure[y]]]] == True

In[31]:= or[equal[HULL[x_], HULL[y_]], not[equal[Aclosure[x_], Aclosure[y_]]]] := True
```

Unfortunately, one can not at present derive a reverse implication here because it remains unknown whether **Aclosure[x]** is equal to **fix[HULL[x]]**. All that is currently known is an inclusion:

```
In[32]:= subclass[Aclosure[x], fix[HULL[x]]]

Out[32]= True
```

In the special case of sets, one does have such an equation:

```
In[33]:= implies[member[x, V], equal[Aclosure[x], fix[HULL[x]]]]

Out[33]= True
```

There are no known counterexamples to the conjecture that this extends to proper classes, but no proof that **Aclosure[x]** and **fix[HULL[x]]** are equal is known either. It is also unkown whether the following statement holds for proper classes:

```
In[34]:= HULL[fix[HULL[x]]] == HULL[x]

Out[34]= HULL[fix[HULL[x]]] == HULL[x]
```

All that is known at present is this rule, which implies that the above formula holds when **x** is a set.

```
In[35]:= HULL[Aclosure[x]]
Out[35]= HULL[x]
```