HULL[intersection[x,y]] rules

Johan G. F. Belinfante
2004 October 7

```
In[1]:= SetDirectory["i:"]; << goedel62.05a; << tools.m

        :Package Title: goedel62.05a          2004 October 5 at 6:05 a.m.

        It is now:  2004 Oct 7 at 10:46

        Loading Simplification Rules

        TOOLS.M                      Revised 2004 September 25

        weightlimit = 40
```

summary

If **x** is a subclass of **y**, then the composite of the functions **HULL[x]** and **HULL[y]** in either order is equal to **HULL[x]**. This result, which is analogous to a corresponding rule for the composite of **CORE[x]** and **CORE[y]**, is derived in this notebook. The method used to prove the **CORE** result does not generalize to the case of **HULL**, so a different argument needs to be used. The main fact used in the derivation is that the composite of **HULL[x]** and **HULL[y]** is a **HULL** function if **fix[HULL[y]]** is invariant under **HULL[x]**.

```
In[2]:= implies[invariant[HULL[x], fix[HULL[y]]], equal[composite[HULL[x], HULL[y]],
          HULL[intersection[fix[HULL[x]], fix[HULL[y]]]]]]

Out[2]= True
```

One can avoid the condition **subclass[x,y]** by replacing **x** with **intersection[x,y]**. This trick allows one to replace some reasoning with rewrites, but the price one pays for this is that the needed rewrite rules must first be derived. Hopefully other applications in the future may also be able to take advantage of such **intersection** rules.

invariance results

The fact that **fix[HULL[x]]** depends monotonically on **x** yields this **intersection** rewrite rule:

```
In[3]:= SubstTest[implies, subclass[w, x],
         subclass[fix[HULL[w]], fix[HULL[x]]], w → intersection[x, y]]

Out[3]= subclass[fix[HULL[intersection[x, y]]], fix[HULL[x]]] == True

In[4]:= subclass[fix[HULL[intersection[x_, y_]]], fix[HULL[x_]]] := True
```

Equivalently:

```
In[5]:= equal[intersection[fix[HULL[intersection[x, y]]], fix[HULL[x]]],
         fix[HULL[intersection[x, y]]]]

Out[5]= True
```

This is made into a new rewrite rule:

```
In[6]:= intersection[fix[HULL[x_]], fix[HULL[intersection[x_, y_]]]] :=
         fix[HULL[intersection[x, y]]]
```

Corollary:

```
In[7]:= ImageComp[HULL[x], id[fix[HULL[x]]], fix[HULL[intersection[x, y]]]] // Reverse

Out[7]= image[HULL[x], fix[HULL[intersection[x, y]]]] == fix[HULL[intersection[x, y]]]

In[8]:= image[HULL[x_], fix[HULL[intersection[x_, y_]]]] :=
         fix[HULL[intersection[x, y]]]
```

It follows that the composite of these **HULL** functions is a **HULL** function:

```
In[9]:= SubstTest[implies, invariant[HULL[x], fix[HULL[z]]],
         equal[composite[HULL[x], HULL[z]],
          HULL[intersection[fix[HULL[x]], fix[HULL[z]]]]], z → intersection[x, y]]

Out[9]= equal[composite[HULL[x], HULL[intersection[x, y]]],
          HULL[intersection[x, y]]] == True

In[10]:= composite[HULL[x_], HULL[intersection[x_, y_]]] := HULL[intersection[x, y]]
```

composite in the reverse order

Lemma.

```
In[11]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → fix[HULL[intersection[x, y]]],
           v → fix[HULL[x]], w → HULL[intersection[x, y]]}]

Out[11]= subclass[fix[HULL[intersection[x, y]]],
           image[HULL[intersection[x, y]], fix[HULL[x]]]] == True

In[12]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The reverse inclusion also holds, so one contains an equation:

```
In[13]:= SubstTest[and, subclass[u, v],
          subclass[v, u], {u -> fix[HULL[intersection[x, y]]],
           v -> image[HULL[intersection[x, y]], fix[HULL[x]]]}]

Out[13]= True == equal[fix[HULL[intersection[x, y]]],
           image[HULL[intersection[x, y]], fix[HULL[x]]]]

In[14]:= image[HULL[intersection[x_, y_]], fix[HULL[x_]]] :=
          fix[HULL[intersection[x, y]]]
```

Corollary.

```
In[15]:= SubstTest[implies, invariant[HULL[z], fix[HULL[x]]],
          equal[composite[HULL[z], HULL[x]],
           HULL[intersection[fix[HULL[z]], fix[HULL[x]]]], z → intersection[x, y]]

Out[15]= equal[composite[HULL[intersection[x, y]], HULL[x]],
           HULL[intersection[x, y]]] == True

In[16]:= composite[HULL[intersection[x_, y_]], HULL[x_]] := HULL[intersection[x, y]]
```

replacing intersection with subclass literals

If one prefers reasoning to rewrites, it may be useful to replace the **intersection** rules derived above with equivalent statements involving **subclass** literals, which one can do as follows:

```
In[17]:= SubstTest[implies, equal[z, intersection[x, y]],
          equal[composite[HULL[z], HULL[y]], HULL[z]], z → x]

Out[17]= or[equal[composite[HULL[x], HULL[y]], HULL[x]], not[subclass[x, y]]] == True

In[18]:= or[equal[composite[HULL[x_], HULL[y_]], HULL[x_]],
           not[subclass[x_, y_]]] := True

In[19]:= SubstTest[implies, equal[z, intersection[x, y]],
          equal[composite[HULL[y], HULL[z]], HULL[z]], z → x]

Out[19]= or[equal[composite[HULL[y], HULL[x]], HULL[x]], not[subclass[x, y]]] == True

In[20]:= or[equal[composite[HULL[y_], HULL[x_]], HULL[x_]],
           not[subclass[x_, y_]]] := True
```

some applications

The intersection of the class **SYM** of all small symmetric relations and the class **TRV** of all small transitive relations is the class **EQV** of all small equivalence relations.

```
In[21]:= SubstTest[composite, HULL[x], HULL[intersection[x, y]], {x → SYM, y → TRV}]

Out[21]= composite[HULL[SYM], HULL[EQV]] == HULL[EQV]

In[22]:= composite[HULL[SYM], HULL[EQV]] := HULL[EQV]

In[23]:= SubstTest[composite, HULL[y], HULL[intersection[x, y]], {x → SYM, y → TRV}]

Out[23]= composite[HULL[TRV], HULL[EQV]] == HULL[EQV]

In[24]:= composite[HULL[TRV], HULL[EQV]] := HULL[EQV]

In[25]:= SubstTest[composite, HULL[intersection[x, y]], HULL[x], {x → SYM, y → TRV}]

Out[25]= composite[HULL[EQV], HULL[SYM]] == HULL[EQV]

In[26]:= composite[HULL[EQV], HULL[SYM]] := HULL[EQV]

In[27]:= SubstTest[composite, HULL[intersection[x, y]], HULL[y], {x → SYM, y → TRV}]

Out[27]= composite[HULL[EQV], HULL[TRV]] == HULL[EQV]

In[28]:= composite[HULL[EQV], HULL[TRV]] := HULL[EQV]
```

Some further corollaries can be derived using **ImageComp**.

```
In[29]:= ImageComp[HULL[EQV], HULL[TRV], V] // Reverse

Out[29]= image[HULL[EQV], TRV] == EQV
```

```
In[30]:= image[HULL[EQV], TRV] := EQV

In[31]:= ImageComp[HULL[SYM], HULL[EQV], V] // Reverse

Out[31]= image[HULL[SYM], EQV] == EQV

In[32]:= image[HULL[SYM], EQV] := EQV

In[33]:= ImageComp[HULL[TRV], HULL[EQV], V] // Reverse

Out[33]= image[HULL[TRV], EQV] == EQV

In[34]:= image[HULL[TRV], EQV] := EQV
```

---

comment

The composite of **HULL[TRV]** and **HULL[SYM]** is **HULL[EQV]**, reflecting the fact that the transitive closure of a symmetric relation is symmetric.

```
In[35]:= composite[HULL[TRV], HULL[SYM]]

Out[35]= HULL[EQV]

In[36]:= implies[SYMMETRIC[x], SYMMETRIC[trv[x]]]

Out[36]= True
```

On the other hand, nothing much is known about the composite in the reverse order because **TRV** is not invariant under **HULL[SYM]**; in general, the union of a transitive relation and its inverse need not be a transitive relation.

```
In[37]:= composite[HULL[SYM], HULL[TRV]]

Out[37]= composite[HULL[SYM], HULL[TRV]]

In[38]:= invariant[HULL[SYM], TRV]

Out[38]= subclass[image[HULL[SYM], TRV], TRV]
```

The symmetric hull of a relation is its union with its inverse, and the transitive hull is the union of the relation with all its finite powers. The equivalential hull of a relation is the smallest equivalence relation that contains the given relation, which is the same thing as the transitive hull of its symmetric hull.

a result about fix[HULL[x]]

In general **fix[HULL[x]]** behaves somewhat better than the closely related constructor **Aclosure[x]**. These constructors coincide for sets, and for many proper classes as well. Neither in general preserves intersections, but one does have the following weaker result:

```
In[39]:= SubstTest[implies, and[equal[u, fix[HULL[u]]], equal[v, fix[HULL[v]]]],
          equal[intersection[u, v], fix[HULL[intersection[u, v]]]],
          {u → fix[HULL[x]], v → fix[HULL[y]]}]

Out[39]= equal[fix[HULL[intersection[fix[HULL[x]], fix[HULL[y]]]]],
           intersection[fix[HULL[x]], fix[HULL[y]]]] == True

In[40]:= fix[HULL[intersection[fix[HULL[x_]], fix[HULL[y_]]]]] :=
          intersection[fix[HULL[x]], fix[HULL[y]]]
```

serendipity: a general result

The following general result was discovered in the course of reifying various inclusions involving **hull** expressions.

```
In[41]:= subclass[composite[x, E], E] // AssertTest

Out[41]= subclass[composite[x, E], E] == subclass[composite[Id, x], S]

In[42]:= subclass[composite[x_, E], E] := subclass[composite[Id, x], S]
```

The results obtained using reification turned out to be weaker than existing formulas, and so in the final analysis this rule was not needed. To illustrate, consider the following rule:

```
In[43]:= SubstTest[implies, subclass[w, x],
          subclass[hull[x, z], hull[w, z]], w → intersection[x, y]]

Out[43]= subclass[hull[x, z], hull[intersection[x, y], z]] == True

In[44]:= subclass[hull[x_, z_], hull[intersection[x_, y_], z_]] := True
```

Using reification, one obtains:

```
In[45]:= Map[equal[0, #] &, SubstTest[reify, z,
           dif[hull[x, z], hull[w, z]], w → intersection[x, y]]] // Reverse

Out[45]= subclass[composite[HULL[intersection[x, y]], inverse[HULL[x]]], S] == True
```

The same inclusion can also be derived without resorting to reification as follows:

```
In[46]:= SubstTest[implies, subclass[w, x],
          subclass[HULL[w], composite[S, HULL[x]]], w → intersection[x, y]]

Out[46]= subclass[HULL[intersection[x, y]], composite[S, HULL[x]]] == True

In[47]:= subclass[HULL[intersection[x_, y_]], composite[S, HULL[x_]]] := True

In[48]:= SubstTest[implies, subclass[u, v],
          subclass[composite[u, w], composite[v, w]],
          {u → HULL[intersection[x, y]],
           v → composite[S, HULL[x]], w → inverse[HULL[x]]}]

Out[48]= subclass[composite[HULL[intersection[x, y]], inverse[HULL[x]]], S] == True
```

The presence of **S** in these formulas makes them appear to be too weak to obtain significant results about **HULL[intersection[x,y]]**.

---

hull results

One can derive formulas for **hull[x,y]** analogous to those derived for the corresponding **HULL[x]** functions.

```
In[50]:= SubstTest[APPLY, composite[u, v], z,
           {u → HULL[x], v → HULL[intersection[x, y]]}] // Reverse

Out[50]= hull[x, hull[intersection[x, y], z]] == hull[intersection[x, y], z]

In[51]:= hull[x_, hull[intersection[x_, y_], z_]] := hull[intersection[x, y], z]

In[52]:= SubstTest[APPLY, composite[u, v], z,
           {u → HULL[intersection[x, y]], v → HULL[x]}] // Reverse

Out[52]= hull[intersection[x, y], hull[x, z]] == hull[intersection[x, y], z]

In[53]:= hull[intersection[x_, y_], hull[x_, z_]] := hull[intersection[x, y], z]
```

As before, one can replace **intersection** with **subclass**:

```
In[54]:= SubstTest[implies, equal[w, intersection[x, y]],
          equal[hull[w, hull[y, z]], hull[w, z]], w → x]

Out[54]= or[equal[hull[x, z], hull[x, hull[y, z]]], not[subclass[x, y]]] == True

In[55]:= or[equal[hull[x_, z_], hull[x_, hull[y_, z_]]], not[subclass[x_, y_]]] := True

In[56]:= SubstTest[implies, equal[w, intersection[x, y]],
          equal[hull[y, hull[w, z]], hull[w, z]], w → x]

Out[56]= or[equal[hull[x, z], hull[y, hull[x, z]]], not[subclass[x, y]]] == True

In[57]:= or[equal[hull[x_, z_], hull[y_, hull[x_, z_]]], not[subclass[x_, y_]]] := True
```