hull[x,y]

Johan G. F. Belinfante
2004 March 29

```
In[1]:= << goedel55.23c; << tools.m;

    :Package Title: goedel55.23c          2004 March 23 at 11:45 p.m.

    It is now:  2004 Mar 31 at 10:17

    Loading Simplification Rules

    TOOLS.M                      Revised 2004 March 14

    weightlimit = 40

In[2]:= Begin["Goedel‘Private‘"];
```

summary

A wrapped membership rule is introduced for **hull[x,y]**, and replacements for a large number of existing rewrite rules are derived. By delaying the normalization of **hull**, existing rules can conveniently be used to help derive their replacements.

definitions

The definition of **hull** is given in wrapped form. Unless special measures are taken, a default rule for wrapped membership would prevent the **GOEDEL** program from recognizing this new definition. To get around this problem, the dummy variable in **class** is called **u_** so that sorting the down values for **class** will cause the new rule to precede the default rule.

```
In[3]:= class[u_, member[v_, hull[x_, y_]]] := Module[{w = Unique[]},
          class[u, forall[w, implies[and[member[w, x], subclass[y, w]], member[v, w]]]]]
```

The wrapped definition of **trv[x]** will also be replaced:

```
In[4]:= class[u_, member[x_, trvnew[y_]]] := Module[{z = Unique[]}, class[u,
           exists[z, and[member[x, hull[TRV, z]], subclass[z, y], subclass[z, cart[V, V]]]]]]

In[5]:= DownValues[class] = Sort[DownValues[class]];
```

The correctness of this replacement for **trv[x]** will now be verified, but no further use of **trvnew[x]** will be made of this in the rest of this notebok.

```
In[6]:= trvnew[x] // Normality

Out[6]= True
```

Normalization

The new constructor will not immediately be normalized, but instead **Normality** will be used to derive a temporary equality rule:

```
In[7]:= Map[equal[hull[x, y], #] &, hull[x, y] // Normality // Reverse]

    Reverse::normal : Nonatomic expression expected at position 1 in Reverse[True]. More…

Out[7]= equal[True, hull[x, y]]
```

```
In[8]:= equal[A[intersection[x_, image[S, singleton[y_]]]], hull[x_, y_]] := True
```

When **hull** is later normalized, the equality rule will no longer be needed. Later in this noteook it will be replaced with a rewrite rule, but only after various replacement rules have been derived.

deriving formulas for hull using equality

Delaying the normalization of **hull** allows one to take advantage of existing rewrite rules to derive the needed replacement rules using equality substitution. For example:

```
In[9]:= SubstTest[implies, and[equal[u, v], subclass[y, v]], subclass[y, u],
         {u -> hull[x, y], v -> A[intersection[x, image[S, singleton[y]]]]}]

Out[9]= subclass[y, hull[x, y]] == True
```

```
In[10]:= subclass[y_, hull[x_, y_]] := True
```

rules for argument 0

Two rules can be derived when one of the arguments is the empty set.

```
In[11]:= hull[0, x] // Normality

Out[11]= True
```

```
In[12]:= hull[0, x_] := V
```

```
In[13]:= hull[x, 0] // Normality

Out[13]= True
```

```
In[14]:= hull[x_, 0] := A[x]
```

rules for argument V

When one of the aruments of **hull** is the universal class **V**, one has:

```
In[15]:= hull[V, x] // Normality

Out[15]= True

In[16]:= hull[V, x_] := union[x, complement[image[V, singleton[x]]]]

In[17]:= hull[x, V] // Normality

Out[17]= True

In[18]:= hull[x_, V] := V
```

sethood rules

The class **hull[x,y]** need not be a set. The following sethood rules hold:

```
In[19]:= member[hull[x, y], V] // AssertTest

Out[19]= True

In[20]:= member[hull[x_, y_], V] := member[y, image[inverse[S], x]]
```

Note that when **hull[x,y]** is not a set, it must be **V**.

```
In[21]:= equal[V, hull[x, y]] // AssertTest

Out[21]= True

In[22]:= equal[V, hull[x_, y_]] := not[member[y, image[inverse[S], x]]]
```

equality rules for hull

One needs lemmas to derive equality substitution rules for each of the arguments of **hull**.

```
In[23]:= SubstTest[implies, equal[u, v], equal[image[w, u], image[w, v]],
          {u -> x, v -> y, w -> composite[complement[inverse[E]], id[image[S, singleton[z]]]]}]

Out[23]= or[equal[A[intersection[x, image[S, singleton[z]]]],
            A[intersection[y, image[S, singleton[z]]]]], not[equal[x, y]]] == True

In[24]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed

In[25]:= implies[equal[x, y], equal[hull[x, z], hull[y, z]]] // AssertTest

Out[25]= True

In[26]:= or[equal[hull[x_, z_], hull[y_, z_]], not[equal[x_, y_]]] := True
```

```
In[27]:= SubstTest[implies, equal[u, v], equal[image[w, u], image[w, v]],
          {u -> image[S, singleton[y]], v -> image[S, singleton[z]],
           w -> composite[complement[inverse[E]], id[x]]}]

Out[27]= or[equal[A[intersection[x, image[S, singleton[y]]]],
            A[intersection[x, image[S, singleton[z]]]]],
           not[equal[singleton[y], singleton[z]]]] == True

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

In this case, some additional reasoning is required:

```
In[29]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 -> equal[y, z], p2 -> equal[singleton[y], singleton[z]],
            p3 -> equal[A[intersection[x, image[S, singleton[y]]]],
              A[intersection[x, image[S, singleton[z]]]]]}]]

Out[29]= or[equal[A[intersection[x, image[S, singleton[y]]]],
            A[intersection[x, image[S, singleton[z]]]]], not[equal[y, z]]] == True

In[30]:= (% /. {x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

Finally, one needs to use **AssertTest**:

```
In[31]:= implies[equal[y, z], equal[hull[x, y], hull[x, z]]] // AssertTest

Out[31]= True

In[32]:= or[equal[hull[x_, y_], hull[x_, z_]], not[equal[y_, z_]]] := True
```

monotonicity properties of hull

The use of **AssertTest** yields a monotonicity property with respect to the second argument:

```
In[33]:= implies[subclass[y, z], subclass[hull[x, y], hull[x, z]]] // AssertTest

Out[33]= True

In[34]:= or[not[subclass[y_, z_]], subclass[hull[x_, y_], hull[x_, z_]]] := True
```

For the first argument, monotonicity does not hold, but instead one has an antitone property, which one can derive using **AssertTest**.

```
In[35]:= implies[subclass[x, y], subclass[hull[y, z], hull[x, z]]] // AssertTest

Out[35]= True

In[36]:= or[not[subclass[x_, y_]], subclass[hull[y_, z_], hull[x_, z_]]] := True
```

idempotence of hull

To derive the idempotence of **hull**, the use of **AssertTest** does not suffice. In one direction, no new rule is needed

```
In[37]:= subclass[hull[x, y], hull[x, hull[x, y]]]

Out[37]= True
```

From the idempotence of **HULL[x]** one finds:

```
In[38]:= Map[assert[implies[member[y, image[inverse[S], x]],
              member[A[intersection[x, image[S, singleton[y]]]], #]]] &,
            ImageComp[HULL[x], HULL[x], singleton[y]]] // Reverse

Out[38]= or[not[member[y, image[inverse[S], x]]], subclass[A[
              intersection[x, image[S, singleton[A[intersection[x, image[S, singleton[y]]]]]]]],
            A[intersection[x, image[S, singleton[y]]]]]] == True

In[39]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

This yields a partial result:

```
In[40]:= implies[member[y, image[inverse[S], x]],
            equal[hull[x, hull[x, y]], hull[x, y]]] // AssertTest

Out[40]= or[equal[hull[x, y], hull[x, hull[x, y]]], not[member[y, image[inverse[S], x]]]] ==
           or[not[member[y, image[inverse[S], x]]], subclass[hull[x, hull[x, y]], hull[x, y]]]

In[41]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

        RuleDelayed::rhs : Pattern y_ appears on the right-hand side of rule
           or[equal[hull[x_, hull[x_, y_]], hull[x_, y_]], not[member[y_, image[inverse[S], x_]]]] :>
            or[not[member[y_, image[inverse[S], x_]]], subclass[hull[x_, hull[x_, <<2>>]], <<1>>]]. More…

        RuleDelayed::rhs : Pattern x_ appears on the right-hand side of rule
           or[equal[hull[x_, hull[x_, y_]], hull[x_, y_]], not[member[y_, image[inverse[S], x_]]]] :>
            or[not[member[y_, image[inverse[S], x_]]], subclass[hull[x_, hull[x_, <<2>>]], <<1>>]]. More…

        RuleDelayed::rhs : Pattern x_ appears on the right-hand side of rule
           or[equal[hull[x_, hull[x_, y_]], hull[x_, y_]], not[member[y_, image[inverse[S], x_]]]] :>
            or[not[member[y_, image[inverse[S], x_]]], subclass[hull[x_, hull[x_, <<2>>]], <<1>>]]. More…

        General::stop : Further output of RuleDelayed::rhs will be suppressed during this calculation. More…
```

For the case that **y** does not belong to **image[inverse[S],x]** one needs two lemmas. This is the first:

```
In[42]:= SubstTest[implies, and[equal[u, v], member[v, w]], member[u, w],
             {u -> V, v -> hull[x, y], w -> image[inverse[S], x]}]

Out[42]= or[member[y, image[inverse[S], x]],
            not[member[hull[x, y], image[inverse[S], x]]]] == True

In[43]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The second lemma is this:

```
In[44]:= SubstTest[implies, and[equal[u, v], equal[v, w]],
           equal[u, w], {u -> hull[x, hull[x, y]], v -> V, w -> hull[x, y]}]

Out[44]= or[equal[hull[x, y], hull[x, hull[x, y]]],
            member[y, image[inverse[S], x]], member[hull[x, y], image[inverse[S], x]]] == True

In[45]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The idempotence result now follows:

```
In[46]:= SubstTest[and, implies[p1, p2],
           implies[not[p1], or[p2, p3]], implies[not[p1], not[p3]],
            {p1 -> member[y, image[inverse[S], x]], p2 -> equal[hull[x, y], hull[x, hull[x, y]]],
            p3 -> member[hull[x, y], image[inverse[S], x]]}] // MapNotNot // Reverse

Out[46]= equal[hull[x, y], hull[x, hull[x, y]]] == or[not[member[y_, image[inverse[S], x_]]],
           subclass[hull[x_, hull[x_, y_]], hull[x_, y_]]]

In[47]:= Equal[hull[x, y], hull[x, hull[x, y]]] // Reverse

Out[47]= hull[x, hull[x, y]] == hull[x, y]

In[48]:= hull[x_, hull[x_, y_]] := hull[x, y]
```

miscellaneous hull rules

The rules derived in this section are replacements for existing rewrite rules for **hull**.

```
In[49]:= or[equal[hull[x, y], U[image[HULL[x], P[y]]]],
           not[member[y, image[inverse[S], x]]]] // AssertTest

Out[49]= True

In[50]:= or[equal[hull[x_, y_], U[image[HULL[x_], P[y_]]]],
           not[member[y_, image[inverse[S], x_]]]] := True

In[51]:= or[equal[hull[x, union[y, z]], union[hull[x, y], hull[x, z]]],
           not[subclass[image[CUP, cart[x, x]], x]]] // AssertTest

Out[51]= True

In[52]:= or[equal[hull[x_, union[y_, z_]], union[hull[x_, y_], hull[x_, z_]]],
           not[subclass[image[CUP, cart[x_, x_]], x_]]] := True

In[53]:= subclass[image[trv[x], y], hull[invar[x], y]] // AssertTest

Out[53]= True

In[54]:= subclass[image[trv[x_], y_], hull[invar[x_], y_]] := True
```

examples

Rewrite rules that eliminate **hull** can often be derived using **Normality**. For example:

```
In[55]:= hull[image[S, singleton[x]], y] // Normality

Out[55]= True

In[56]:= hull[image[S, singleton[x_]], y_] :=
          union[x, y, complement[image[V, singleton[x]]], complement[image[V, singleton[y]]]]

In[57]:= hull[FULL, x] // Normality

Out[57]= True
```

```
In[58]:= hull[FULL, x_] := union[complement[image[V, singleton[x]]], tc[x]]
```

The following rules replace two other existing rules:

```
In[59]:= hull[ACYCLIC, x] // Normality

Out[59]= True

In[60]:= hull[ACYCLIC, x_] :=
          union[x, complement[image[V, intersection[ACYCLIC, singleton[x]]]]]

In[61]:= hull[range[POWER], x] // Normality

Out[61]= True

In[62]:= hull[range[POWER], x_] := union[complement[image[V, singleton[x]]], P[U[x]]]
```

hull[TRV,x]

The case of **hull[TRV,x]** is of special interest in connection with the transitive closure **trv[x]**. For sets, these are equal, but not for proper classes.

```
In[63]:= subclass[trv[x], hull[TRV, x]] // AssertTest

Out[63]= True

In[64]:= subclass[trv[x_], hull[TRV, x_]] := True

In[65]:= SubstTest[implies, and[equal[u, v], equal[u, w]], equal[v, w],
          {u -> A[intersection[TRV, image[S, singleton[x]]]], v -> hull[TRV, x], w -> trv[x]}]

Out[65]= or[equal[hull[TRV, x], trv[x]], not[member[x, V]], not[subclass[x, cart[V, V]]]] == True

In[66]:= (% /. x -> x_) /. Equal -> SetDelayed

In[67]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
          {u -> A[intersection[TRV, image[S, singleton[x]]]], v -> hull[TRV, x], w -> trv[x]}]

Out[67]= or[and[member[x, V], subclass[x, cart[V, V]]], not[equal[hull[TRV, x], trv[x]]]] == True

In[68]:= (% /. x -> x_) /. Equal -> SetDelayed

In[69]:= equiv[equal[trv[x], hull[TRV, x]], and[member[x, V], subclass[x, cart[V, V]]]]

Out[69]= True

In[70]:= equal[hull[TRV, x_], trv[x_]] := and[member[x, V], subclass[x, cart[V, V]]]

In[71]:= subclass[hull[TRV, x], cart[V, V]] // AssertTest

Out[71]= True

In[72]:= subclass[hull[TRV, x_], cart[V, V]] := and[member[x, V], subclass[x, cart[V, V]]]
```

```
In[73]:= Map[equal[0, fix[#]] &, hull[TRV, x] // Normality]

Out[73]= True

In[74]:= equal[0, fix[hull[TRV, x_]]] :=
          and[equal[0, fix[trv[x]]], member[x, V], subclass[x, cart[V, V]]]

In[75]:= TRANSITIVE[composite[Id, hull[TRV, x]]] // AssertTest

Out[75]= TRANSITIVE[composite[Id, hull[TRV, x]]] == True

In[76]:= TRANSITIVE[composite[Id, hull[TRV, x_]]] := True

In[77]:= TRANSITIVE[hull[TRV, x]] // AssertTest

Out[77]= TRANSITIVE[hull[TRV, x]] == and[member[x, V], subclass[x, cart[V, V]]]

In[78]:= TRANSITIVE[hull[TRV, x_]] := and[member[x, V], subclass[x, cart[V, V]]]
```

union rules

To derive rewrite rules in which **hull** appears on the right side, it is convenient to normalize this constructor:

```
In[79]:= hull[x, y] // Normality // Reverse

    Reverse::normal : Nonatomic expression expected at position 1 in Reverse[True]. More…

Out[79]= Reverse[True]

In[80]:= A[intersection[x_, image[S, singleton[y_]]]] := hull[x, y]
```

For example, one can derive this rule:

```
In[81]:= hull[union[x, y], z] // Normality

Out[81]= True

In[82]:= hull[union[x_, y_], z_] := intersection[hull[x, z], hull[y, z]]
```

Aclosure rules

```
In[83]:= hull[Aclosure[x], y] // Normality

Out[83]= True

In[84]:= hull[Aclosure[x_], y_] := hull[x, y]

In[85]:= SubstTest[implies, member[y, fix[z]], member[pair[y, y], z], z -> HULL[x]] // MapNotNot

Out[85]= or[equal[y, hull[x, y]], not[member[y, fix[HULL[x]]]]] == True

In[86]:= or[equal[y_, hull[x_, y_]], not[member[y_, fix[HULL[x_]]]]] := True
```

```
In[87]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], not[implies[p1, p3]], {p1 -> member[y, Aclosure[x]],
            p2 -> member[y, fix[HULL[x]]], p3 -> equal[y, hull[x, y]]}]]

Out[87]= or[equal[y, hull[x, y]], not[member[y, Aclosure[x]]]] == True

In[88]:= or[equal[y_, hull[x_, y_]], not[member[y_, Aclosure[x_]]]] := True
```

The following rule is new:

```
In[89]:= hull[x, union[y, complement[image[V, z]]]] // Normality

Out[89]= True

In[90]:= hull[x_, union[y_, complement[image[V, z_]]]] :=
          union[complement[image[V, z]], hull[x, y]]
```

rules with hull on the right side

There are a few rewrite rules in which **hull[x,y]** appears only on the right side. Here are two of these:

```
In[91]:= member[x, HULL[y]]

Out[91]= and[equal[hull[y, first[x]], second[x]], member[first[x], image[inverse[S], y]]]

In[92]:= member[pair[x, y], HULL[z]]

Out[92]= and[equal[y, hull[z, x]], member[x, V], member[y, V]]
```

The existing rules that will be removed are:

```
In[93]:= InfoMatch[member[x_, HoldPattern[HULL[y_]]]]

Out[93]//TableForm=
       member[pair[x_, y_], HULL[z_]] := and[equal[A[intersection[z, image[S, singleton[x]]]]
       member[x_, HULL[y_]] := and[equal[A[intersection[y, image[S, singleton[first[x]]]]], s
```

The old rules are removed, and replaced with new ones:

```
In[94]:= member[x_, HULL[y_]] =.

In[95]:= member[x_, HULL[y_]] :=
          and[equal[hull[y, first[x]], second[x]], member[first[x], image[inverse[S], y]]]

In[96]:= member[pair[x_, y_], HULL[z_]] =.

In[97]:= member[pair[x_, y_], HULL[z_]] := and[equal[y, hull[z, x]], member[x, V], member[y, V]]
```

There are some related rules of the same type.

```
In[98]:= member[x_, composite[y_, HULL[z_]]] =.

In[99]:= member[x, composite[y, HULL[z]]] // AssertTest

Out[99]= member[x, composite[y, HULL[z]]] == and[member[first[x], image[inverse[S], z]],
           member[pair[hull[z, first[x]], second[x]], y]]
```

In[100]:=
```
member[x_, composite[y_, HULL[z_]]] := and[
  member[first[x], image[inverse[S], z]], member[pair[hull[z, first[x]], second[x]], y]]
```

In[101]:=
```
member[x_, composite[inverse[HULL[y_]], z_]] =.
```

In[102]:=
```
member[x, composite[inverse[HULL[y]], z]] // AssertTest
```

Out[102]=
```
member[x, composite[inverse[HULL[y]], z]] ==
 and[member[pair[first[x], hull[y, second[x]]], z],
  member[second[x], image[inverse[S], y]]]
```

In[103]:=
```
member[x_, composite[inverse[HULL[y_]], z_]] :=
 and[member[pair[first[x], hull[y, second[x]]], z],
  member[second[x], image[inverse[S], y]]]
```

inverse image rule

Here is another one that needs to be replaced:

In[104]:=
```
member[x, image[inverse[HULL[y]], z]]
```

Out[104]=
```
and[member[x, V], member[hull[y, x], z]]
```

In[105]:=
```
member[x_, image[inverse[HULL[y_]], z_]] =.
```

The replacement rule is simpler:

In[106]:=
```
member[x, image[inverse[HULL[y]], z]] // AssertTest
```

Out[106]=
```
member[x, image[inverse[HULL[y]], z]] == member[hull[y, x], z]
```

In[107]:=
```
member[x_, image[inverse[HULL[y_]], z_]] := member[hull[y, x], z]
```

Note that when **hull[y,x]** is a set, then **x** also must be a set because **x** is contained in **hull[y,x]**.

In[108]:=
```
SubstTest[implies, and[subclass[y, v], member[v, z]], member[y, V], v -> hull[x, y]]
```

Out[108]=
```
or[member[y, V], not[member[hull[x, y], z]]] == True
```

In[109]:=
```
or[member[y_, V], not[member[hull[x_, y_], z_]]] := True
```

In[110]:=
```
equiv[and[member[y, V], member[hull[x, y], z]], member[hull[x, y], z]]
```

Out[110]=
```
True
```

In[111]:=
```
and[member[y_, V], member[hull[x_, y_], z_]] := member[hull[x, y], z]
```

the APPLY rule

The **APPLY** rule is another rule that will need to be replaced. The old rule produces this result:

In[112]:=
```
APPLY[HULL[x], y]
```

Out[112]=
```
hull[x, y]
```

In[113]:=
```
APPLY[HULL[x_], y_] =.
```

To derive the replacement, some lemmas are needed.

In[114]:=
```
image[V, singleton[hull[x, y]]] // Normality
```

Out[114]=
```
image[V, singleton[hull[x, y]]] == image[V, intersection[x, image[S, singleton[y]]]]
```

In[115]:=
```
image[V, singleton[hull[x_, y_]]] := image[V, intersection[x, image[S, singleton[y]]]]
```

In[116]:=
```
equal[union[complement[image[V, intersection[x, image[S, singleton[y]]]]], hull[x, y]],
 hull[x, y]]
```

Out[116]=
```
True
```

In[117]:=
```
union[complement[image[V, intersection[x_, image[S, singleton[y_]]]]], hull[x_, y_]] :=
 hull[x, y]
```

In[118]:=
```
SubstTest[A, image[w, singleton[y]], w -> HULL[x]] // Reverse
```

Out[118]=
```
APPLY[HULL[x], y] == hull[x, y]
```

In[119]:=
```
APPLY[HULL[x_], y_] := hull[x, y]
```

The vertical section rule is closely related:

In[120]:=
```
image[HULL[x], singleton[y]]
```

Out[120]=
```
singleton[hull[x, y]]
```

In[121]:=
```
image[HULL[x_], singleton[y_]] =.
```

The new **APPLY** rule can be used to derive the replacement rule:

In[122]:=
```
SubstTest[image, funpart[w], singleton[y], w -> HULL[x]]
```

Out[122]=
```
image[HULL[x], singleton[y]] == singleton[hull[x, y]]
```

In[123]:=
```
image[HULL[x_], singleton[y_]] := singleton[hull[x, y]]
```

reify rule

In[124]:=
```
SubstTest[reify, x, A[intersection[f[x], image[z, singleton[g[x]]]]], z -> S]
```

Out[124]=
```
reify[x, hull[f[x], g[x]]] == composite[Id, complement[composite[complement[inverse[E]],
    intersection[composite[S, VERTSECT[reify[x, g[x]]]], reify[x, f[x]]]]]]
```

In[125]:=
```
reify[x_, hull[y_, z_]] := composite[Id, complement[composite[complement[inverse[E]],
    intersection[composite[S, VERTSECT[reify[x, z]]], reify[x, y]]]]]
```